Date: Jan 7, 2022

Exp 1 (Basic arithmetic operations)

Aim: To verify the basic arithmetic operations using the 8086 processor by MASM 611 assembler.

Tool Used:
Assembler - MASM 611

Algorithm:

1. Inside the code segment, move the first value to the accumulator (AX) register.
2. Move the second number to the base (BX) register.
3. Perform add/sub/mul/div operation of the AX and BX register and store the final value in the accumulator.
4. Then halt the operation and end the code segment and the start of the segment.
5. Then run the code after assembling it.

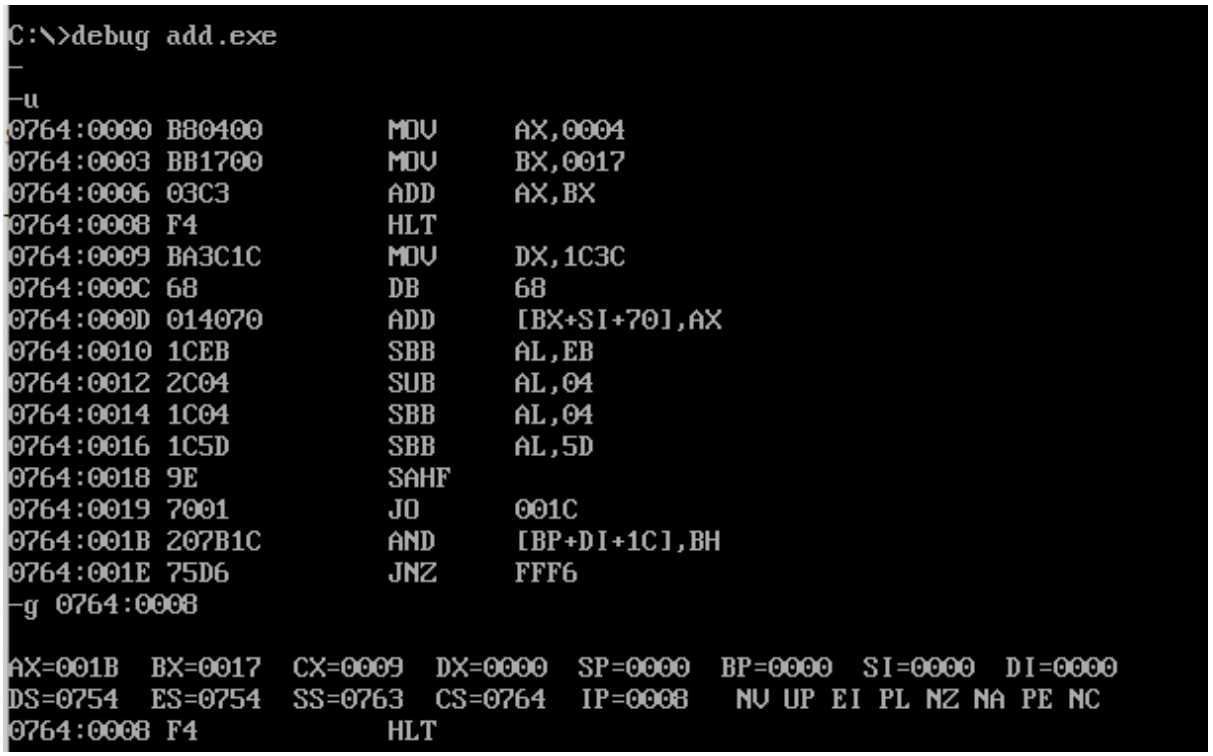
1. Addition:
Code:

```
File  Edit  Search  Options
                ADD.ASM
code segment
assume cs:code
start : mov ax, 04h
mov bx, 17h
add ax,bx
hlt
code ends
end
```

Sample Input: 04H, 17H
Sample Output: 1B

Register/ Memory Contents for I/O:
AX= 04H, BX=17H
After Operation: AX = 001B

Snapshot of the Output:



```
C:\>debug add.exe
-
-u
0764:0000 B80400        MOV     AX,0004
0764:0003 BB1700        MOV     BX,0017
0764:0006 03C3          ADD     AX,BX
0764:0008 F4            HLT
0764:0009 BA3C1C        MOV     DX,1C3C
0764:000C 68            DB      68
0764:000D 014070        ADD     [BX+SI+70],AX
0764:0010 1CEB          SBB     AL,EB
0764:0012 2C04          SUB     AL,04
0764:0014 1C04          SBB     AL,04
0764:0016 1C5D          SBB     AL,5D
0764:0018 9E            SAHF
0764:0019 7001          JO      001C
0764:001B 207B1C        AND     [BP+DI+1C],BH
0764:001E 75D6          JNZ     FFF6
-g 0764:0008

AX=001B  BX=0017  CX=0009  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=0754  ES=0754  SS=0763  CS=0764  IP=0008   NV UP EI PL NZ NA PE NC
0764:0008 F4            HLT
```

2. Subtraction:

Code:

```
File  Edit  Search  Options                                    Help
                        SUB.ASM
code segment
assume cs:code
start : mov ax, 07h
mov bx, 02h
sub ax,bx
hlt
code ends
end
```

Sample Input: 07H, 02H
Sample Output: 5
Register/ Memory Contents for I/O:
AX= 07H, BX=02H
After Operation: AX = 0005

Snapshot of the Output:

```
C:\>debug sub.exe
-u
0764:0000 B80700        MOV     AX,0007
0764:0003 BB0200        MOV     BX,0002
0764:0006 2BC3          SUB     AX,BX
0764:0008 F4            HLT
0764:0009 BA3C1C        MOV     DX,1C3C
0764:000C 68            DB      68
0764:000D 014070        ADD     [BX+SI+70],AX
0764:0010 1CEB          SBB     AL,EB
0764:0012 2C04          SUB     AL,04
0764:0014 1C04          SBB     AL,04
0764:0016 1C5D          SBB     AL,5D
0764:0018 9E            SAHF
0764:0019 7001          JO      001C
0764:001B 207B1C        AND     [BP+DI+1C],BH
0764:001E 75D6          JNZ     FFF6
-g 0764:0008

AX=0005  BX=0002  CX=0009  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=0754  ES=0754  SS=0763  CS=0764  IP=0008   NV UP EI PL NZ NA PE NC
0764:0008 F4            HLT
```

Multiplication:

Code:

```
File  Edit  Search  Options                                        Help
                              MUL.ASM
code segment
assume cs:code
start : mov ax, 04h
mov bx, 03h
mul bx
hlt
code ends
end
```

Sample Input: 04H, 03H
Sample Output: 1B
Register/ Memory Contents for I/O:
AX= 04H, BX=03H
After Operation: AX = 000C
Snapshot of the Output:

```
C:\>debug mul.exe
-u
0764:0000 B80400        MOV     AX,0004
0764:0003 BB0300        MOV     BX,0003
0764:0006 F7E3          MUL     BX
0764:0008 F4            HLT
0764:0009 BA3C1C        MOV     DX,1C3C
0764:000C 68            DB      68
0764:000D 014070        ADD     [BX+SI+70],AX
0764:0010 1CEB          SBB     AL,EB
0764:0012 2C04          SUB     AL,04
0764:0014 1C04          SBB     AL,04
0764:0016 1C5D          SBB     AL,5D
0764:0018 9E            SAHF
0764:0019 7001          JO      001C
0764:001B 207B1C        AND     [BP+DI+1C],BH
0764:001E 75D6          JNZ     FFF6
-g 0764:0008

AX=000C  BX=0003  CX=0009  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=0754  ES=0754  SS=0763  CS=0764  IP=0008   NV UP EI PL NZ NA PO NC
0764:0008 F4            HLT
```

Division:

Code:

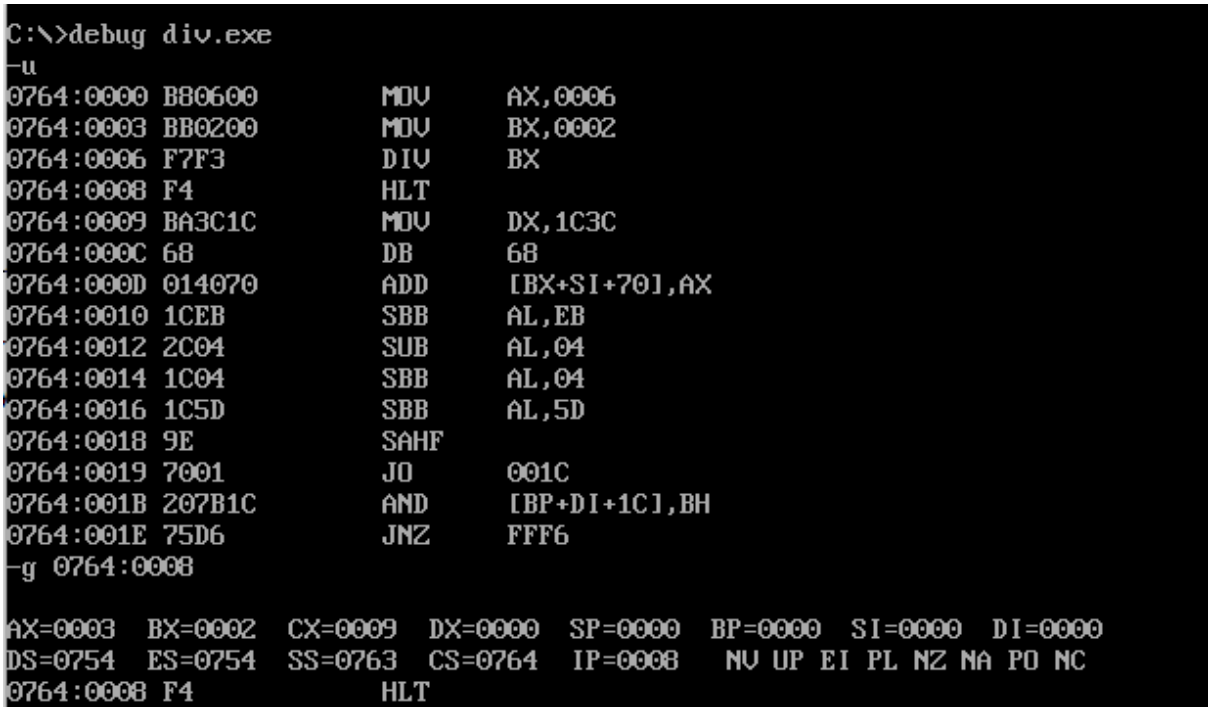
```
C:\>edit div.asm
```

```
File  Edit  Search  Options                                    Help
                          DIV.ASM
code segment
assume cs:code
start : mov ax, 06h
mov bx, 02h
div bx
hlt
code ends
end
```

Assembling and linking the file to create an executable file:

```
C:\>masm div.asm
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1993.  All rights reserved.

 Invoking: ML.EXE /I. /Zm /c /Ta div.asm

Microsoft (R) Macro Assembler Version 6.11
Copyright (C) Microsoft Corp 1981-1993.  All rights reserved.

 Assembling: div.asm

C:\>link div.obj

Microsoft (R) Segmented Executable Linker  Version 5.31.009 Jul 13 1992
Copyright (C) Microsoft Corp 1984-1992.  All rights reserved.

Run File [div.exe]:
List File [nul.map]:
Libraries [.lib]:
Definitions File [nul.def]:
LINK : warning L4021: no stack segment
LINK : warning L4038: program has no starting address
```

Sample Input: 06H, 02H
Sample Output: 3
Register/ Memory Contents for I/O:
AX= 06H, BX=02H
After Operation: AX = 0003
Snapshot of the Output:



Result:
Hence, all operation are verified using the MASM application using DOSBOX.

Reg no. :20BCE1161 **Name: Shubhangi Agrawal**

Date: Jan 21, 2022 **Exp 2 Average of an Array**

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Aim: To find the sum and average of an array using the 8086 processor by MASM 611 assembler.

Tool Used:
Assembler - MASM 611

Algorithm:

1. In the data segment, declare the array using data type DB (Define Byte) for 8-bit and DW (Define Word) for 16-bit values.
2. Inside the code segment, first, move the data to AX register and then to DATA SEGMENT.
3. Move the value 6 (counter value) to the CX register.
4. Initialize AX= 0H
5. Start the loop.
6. Add the first element of the array to the value in AX
7. Increase the SI value by one (for 8 bit) or two (for 16-bit) to get the index of the next element of the array.
8. Decrease the value of CX by one
9. Keep repeating step 6 to 8 till CX is reduced to 0.
10. Divide AX by array size and store in AX
11. Then halt the operation and end the code segment and the start of the segment.
12. Then run the code after assembling it.

8-bit:

Code:

File Edit Search Options

AVG.ASM

```
DATA SEGMENT
        NUM1 DB 2H,6H,3H,1H,5H,7H
DATA ENDS
CODE SEGMENT
        ASSUME DS:DATA, CS:CODE
START:
        MOV AX,DATA
        MOV DS,AX
        MOV CX,6
        LEA SI,NUM1
        MOV AX, 0000H
LOOP1:  ADD AX,[SI]
        INC SI
        DEC CX
        JNZ LOOP1
        MOV BL,06H
        MOV AH,00H
        DIV BL
        HLT
CODE ENDS
END START
```

Sample Input: 2H, 6H,3H,1H,5H,7H
Sample Output: R: 0H, Q: 4H (Q=24/6 = 4, R=0)
Register/ Memory Contents for I/O:
AX= 04H, BX=17H
After Operation: AX = 001B
Snapshot of the Output:

```
C:\>debug avg.exe
-u
0765:0000 B86407        MOV     AX,0764
0765:0003 8ED8          MOV     DS,AX
0765:0005 B90600        MOV     CX,0006
0765:0008 8D360000      LEA     SI,[0000]
0765:000C B80000        MOV     AX,0000
0765:000F 0304          ADD     AX,[SI]
0765:0011 46            INC     SI
0765:0012 49            DEC     CX
0765:0013 75FA          JNZ     000F
0765:0015 B306          MOV     BL,06
0765:0017 B400          MOV     AH,00
0765:0019 F6F3          DIV     BL
0765:001B F4            HLT
0765:001C 40            INC     AX
0765:001D 7D22          JGE     0041
0765:001F 6C            DB      6C
-g 0765:001B

AX=0004  BX=0006  CX=0000  DX=0000  SP=0000  BP=0000  SI=0006  DI=0000
DS=0764  ES=0754  SS=0763  CS=0765  IP=001B   NV UP EI PL ZR NA PE NC
0765:001B F4            HLT
```

16-bit:

AVG.ASM

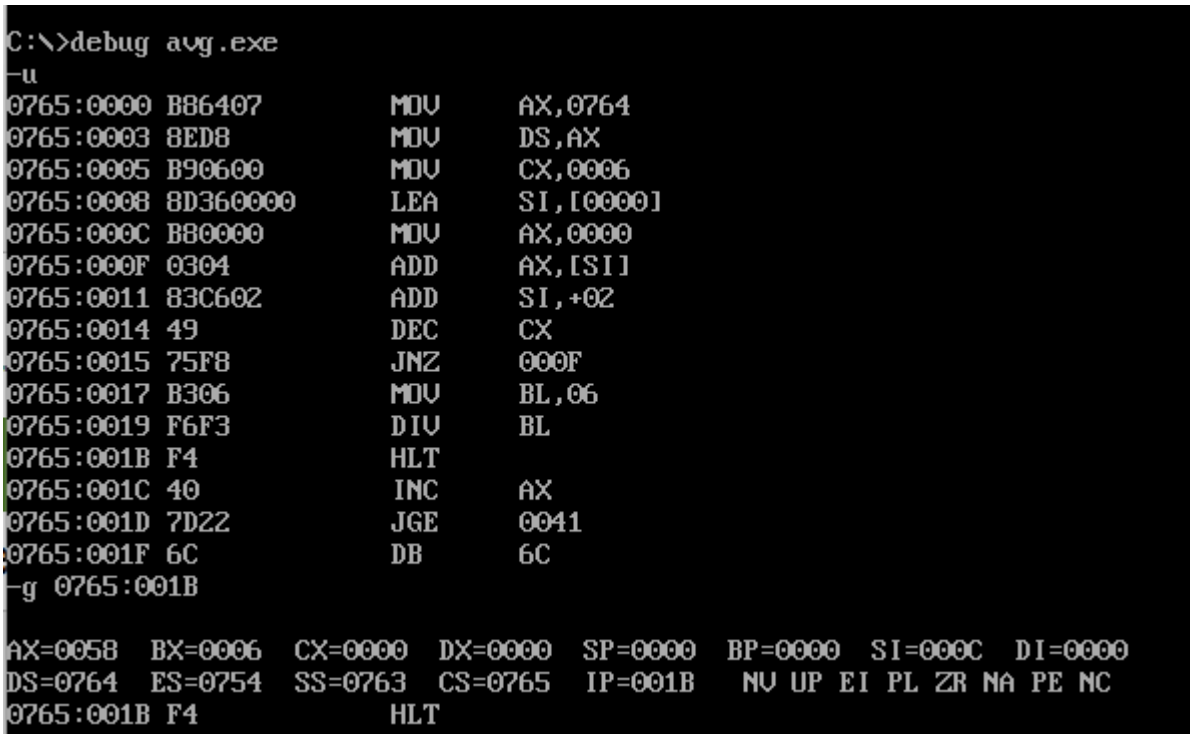
```
DATA SEGMENT
        NUM1 DW 10H,90H,40H,50H,60H,80H
DATA ENDS
CODE SEGMENT
        ASSUME DS:DATA, CS:CODE
START:
        MOV AX,DATA
        MOV DS,AX
        MOV CX,6
        LEA SI,NUM1
        MOV AX, 0000H
LOOP1:  ADD AX,[SI]
        ADD SI,02H
        DEC CX
        JNZ LOOP1
        MOV BL,06H
        DIV BL
        HLT
CODE ENDS
END START
```

Sample Input: 10H, 90H, 40H, 50H, 60H, 80H
Sample Output: 0058 (10H+ 90H+ 40H+ 50H+ 60H+ 80H)/(7H) = 210H/7H = 58H

Register/ Memory Contents for I/O:

AX= 0H, BX=6H, CX= 06H,
After operation: AX = 0058, BX=6H, CX=0H

Snapshot of the Output:



Result:
Hence, Hence, the required average was calculated (both 8-bit and 16-bit) using MASM on DOSBOX.

Reg no. :20BCE1161 **Name: Shubhangi Agrawal**

Date: Jan 28, 2022 **Exp 3 Bubble sort and max/min**

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Aim: To sort an array in ascending and descending order by Bubble sort using the 8086 processor by MASM 611 assembler and also find maximum and minimum in the array.

Tool Used:
Assembler - MASM 611

Algorithm:

1. In the data segment, define the array.
2. Inside the code segment, first, move the data to AX register and then to DATA SEGMENT.
3. Move the value 4 (counter value),i.e., size of the array to the CH register.
4. Start the loop 1.
5. Load effective address of array in source index register.
6. Start loop 2
7. Move the size of the array to CL register.
8. Move the value at source index register in AL register.
9. Move the value next to the source index register in the BL register.
10. Compare value at AL and BL register.
11. If value at AL register is greater than value at BL register, go to step 11, else swap the value at SI and SI+1
12. Increase the SI pointer
13. Decrease the value at CL register.
14. Continue step 7 to 10 until CL is reduced to zero.
15. Decrease the value at CH register.
16. Continue step 5 to 14 until CH is reduced to zero.
17. Increase the SI value by one to get the index of the next element of the array.
18. Decrease the value of CX by one
19. Keep repeating step 6 to 8 till CX is reduced to 0.
20. Then halt the operation and end the code segment and the start of the segment.
21. Then run the code after assembling it.

Ascending order:

```
DATA SEGMENT
STRING1 DB 99H,12H,56H,45H,36H
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX

MOV CH,04H

UP2: MOV CL,04H
LEA SI,STRING1

UP1: MOV AL,[SI]
MOV BL,[SI+1]
CMP AL,BL
JC DOWN
MOV DL,[SI+1]
XCHG [SI],DL
MOV [SI+1],DL
```

```
DOWN: INC SI
DEC CL
JNZ UP1
DEC CH
JNZ UP2


CODE ENDS
END START
```

Sample Input: 99H, 12H,56H,45H,36H
Sample Output: 12H 36H 45H 56H 99H
Register/ Memory Contents for I/O:

AX= 077A, BX=0099, CX=0000,DX=0045,SI=0004, DS=0764

Before operation: DS 0764:0000 0004 - 99H, 12H,56H,45H,36H
After operation: DS 0764:0000 0004 - 12H 36H 45H 56H 99H

```
C:\>debug asc.exe
-u
0765:0000 B86407        MOV     AX,0764
0765:0003 8ED8          MOV     DS,AX
0765:0005 B504          MOV     CH,04
0765:0007 B104          MOV     CL,04
0765:0009 8D360000      LEA     SI,[0000]
0765:000D 8A04          MOV     AL,[SI]
0765:000F 8A5C01        MOV     BL,[SI+01]
0765:0012 38D8          CMP     AL,BL
0765:0014 7208          JB      001E
0765:0016 8A5401        MOV     DL,[SI+01]
0765:0019 8614          XCHG    DL,[SI]
0765:001B 885401        MOV     [SI+01],DL
0765:001E 46            INC     SI
0765:001F FEC9          DEC     CL
-g

AX=077A  BX=0099  CX=0000  DX=0045  SP=20B8  BP=0000  SI=0004  DI=0000
DS=0764  ES=0754  SS=0000  CS=0000  IP=C0F4   NV UP EI PL NZ NA PO CY
0000:C0F4 CC            INT     3
-D 0764:0000 0004
0764:0000  12 36 45 56 99
```

Descending order:

```
DATA SEGMENT
STRING1 DW 99H,12H,56H,45H,36H
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX

MOV CH,04H

UP2: MOV CL,04H
LEA SI,STRING1

UP1: MOV AL,[SI]
MOV BL,[SI+1]
CMP AL,BL
JNC DOWN
MOV DL,[SI+1]
XCHG [SI],DL
```

```
MOV [SI+1],DL

DOWN: INC SI
DEC CL
JNZ UP1
DEC CH
JNZ UP2

INT 3
CODE ENDS
END START
```

Sample Input: 99H, 12H,56H,45H,36H
Sample Output: 99H 56H 45H 36H 12H
Register/ Memory Contents for I/O:

AX= 0736, BX=0012, CX=0000,DX=0012,SI=0004,DS=0764

```
C:\>debug desc.exe
-u
0765:0000 B86407        MOV     AX,0764
0765:0003 8ED8          MOV     DS,AX
0765:0005 B504          MOV     CH,04
0765:0007 B104          MOV     CL,04
0765:0009 8D360000      LEA     SI,[0000]
0765:000D 8A04          MOV     AL,[SI]
0765:000F 8A5C01        MOV     BL,[SI+01]
0765:0012 38D8          CMP     AL,BL
0765:0014 7308          JNB     001E
0765:0016 8A5401        MOV     DL,[SI+01]
0765:0019 8614          XCHG    DL,[SI]
0765:001B 885401        MOV     [SI+01],DL
0765:001E 46            INC     SI
0765:001F FEC9          DEC     CL
-g

AX=0736  BX=0012  CX=0000  DX=0012  SP=0000  BP=0000  SI=0004  DI=0000
DS=0764  ES=0754  SS=0763  CS=0765  IP=0027   NV UP EI PL ZR NA PE NC
0765:0027 CC            INT     3
-D 0764:0000 0004
0764:0000  99 56 45 36 12                                .VE6.
```

Maximum and Minimum:

Algorithm:

1. the data segment, define the array.

2. Inside the code segment, first, move the data to AX register and then to DATA SEGMENT.
3. Move the value 5 (counter value),i.e., size of the array to the CL register.
4. Load effective address of array in source index register.
5. Initialize BL register with 0H or 0FFH for maximum or minimum respectively.
6. Start the loop 1.
7. Move the first element of the array in AL register
8. If the value in the current index is greater or smaller for maximum or minimum respectively than value in BL, go to step 11
9. Increase the SI pointer to go to next value
10. Decrease the loop counter value
11. If the loop counter value is not equal to zero, go back to step 6, else stop
12. Move the value in AL register to BL register
13. Go back to step 8

Maximum:

MAX.ASM

```
DATA SEGMENT
STRING1 DB 5H,2H,6H,9H,3H
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE, DS:DATA
START:
MOV AX,DATA
MOV DS,AX
MOV AH,00H
MOV CL,05H
MOV BL,0H
LEA SI,STRING1
LOOP1:
MOV AL,[SI]
CMP AL,BL
JNC REPLACE
BACK:
INC SI
DEC CL
JNZ LOOP1
HLT
```

Sample Input: 5H, 2H,6H,9H,3H
Sample Output: 9H (stored in BX register)
Register/ Memory Contents for I/O:

AX= 0003, BX=0009, DS=0764
Before operation: BX= 0H
After operation: BX = 09H

```
C:\>debug max.exe
-u
0765:0000 B86407        MOV     AX,0764
0765:0003 8ED8          MOV     DS,AX
0765:0005 B400          MOV     AH,00
0765:0007 B105          MOV     CL,05
0765:0009 B300          MOV     BL,00
0765:000B 8D360000      LEA     SI,[0000]
0765:000F 8A04          MOV     AL,[SI]
0765:0011 38D8          CMP     AL,BL
0765:0013 7306          JNB     001B
0765:0015 46            INC     SI
0765:0016 FEC9          DEC     CL
0765:0018 75F5          JNZ     000F
0765:001A F4            HLT
0765:001B 8AD8          MOV     BL,AL
0765:001D EBF6          JMP     0015
0765:001F 6C            DB      6C
-g 0765:001A

AX=0003  BX=0009  CX=0000  DX=0000  SP=0000  BP=0000  SI=0005  DI=0000
DS=0764  ES=0754  SS=0763  CS=0765  IP=001A   NV UP EI PL ZR NA PE CY
0765:001A F4            HLT
```

Minimum:

MIN.ASM

```
DATA SEGMENT
STRING1 DB 5H,2H,6H,9H,3H
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE, DS:DATA
START:
MOV AX,DATA
MOV DS,AX
MOV AH,00H
MOV CL,05H
MOV BL,0FFH
LEA SI,STRING1
LOOP1:
MOV AL,[SI]
CMP AL,BL
JC REPLACE
BACK:
INC SI
DEC CL
JNZ LOOP1
HLT
```

Sample Input: 5H, 2H,6H,9H,3H

Sample Output: 9H (stored in BX register)
Register/ Memory Contents for I/O:

AX= 0003, BX=0002, DS=0764, SI= 0005
Before operation: BX= 0H
After operation: BX = 02H

```
C:\>debug min.exe
-u
0765:0000 B86407        MOV     AX,0764
0765:0003 8ED8          MOV     DS,AX
0765:0005 B400          MOV     AH,00
0765:0007 B105          MOV     CL,05
0765:0009 B3FF          MOV     BL,FF
0765:000B 8D360000      LEA     SI,[0000]
0765:000F 8A04          MOV     AL,[SI]
0765:0011 38D8          CMP     AL,BL
0765:0013 7206          JB      001B
0765:0015 46            INC     SI
0765:0016 FEC9          DEC     CL
0765:0018 75F5          JNZ     000F
0765:001A F4            HLT
0765:001B 8AD8          MOV     BL,AL
0765:001D EBF6          JMP     0015
0765:001F 6C            DB      6C
-g 0765:001A

AX=0003  BX=0002  CX=0000  DX=0000  SP=0000  BP=0000  SI=0005  DI=0000
DS=0764  ES=0754  SS=0763  CS=0765  IP=001A   NV UP EI PL ZR NA PE NC
0765:001A F4            HLT
```

Result:
Hence, the given array is sorted in ascending and descending order and maximum and minimum are found using MASM on DOSBOX.

Reg no. :20BCE1161 **Name: Shubhangi Agrawal**

Date: Jan 28, 2022

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Exp 4 Permutation and Combination

Aim: To find the Permutation and Combination with given n and r using the 8086 processor by MASM 611 assembler.

Tool Used:
Assembler - MASM 611

Algorithm:

1. In the data segment the offset value has been set to 1000h.
2. The value of n and r have been initialised to the variables.
3. “temp”, “tmp”,”tmpp” stores (n-r)!, n!, r! respectively.
4. The value of permutation is stored in npr and value of combination is stored in ncr.
5. End of data segment.
6. In the code segment, register BX holds the value (n-r), and this value is compared with value 1, if its equal to 1, temp stores 1 and loop2 is executed else loop1 is executed.
7. In loop1, it calls for fact procedure to calculate the factorial (n-r).
8. After the loop1 is halted the value in AX register is moved to tmp variable.
9. The loop2 calls fact to calculate the factorial of n.
10. The loop3 calls fact to calculate the factorial of r.
11. The tmp value is moved to AX register and temp value is moved to BX register
12. The content of AX is divided by BX and quotient is moved to npr variable.
13. The content of AX register is cleared.
14. Again, the value npr is moved to AX register, and value of tmpp is moved to BX register,
15. The content of AX register is divided by the content of the BX register, and the result is stored in the ncr variable.
16. Then halt the operation and end the code segment and the start of the segment.
17. Then run the code after assembling it.

Code

PERMCOMB.ASM

```
assume cs:code,ds:data
data segment
org 1000h
n equ 05h
r equ 02h
temp dw ? ; stores (n-r)!
tmp dw ?; stores n!
tmpp dw ? stores r!
npr dw ?
ncr dw ?
data ends
code segment
start:
mov ax,data
mov ds,ax
xor ax,ax
xor dx,dx; cleared for 16-bit division
mov bx,n-r
cmp bx,01h
jnz lp1
mov temp,01h
```

PERMCOMB.ASM

```
jmp lp2
lp1:call fact
mov temp,ax
xor ax,ax
lp2:mov bx,n
call fact
mov tmp,ax
xor bx,bx
lp3:mov bx,r
call fact
mov tmpp,ax
xor ax,ax
mov ax,tmp
mov bx,temp
div bx ; (dx|ax)/bx, quo-ax
mov npr,ax
xor ax,ax
mov ax,npr
mov bx,tmpp
div bx
mov ncr,ax
```

```
hlt
fact proc
mov cx,00h
mov cx,bx
dec cx
mov ax, bx
proc_lp: dec bx
mul bx
loop proc_lp
ret
fact endp
code ends
end start
```

Sample Input:

N= 5

R= 2

Sample Output:

nPr= 5P2 = 20 = 14h

nCr= 5C2= 10= 0Ah

Register/ Memory Contents for I/O:

AX= 000A BX= 0002 CX= 0000 DX=0000 SP=0000 BP=0000 SI=0000

```
0865:0043 33C0          XOR     AX,AX
0865:0045 A10610        MOV     AX,[1006]
0865:0048 8B1E0410      MOV     BX,[1004]
0865:004C F7F3          DIV     BX
0865:004E A30810        MOV     [1008],AX
0865:0051 F4            HLT
0865:0052 B90000        MOV     CX,0000
0865:0055 8BCB          MOV     CX,BX
0865:0057 49            DEC     CX
0865:0058 8BC3          MOV     AX,BX
0865:005A 4B            DEC     BX
0865:005B F7E3          MUL     BX
0865:005D E2FB          LOOP    005A
0865:005F C3            RET
0865:0060 0E            PUSH    CS
0865:0061 53            PUSH    BX
0865:0062 E8EE00        CALL    0153
-g 0051

AX=000A  BX=0002  CX=0000  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=0764  ES=0754  SS=0763  CS=0865  IP=0051   NV UP EI PL ZR NA PE NC
0865:0051 F4            HLT
```

```
-d 0764:1000 100A
0764:1000  06 00 78 00 02 00 14 00-0A 00 00                ..x........
```

Result:

Hence,the permutation and combination is found with given value of n and r using MASM on DOSBOX.

Reg no. :20BCE1161 Name: Shubhangi Agrawal

Date: Feb 26, 2022 Exp 5 Logical Operation

Aim: To perform logical operations (AND,OR,NOT,XOR,,SHL,SHR) and conversion of BCD to Hexadecimal and BCD to ASCII.

Tool Used:

Assembler - MASM 611

Algorithm(AND,OR,NOT,XOR):

1. Move a number in AX register (here FFF0H)
2. Move another number in BX register (here 0000H)
3. Perform NOT operation of AX, so AX stores the result of NOT operation
4. Perform OR operation of BX and AX and store in BX, so BX stores the result of OR operation
5. Move a number in DX register and perform AND with another number (here 1 and 1), so DX stores the result of AND operation.
6. Move some number in CX register and perform XOR with another number(here 0 and value in AX register), so CX stores the result of XOR operation.
7. Then halt the operation and end the code segment and the start of the segment.
8. Then run the code after assembling it.

AND,OR,NOT,XOR

logical.asm

```
1    CODE SEGMENT
2    ASSUME CS:CODE
3    START:
4    MOV AX,0FFF0H ;AX=FFF0
5    MOV BX,0000H ;BX=0
6    NOT AX ;AX=000F
7    OR BX,AX ;BX= AX OR BX =000F OR 0000 -> BX=000F
8    MOV DX,01H
9    AND DX,01H      ; DX= 1 AND 1 =1
10   MOV CX,0000H
11   XOR CX,AX ; CX= CX XOR AX -> 0000 XOR 000F -> 000F
12   HLT
13   CODE ENDS
14   END START
15
16
```

Sample Input: AX=FFF0H, BX=0000H,CX=0000H,DX=0001H
Sample Output: AX= 000F, BX=000FH,CX=000FH,DX=0001H
Register/ Memory Contents for I/O:

AX=FFF0H, BX=0000H,CX=0000H,DX=0001H,SI=0000, DS=0754

Before operation:AX=FFF0H, BX=0000H,CX=0000H,DX=0001H
After operation: AX= 000F, BX=000FH,CX=000FH,DX=0001H

```
C:\>debug logical.exe
-u
0764:0000 B8F0FF        MOV     AX,FFF0
0764:0003 BB0000        MOV     BX,0000
0764:0006 F7D0          NOT     AX
0764:0008 0BD8          OR      BX,AX
0764:000A BA0100        MOV     DX,0001
0764:000D 83E201        AND     DX,+01
0764:0010 B90000        MOV     CX,0000
0764:0013 33C8          XOR     CX,AX
0764:0015 F4            HLT
0764:0016 1C5D          SBB     AL,5D
0764:0018 9E            SAHF
0764:0019 7001          JO      001C
0764:001B 207B1C        AND     [BP+DI+1C],BH
0764:001E 75D6          JNZ     FFF6
-g 0015

AX=000F  BX=000F  CX=000F  DX=0001  SP=0000  BP=0000  SI=0000  DI=0000
DS=0754  ES=0754  SS=0763  CS=0764  IP=0015   NV UP EI PL NZ NA PE NC
0764:0015 F4            HLT
```

Algorithm(SHL,SHR):

1. Move a number in BX register.
2. Perform SHL operation of AX, with a count, so AX stores the result of SHL operation
3. Move a number in AX register.
4. Perform SHR operation of AX, with a count, so AX stores the result of SHR operation
5. Then halt the operation and end the code segment and the start of the segment.
6. Then run the code after assembling it.

SHL,SHR:

logical.asm

```
1    CODE SEGMENT
2    ASSUME CS:CODE
3    START:
4    MOV BX,05H
5    SHL BX,01H
6    MOV AX,05H
7    SHR AX,01H
8    HLT
9    CODE ENDS
10   END START
```

Sample Input: 5
Sample Output: SHL: 10, SHR: 2
Register/ Memory Contents for I/O:

```
C:\>debug logical.exe
-u
0764:0000 BB0500        MOV     BX,0005
0764:0003 D1E3          SHL     BX,1
0764:0005 B80500        MOV     AX,0005
0764:0008 D1E8          SHR     AX,1
0764:000A F4            HLT
0764:000B 1C68          SBB     AL,68
0764:000D 014070        ADD     [BX+SI+70],AX
0764:0010 1CEB          SBB     AL,EB
0764:0012 2C04          SUB     AL,04
0764:0014 1C04          SBB     AL,04
0764:0016 1C5D          SBB     AL,5D
0764:0018 9E            SAHF
0764:0019 7001          JO      001C
0764:001B 207B1C        AND     [BP+DI+1C],BH
0764:001E 75D6          JNZ     FFF6
-g 000A

AX=0002  BX=000A  CX=000B  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=0754  ES=0754  SS=0763  CS=0764  IP=000A   NV UP EI PL NZ AC PO CY
0764:000A F4            HLT
```

Before operation:AX=0005, BX=0005
After operation: AX= 0002, BX=000A

Algorithm(BCD to ASCII):

1. Move the number to be converted to ASCII in the AL register.
2. Copy the content of AL register in BL register.
3. Store 4 in CL register
4. Perform ROR operation on AL register to store first 4 bits in AL register.
5. Add the contents of AL register with 30H, which is the hex value of ascii of '0', so AL register contains the ASCII of first digit.
6. Store the AL register content in AH register.
7. Now, to do the same for the 2nd digit, we clear the first 4 bits of BL register by performing AND operation with 0FH.
8. Now add 30h to BL register.
9. Store the value got in BL register to AL register.
10. Now AX register contains the ASCII form which we wanted
11. Then halt the operation and end the code segment and the start of the segment.
12. Then run the code after assembling it.

b2ascii.asm

```
1    code segment
2    assume cs:code
3    start: mov al,07h
4    mov bl,al
5    mov cl,04h
6    ror al,cl
7    and al,0fh
8    add al,30h
9    mov ah,al
10   and bl,0fh
11   add bl,30h
12   mov al,bl
13   HLT
14   code ends
15   end start
```

Sample Input: 07H
Sample Output: 3037H
Register/ Memory Contents for I/O:

```
C:\>debug b2ascii.exe
-u
0764:0000 B007          MOV     AL,07
0764:0002 8AD8          MOV     BL,AL
0764:0004 B104          MOV     CL,04
0764:0006 D2C8          ROR     AL,CL
0764:0008 240F          AND     AL,0F
0764:000A 0430          ADD     AL,30
0764:000C 8AE0          MOV     AH,AL
0764:000E 80E30F        AND     BL,0F
0764:0011 80C330        ADD     BL,30
0764:0014 8AC3          MOV     AL,BL
0764:0016 F4            HLT
0764:0017 5D            POP     BP
0764:0018 9E            SAHF
0764:0019 7001          JO      001C
0764:001B 207B1C        AND     [BP+DI+1C],BH
0764:001E 75D6          JNZ     FFF6
-g 0016

AX=3037  BX=0037  CX=0004  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=0754  ES=0754  SS=0763  CS=0764  IP=0016   NV UP EI PL NZ NA PO NC
0764:0016 F4            HLT
```

Before operation:AL= 07,BL=07
After operation: AX= 3037

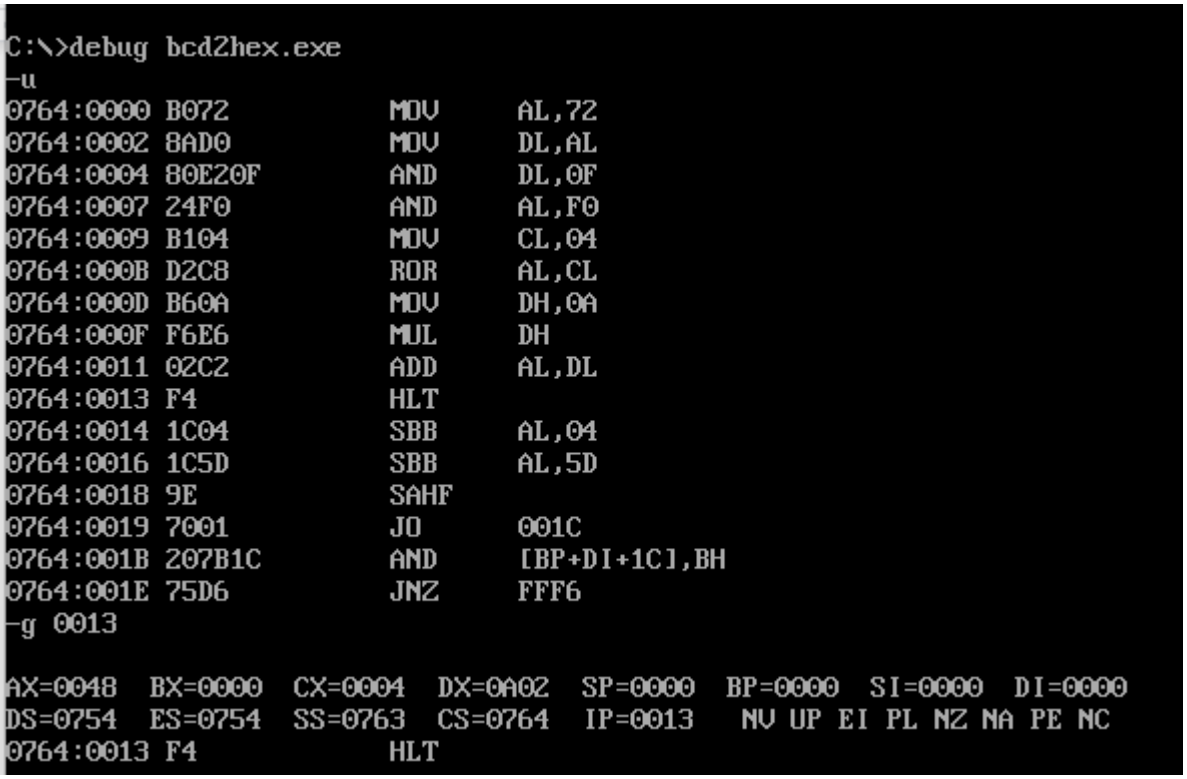
Algorithm(BCD to HEX):

1. Move the number to be converted to HEX in the AL register.
2. Copy the content of AL register in the DL register.
3. Perform AND operation of DL register with 0fh to clear higher 4 bits, so we got the lower 4 bits in DL register.
4. Perform AND operation of AL register with f0h to clear lower 4 bits, so we got the higher 4 bits in AL register.
5. Perform ROR operation by 4 count to AL register, so that we get the 4 bits got in step 4 stored in the lower 4 bits
6. Now multiply the higher 4 with 10 (that is stored in AL register) and add it with the lower 4 bits which in turn will be stored in the register in hexadecimal.
13. Now AX register contains the hexadecimal form which we wanted
14. Then halt the operation and end the code segment and the start of the segment.

15. Then run the code after assembling it.

bcd2hex.asm

```
1    assume cs:code
2    code segment
3    start:
4    mov al,72h
5    mov dl,al
6    and dl,0fh
7    and al,0f0h
8    mov cl,4
9    ror al,cl
10   mov dh,0AH
11   mul dh
12   add al,dl
13   HLT
14   code ends
15   end start
16
```



Result:

Hence, logical operations are performed and conversion of BCD to Hexadecimal and BCD to ASCII are done using MASM on DOSBOX.

Reg no. :20BCE1161 Name: Shubhangi Agrawal

Date: March 3, 2022

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Exp 6 - Length and Reverse of a string

Aim: To find the length of a string and reverse a string with a given string using the 8086 processor by MASM 611 assembler.

Tool Used:
Assembler - MASM 611

Algorithm (length of the string):

1. In the data segment, define the string.
2. Inside the code segment, first, move the data to AX register and then to DATA SEGMENT.
3. Move the value 0 (initialise length with 0) to the CX register.
4. Move '$' in the AL register (the end of the string contains a dollar sign).
5. Load effective address of string in source index register.
6. Start the loop
7. If value at SI register equals value in AL register, i.e. '$' sign, then go to step 11, else go to step 8.
8. Increment the source index register value.
9. Increase the CX register value (that stores the length of the string).
10. Go to step 7.
11. Then halt the operation and end the code segment and the start of the segment.
12. Then run the code after assembling it.

lenstr.asm

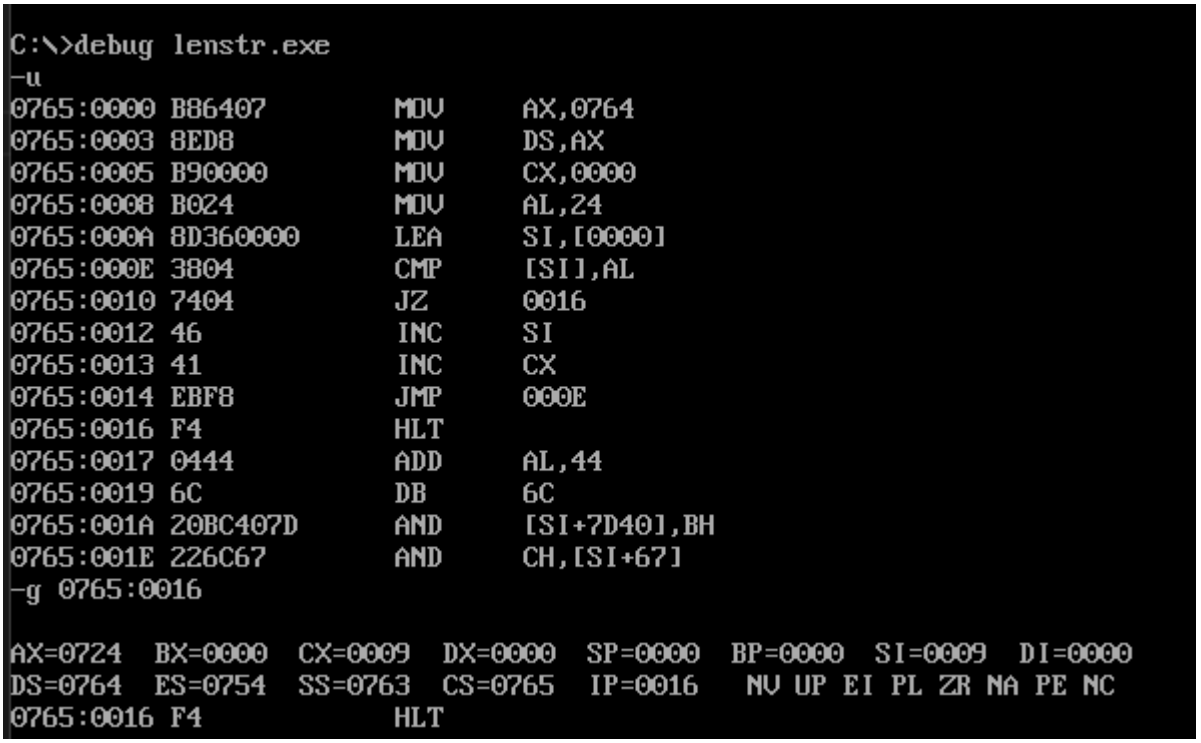
```
1   data segment
2   string db "20bce1161$"
3   data ends
4   code segment
5   assume cs:code, ds:data
6   start:
7   mov ax,data
8   mov ds,ax
9   mov cx,0h
10  mov al,'$'
11  lea si,string
12  label1:
13  cmp [si],al
14  je label2
15  inc si
16  inc cx
17  jmp label1
18  label2:
19  hlt
20  code ends
21  end start
22
```

Sample Input: 20bce1161
Sample Output: CX= 9 (length of string)

Register/ Memory Contents for I/O:

AX= 0724, BX=0000, CX=0009,DX=0000,SI=0009, DS=0764

Before operation: CX= 0000
After operation: CX=0009



Algorithm (reverse of the string):

1. In the data segment, define the string.
2. Inside the code segment, first, move the data to AX register and then to DATA SEGMENT.
3. Move the length of the string to the BX register (to be used as the index from the end of the string with which we need to swap) .
4. Load effective address of string in source index register.
5. Move length/2 in AX register.
6. Move 0 to CX register (loop counter value)
7. Start the loop
8. Move value in SI register in DL register.
9. Swap/ Exchange the value in DL register and value at effective address of SI+BX register.(So, basically we're swapping ith and (n-ith) index of string)
10. Now move the value at DL register to the address at source index register.
11. Decrement the value at BX register by 2.
12. Increase the loop counter value, i.e., CX register.
13. Increment the source index register to go the next index in the string.
14. If loop counter value (CX) is less than length/2, i.e., stored in AX register, go to step 8, else go to step 15.

15. Halt the operation and end the code segment and the start of the segment.
16. Then run the code after assembling it.

ASM revstr.asm

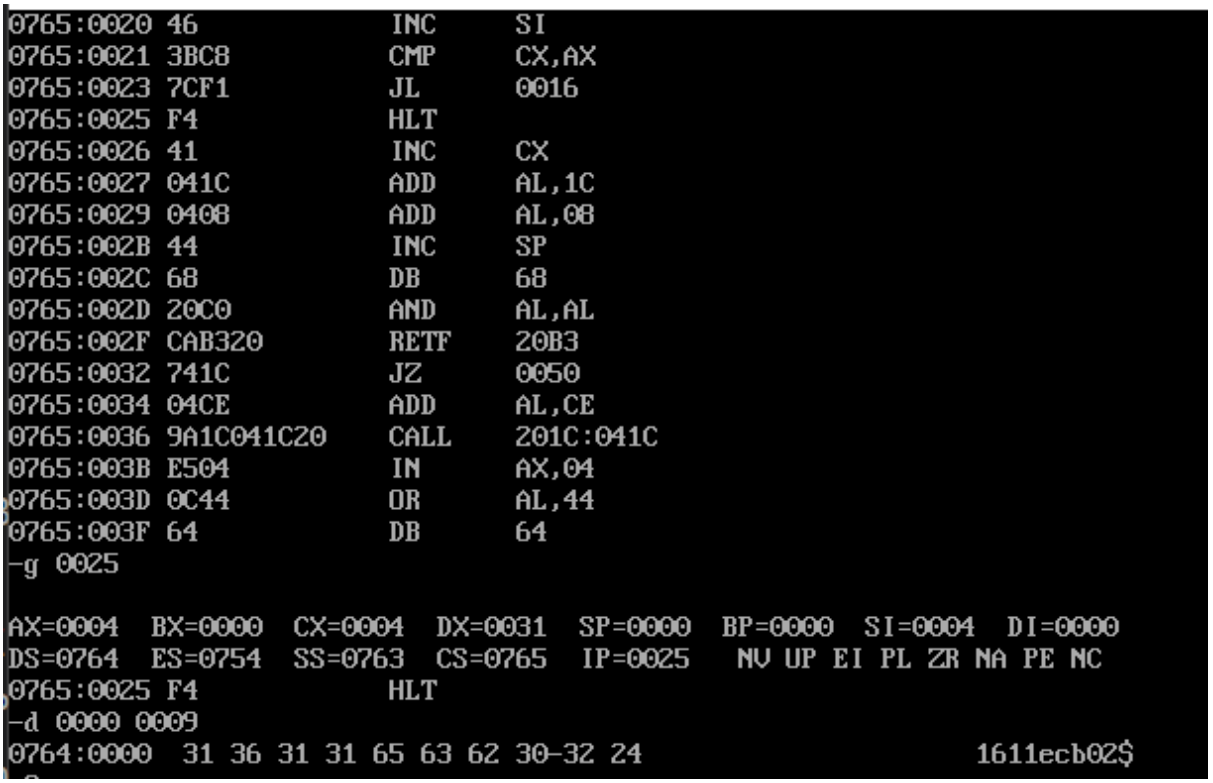
```
1    data segment
2    string db "20bce1161$"
3    data ends
4    code segment
5    assume cs:code, ds:data
6    start:
7    mov ax,data
8    mov ds,ax
9    mov bx,8h
10   lea si,string
11   mov ax,bx
12   mov cx,02h
13   div cx
14   mov cx,0h
15   label1:
16   MOV DL,[SI]
17   XCHG DL,[SI+bx]
18   MOV [SI],DL
19   sub bx,2
20   inc cx
21   inc si
22   cmp cx,ax
23   jl label1
24   hlt
25   code ends
26   end start
```

Output:

Sample Input: 20bce1161
Sample Output: 1611ecb02
Register/ Memory Contents for I/O:

AX= 0004, BX=0000, CX=0004,DX=0031,SI=0004, DS=0764

Before operation: DS 0764:0000 0009 - 20bce1161$
After operation: DS 0764:0000 0009 - 1611ecb02$



Result:
Hence, the length of the given is found and the string is reversed using MASM on DOSBOX.

Reg no. :20BCE1161 Name: Shubhangi Agrawal

Date: March 5, 2022 Exp 7 Square root & Parity check

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Aim: To find the square root of a given number and also find the parity of the number using the 8086 processor by MASM 611 assembler

Tool Used:
Assembler - MASM 611

Algorithm:

1. Move the number in the CX register.
2. Move 0 in AX register, loop counter
3. Start the loop
4. Increase AX by 1
5. Copy the content of AX in BX register.
6. Multiply AX with AX, i.e., square of AX
7. If AX is equal to or greater than the number of which we want to find the square root, go to step 10, else continue
8. Restore the value of loop counter by copying the content in BX to AX register.
9. Go to step 4.
10. Then halt the operation and end the code segment and the start of the segment.
11. Then run the code after assembling it.

Square root of a number:

sqrtn.asm

```
assume cs:code
code segment
start:
mov cx,9h
mov ax,0h
label1:
inc ax
mov bx,ax
mul ax
cmp ax,cx
jae label2
mov ax,bx
jmp label1
label2:
HLT
code ends
end start
```

Sample Input:

AX = 09H

Sample Output:

BX = 0003

Register/ Memory Contents for I/O:

AX=0009, BX=0003, CX=0009, DX=0000, DS=0754

```
C:\>debug sqrtn.exe
-u
0764:0000 B90900        MOV     CX,0009
0764:0003 B80000        MOV     AX,0000
0764:0006 40            INC     AX
0764:0007 8BD8          MOV     BX,AX
0764:0009 F7E0          MUL     AX
0764:000B 3BC1          CMP     AX,CX
0764:000D 7304          JNB     0013
0764:000F 8BC3          MOV     AX,BX
0764:0011 EBF3          JMP     0006
0764:0013 F4            HLT
0764:0014 1C04          SBB     AL,04
0764:0016 1C5D          SBB     AL,5D
0764:0018 9E            SAHF
0764:0019 7001          JO      001C
0764:001B 207B1C        AND     [BP+DI+1C],BH
0764:001E 75D6          JNZ     FFF6
-g 0013

AX=0009  BX=0003  CX=0009  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=0754  ES=0754  SS=0763  CS=0764  IP=0013   NV UP EI PL ZR NA PE NC
0764:0013 F4            HLT
```

Algorithm (Parity of a number):

1. In the data segment, define the messages to be displayed(odd parity/even parity).
2. Inside the code segment, first, move the data to AX register and then to DATA SEGMENT.
3. Move the number to the CX register and 0 to the BX register (holding the count of set bits).
4. Start loop1
5. If the number equals zero, go to step 9.
6. Shift rotate CX by 1 to check the last bit.
7. If the carry flag is 1 then increase the counter, else continue
8. Go to step 5.
9. Perform AND operation of BX with 1 to check whether it's odd or even
10. If the count of set bits which is stored in BX is odd, print odd, else print even
11. End the code segment and the start of the segment.
12. Then run the code after assembling it.

Code:

parity.asm

```
1   data segment
2   msg1 db "even$"
3   msg2 db "odd$"
4   data ends
5   assume cs:code
6   code segment
7   start:
8   mov ax,data
9   mov ds,ax
10  mov cx,2h
11  mov bx,0h
12  loop1:
13  cmp cx,0h
14  je stop
15  shr cx,1
16  jc cnt
17  jmp loop1
18  cnt:
19  inc bx
20  jmp loop1
21  stop:
22  AND BX,0001h
23  cmp bx,0001h
24  je odd
25  lea dx,msg1
26  ans:
27  mov ah,09h
28  int 21h
29  mov ah,4ch
30  int 21h
31  odd:
32  lea dx,msg2
33  jmp ans
34  code ends
35  end start
```

Sample Input:

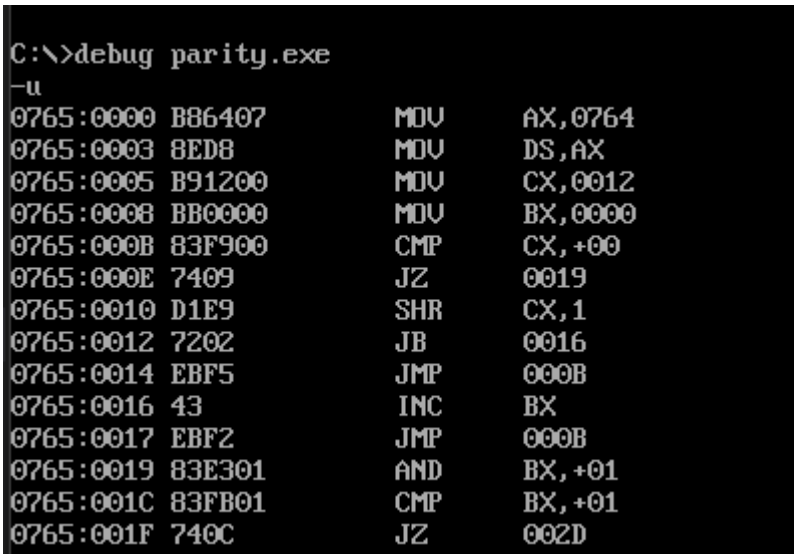
AX = 12h

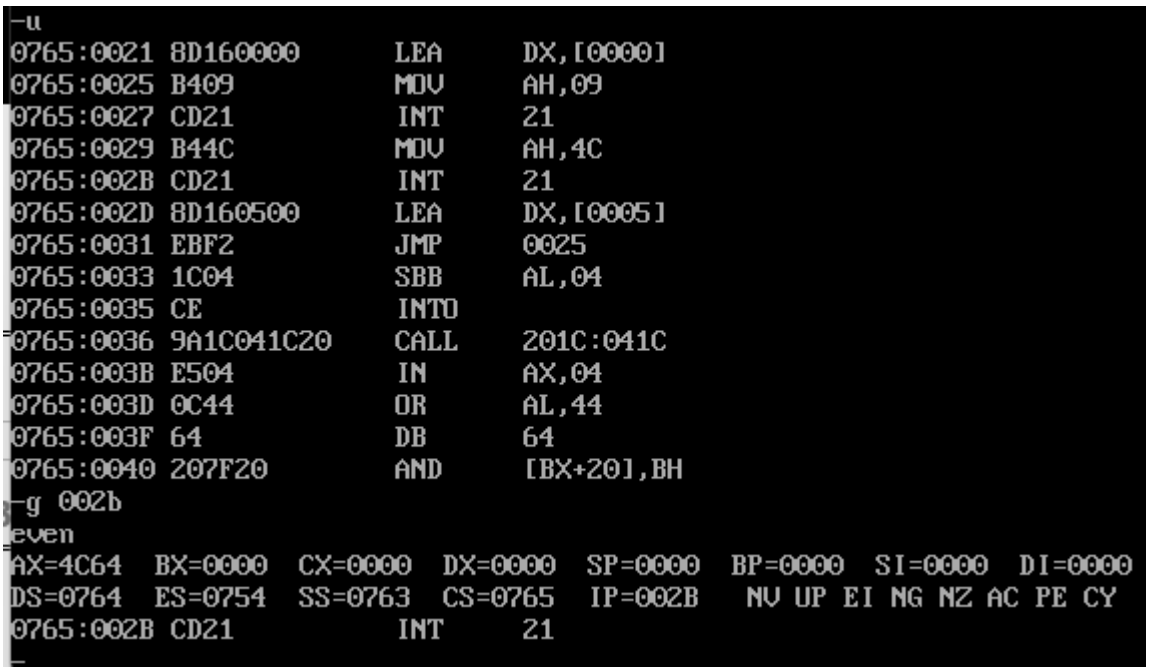
Sample Output:

even

Register/ Memory Contents for I/O:

AX=4C64, BX=0000, CX=0012, DX=0005, DS=0764





Result:

Hence, the square root of a given number is found and also parity of a given number is found using MASM on DOSBOX.

Reg no. :20BCE1161 **Name: Shubhangi Agrawal**

Date: March 25, 2022

Exp 8 Area of Square

Aim: To find the area of the square with the given side using 8087 processor by MASM 611 assembler.

Tool Used:
Assembler - MASM 611

Algorithm:

1. In the data segment the side value is initialised to a value and sqarea is defined for area of square.
2. Data is moved to AX register and then to data segment register
3. Finit command is given to signal the initialization of 8087 commands and registers.
4. The value in side variable is loaded and stored in st(4)
5. The value at st(4) is multiplied with st(4) and loaded at top of stack st(0).
6. Store the value in the sqarea variable.
7. Then halt the operation and end the code segment and the start of the segment.
8. Then run the code after assembling it.

Code

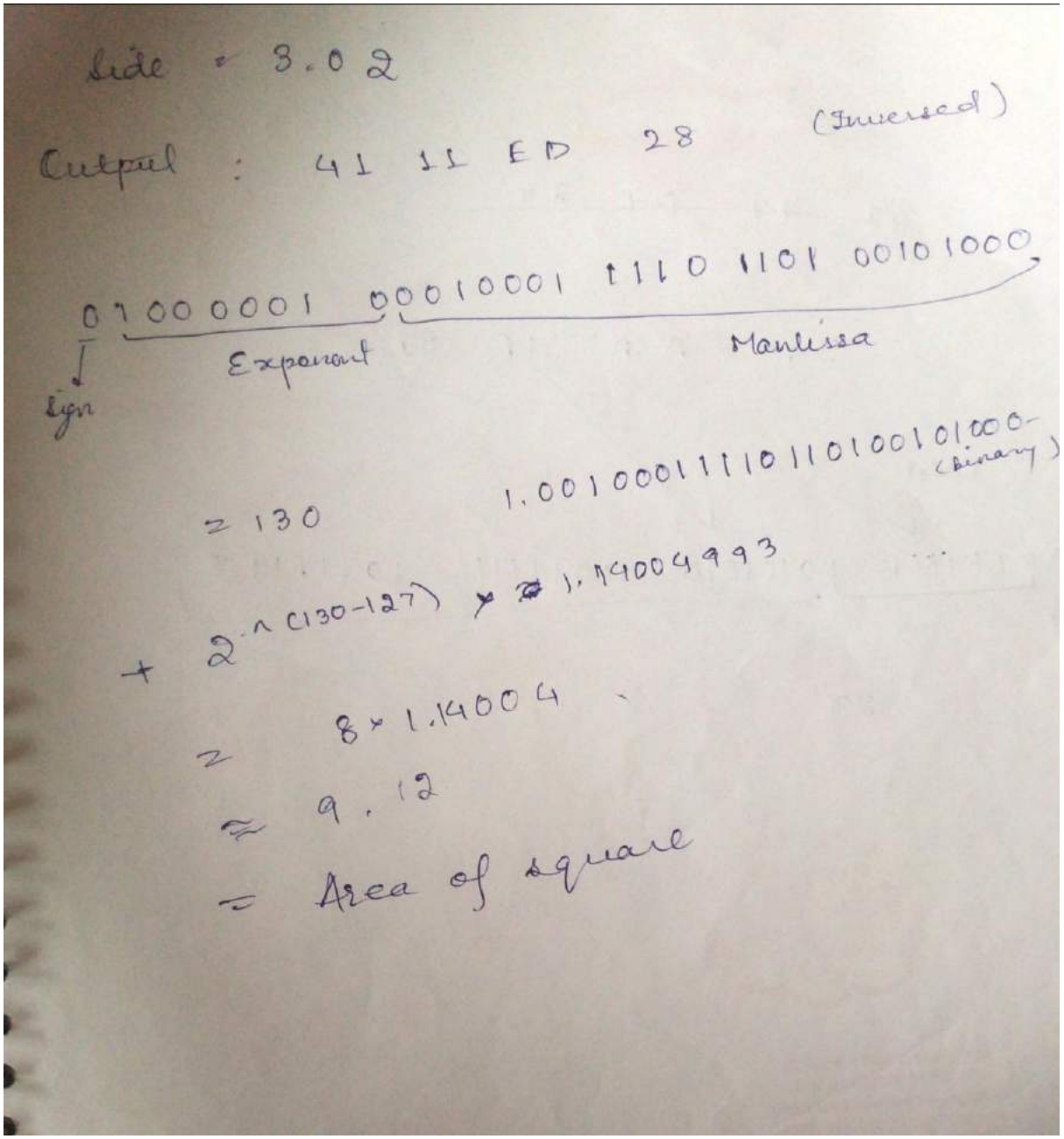
SQAREA.ASM

```
1
2    data segment
3    side dd 3.02
4    sqarea dd 01 dup(?)
5    data ends
6    code segment
7    assume cs:code, ds:data
8    .8087
9    start:
10   mov ax, data
11   mov ds, ax
12   finit
13   fld side
14   fst st(4)
15   fmul st(0), st(4)
16   fst sqarea
17   hlt
18   code ends
19   end start
```

Sample Input:
Side = 3.02
Sample Output:
Area = 9.04
Register/ Memory Contents for I/O:
AX= 0764 BX= 0000 CX= 0029 DX=0000 SP=0000 SI=0000

```
-u
0765:0000 B86407        MOV     AX,0764
0765:0003 8ED8          MOV     DS,AX
0765:0005 9B            WAIT
0765:0006 DBE3                  FINIT
0765:0008 9B            WAIT
0765:0009 D9060000              FLD     DWORD PTR [0000]
0765:000D 9B            WAIT
0765:000E DDD4                  FST     ST(4)
0765:0010 9B            WAIT
0765:0011 D8CC                  FMUL    ST,ST(4)
0765:0013 9B            WAIT
0765:0014 D9160400              FST     DWORD PTR [0004]
0765:0018 F4            HLT
0765:0019 6C            DB      6C
0765:001A 20BC407D      AND     [SI+7D40],BH
0765:001E 226C67        AND     CH,[SI+67]
-g 0765:0018

AX=0764  BX=0000  CX=0029  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=0764  ES=0754  SS=0763  CS=0765  IP=0018   NV UP EI PL NZ NA PO NC
0765:0018 F4            HLT
-D 0764:0000 0007
0764:0000  E0 BE A0 40 4D DE C9 41                              ...@M..A
```

Manual calculation



Result:
Hence,the area of square is found with a given value of side using MASM on DOSBOX.

Reg no. :20BCE1161 **Name: Shubhangi Agrawal**

Date: March 25, 2022

Exp 9 Volume of sphere

Aim: To find the volume of sphere with the given radius using 8087 processor by MASM 611 assembler.

Tool Used:
Assembler - MASM 611

Algorithm:

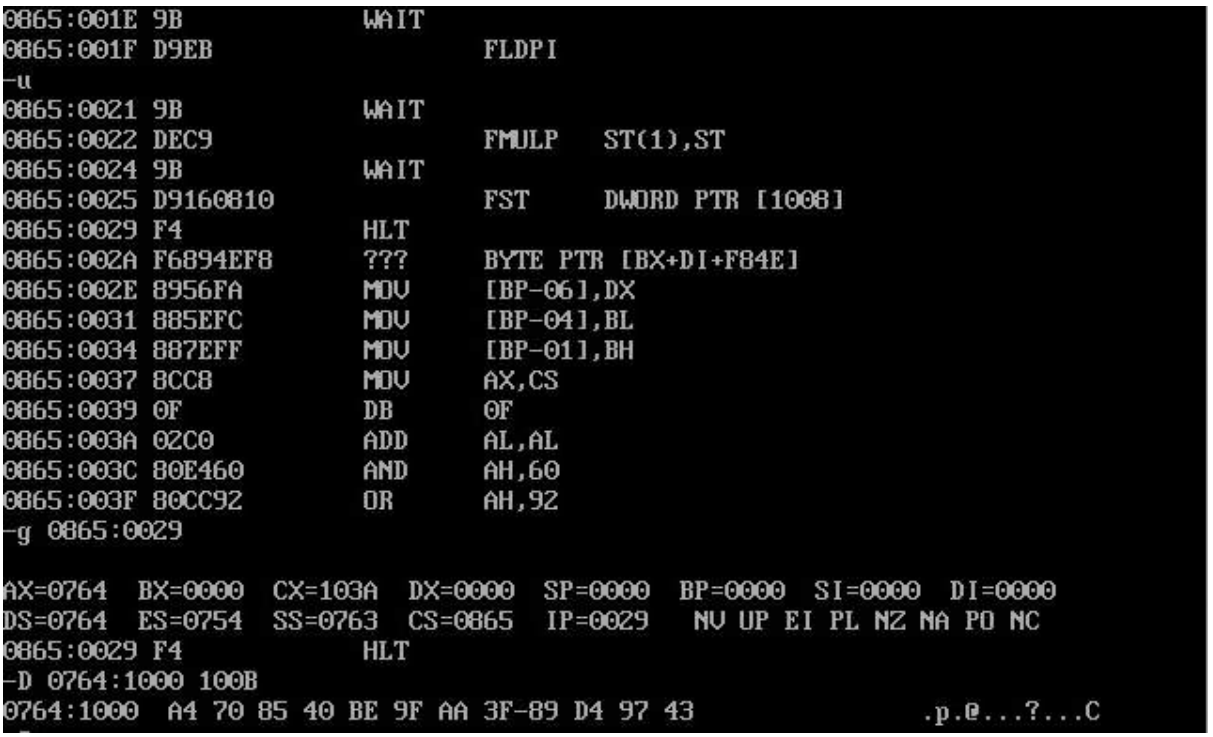
1. In the data segment the radius variable is initialised to a value, const value as 1.333, and volume is defined for volume of sphere.
2. Data is moved to AX register and then to data segment register
3. Finit command is given to signal the initialization of 8087 commands and registers.
4. The value in radius variable is loaded and stored in st(4)
5. The value at st(4) is multiplied with st(4) and loaded at top of stack st(0) and this is done twice as we need r to the power 3.
6. Const value is loaded (4/3) and multiplied with resultant value at st(0).
7. Pi value is loaded and multiplied with resultant value at st(0).
6. Store the resultant value in the volume variable.
7. Then halt the operation and end the code segment and the start of the segment.
8. Then run the code after assembling it.

Code

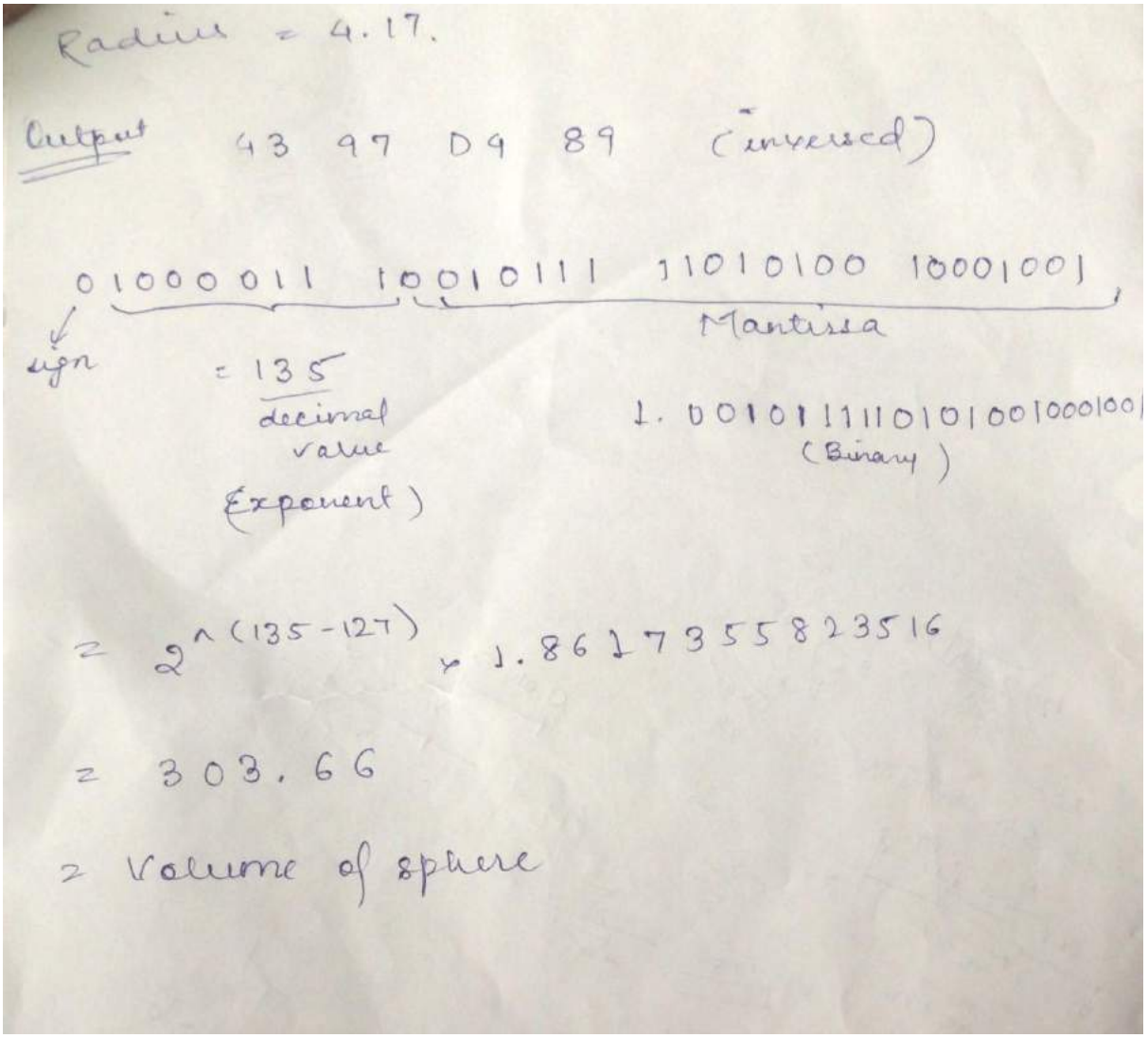
spvol2.asm

```
1    data segment
2    org 1000h
3    radius dd 4.17
4    const dd 1.333
5    volume dd 01 dup(?)
6    data ends
7    code segment
8    assume cs:code,ds:data
9    .8087
10   start:
11   mov ax,data
12   mov ds,ax
13   finit
14   fld radius
15   fst st(4)
16   fmul st(0),st(4)
17   fmul st(0),st(4)
18   fld const
19   fmul
20   fldpi
21   fmul
22   fst volume
23   hlt
24   code ends
25   end start
26   end
```

Sample Input:
Side = 4.17
Sample Output:
Area = 303.74
Register/ Memory Contents for I/O:
AX= 0764 BX= 0000 CX= 0029 DX=0000 SP=0000 SI=0000



Manual calculation



Result:
Hence,the volume of sphere is found with a given value of radius using MASM on DOSBOX.

Reg no. :20BCE1161 **Name: Shubhangi Agrawal**

Date: April 8, 2022

Exp 10 Arithmetic Operations - Vi 8086 Kit

Aim: To perform Arithmetic Operations using the Vi 8086 Kit.

Tool Used:
Vi 8086 Kit

Algorithm:

1. Enter the Program
    a) Switch on the kit
    b) Check Display: It will show #
    c) Press A (Line Assembler) and then Enter
    d) Give Start Address: 1000, Output Address: 1500
    e) 1000: Load the program
2. Execute the program
    a) Press reset
    b) Write Go 1000 and then press Enter. It shows Executing
    c) Press Reset
    d) Enter #SB and the output Memory Location (here 1500)
    e) Enter GB and start address and end address
    f) SB Enter 1500 and check the output

Addition

```
MOV AX, 2561
MOV BX, 5561
ADD AX,BX
HLT
```

Sample Input : AX=2561, BX=5561
Sample Output : AX=7AC2

Register/ Memory Contents for I/O:
AX= 7AC2, BX= 5561

Subtraction

MOV AX, 6517
MOV BX,0003
SUB AX,BX
HLT

Sample Input : AX=6517, BX=0003
Sample Output : AX=6514

Register/ Memory Contents for I/O:
AX= 6514, BX= 0003

Multiplication

MOV AX, 0020
MOV BX,0008
MUL BX
HLT

Sample Input : AX=0020, BX=0008
Sample Output : AX=0100

Register/ Memory Contents for I/O:
AX= 0100, BX= 0008

Division

MOV AX, 0008
MOV BX,0004
DIV BX
HLT

Sample Input : AX=0008, BX=0004
Sample Output : AX=0002 (AH= 0 (remainder) , AL =2 (quotient))

Register/ Memory Contents for I/O:
AX= 0002, BX= 0004

Result:

Hence, arithmetic Operations were performed using the Vi 8086 ToolBox and results are got.

Reg no. :20BCE1161 **Name: Shubhangi Agrawal**

Date: April 8, 2022

Exp 11 - LCM and GCD - Vi 8086 Kit

Aim: To find LCM and GCD using Vi 8086 Kit.

Tool Used:
Vi 8086 Kit

Algorithm:

1. Enter the Program
    a) Switch on the kit
    b) Check Display: It will show #
    c) Press A (Line Assembler) and then Enter
    d) Give Start Address: 1000, Output Address: 1500
    e) 1000: Load the program
2. Execute the program
    a) Press reset
    b) Write Go 1000 and then press Enter. It shows Executing
    c) Press Reset
    d) Enter #SB and the output Memory Location (here 1500)
    e) Enter GB and start address and end address
    f) SB Enter 1500 and check the output

GCD

```
MOV SI, 1500H
MOV AX,0006
MOV BX,0008
CMP AX,BX
JE 1019
JA 1014
XCHG AX,BX
SUB AX,BX
JMP 100C
MOV [SI],AX
HLT
```

Sample Input : AX = 0006H, BX = 0008H
Sample Output : 1500 - 02
1501 - 00
So, GCD: 0002

OUTPUT:

LCM

```
MOV SI,1500H
MOV AX,0012H
MOV BX,0008H
CMP AX,BX
JE 1019
JA 1014
XCHG AX,BX
SUB AX,BX
JMP 100C
MOV [SI],AX
MOV AX,0012H
MOV BX,0008H
MUL BX
MOV BL,[SI]
DIV BL
MOV [SI+02H],AX
HLT
```

Sample Input : AX = 0012H, BX = 0008H
Sample Output : 0048

Result:
Hence, Hence, LCM and GCD was found using the Vi 8086 ToolBox

Reg no. :20BCE1161 **Name: Shubhangi Agrawal**

Date: April 9, 2022

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Exp 12 -7 Segment Display - Vi 8086 Kit

Aim: To display numbers from 0-7 on the 7 segment display using the Vi 8086 Kit.

Tool Used:

Vi 8086 Kit
7 segment display

Algorithm:

1. Enter the Program
    a) Switch on the kit
    b) Check Display: It will show #
    c) Press A (Line Assembler) and then Enter
    d) Give Start Address: 1000, Output Address: 1500
    e) 1000: Load the program
2. Execute the program
    a) Press reset
    b) Enter SB 1200 to enter HEX values for loading numbers 0-7
    c) Press Reset
    d) Enter #SB and the output Memory Location (here 1500)
    e) Give the values 3F, 06,4B,4F,66,6D,7D,07 for each byte from 1200 and press enter every time you enter a hex value.
    f) Press Reset
    g) Enter GO 1000
    h) It will show “Executing….” on 8086 kit and the seven segment display will display the numbers 0-7.

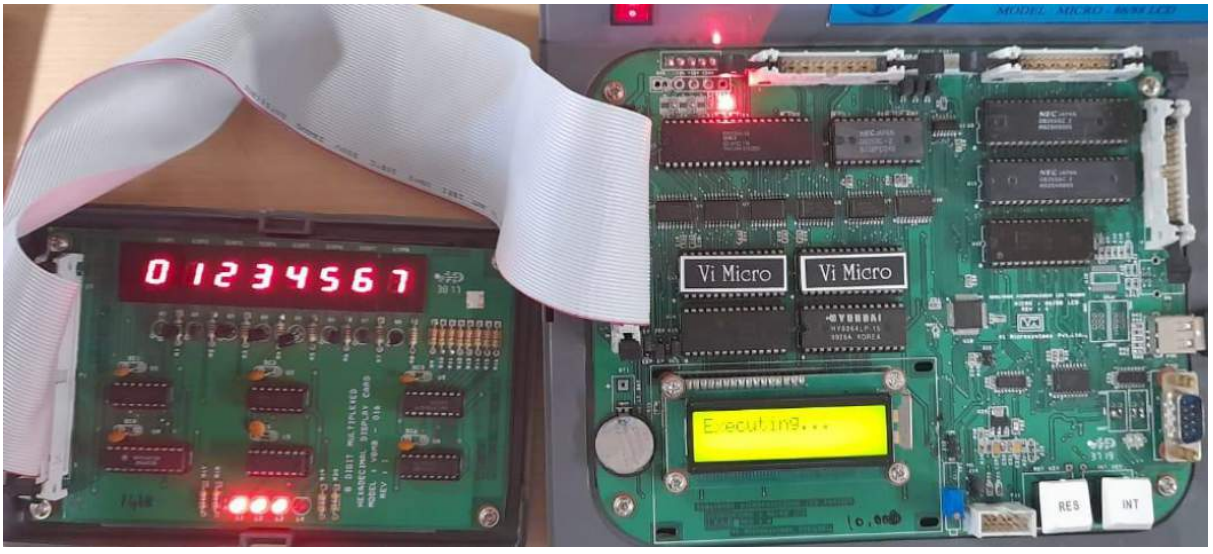
GCD

```
MOV SI,1200H
MOV BL,08H
DEC BL
```

```
MOV AL,BL
OUT C0,AL
MOV AL,[SI]
OUT C8,AL
CALL 1020
INC SI
CMP BL,00
JNZ 1007
JMP 1000
MOV CL,02
MOV AL,0FF
DEC AL
JNZ 1026
DEC CL
JNZ 1023
RET
```

Sample Input : Sample Input is explained in the procedure.
Sample Output : 0 1 2 3 4 5 6 7

OUTPUT:



Result:
Hence the numbers from 0-7is displayed on the 7 segment display using the Vi 8086 Kit.

Date: 22-4-2022 Exp13 Analog to Digital Convertor

Aim:

To interface ADC with the 8086 processor kit

Tool Used:

8086 kit, ADC kit, Multimeter, System bus

Algorithm:

1. Switch on the kit.
2. Interface the 8086 kit with ADC converter using system bus.
3. In the 8086 kit, check display, it'd be showing #
4. Press A (Line assembler)
5. Start address: 1000
6. 1000: Load the program
7. Move 10H in AL register to enable address bus, i.e., ALE high
8. Transfer the data from accumulator register to I/O port C8.
9. Move 18H in AL register to enable data bus, i.e., ALE low
10. Transfer the data from accumulator register to I/O port C8.
11. Halt the program.
12. Now to execute the program, press reset on 8086 kit and the enter GO 100
13. The 8086 kit shows "Executing…"
14. Now, vary the potentiometer and record the output from LEDs while observing the voltage using a multimeter.

Program:

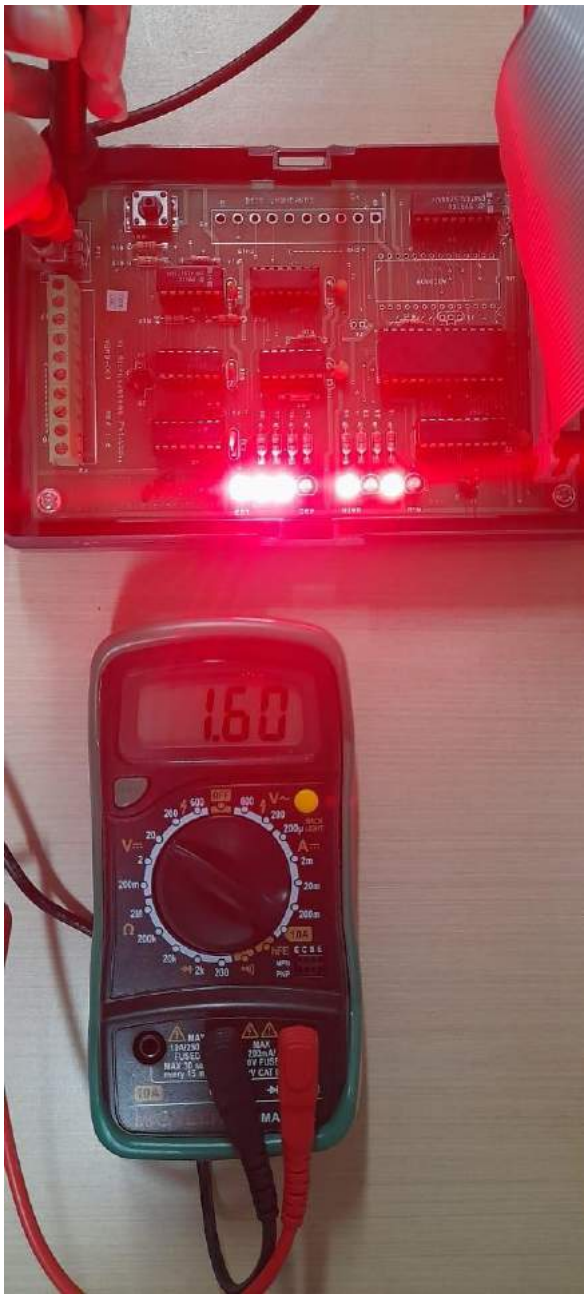
```
MOV AL, 10H        ;ALE Low
OUT C8, AL
MOV AL, 18H        ;ALE High
OUT C8, AL
```

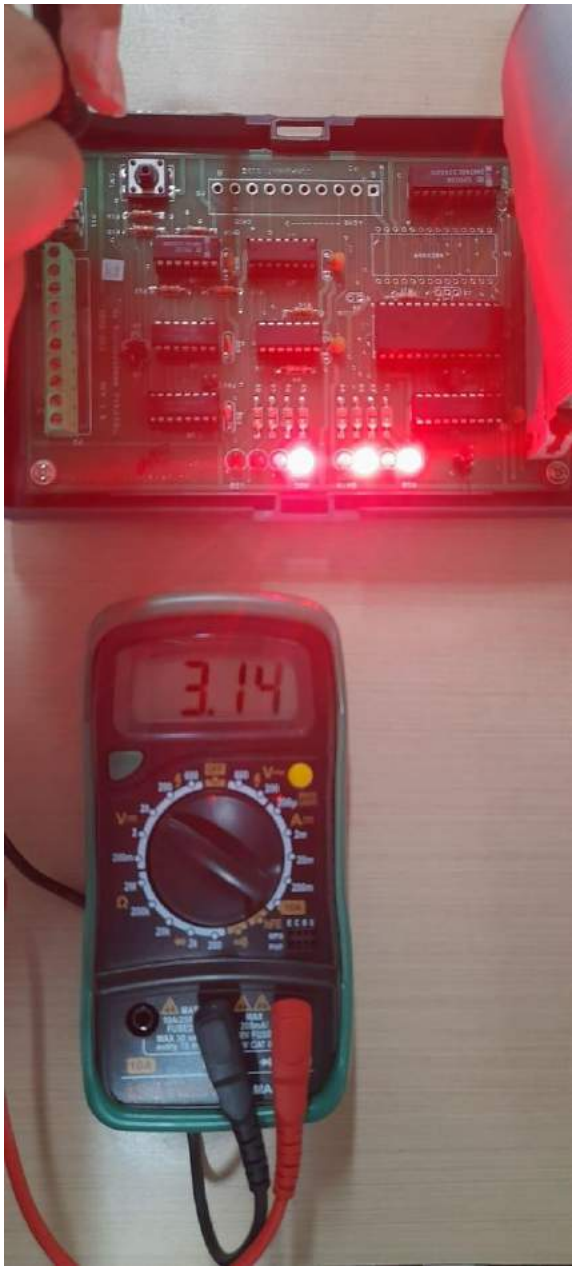
HLT

Sample input: Different values of voltages as noted in potentiometer in the images.

Sample output: Digital signals in binary form in the 8 LEDs representing 8 bits based on the values of voltages given.

Snapshot of the Output:





Result: Hence the ADC was interfaced with Vi 8086 kit and ADC conversion is verified.