Exp No:1

8 bit arithmetic operations

Name: Shreya Sriram
Register Number:195001106
Date: 12/07/2021

AIM:

To perform 8 bit operations – addition, subtraction, multiplication, division

PROCEDURE FOR EXECUTING MASM:

1. Open DosBox, mount masm to d drive

```
Z:\>mount D D:/masm
Drive D is mounted as local directory D:/masm\
```

2. Navigate to d drive

```
Z:\>d:
```

3. Assemble code

```
D:\>masm 8BITADD.ASM
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1993.  All rights reserved.

 Invoking: ML.EXE /I. /Zm /c /Ta 8BITADD.ASM

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

 Assembling: 8BITADD.ASM
```

4. Link the object file to create .exe file(executable)

```
D:\>link 8BITADD.OBJ

   Microsoft Object Linker V2.01 (Large)
(C) Copyright 1982, 1983 by Microsoft Inc.

Run File [8BITADD.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
Warning: No STACK segment

There was 1 error detected.
```

5. Run the .exe file using debug mode

6. Use commands like

a) –u (to view unassembled code)
b) –d segment:offset to display values of variables in memory
c) –e segment:offset to change values of variable
d) –g to execute code

7. –q to quit from debug mode
8. Exit to quit dosbox

ALGORITHM:

Addition:

a. Declare variables – 2 operands a and b, res(to store result), carry in data segment
b. Move first operand to ah (half word accumulator) and second operand to bh
c. Use add ah,bh to store the result in ah
d. Store 1 in ch if carry exists, else branch to e (using branch)
e. Move the contents from ah to res, and ch to carry
f. Terminate the program

Subtraction:

a. Declare variables – 2 operands a and b, res(to store result), carry in the data segment
b. Move first operand to ah (half word accumulator) and second to bh
c. Use sub ah,bh to store result in accumulator
d. If there is a carry, store 1 in ch and perform neg ah (to obtain 2s complement). Else branch to e
e. Move result from ah to res and ch to carry

Multiplication:

a. Declare variables – 2 operands a and b, result (16 bit) in the data segment
b. Move first operand to al and second to bl (16 bit accumulators)
c. Use mul bl to perform multiplication and store result in ax accumulator
d. Move result from ax to result

Division:

a. Declare the variables – 2 operands a and b, res(to store result), remainder in data segment
b. Move first operand a to al and second operand to bl. Initialize ah to 0

c. Use div bl to store the result in al. (ax/bl is performed)
d. Remainder is stored in ah
e. Move contents of al to result, and ah to remainder

PROGRAM

Addition:

PROGRAM	**COMMENTS**
assume cs:code,ds:data	Declare code segment as code, data segment as data
data segment	Begin data segment
a db 11h	Initialize a to 11
b db 99h	Initialize b to 99
res db 00H	Initialize res to 0
carry db 00H	Initialize carry to 0
data ends	Data segment ends
code segment	Begin code segment
org 0100h	Set origin as 100
start: mov ax,data	Move contents from data to ax
mov ds,ax	Move contents from ax to ds
mov ah,a	Move a to ah
mov bh,b	Move b to bh
mov ch,00h	Initialize ch(to store carry) to 0
add ah,bh	Add ah+bh and store in ah
jnc here	If no carry, jump to label here
inc ch	Else,Increment carry
here: mov res,ah	Move contents from ah to res
mov carry,ch	Move contents from ch to carry
mov ah,4ch	Move 4ch to ah
int 21h	This step generates an interrupt for terminating the program
code ends	Code segment ends
end start	End of start

Subtraction:

PROGRAM	COMMENTS
assume cs:code,ds:data	Declare code segment as code, data segment as data
data segment	Begin data segment
a db 11h	Initialize a to 11
b db 99h	Initialize b to 99
res db 00H	Initialize res to 0
carry db 00H	Initialize carry to 0
data ends	Data segment ends
code segment	Begin code segment
org 0100h	Set origin as 100
start: mov ax,data	Move contents from data to ax
mov ds,ax	Move contents from ax to ds
mov ah,a	Move a to ah
mov bh,b	Move b to bh
mov ch,00h	Initialize ch(to store carry) to 0
sub ah,bh	Perform ah-bh and store in ah
jnc here	If no carry, jump to label here
inc ch	Else, Increment carry
neg ah	Find twos complement of ah
here: mov res,ah	Move contents from ah to res
mov carry,ch	Move carry from ch to carry
mov ah,4ch	Move 4ch to ah
int 21h	This step generates an interrupt for terminating the program
code ends	Code segment ends
end start	End of start

Multiplication:

PROGRAM	COMMENTS
assume cs:code,ds:data	Declare code segment as code, data segment as data
data segment	Begin data segment
a db 11h	Initialize a to 11
b db 99h	Initialize b to 99
res db 00H	Initialize res to 0
data ends	Data segment ends
code segment	Begin code segment
org 0100h	Set origin as 100
start: mov ax,data	Move contents from data to ax
mov ds,ax	Move contents from ax to ds

mov al,a	Move a to al
mov bl,b	Move b to bl
mov ch,00h	Initialize carry to 0
mul bl	Perform ax*bl and store in ax
here: mov res,ax	Move contents from ax to res
mov ah,4ch	Move 4ch to ah
int 21h	This step generates an interrupt for terminating the program
code ends	Code segment ends
end start	End of start

Addition:

PROGRAM	COMMENTS
assume cs:code,ds:data	Declare code segment as code, data segment as data
data segment	Begin data segment
a db 11h	Initialize a to 11
b db 99h	Initialize b to 99
res db 00H	Initialize res to 0
data ends	Data segment ends
code segment	Begin code segment
org 0100h	Set origin as 100
start: mov ax,data	Move contents from data to ax
mov ds,ax	Move contents from ax to ds
mov ah,00h	Initialize ah to 0
mov al,a	Move a to al
mov bl,b	Move b to bl
div bl	Perform al/bl and store quotient in al
here: mov res,al	Move contents from al to res
mov remainder,ah	Move carry from ah to remainder
mov ah,4ch	Move 4ch to ah
int 21h	This step generates an interrupt for terminating the program
code ends	Code segment ends
end start	End of start

OUTPUT SCREENSHOTS

1) Addition

```
D:\EX1>masm 8bitadd.asm
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1993.  All rights reserved.

 Invoking: ML.EXE /I. /Zm /c /Ta 8bitadd.asm

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

 Assembling: 8bitadd.asm

D:\EX1>link 8bitadd.obj

   Microsoft Object Linker V2.01 (Large)
(C) Copyright 1982, 1983 by Microsoft Inc.

Run File [8BITADD.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
Warning: No STACK segment

There was 1 error detected.
```

```
D:\EX1>debug 8bitadd.exe
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 8A260000      MOV     AH,[0000]
076B:0109 8A3E0100      MOV     BH,[0001]
076B:010D B500          MOV     CH,00
076B:010F 02E7          ADD     AH,BH
076B:0111 7302          JNB     0115
076B:0113 FEC5          INC     CH
076B:0115 88260200      MOV     [0002],AH
076B:0119 882E0300      MOV     [0003],CH
076B:011D B44C          MOV     AH,4C
076B:011F CD21          INT     21
-d 076A:0000
076A:0000  11 99 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-q
```

```
-D 076A:0000
076A:0000  11 99 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-E 076A:0000
076A:0000  11.23   99.47

-G

Program terminated normally
-D 076A:0000
076A:0000  23 47 6A 00 00 00 00 00-00 00 00 00 00 00 00 00   #Gj............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

2) <u>Subtraction</u>

```
D:\EX1>masm 8bitsub.asm
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1993.  All rights reserved.

 Invoking: ML.EXE /I. /Zm /c /Ta 8bitsub.asm

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

 Assembling: 8bitsub.asm

D:\EX1>link 8bitsub.obj

   Microsoft Object Linker V2.01 (Large)
(C) Copyright 1982, 1983 by Microsoft Inc.

Run File [8BITSUB.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
Warning: No STACK segment

There was 1 error detected.
```

```
D:\EX1>debug 8bitsub.exe
-U
076B:0100 B86A07          MOV     AX,076A
076B:0103 8ED8            MOV     DS,AX
076B:0105 8A260000        MOV     AH,[0000]
076B:0109 8A3E0100        MOV     BH,[0001]
076B:010D B500            MOV     CH,00
076B:010F 2AE7            SUB     AH,BH
076B:0111 7304            JNB     0117
076B:0113 FEC5            INC     CH
076B:0115 F6DC            NEG     AH
076B:0117 88260200        MOV     [0002],AH
076B:011B 882E0300        MOV     [0003],CH
076B:011F B44C            MOV     AH,4C
```

```
-e 076A:0000
076A:0000  11.45   99.29

-G

Program terminated normally
-d 076A:0000
076A:0000  45 29 1C 00 00 00 00 00-00 00 00 00 00 00 00 00   E).............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

3) <u>Multiplication</u>

```
D:\EX1>masm 8bitmul.asm
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1993.  All rights reserved.

 Invoking: ML.EXE /I. /Zm /c /Ta 8bitmul.asm

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

 Assembling: 8bitmul.asm

D:\EX1>link 8bitmul.obj

   Microsoft Object Linker V2.01 (Large)
(C) Copyright 1982, 1983 by Microsoft Inc.

Run File [8BITMUL.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
Warning: No STACK segment

There was 1 error detected.
```

```
D:\EX1>debug 8bitmul.exe
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 A00000        MOV     AL,[0000]
076B:0108 8A1E0100      MOV     BL,[0001]
076B:010C F6E3          MUL     BL
076B:010E A30200        MOV     [0002],AX
076B:0111 B44C          MOV     AH,4C
076B:0113 CD21          INT     21
076B:0115 0000          ADD     [BX+SI],AL
076B:0117 807EFD00      CMP     BYTE PTR [BP-03],00
076B:011B B0FF          MOV     AL,FF
076B:011D 7701          JA      0120
076B:011F 40            INC     AX
-d 076a:0000
076A:0000  11 99 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-S
```

```
-e 076a:0000
076A:0000  11.34   99.56

-g

Program terminated normally
-d 076a:0000
076A:0000  34 56 78 11 00 00 00 00-00 00 00 00 00 00 00 00   4Vx.............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-ss
```

4) <u>Division</u>

```
D:\EX1>masm 8bitdiv.asm
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1993.  All rights reserved.

 Invoking: ML.EXE /I. /Zm /c /Ta 8bitdiv.asm

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

 Assembling: 8bitdiv.asm

D:\EX1>link 8bitdiv.obj

   Microsoft Object Linker V2.01 (Large)
(C) Copyright 1982, 1983 by Microsoft Inc.

Run File [8BITDIV.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
Warning: No STACK segment
```

```
D:\EX1>debug 8bitdiv.exe
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 B400          MOV     AH,00
076B:0107 A00000        MOV     AL,[0000]
076B:010A 8A1E0100      MOV     BL,[0001]
076B:010E F6F3          DIV     BL
076B:0110 A20200        MOV     [0002],AL
076B:0113 88260300      MOV     [0003],AH
076B:0117 B44C          MOV     AH,4C
076B:0119 CD21          INT     21
076B:011B B0FF          MOV     AL,FF
076B:011D 7701          JA      0120
076B:011F 40            INC     AX
```

```
-E 076A:0000
076A:0000  11.34   99.3

-G

Program terminated normally
-D 076A:0000
076A:0000  34 03 11 01 00 00 00 00-00 00 00 00 00 00 00 00   4...............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-S
```

Result:

The arithmetic operations have been performed and executed correctly using masm

Exp No:2

16 bit arithmetic operations

Name: Shreya Sriram
Register Number:195001106
Date: 19/07/2021

AIM:

To perform 16 bit operations – addition, subtraction, multiplication, division

ALGORITHM:

Addition:

a. Declare variables – 2 operands a and b, res(to store result), carry in data segment
b. Move first operand to ax word accumulator) and second operand to bx
c. Use add ax,bx to store the result in ax
d. Store 1 in ch if carry exists, else branch to e (using branch)
e. Move the contents from ax to res, and ch to carry
f. Terminate the program

Subtraction:

a. Declare variables – 2 operands a and b, res(to store result), carry in the data segment
b. Move first operand to ax(word accumulator) and second to bx
c. Use sub ax,bx to store result in accumulator
d. If there is a carry, store 1 in ch and perform neg ax (to obtain 2s complement). Else branch to e
e. Move result from ax to res and ch to carry

Multiplication:

a. Declare variables – 2 operands a and b, res1,res2 (16 bit) in the data segment
b. Move first operand to ax and second to bx
c. Use mul bx to perform multiplication and store result in dxax accumulator
d. Move result from dxax to result

Division:

a. Declare the variables – 2 operands a and b, res1,res2(to store result), remainder in data segment
b. Move first operand a to ax and second operand to bx. Initialize ah to 0

c. Use div bx to store the result in dxax. (dxax/bx is performed)
d. Remainder is stored in ah
e. Move contents of ax to res1 and bx to res2, and ah to remainder

PROGRAM

Addition:

PROGRAM	COMMENTS
assume cs:code,ds:data	Declare code segment as code, data segment as data
data segment	Begin data segment
a dw 0011h	Initialize a to 0011
b dw 0099h	Initialize b to 0099
res dw 0000H	Initialize res to 0000
carry db 00H	Initialize carry to 0
data ends	Data segment ends
code segment	Begin code segment
org 0100h	Set origin as 100
start: mov ax,data	Move contents from data to ax
mov ds,ax	Move contents from ax to ds
mov ax,a	Move a to ax
mov bx,b	Move b to bx
mov ch,00h	Initialize ch(to store carry) to 0
add ax,bx	Add ax+bx and store in ax
jnc here	If no carry, jump to label here
inc ch	Else,Increment carry
here: mov res,ax	Move contents from ax to res
mov carry,ch	Move contents from ch to carry
mov ah,4ch	Move 4ch to ah
int 21h	This step generates an interrupt for terminating the program
code ends	Code segment ends
end start	End of start

Subtraction:

PROGRAM	COMMENTS
assume cs:code,ds:data	Declare code segment as code, data segment as data
data segment	Begin data segment
a dw 0011h	Initialize a to 0011
b dw 0099h	Initialize b to 0099
res dw 0000H	Initialize res to 0000h
carry db 00H	Initialize carry to 0
data ends	Data segment ends
code segment	Begin code segment
org 0100h	Set origin as 100
start: mov ax,data	Move contents from data to ax
mov ds,ax	Move contents from ax to ds
mov ax,a	Move a to ax
mov bx,b	Move b to bx
mov ch,00h	Initialize ch(to store carry) to 0
sub ax,bx	Perform ax-bx and store in ax
jnc here	If no carry, jump to label here
inc ch	Else, Increment carry
neg ax	Find twos complement of ax
here: mov res,ax	Move contents from ax to res
mov carry,ch	Move carry from ch to carry
mov ah,4ch	Move 4ch to ah
int 21h	This step generates an interrupt for terminating the program
code ends	Code segment ends
end start	End of start

Multiplication:

PROGRAM	COMMENTS
assume cs:code,ds:data	Declare code segment as code, data segment as data
data segment	Begin data segment
a dw 0011h	Initialize a to 11
b dw 0099h	Initialize b to 99
res1 db 00H	Initialize res1 to 0(to store last 16bits)
res2 db 00H	Initialize res2 to 0(to store first 16bits)
data ends	Data segment ends
code segment	Begin code segment

org 0100h	Set origin as 100
start: mov ax,data	Move contents from data to ax
mov ds,ax	Move contents from ax to ds
mov ax,a	Move a to ax
mov bx,b	Move b to bx
mov ch,00h	Initialize carry to 0
mul bx	Perform ax*bx and store in dxax
here: mov res1,ax	Move contents from ax to res1
mov res2,dx	Move contents from dx to res2
mov ah,4ch	Move 4ch to ah
int 21h	This step generates an interrupt for terminating the program
code ends	Code segment ends
end start	End of start

Division

PROGRAM	COMMENTS
assume cs:code,ds:data	Declare code segment as code, data segment as data
data segment	Begin data segment
a dw 0011h	Initialize a to 0011
b dw 0099h	Initialize b to 0099
res1 dw 0000H	Initialize res1 to 0
res2 dw 0000H	Initialize res2 to 0
remainder db 00H	Initialize remainder to 0
data ends	Data segment ends
code segment	Begin code segment
org 0100h	Set origin as 100
start: mov ax,data	Move contents from data to ax
mov ds,ax	Move contents from ax to ds
mov ah,00h	Initialize ah to 0
mov ax,a	Move a to ax
mov bx,b	Move b to bx
div bx	Perform ax/bx and store quotient in ax
here: mov res1,ax	Move contents from ax to res1
mov res2,bx	Move contents of bx to res2
mov remainder,ah	Move carry from ah to remainder
mov ah,4ch	Move 4ch to ah
int 21h	This step generates an interrupt for terminating the program
code ends	Code segment ends
end start	End of start

OUTPUT SCREENSHOTS

1) Addition

```
D:\EX2>masm 16bitadd.asm
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1993.  All rights reserved.

 Invoking: ML.EXE /I. /Zm /c /Ta 16bitadd.asm

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

 Assembling: 16bitadd.asm

D:\EX2>link 16bitadd.obj

   Microsoft Object Linker V2.01 (Large)
(C) Copyright 1982, 1983 by Microsoft Inc.

Run File [16BITADD.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
Warning: No STACK segment

There was 1 error detected.
```

```
D:\EX2>debug 16bitadd.exe
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 A10000        MOV     AX,[0000]
076B:0108 8B1E0200      MOV     BX,[0002]
076B:010C B500          MOV     CH,00
076B:010E 03C3          ADD     AX,BX
076B:0110 7302          JNB     0114
076B:0112 FEC5          INC     CH
076B:0114 A30400        MOV     [0004],AX
076B:0117 882E0600      MOV     [0006],CH
076B:011B B84C00        MOV     AX,004C
076B:011E CD21          INT     21
-d 076a:0000
076A:0000  11 00 99 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

```
-u
076A:0020 0000            ADD     [BX+SI],AL
076A:0022 0000            ADD     [BX+SI],AL
076A:0024 0000            ADD     [BX+SI],AL
076A:0026 0000            ADD     [BX+SI],AL
076A:0028 0000            ADD     [BX+SI],AL
076A:002A 0000            ADD     [BX+SI],AL
076A:002C 0000            ADD     [BX+SI],AL
076A:002E 0000            ADD     [BX+SI],AL
076A:0030 0000            ADD     [BX+SI],AL
076A:0032 0000            ADD     [BX+SI],AL
076A:0034 0000            ADD     [BX+SI],AL
076A:0036 0000            ADD     [BX+SI],AL
076A:0038 0000            ADD     [BX+SI],AL
076A:003A 0000            ADD     [BX+SI],AL
076A:003C 0000            ADD     [BX+SI],AL
076A:003E 0000            ADD     [BX+SI],AL
```

```
-d 076a:0000
076A:0000  11 00 99 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
-e 076a:0000
076A:0000  11.BA   00.AA   99.AC   00.00

-g

Program terminated normally
-d 076a:0000
076A:0000  BA AA AC 00 66 AB 00 00-00 00 00 00 00 00 00 00    ....f...........
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
```

2) <u>Subtraction</u>

```
D:\EX2>masm 16bitsub.asm
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1993.  All rights reserved.

 Invoking: ML.EXE /I. /Zm /c /Ta 16bitsub.asm

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

 Assembling: 16bitsub.asm

D:\EX2>link 16bitsub.obj

   Microsoft Object Linker V2.01 (Large)
(C) Copyright 1982, 1983 by Microsoft Inc.

Run File [16BITSUB.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
Warning: No STACK segment

There was 1 error detected.
```

```
D:\EX2>debug 16bitsub.exe
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 A10000        MOV     AX,[0000]
076B:0108 8B1E0200      MOV     BX,[0002]
076B:010C B500          MOV     CH,00
076B:010E 2BC3          SUB     AX,BX
076B:0110 7304          JNB     0116
076B:0112 FEC5          INC     CH
076B:0114 F7D8          NEG     AX
076B:0116 A30400        MOV     [0004],AX
076B:0119 882E0600      MOV     [0006],CH
076B:011D B44C          MOV     AH,4C
076B:011F CD21          INT     21
-d 076a:0000
076A:0000  11 00 99 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
-S
```

```
-e 076a:0000
076A:0000  11.BA   00.AA   99.AC   00.00

-g

Program terminated normally
-d 076a:0000
076A:0000  BA AA AC 00 0E AA 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-S
```

3) Multiplication

```
D:\EX2>masm 16bitmul.asm
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1993.  All rights reserved.

 Invoking: ML.EXE /I. /Zm /c /Ta 16bitmul.asm

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

 Assembling: 16bitmul.asm

D:\EX2>link 16bitmul.obj

   Microsoft Object Linker V2.01 (Large)
(C) Copyright 1982, 1983 by Microsoft Inc.

Run File [16BITMUL.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
Warning: No STACK segment

There was 1 error detected.
```

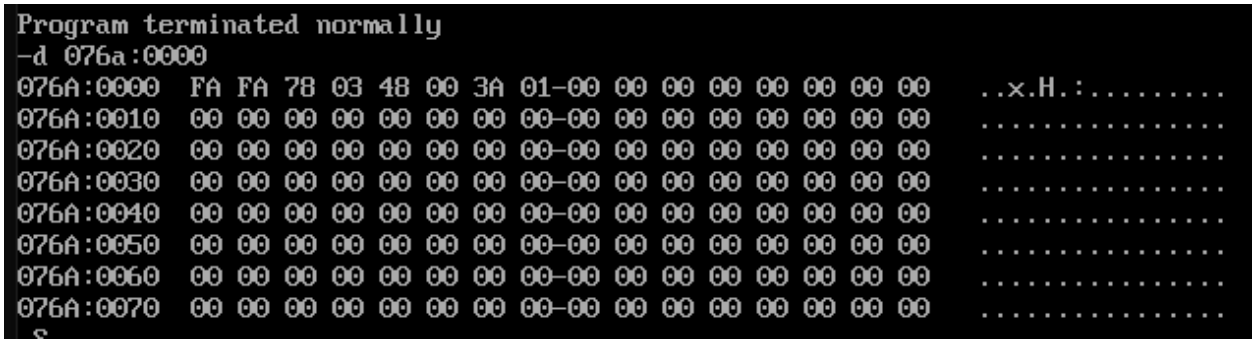
```
D:\EX2>debug 16bitmul.exe
-U
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 A10000        MOV     AX,[0000]
076B:0108 8B1E0200      MOV     BX,[0002]
076B:010C F7E3          MUL     BX
076B:010E A30400        MOV     [0004],AX
076B:0111 89160600      MOV     [0006],DX
076B:0115 B44C          MOV     AH,4C
076B:0117 CD21          INT     21
076B:0119 FD            STD
076B:011A 00B0FF77      ADD     [BX+SI+77FF],DH
076B:011E 01408B        ADD     [BX+SI-75],AX
-E 076A:0000
076A:0000  11.BA   00.BA   99.AA   00.AA

-G

Program terminated normally
```

```
Program terminated normally
-D 076A:0000
076A:0000  BA BA AA AA 84 83 7B 7C-00 00 00 00 00 00 00 00   ......{|........
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

4) Division

```
D:\EX2>masm 16bitdiv.asm
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1993.  All rights reserved.

 Invoking: ML.EXE /I. /Zm /c /Ta 16bitdiv.asm

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

 Assembling: 16bitdiv.asm

D:\EX2>link 16bitdiv.obj

   Microsoft Object Linker V2.01 (Large)
(C) Copyright 1982, 1983 by Microsoft Inc.

Run File [16BITDIV.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
Warning: No STACK segment

There was 1 error detected.
```

```
D:\EX2>debug 16bitdiv.exe
-u
076B:0100 B86A07          MOV     AX,076A
076B:0103 8ED8            MOV     DS,AX
076B:0105 B400            MOV     AH,00
076B:0107 A10000          MOV     AX,[0000]
076B:010A 8B1E0200        MOV     BX,[0002]
076B:010E F7F3            DIV     BX
076B:0110 A30400          MOV     [0004],AX
076B:0113 89160600        MOV     [0006],DX
076B:0117 88260800        MOV     [0008],AH
076B:011B B44C            MOV     AH,4C
076B:011D CD21            INT     21
076B:011F 40              INC     AX
-e 076a:0000
076A:0000  11.FA    00.FA    99.78    00.03

-g

Program terminated normally
```



Result:

The arithmetic operations have been performed and executed correctly using MASM

Exp No:3 # String Operations

Name: Shreya Sriram
Register Number:195001106
Date: 06/07/2021

AIM:

To perform the following string manipulation operations using MASM

1. Moving a string of bytes
2. Comparing 2 strings of bytes
3. Searching a byte in a string
4. Moving a string of bytes without using string instructions

a. Moving a string of bytes

Algorithm:

- Declare the source string and string length in the data segment
- Declare the location of destination string in the extra segment
- Move the offset of source string into SI
- Move the offset of destination string into DI
- Move the string length value into CX count register
- Use **REP MOVSB** instruction to move the data at SI to DI

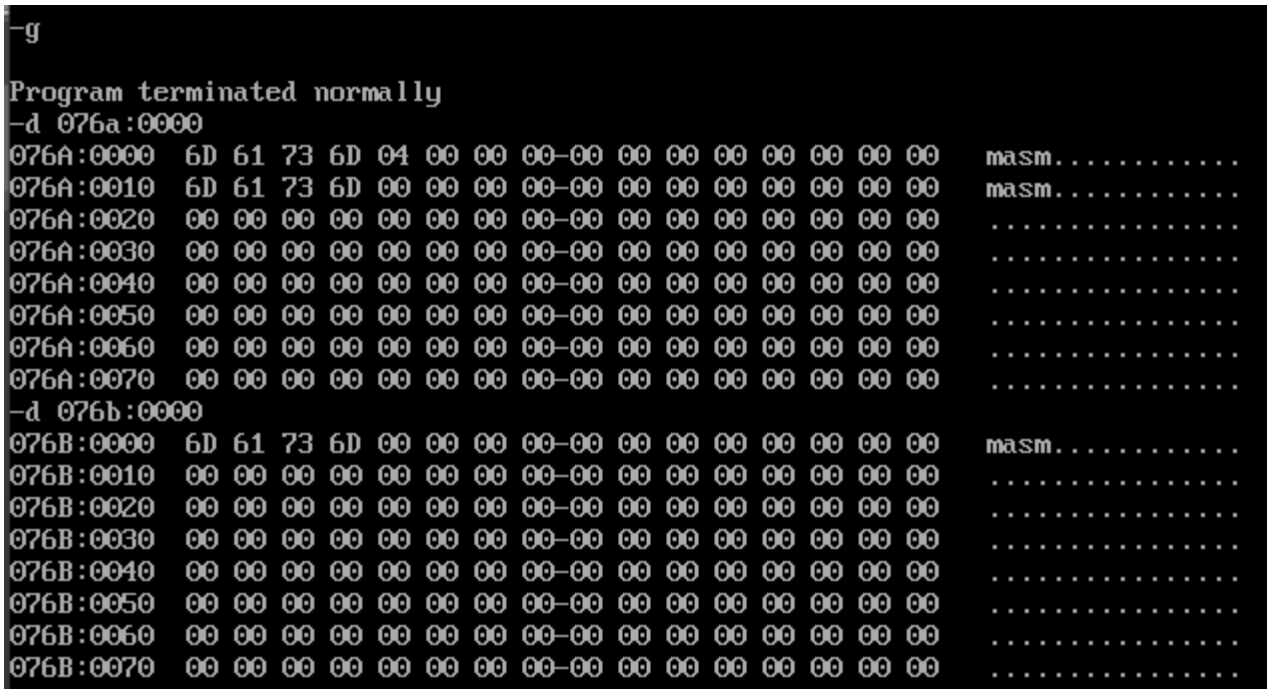
PROGRAM CODE	COMMENTS
assume cs:code,ds:data,es:extra	Declare the code,data,extra segments
data segment	Start the data segment
source db 'abcd'	Initialise the source string in bytes
strlen dw 0004h	Initialise the source string length
data ends	End the data segment
extra segment	Start the extra segment
dest db 4 dup(0)	Declare the destination string
extra ends	End the extra segment
code segment	Start the code segment
org 0100h	Enter starting address of the code segment
start :	Label start:
mov ax,data	Move the data segment to DS through the AX register

mov ds,ax	

	Move the extra segment to ES through the AX register SI = [source] DI = [dest] CX = strlen Clear the direction flag Move the string from DI to SI until CX =0 Terminate the program End code segment End start label

OUTPUT:

```
-u
076C:0100 B86A07        MOV     AX,076A
076C:0103 8ED8          MOV     DS,AX
076C:0105 B86B07        MOV     AX,076B
076C:0108 8EC0          MOV     ES,AX
076C:010A BE0000        MOV     SI,0000
076C:010D BF0000        MOV     DI,0000
076C:0110 8B0E0400      MOV     CX,[0004]
076C:0114 FC            CLD
076C:0115 F3            REPZ
076C:0116 A4            MOVSB
076C:0117 B44C          MOV     AH,4C
076C:0119 CD21          INT     21
076C:011B 83FA10        CMP     DX,+10
076C:011E B0FF          MOV     AL,FF
-d 076a:0000
076A:0000  6D 61 73 6D 04 00 00 00-00 00 00 00 00 00 00 00   masm............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```



b. Comparing 2 strings of bytes

Algorithm:

- Declare and initialise string1, length of string1, memory location to store the result in the data segment
- Declare and initialise string2 in the extra segment
- Move the offset of string1 into SI
- Move the offset of string2 into DI
- Store the length of string1 in both BX and CX registers
- Use the REPE CMPSB instruction to compare the two strings. After the end of loop if ZF is not set, go to not equal label, else go to equal label
- not equal label : Subtract CX from BX (BX=BX-CX) to get the index of the first mismatch and store the value in result
- equal label: Implies the strings are equal. Hence store the value 0000 in result

PROGRAM:

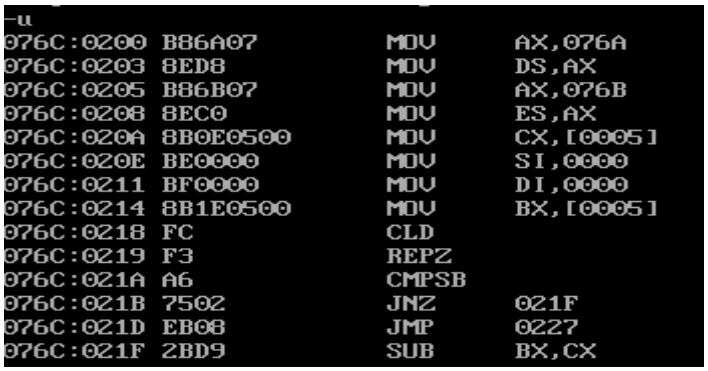
PROGRAM CODE	COMMENTS

assume cs:code,ds:data,es:extra data segment	Declare the code,data,extra segments Start the data segment
str1 db 'abcde'	Initialise the string1 of bytes
strlen dw 0005h	Initialise the source string length Declare result to store the output
result dw 0000h	
data ends	End the data segment
extra segment	Start the extra segment
str2 db 'abcbe'	Declare the string2 of bytes
extra ends	End the extra segment
code segment	Start the code segment
org 0200h	Enter starting address of the code segment
start:	Label start:
mov ax,data	Move the data segment to DS through the AX register

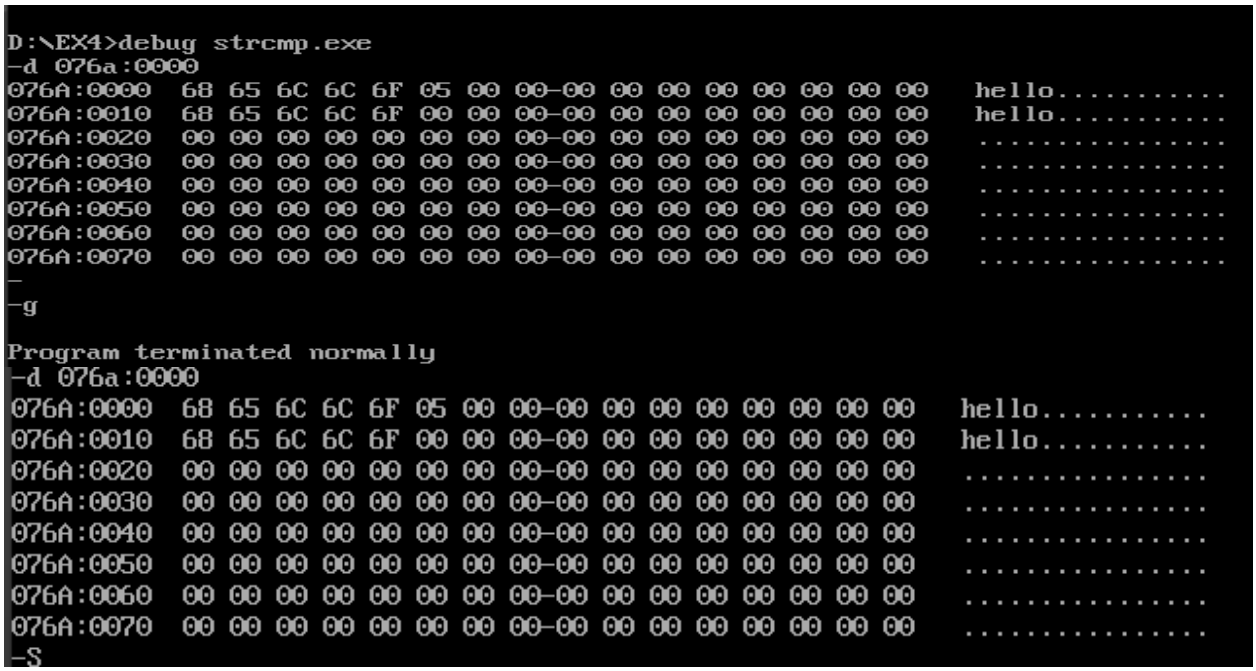
mov ds,ax	Move the extra segment to ES through the AX register
mov ax,extra	CX = strlen
mov es,ax	SI = [str1]
mov cx,strlen	DI = [str2]
mov si,offset str1	BX = strlen
mov di,offset str2	Clear the direction flag
mov bx,strlen	Compare Strings at SI and DI
cld	If ZF is not set go to notequal label Jump to equal label
repe cmpsb	Notequal label
jne notequal	BX = BX – CX
jmp equal	result = BX
notequal:	Jump to Exit label
sub bx,cx	Equal label
mov result,bx	result = 0000h
jmp exit	Exit label
equal:	Terminate the program
mov result,0000h	
exit:	End code segment
mov ah,4ch	

int 21h code ends end start	End start label

OUTPUT:



```
-u
076C:0200 B86A07        MOV     AX,076A
076C:0203 8ED8          MOV     DS,AX
076C:0205 B86B07        MOV     AX,076B
076C:0208 8EC0          MOV     ES,AX
076C:020A 8B0E0500      MOV     CX,[0005]
076C:020E BE0000        MOV     SI,0000
076C:0211 BF0000        MOV     DI,0000
076C:0214 8B1E0500      MOV     BX,[0005]
076C:0218 FC            CLD
076C:0219 F3            REPZ
076C:021A A6            CMPSB
076C:021B 7502          JNZ     021F
076C:021D EB08          JMP     0227
076C:021F 2BD9          SUB     BX,CX
```



```
D:\EX4>debug strcmp.exe
-d 076a:0000
076A:0000  68 65 6C 6C 6F 05 00 00-00 00 00 00 00 00 00 00   hello...........
076A:0010  68 65 6C 6C 6F 00 00 00-00 00 00 00 00 00 00 00   hello...........
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
-g

Program terminated normally
-d 076a:0000
076A:0000  68 65 6C 6C 6F 05 00 00-00 00 00 00 00 00 00 00   hello...........
076A:0010  68 65 6C 6C 6F 00 00 00-00 00 00 00 00 00 00 00   hello...........
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-S
```

c. Searching a byte in a string

Algorithm:

- Declare and initialize the string in the extra segment
- Declare and initialize the string length, memory to store index of occurrence of substring (indexval), and the substring itself in the data segment
- Move the substring into the AL register
- Move the string length into the BX and CX registers
- Move offset of string into DI
- Compare the byte (substring) stored in AL register to the byte pointed to by the DI using REPNE SCASB instruction, the instruction terminates when the substring is found and the number of comparisons is found in the CX register
- If the ZF is not set, go to absent label, else go to present label
- present label : store the value 0000h in indexval and go to exit label
- absent label : subtract CX from BX (BX=BX-CX) to get the index of the first occurrence and then move it to indexval

PROGRAM:

PROGRAM CODE	COMMENTS
assume cs:code,ds:data,es:extra data segment	Declare the code,data,extra segments Start the data segment
subst db 'd'	Initialise the substring
strlen dw 0004h	Initialise the length of the string
indexval dw 0000h	Declare indexval to store the index of occurence
data ends	End the data segment
extra segment	Start the extra segment
str db 'abcde'	Declare and initialize the string of bytes End the extra segment
extra ends	Start the code segment
code segment	Enter starting address of the code segment
org 0300h	Label start:
start:	Move the data segment to DS through the AX register
mov ax,data	
mov ds,ax	
mov ax,extra	Move the extra segment to ES through the AX register
mov es,ax	AL =substring
mov al,subst	BX = strlen
mov bx,strlen	

mov cx,strlen
mov di,offset str
cld
repne scasb
 	
jne absent
jmp present
absent:
mov indexval,0000h
jmp exit
present:
sub bx,cx
	CX = strlen
	DI =[dest]
	Clear the direction flag
	
	Search for substring stored in AL in the string at DI until found or CX=0
	If Zero flag is not set go to absent label Jump to present label
	absent label
	indexval = 0000h(substring not present)
	
	Jump to Exit label
	Present label
	BX = BX - CX
	indexval = BX

mov indexval,bx
exit:
mov ah,4ch
int 21h
code ends
 	
end start
	Exit label
	Terminate the program
	
	End code segment
	End start label

SNAPSHOTS:

OUTPUT:

```
D:\EX4>debug search.exe
-u
076C:0300 B86A07        MOV     AX,076A
076C:0303 8ED8          MOV     DS,AX
076C:0305 B86B07        MOV     AX,076B
076C:0308 8EC0          MOV     ES,AX
076C:030A A00000        MOV     AL,[0000]
076C:030D 8B1E0100      MOV     BX,[0001]
076C:0311 8B0E0100      MOV     CX,[0001]
076C:0315 BF0000        MOV     DI,0000
076C:0318 FC            CLD
076C:0319 F2            REPNZ
076C:031A AE            SCASB
076C:031B 7502          JNZ     031F
076C:031D EB08          JMP     0327
076C:031F C70603000000  MOV     WORD PTR [0003],0000
```

```
-d 076a:0000
076A:0000  64 04 00 00 00 00 00 00-00 00 00 00 00 00 00 00   d...............
076A:0010  61 62 63 64 65 00 00 00-00 00 00 00 00 00 00 00   abcde...........
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-g

Program terminated normally
-d 076a:0000
076A:0000  64 04 00 04 00 00 00 00-00 00 00 00 00 00 00 00   d...............
076A:0010  61 62 63 64 65 00 00 00-00 00 00 00 00 00 00 00   abcde...........
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

d.Moving a string without using string instructions

Algorithm:

- Declare and initialize the source string in the data segment
- Declare string length and destination string location also in data segment
- Move the offset of source string into SI
- Move the offset of destination string into DI
- Move the string length value into CX count register
- Repeat the following statements until CX register value becomes 0 (Loop start)
  - Move the value at [SI] into AL register
  - Move the value in AL register to the memory location [DI]

- Increment SI and DI
- Decrement CX
- (Loop end)
- Terminate the program

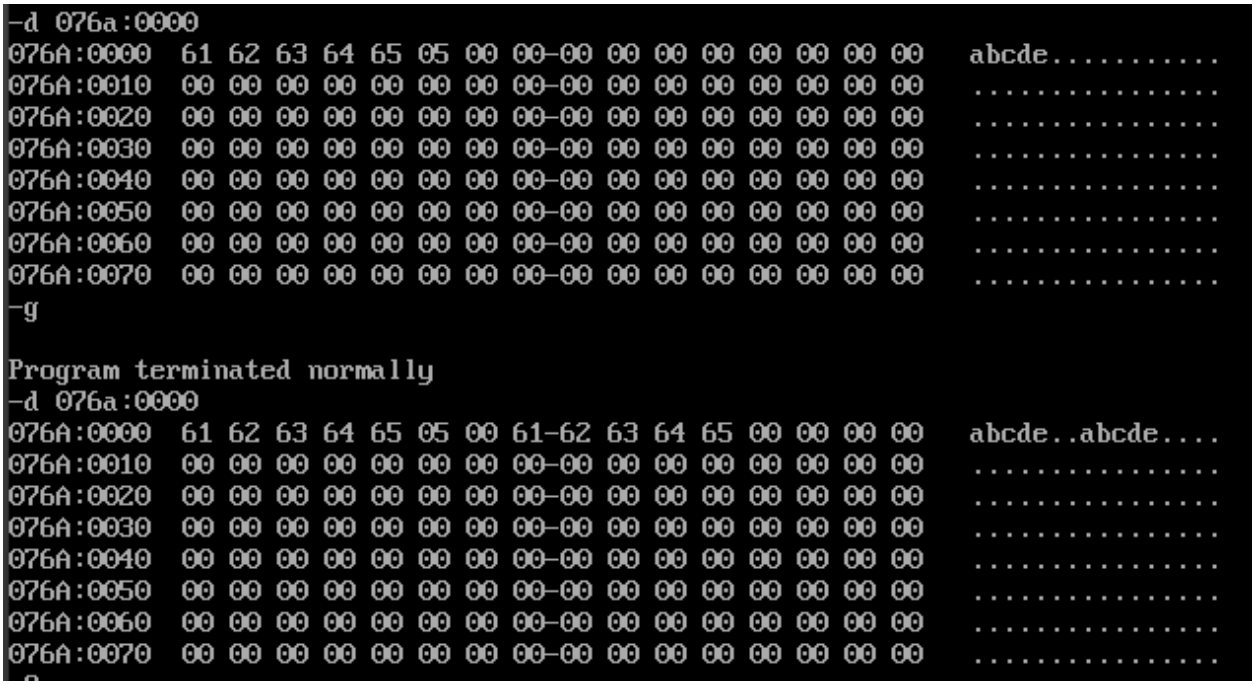
PROGRAM:

PROGRAM CODE	COMMENTS
assume cs:code,ds:data data segment source db 'abcde' strlen dw 0005h dest db 5 dup(0) data ends code segment org 0500h	Declare the code, data segments Start the data segment Initialise the source string Initialise the length of the string Declare destination location End the data segment Start the code segment Enter starting address of the code segment

start:
mov ax,data
mov ds,ax
mov si,offset source
mov di,offset dest
mov cx,strlen
looper:
mov al,[si]
mov [di],al
inc si
inc di
dec cx
jnz looper
mov ah,4ch
int 21h
code ends
end start
	Label start:
	Move the data segment to DS through the AX register
	SI = [source]
	DI = [dest]
	CX = strlen
	Loop label
	AL = [SI]
	[DI] = AL
	SI = SI + 1
	DI = DI +1
	CX = CX – 1
	
	Jump to loop label if ZF is not set Terminate the program
	
	End code segment
	End start label

OUTPUT:

```
-u
076B:0522 50            PUSH    AX
076B:0523 B8C200        MOV     AX,00C2
076B:0526 50            PUSH    AX
076B:0527 9AA701FD02    CALL    02FD:01A7
076B:052C 8A5E0A        MOV     BL,[BP+0A]
076B:052F B700          MOV     BH,00
076B:0531 D1E3          SHL     BX,1
076B:0533 8B87AA2A      MOV     AX,[BX+2AAA]
076B:0537 8946F8        MOV     [BP-08],AX
076B:053A 83F8FF        CMP     AX,-01
076B:053D 7509          JNZ     0548
076B:053F B8C300        MOV     AX,00C3
-s
```



RESULT:

The above mentioned 4 string manipulation operations using the mentioned instructions has been performed and the result is verified.

Exp No:4 # Code Conversion

Name: Shreya Sriram
Register Number:195001106
Date: 02/08/2021

AIM:

To convert BCD to hexadecimal and vice versa

ALGORITHM:

BCD to Hexadecimal

- Declare the variables bcd, first, second and hex to store the bcd number, first digit, second digit and hexadecimal number
- Move bcd number to the al register
- Move 0FH to bl register
- Perform al and bl and store the result in al
- This fetches the second digit and it is moved to second
- Move the bcd number to the al register
- Left shift 0FH four times to bl
- Perform al and bl and move result to first
- Initialize bl to 0Ah
- Multiply al and bl
- Add second to al
- Move the result to hex
- Terminate the program

Hexadecimal to BCD

- Declare the variables bcd and hex to store the bcd number and hexadecimal number
- Move hexadecimal to al
- Move 64h to cl
- Divide al/cl
- Move al to bl
- Move 00h to ah
- Move 00Ah to cl
- Divide al/cl
- Move al to dh and ah to dl
- Move 04h to cl and right rotate by 4 positions
- Add dh and dl
- Move dx to bcd number
- Terminate

CODE:

BCD to Hexadecimal

CODE	REMARKS
assume cs:code,ds:data	
data segment	Data segment starts
bcd db 34h	Initialise bcd to 34h
first db 00H	Initialise first to 0
second db 00H	Initialise second to 0
hex db 00H	Initialise hex to 0
data ends	Data segment ends
code segment	Code segment starts
org 0100h	
start: mov ax, data	Mov data to ax
mov ds,ax	Move ax to ds
mov al,bcd	Move bcd to al
mov bl,0FH	Move 0Fh to bl
and al,bl	Perform al and bl
mov second,al	Move al to second
mov al,bcd	Move bcd to al
mov cl, 4	Initialize cl to 4
shl bl,cl	Left shift bl 4 times
and al,bl	Perform al and bl
shr al,cl	Right shift al 4 times
mov first,al	Move al to first(first digit)
mov al,first	Move first to al
mov bl,0Ah	Move 0Ah to bl
mov ch,00h	Move 00h to ch
mul bl	Multiply al*bl and store in al
add al,second	Add second and al
mov hex,al	Move al to hex
mov ah,4ch	Move 4ch to ah
int 21h	Terminate the code
code ends	Code ends
end start	

Hexadecimal to BCD

CODE	REMARKS
assume cs:code, ds:data	
data segment	Data segment starts
hex db 0FFh	Initialise bcd to 0FFh
bcd dw 00h	Initialise hex to 0
data ends	Data segment ends

code segment	Code segment starts
org 0100h	
start:	
mov ax, data	Mov data to ax
mov ds, ax	Move ax to ds
mov ch, 00h	Move 00h to ch
mov ah, 00h	Move ooh to ah
mov al, hex	Move hex to al
mov cl, 64h	Move 64h to cl
div cl	Divide al/cl
mov bl, al	Move al to bl
mov al, ah	Move ah to al
mov ah, 00h	Move 00h to ah
mov cl, 00Ah	Move 00Ah to cl
div cl	Divide al/cl
mov dh, al	Move al to dh
mov dl,ah	Move ah to dl
mov cl, 04h	Move 04h to cl
ror dh, cl	Right rotate dh 4 times
add dh, dl	Add dh and dl and store in dh
mov dl, bl	Move bl to dl
mov bcd, dx	Move dx to bcd
mov ah, 4ch	Move 4ch to ah
int 21h	Terminate the code
code ends	Code ends
end start	

OUTPUT:

BCD TO HEXADECIMAL

```
D:\EX5>debug bcdhex.exe
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 A00000        MOV     AL,[0000]
076B:0108 B30F          MOV     BL,0F
076B:010A 22C3          AND     AL,BL
076B:010C A20200        MOV     [0002],AL
076B:010F A00000        MOV     AL,[0000]
076B:0112 B104          MOV     CL,04
076B:0114 D2E3          SHL     BL,CL
076B:0116 22C3          AND     AL,BL
076B:0118 D2E8          SHR     AL,CL
076B:011A A20100        MOV     [0001],AL
076B:011D A00100        MOV     AL,[0001]
-d 076a:0000
076A:0000  34 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   4...............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

```
-g

Program terminated normally
-d 076a:0000
076A:0000  34 03 04 22 00 00 00 00-00 00 00 00 00 00 00 00   4.".............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

HEXADECIMAL TO BCD

```
-u
076B:0120 D2CE          ROR     DH,CL
076B:0122 02F2          ADD     DH,DL
076B:0124 8AD3          MOV     DL,BL
076B:0126 89160100      MOV     [0001],DX
076B:012A B44C          MOV     AH,4C
076B:012C CD21          INT     21
076B:012E B0FF          MOV     AL,FF
076B:0130 7201          JB      0133
076B:0132 40            INC     AX
076B:0133 59            POP     CX
076B:0134 22C1          AND     AL,CL
076B:0136 D0D8          RCR     AL,1
076B:0138 7203          JB      013D
076B:013A E98300        JMP     01C0
076B:013D 8A46FD        MOV     AL,[BP-03]
```

```
-g

Program terminated normally
-d 076a:0000
076A:0000  FF 02 55 00 00 00 00 00-00 00 00 00 00 00 00 00   ..U.............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

RESULT:

Code conversion from BCD to Hexadecimal and vice versa is coded and tested.

Exp No:5 # Matrix Operations

Name: Shreya Sriram
Register Number:195001106
Date: 09/08/2021

AIM:

To perform sum and difference of two matrices

ALGORITHM:

Addition:

- Declare and initialize row1 and col1
- Declare and initialize mat_a and mat_b
- Move data to ax, ax to ds, ax to es
- Move row1 to al and row2 to bl
- Compare al and bl, if they are not equal, go to finish
- Move col1 to al and col2 to bl
- Compare al and bl, if they are not equal, go to finish
- Move row1 to al and col1 to cl
- Move offset of mat_a to SI and mat_b to DI and offset of mat_res to bx
- Move ax to cx
- Move address of SI to al, add this to the address of DI
- Move al to the address of BX
- Increment SI, DI, BL
- Terminate the program

Subtraction:

- Declare and initialize row1 and col1
- Declare and initialize mat_a and mat_b
- Move data to ax, ax to ds, ax to es
- Move row1 to al and row2 to bl
- Compare al and bl, if they are not equal, go to finish
- Move col1 to al and col2 to bl
- Compare al and bl, if they are not equal, go to finish
- Move row1 to al and col1 to cl
- Move offset of mat_a to SI and mat_b to DI and offset of mat_res to bx
- Move ax to cx
- Move address of SI to al, subtract this from the address of DI
- Move al to the address of BX

- Increment SI, DI, BL
- Terminate the program

CODE:

Addition:

CODE	**REMARKS**
assume cs:code,ds:data	
data segment	Data segment starts
row1 db 02h	Initialize row1 to 02h
col1 db 02h	Initialize col1 to 02h
mat_a db 01H,02H,03H,04H	Initialize matrix a
org 0010h	
row2 db 02h	Initialise row2 to 02h
col2 db 02h	Initialize col2 to 02h
mat_b db 05H,06H,07H,08H	Initialize matrix b
org 0020h	
mat_res db ?	Store result matrix
data ends	Data segment ends
code segment	Code segment begins
start:	
mov ax, data	Move data to ax
mov ds, ax	Move ax to ds
mov es, ax	Move ax to es
CHK: MOV AL, ROW1	Move row1 to al
MOV BL, ROW2	Move row2 to bl
CMP AL,BL	Compare al and bl
JNZ FINISH	If they are not equal, move to end
MOV AL, COL1	Move col1 to al
MOV BL, COL2	Move col2 to bl
CMP AL,BL	Compare al and bl
JNZ FINISH	If not equal, move to end
mov al, row1	Move row1 to al
mov cl, col1	Move col1 to cl
mul cl	Multiply al*cl
mov SI, offset mat_a	Move the offset of matrix a to SI
mov DI, offset mat_b	Move the offset of matrix b to DI
mov bx,offset mat_res	Move the offset of matrix res to bx
mov cx,ax	Move ax to cx
WRK: MOV AL,[SI]	Move SI address to al
ADD AL,[DI]	Add DI address to al
MOV [BX],AL	Move the address of bx to al
INC SI	Increment SI
INC DI	Increment DI

INC BL	Increment bl
LOOP WRK	
FINISH:	
mov ah, 4ch	Terminate the program
int 21h	
code ends	Code ends
end start	

Subtraction:

CODE	**REMARKS**
assume cs:code,ds:data	
data segment	Data segment starts
row1 db 02h	Initialize row1 to 02h
col1 db 02h	Initialize col1 to 02h
mat_a db 01H,02H,03H,04H	Initialize matrix a
org 0010h	
row2 db 02h	Initialise row2 to 02h
col2 db 02h	Initialize col2 to 02h
mat_b db 05H,06H,07H,08H	Initialize matrix b
org 0020h	
mat_res db ?	Store result matrix
data ends	Data segment ends
code segment	Code segment begins
start:	
mov ax, data	Move data to ax
mov ds, ax	Move ax to ds
mov es, ax	Move ax to es
CHK: MOV AL, ROW1	Move row1 to al
MOV BL, ROW2	Move row2 to bl
CMP AL,BL	Compare al and bl
JNZ FINISH	If they are not equal, move to end
MOV AL, COL1	Move col1 to al
MOV BL, COL2	Move col2 to bl
CMP AL,BL	Compare al and bl
JNZ FINISH	If not equal, move to end
mov al, row1	Move row1 to al
mov cl, col1	Move col1 to cl
mul cl	Multiply al*cl
mov SI, offset mat_a	Move the offset of matrix a to SI
mov DI, offset mat_b	Move the offset of matrix b to DI
mov bx,offset mat_res	Move the offset of matrix res to bx
mov cx,ax	Move ax to cx
WRK: MOV AL,[SI]	Move SI address to al

SUB AL,[DI]	Subtract DI address to al
MOV [BX],AL	Move the address of bx to al
INC SI	Increment SI
INC DI	Increment DI
INC BL	Increment bl
LOOP WRK	
FINISH:	
mov ah, 4ch	Terminate the program
int 21h	
code ends	Code ends
end start	

OUTPUT:

Addition:

```
D:\EX6>debug matadd.exe
-u
076D:0000 B86A07        MOV     AX,076A
076D:0003 8ED8          MOV     DS,AX
076D:0005 8EC0          MOV     ES,AX
076D:0007 A00000        MOV     AL,[0000]
076D:000A 8A1E1000      MOV     BL,[0010]
076D:000E 38D8          CMP     AL,BL
076D:0010 752B          JNZ     003D
076D:0012 A00100        MOV     AL,[0001]
076D:0015 8A1E1100      MOV     BL,[0011]
076D:0019 38D8          CMP     AL,BL
076D:001B 7520          JNZ     003D
076D:001D A00000        MOV     AL,[0000]
```

```
D:\EX6>debug matadd.exe
-d 076a:0000
076A:0000  02 02 01 02 03 04 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  02 02 05 06 07 08 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  B8 6A 07 8E D8 8E C0 A0-00 00 8A 1E 10 00 38 D8   .j............8.
076A:0040  75 2B A0 01 00 8A 1E 11-00 38 D8 75 20 A0 00 00   u+.......8.u ...
076A:0050  8A 0E 01 00 F6 E1 BE 02-00 BF 12 00 BB 20 00 8B   ............. ..
076A:0060  C8 8A 04 02 05 88 07 46-47 FE C3 E2 F4 B4 4C CD   .......FG.....L.
076A:0070  21 B6 2C 3A 46 F8 74 7E-C7 46 FA 00 00 8A 46 F8   !.,:F.t~.F....F.
-g

Program terminated normally
-d 076a:0000
076A:0000  02 02 01 02 03 04 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  02 02 05 06 07 08 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  06 08 0A 0C 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  B8 6A 07 8E D8 8E C0 A0-00 00 8A 1E 10 00 38 D8   .j............8.
076A:0040  75 2B A0 01 00 8A 1E 11-00 38 D8 75 20 A0 00 00   u+.......8.u ...
076A:0050  8A 0E 01 00 F6 E1 BE 02-00 BF 12 00 BB 20 00 8B   ............. ..
076A:0060  C8 8A 04 02 05 88 07 46-47 FE C3 E2 F4 B4 4C CD   .......FG.....L.
076A:0070  21 B6 2C 3A 46 F8 74 7E-C7 46 FA 00 00 8A 46 F8   !.,:F.t~.F....F.
```

Subtraction:

```
-u
076D:0000 B86A07        MOV     AX,076A
076D:0003 8ED8          MOV     DS,AX
076D:0005 8EC0          MOV     ES,AX
076D:0007 A00000        MOV     AL,[0000]
076D:000A 8A1E1000      MOV     BL,[0010]
076D:000E 38D8          CMP     AL,BL
076D:0010 752B          JNZ     003D
076D:0012 A00100        MOV     AL,[0001]
076D:0015 8A1E1100      MOV     BL,[0011]
076D:0019 38D8          CMP     AL,BL
076D:001B 7520          JNZ     003D
076D:001D A00000        MOV     AL,[0000]
```

```
D:\EX6>debug matsub.exe
-d 076a:0000
076A:0000  02 02 01 02 03 04 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  02 02 05 06 07 08 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  B8 6A 07 8E D8 8E C0 A0-00 00 8A 1E 10 00 38 D8   .j............8.
076A:0040  75 2B A0 01 00 8A 1E 11-00 38 D8 75 20 A0 00 00   u+.......8.u ...
076A:0050  8A 0E 01 00 F6 E1 BE 02-00 BF 12 00 BB 20 00 8B   ............. ..
076A:0060  C8 8A 04 2A 05 88 07 46-47 FE C3 E2 F4 B4 4C CD   ...*...FG.....L.
076A:0070  21 B6 2C 3A 46 F8 74 7E-C7 46 FA 00 00 8A 46 F8   !.,:F.t~.F....F.
-g

Program terminated normally
-d 076a:0000
076A:0000  02 02 01 02 03 04 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  02 02 05 06 07 08 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  FC FC FC FC 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  B8 6A 07 8E D8 8E C0 A0-00 00 8A 1E 10 00 38 D8   .j............8.
076A:0040  75 2B A0 01 00 8A 1E 11-00 38 D8 75 20 A0 00 00   u+.......8.u ...
076A:0050  8A 0E 01 00 F6 E1 BE 02-00 BF 12 00 BB 20 00 8B   ............. ..
076A:0060  C8 8A 04 2A 05 88 07 46-47 FE C3 E2 F4 B4 4C CD   ...*...FG.....L.
076A:0070  21 B6 2C 3A 46 F8 74 7E-C7 46 FA 00 00 8A 46 F8   !.,:F.t~.F....F.
```

RESULT:

Addition and subtraction of matrices is implemented and tested

Exp No:6 **Sorting Operations**

Name: Shreya Sriram
Register Number:195001106
Date: 16/08/2021

AIM:

To perform sorting of 8-bit numbers in ascending and descending order

ALGORITHM:

Ascending order:

- Initialize list1
- Move data to ax
- Move ax to ds
- Move 04h to ch
- Move 04h to cl
- Move the address of the lis1 to si
- Move content of si to al and si+1 to bl
- Compare al and bl, if al>bl, jump to down
- Move contents of si+1 to dl, exchange contents of si and dl
- Move si+1 to dl
- Increment si
- Decrement cl, if cl is not zero,move to up2
- Terminate the program

Descending order:

- Initialize list1
- Move data to ax
- Move ax to ds
- Move 04h to ch
- Move 04h to cl
- Move the address of the lis1 to si
- Move content of si to al and si+1 to bl
- Compare al and bl, if al<bl, jump to down
- Move contents of si+1 to dl, exchange contents of si and dl
- Move si+1 to dl
- Increment si
- Decrement cl, if cl is not zero,move to up2
- Terminate the program

CODE:

Ascending order:

CODE	REMARKS
assume cs:code, ds:data	
data segment	Data segment starts
list1 db 96h, 12h, 30h, 80h, 22h	Declare the list of elements
data ends	Data segment ends
code segment	Code segment starts
org 0100h	
start:	Start the program
mov ax,data	Move data to ax
mov ds,ax	Move ax to ds
mov ch,04h	Move ch to 04h
up2:	
mov cl,04h	Move 04h to cl
lea si,list1	Move address of list1 to si
up1:	
mov al,[si]	Move the content of si to al
mov bl,[si+1]	Move the content of si+1 to bl
cmp al,bl	Compare al and bl
jc down	If al<bl, move to down
mov dl,[si+1]	Move the content of si+1 to dl
xchg [si],dl	Exchange [si] to dl
mov [si+1],dl	Move the contents of [si+1] to dl
down:	
inc si	Increment si
dec cl	Decrement cl
jnz up1	If cl is not 0, move to up1
dec ch	Decrement ch
jnz up2	If not zero, jump to up2
mov ah,4ch	Move 4ch to ah
int 21h	Terminate program
code ends	Code segment ends
end start	End start

Descending order:

CODE	REMARKS
assume cs:code, ds:data	
data segment	Data segment starts
list1 db 96h, 12h, 30h, 80h, 22h	Declare the list of elements
data ends	Data segment ends
code segment	Code segment starts
org 0100h	
start:	Start the program
mov ax,data	Move data to ax
mov ds,ax	Move ax to ds

mov ch,04h	Move ch to 04h
up2:	
mov cl,04h	Move 04h to cl
lea si,list1	Move address of list1 to si
up1:	
mov al,[si]	Move the content of si to al
mov bl,[si+1]	Move the content of si+1 to bl
cmp al,bl	Compare al and bl
jnc down	If al>bl, move to down
mov dl,[si+1]	Move the content of si+1 to dl
xchg [si],dl	Exchange [si] to dl
mov [si+1],dl	Move the contents of [si+1] to dl
down:	
inc si	Increment si
dec cl	Decrement cl
jnz up1	If cl is not 0, move to up1
dec ch	Decrement ch
jnz up2	If not zero, jump to up2
mov ah,4ch	Move 4ch to ah
int 21h	Terminate program
code ends	Code segment ends
end start	End start

OUTPUT:

Ascending order:

```
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 B504          MOV     CH,04
076B:0107 B104          MOV     CL,04
076B:0109 8D360000      LEA     SI,[0000]
076B:010D 8A04          MOV     AL,[SI]
076B:010F 8A5C01        MOV     BL,[SI+01]
076B:0112 38D8          CMP     AL,BL
076B:0114 7208          JB      011E
076B:0116 8A5401        MOV     DL,[SI+01]
076B:0119 8614          XCHG    DL,[SI]
076B:011B 885401        MOV     [SI+01],DL
076B:011E 46            INC     SI
076B:011F FEC9          DEC     CL
-d 076a:0000
076A:0000  96 12 30 80 22 00 00 00-00 00 00 00 00 00 00 00   ..0."...........
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

```
-g

Program terminated normally
-d 076a:0000
076A:0000  12 22 30 80 96 00 00 00-00 00 00 00 00 00 00 00   ."0.............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

Descending order:

```
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 B504          MOV     CH,04
076B:0107 B104          MOV     CL,04
076B:0109 8D360000      LEA     SI,[0000]
076B:010D 8A04          MOV     AL,[SI]
076B:010F 8A5C01        MOV     BL,[SI+01]
076B:0112 38D8          CMP     AL,BL
076B:0114 7308          JNB     011E
076B:0116 8A5401        MOV     DL,[SI+01]
076B:0119 8614          XCHG    DL,[SI]
076B:011B 885401        MOV     [SI+01],DL
076B:011E 46            INC     SI
076B:011F FEC9          DEC     CL
-d 076a:0000
076A:0000  96 12 30 80 22 00 00 00-00 00 00 00 00 00 00 00   ..0."...........
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

```
-g

Program terminated normally
-d 076a:0000
076A:0000  96 80 30 22 12 00 00 00-00 00 00 00 00 00 00 00   ..0"............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

RESULT

Sorting operations are performed and tested

Exp No:7

BCD Operations

Name: Shreya Sriram
Register Number:195001106
Date: 23/08/2021

AIM:

To perform BCD addition and subtraction using 8086 microprocessor instructions in masm

ALGORITHM:

a)Addition

- Initialize the 2 numbers,sum and carry both as 0 in the data segment
- Load data from num1 to register AL (first number).
- Load data from num2 to register BL (second number).
- Initialize carry to 0.
- Add these two numbers (contents of register AL and register BL).
- Store the result in packed BCD form using DAA instruction.
- Jump to final steps if there is no carry. Else do the following steps,
- Increment carry.
- Store additional values to sum.
- Terminate the program.

b)Subtraction

- Initialize the 2 numbers, diff and carry both as 0 in the data segment
- Load data from num1 to register AL (first number)
- Load data from num2 to register BL (second number)
- Subtract these two numbers (contents of register AL and register BL)
- Store the result in packed BCD form using DAS instruction
- Jump to final steps if there is no carry. Else do the following steps
- Increment carry.
- Find 10’s complement of the result and store it in packed BCD form using DAS instruction
- Store answer to result.
- Terminate the program.

Program Code:

a) Addition :

Code	Comments
assume cs:code,ds:data	Declare the code and data segments
	Data segment starts
data segment	Initialize num1
num1 db 27h	Initialize num2
num2 db 35h	Initialize sum as 0
sum db 00h	Initialize carry as 0
carry db 00h	End of data segment
data ends	
code segment	Code segment starts
org 0100h	Set starting address of code segment
start: mov ax,data	Start label
mov ds,ax	Move the data segment to DS through the AX register
mov cl,00h	Set CL register as 0
mov al,num1	Move num1 into AL register
mov bl,num2	Move num2 into BL register
add al,bl	AL=AL+BL
DAA	Store the result in packed BCD form
jnc noCarry	If there is no carry, jump to 'here' label
inc cl	Increment CL
noCarry:mov sum,al	Nocarry label
mov carry,cl	Move value at AL into sum
	Move value at CL into carry
mov ah,4ch	
INT 21h	Terminate the program
code ends	
end start	End of code segment
	End of start label

b)Subtraction :

CODE	Comments
assume cs:code,ds:data	Declare the code and data segments
data segment	Data segment starts
num1 db 35h	Initialize num1
num2 db 27h	Initialize num2
diff db 00h	Initialize sum as 0
carry db 00h	Initialize diff as 0
data ends	End of data segment
code segment	Code segment starts
org 0100h	Set starting address of code segment
start: mov ax,data	Move the data segment to DS through the AX register
mov ds,ax	
mov cl,00h	Set CL register as 0
mov al,num1	Move num1 into AL register
mov bl,num2	Move num2 into BL register
sub al,bl	AL=AL-BL
DAS	Store the result in packed BCD form
jnc here	If no carry, jump to here label
mov dl,99h	Set DL as 99
sub dl,al	Increment CL register
inc dl	DL=DL-AL
inc cl	Increment DL
mov al,dl	Move contents of DL into AL
DAS	Store the result in packed BCD form
here: mov diff,al	Label ‘here’
mov carry,cl	Move value at AL into diff
mov ah,4ch	Move value at CL into carry
INT 21h	Terminate the program
code ends	End of code segment
end start	End of start segment

Unassembled Code:

a)Addition

```
D:\>debug bcdADD.exe;
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 B100          MOV     CL,00
076B:0107 A00000        MOV     AL,[0000]
076B:010A 8A1E0100      MOV     BL,[0001]
076B:010E 02C3          ADD     AL,BL
076B:0110 27            DAA
076B:0111 7302          JNB     0115
076B:0113 FEC1          INC     CL
076B:0115 A20200        MOV     [0002],AL
076B:0118 880E0300      MOV     [0003],CL
076B:011C B44C          MOV     AH,4C
076B:011E CD21          INT     21
-
```

b)Subtraction

```
D:\>debug bcdSUB.exe;
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 B100          MOV     CL,00
076B:0107 A00000        MOV     AL,[0000]
076B:010A 8A1E0100      MOV     BL,[0001]
076B:010E 2AC3          SUB     AL,BL
076B:0110 2F            DAS
076B:0111 730B          JNB     011E
076B:0113 B299          MOV     DL,99
076B:0115 2AD0          SUB     DL,AL
076B:0117 FEC2          INC     DL
076B:0119 FEC1          INC     CL
076B:011B 8AC2          MOV     AL,DL
076B:011D 2F            DAS
076B:011E A20200        MOV     [0002],AL
```

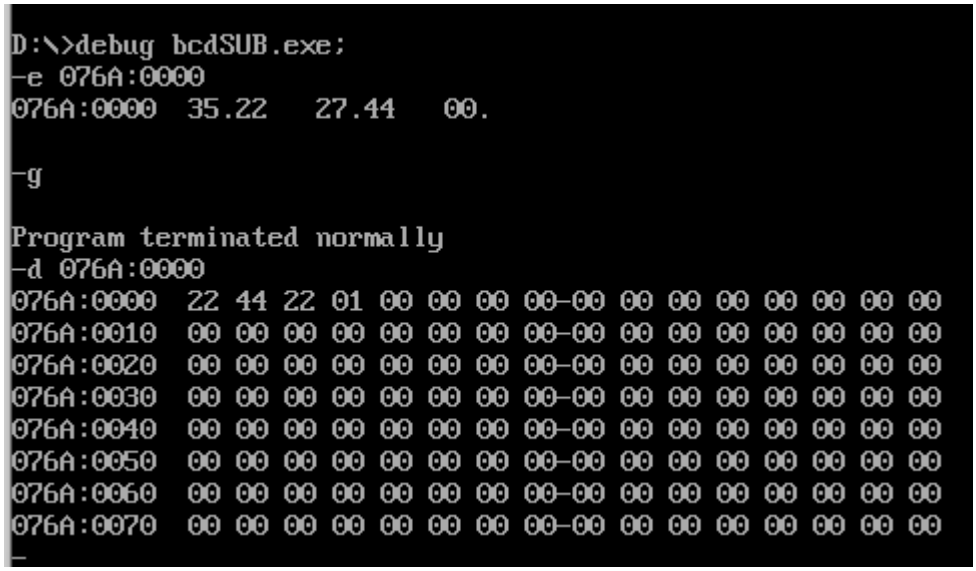
OUTPUT :

a)

```
-d 076A:000
076A:0000  27 35 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
-g

Program terminated normally
-d 076A:0000
076A:0000  27 35 62 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
-
```

b)

```
-d 076A:0000
076A:0000  35 27 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
-g

Program terminated normally
-d 076A:0000
076A:0000  35 27 08 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```



RESULT :

BCD addition and subtraction operations are performed using 8086 instructions in masm and the output is verified.

Exp No:8 # Floating Point Operations

Name: Shreya Sriram
Register Number:195001106
Date: 30/08/2021

AIM:

To implement floating point operations- addition and subtraction using masm

ALGORITHM:

a) Addition

1. Move the starting address of data segment to AX register and move the data from AX register to DS register.

2. Initialize the 8087-stack using FINIT command. This stack will be used for floating point operations.

3. Load the floating-point number from variable X to the top of the stack i.e. ST (0) using FLD command.

4. Now again load the floating-point number from variable Y to the top of the stack i.e. ST (0) using FLD command. The previous stack top contents will be pushed into the stack.

5. Using FADD add ST(0) and ST(1) which stores the result in ST(0).

6. Using FST store the resulting floating-point number from the top of the stack to the variable SUM.

7. Move the hexadecimal value 4C into AH register. INT 21H means invoke the interrupt identified by the hexadecimal number 21. In MS-DOS, invoking interrupt 21h while AH = 4Ch causes the current process to terminate and uses the value of register AL as the exit code of the process.

b) Subtraction:

1.Initialize 8087 stack using FINIT command. This stack is used for floating point operations

2.Load the floating point number from variable X to the top of stack(ST(0)) using FLD command

3.Load the floating point number from variable Y to the top of stack(ST(0)) using FLD command

4.Use FSUB to subtract ST(0) from ST(1) and store the result in ST(0)

5. Use FST to store the resulting floating point number from top of stack to variable DIFF
6. Terminate the program

PROGRAM:

a) Addition

CODE	COMMENTS
assume cs:code, ds:data	Assign code segment to variable code and data segment to variable data
	Start of data segment
data segment	Directive to assign offset address
org 00h	Initialize 2 floating point values
x dd 20.4375	
org 10h	
y dd 20.4375	
org 20h	Declare address space for result
sum dd ?	Data segment ends
data ends	Start of code segment
code segment	

start:
 	
mov ax,data
 	
mov ds,ax
 	
finit
 	
fld x
 	
fld y
 	
fadd st(0),st(1)
 	
fst sum
 	
mov ah,4ch
 	
int 21h
 	
code ends
 	
end start
	Start label
	
	Move the data segment to DS through the AX register
	
	initialize 8087 stack
	
	load X into ST(0)
	
	load Y into ST(0)
	
	ST(0) = X+Y
	
	store ST(0) in sum
	
	setup function-4C of the int21
	
	call BIOS int21 to return to DOS
	
	end of code segment
	
	End of start label

b) Subtraction

Code	Comments

assume cs:code, ds:data	Assign code segment to variable code and data segment to variable data
	Start of data segment
data segment	Directive to assign offset address
org 00h	Initialize 2 floating point values
x dd 20.4375	
org 10h	
y dd 0.125	
org 20h	Declare address space for result
sum dd ?	Data segment ends
data ends	Start of code segment
code segment	

start:	Start label
mov ax,data	Move the data segment to DS through the AX register
mov ds,ax	initialize 8087 stack
finit	load X into ST(0)
fld x	load Y into ST(0)
fld y	ST(0) = X-Y
fsub st(0),st(1)	store ST(0) in diff
fst diff	setup function
mov ah,4ch	4C of the int21
int 21h	call BIOS int21 to return to DOS
code ends	end of code segment
end start	End of start label

Unassembles code :

a)addition

```
D:\>debug floatADD.exe;
-u
076D:0000 B86A07        MOV     AX,076A
076D:0003 8ED8          MOV     DS,AX
076D:0005 9B            WAIT
076D:0006 DBE3                  FINIT
076D:0008 9B            WAIT
076D:0009 D9060000              FLD     DWORD PTR [0000]
076D:000D 9B            WAIT
076D:000E D9061000              FLD     DWORD PTR [0010]
076D:0012 9B            WAIT
076D:0013 D8C1                  FADD    ST,ST(1)
076D:0015 9B            WAIT
076D:0016 D9162000              FST     DWORD PTR [0020]
076D:001A B44C          MOV     AH,4C
076D:001C CD21          INT     21
076D:001E F8            CLC
076D:001F B700          MOV     BH,00
-S
```

B) subtraction

```
D:\>debug flosub.exe
-u
076D:0000 B86A07        MOV     AX,076A
076D:0003 8ED8          MOV     DS,AX
076D:0005 9B            WAIT
076D:0006 DBE3                  FINIT
076D:0008 9B            WAIT
076D:0009 D9060000              FLD     DWORD PTR [0000]
076D:000D 9B            WAIT
076D:000E D9061000              FLD     DWORD PTR [0010]
076D:0012 9B            WAIT
076D:0013 D8E1                  FSUB    ST,ST(1)
076D:0015 9B            WAIT
076D:0016 D9162000              FST     DWORD PTR [0020]
076D:001A B44C          MOV     AH,4C
076D:001C CD21          INT     21
```

Snapshots:

a) Addition

```
-d 076a:0000
076A:0000  00 80 A3 41 00 00 00 00-00 00 00 00 00 00 00 00
076A:0010  00 80 A3 41 00 00 00 00-00 00 00 00 00 00 00 00
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0030  B8 6A 07 8E D8 9B DB E3-9B D9 06 00 00 9B D9 06
076A:0040  10 00 9B D8 C1 9B D9 16-20 00 B4 4C CD 21 F8 B7
076A:0050  00 8A 87 48 2F D0 D8 73-17 E8 B6 00 8A 5E F8 B7
076A:0060  00 8A 87 48 2F D0 D8 73-07 53 B0 01 50 E8 73 01
076A:0070  A0 B6 2C 3A 46 F8 74 7E-C7 46 FA 00 00 8A 46 F8
-g

Program terminated normally
-d 076a:0000
076A:0000  00 80 A3 41 00 00 00 00-00 00 00 00 00 00 00 00
076A:0010  00 80 A3 41 00 00 00 00-00 00 00 00 00 00 00 00
076A:0020  00 80 23 42 00 00 00 00-00 00 00 00 00 00 00 00
076A:0030  B8 6A 07 8E D8 9B DB E3-9B D9 06 00 00 9B D9 06
076A:0040  10 00 9B D8 C1 9B D9 16-20 00 B4 4C CD 21 F8 B7
076A:0050  00 8A 87 48 2F D0 D8 73-17 E8 B6 00 8A 5E F8 B7
076A:0060  00 8A 87 48 2F D0 D8 73-07 53 B0 01 50 E8 73 01
076A:0070  A0 B6 2C 3A 46 F8 74 7E-C7 46 FA 00 00 8A 46 F8
```

b) Subtraction

```
-d 076a:0000
076A:0000  00 80 A3 41 00 00 00 00-00 00 00 00 00 00 00 00
076A:0010  00 00 00 3E 00 00 00 00-00 00 00 00 00 00 00 00
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0030  B8 6A 07 8E D8 9B DB E3-9B D9 06 00 00 9B D9 06
076A:0040  10 00 9B D8 E1 9B D9 16-20 00 B4 4C CD 21 F8 B7
076A:0050  00 8A 87 48 2F D0 D8 73-17 E8 B6 00 8A 5E F8 B7
076A:0060  00 8A 87 48 2F D0 D8 73-07 53 B0 01 50 E8 73 01
076A:0070  A0 B6 2C 3A 46 F8 74 7E-C7 46 FA 00 00 8A 46 F8
-g

Program terminated normally
-d 076a:0000
076A:0000  00 80 A3 41 00 00 00 00-00 00 00 00 00 00 00 00
076A:0010  00 00 00 3E 00 00 00 00-00 00 00 00 00 00 00 00
076A:0020  00 80 A2 C1 00 00 00 00-00 00 00 00 00 00 00 00
076A:0030  B8 6A 07 8E D8 9B DB E3-9B D9 06 00 00 9B D9 06
076A:0040  10 00 9B D8 E1 9B D9 16-20 00 B4 4C CD 21 F8 B7
076A:0050  00 8A 87 48 2F D0 D8 73-17 E8 B6 00 8A 5E F8 B7
076A:0060  00 8A 87 48 2F D0 D8 73-07 53 B0 01 50 E8 73 01
076A:0070  A0 B6 2C 3A 46 F8 74 7E-C7 46 FA 00 00 8A 46 F8
```

Result:

Hence, floating point operations – addition and subtraction were implemented using masm

Exp No:9 **Case Conversions**

Name: Shreya Sriram
Register Number:195001106
Date: 7/09/2021

AIM:

To perform case conversion using 8086 instructions in masm

ALGORITHM:

- Use dos interrupt 21H with AH=01 to read the letters with echo
- Compare the input letter with 60h
- If result is greater, subtract 20h from the input letter
- Else , add 20h to the input letter
- Use AH=2 to display the letters to standard output device
- Move character from AL to DL
- Use AH=2 to display the letters to standard output device
- Terminate the program

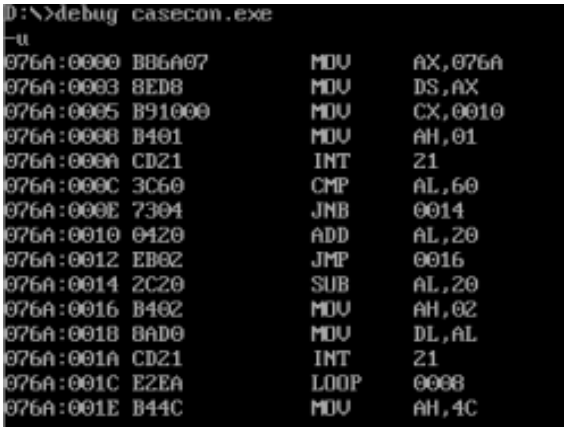
Program :

PROGRAM CODE	COMMENTS

ASSUME CS:CODE,DS:data
data SEGMENT
COUNT equ 10h
data ends
CODE SEGMENT
START:
MOV AX,data
MOV DS,AX
	Declare the code and data segments Data segment starts
	Assign value 10 to COUNT
	End of Data segment
	Code segment starts
	Start label
	Move the data segment to DS through the AX register

MOV CX,COUNT	Move count value into CX register
L1:	Label ‘L1’
MOV AH,1 ; INPUT CHARACTER,	Use dos interrupt 21H with AH=01 to read the letters with echo
INT 21H ; AL = CHARACTER, ASCII(hex) :A-Z=41-5A, a-z=61-7A CMP AL,60H	Compare AL with 60H
JNC UPPER	If carry flag is unset,jump to upper label AL=AL+20
ADD AL,20H	Jump to skip label
JMP SKIP	Label ‘Upper’
UPPER:	AL=AL-20
SUB AL,20H ; CONVERT TO UPPER CASE	Label ‘skip’
SKIP:	Use AH=2 to display the letters to standard output device
MOV AH,2 ; CHARACTER OUTPUT FUNCTION	Move the character into DL register Display to output device using int 21h
MOV DL,AL ; CHARACTER MUST BE IN DL	Loop back to label L1
INT 21H ; DISPLAY THE CHARACTER LOOP L1 ; REPEAT LOOP	Terminate the program
MOV Ah,4CH	End of code segment
INT 21H	End of start label
CODE ENDS	
end start	

UNASSEMBLED CODE



Exp No:10 **Display a string**

Name: Shreya Sriram
Register Number:195001106
Date: 14/09/2021

AIM:

To write and execute 8086 programs for displaying a string.

PROCEDURE:

- Firstly, write the 8086 program using editor(like notepad) and save it with .asm(filename.asm) extension and move to the MASM folder.
  • Now mount the MASM folder in DOSBOX("mount d e:/masm") and then enter into the mounted drive("d:") • Now using "edit filename.asm",we can edit or create a asm file for execution and then save and exit. • Assemble the code using "masm filename.asm" to generate the "filename.obj" file. • Link the file using "link filename.obj;" to generate the executable "filename.exe" file. • Now enter the debug mode using "debug filename.exe" to execute and analyse memory contents. The various commands used in debug mode are as follows:-
    - U :- To display unassembled code
    - D :- Used as 'D segment:offset' to see the content of memory locations starting from segment:offset address.
    - E:- To change the value in memory
    - G:- To execute
    - Q:- To quit

ALGORITHM:

1. START: Move the starting address of data segment to AX register and move the data from AX register to DS register.

   2. Move 9H to AH register.

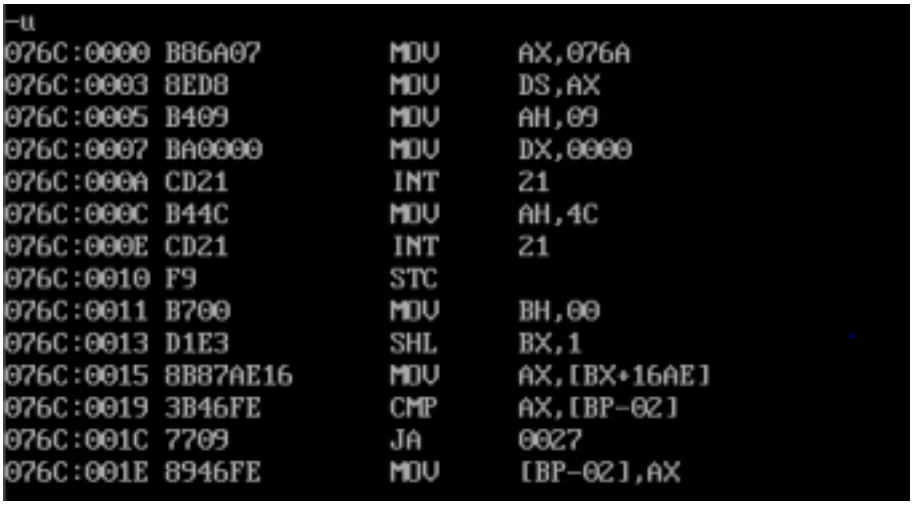
   3. Calling int 21H with AH == 9 will display the contents from the offset stored in DX register.

4. Move the hexadecimal value 4C into AH register. INT 21H means invoke the interrupt identified by the hexadecimal number 21. In MS-DOS, invoking interrupt 21h while AH = 4Ch causes the current process to terminate and uses the value of register AL as the exit code of the process.

	Program	Comments
START:	MOV AX, DATA MOV DS, AX	Transferring the data from DATA to AX register and from AX register to DS register.
	MOV AH,9	DOS FUNCTION #9
	MOV DX, OFFSET MESSAGE	OFFSET OF THE STRING.
	MOV AH,4CH	Setup function-4C of the int21.
	INT 21H	Call BIOS int21 to return to DOS.

SNAPSHOT:

UNASSEMBLED CODE:



SAMPLE I/O:

Result:

Thus the 8086 programs for displaying string is executed successfully using DOS-BOX.

Exp No:11

Display system date and time

Name: Shreya Sriram
Register Number:195001106
Date: 14/09/2021

AIM:

To write and execute 8086 programs for displaying system date and time.

PROCEDURE:

- Firstly, write the 8086 program using editor(like notepad) and save it with .asm(filename.asm) extension and move to the MASM folder.
  • Now mount the MASM folder in DOSBOX("mount d e:/masm") and then enter into the mounted drive("d:") • Now using "edit filename.asm",we can edit or create a asm file for execution and then save and exit. • Assemble the code using "masm filename.asm" to generate the "filename.obj" file.
  • Link the file using "link filename.obj;" to generate the executable "filename.exe" file. • Now enter the debug mode using "debug filename.exe" to execute and analyse memory contents. The various commands used in debug mode are as follows:-
    - U :- To display unassembled code
    - D :- Used as 'D segment:offset' to see the content of memory locations starting from segment:offset address.
    - E:- To change the value in memory
    - G:- To execute
    - Q:- To quit

ALGORITHM:

A) System date

1. START: Move the starting address of data segment to AX register and move the data from AX register to DS register.
2. Move 2ah to AH register.
3. Calling int 21H with 2a in AH register will return year in CX register, month in DH register, day in DL register and day of the week in AL register.
4. Move the offset of the variable DAY in SI register.
5. Move the contents stored in DL register to the location in SI register.
6. Move the offset of the variable MONTH in SI register.

7. Move the contents stored in DH register to the location in SI register.

8. Move the offset of the variable YEAR in SI register.
9. Move the contents stored in CX register to the location in SI register.

10. Move the hexadecimal value 4C into AH register. INT 21H means invoke the interrupt identified by the hexadecimal number 21. In MS-DOS, invoking interrupt 21h while AH = 4Ch causes the current process to terminate and uses the value of register AL as the exit code of the process.

	Program	**Comments**
START:	MOV AX, DATA MOV DS, AX	Transferring the data from DATA to AX register and from AX register to DS register.
	MOV AH, 2AH	AH <- 2AH
	INT 21H	INT 21h /AH=2Ah - get system date
	MOV SI, OFFSET DAY	SI <- DAY
	MOV [SI], DL	[SI] <- DL
	MOV SI, OFFSET MONTH	SI <- MONTH
	MOV [SI], DH	[SI] <- DH
	MOV SI, OFFSET YEAR	SI <- YEAR
	MOV [SI], CX	[SI] <- CX
	MOV AH,4CH	Setup function-4C of the int21.
	INT 21H	Call BIOS int21 to return to DOS.

SNAPSHOT:

UNASSEMBLED CODE:

```
-u
073F:0100 800D00        OR      BYTE PTR [DI],00
073F:0103 0B7379        OR      SI,[BP+DI+79]
073F:0106 7364          JNB     016C
073F:0108 61            DB      61
073F:0109 7465          JZ      0170
073F:010B 2E            CS:
073F:010C 61            DB      61
073F:010D 736D          JNB     017C
073F:010F FC            CLD
073F:0110 96            XCHG    SI,AX
073F:0111 0C00          OR      AL,00
073F:0113 0004          ADD     [SI],AL
073F:0115 44            INC     SP
073F:0116 41            INC     CX
073F:0117 54            PUSH    SP
073F:0118 41            INC     CX
073F:0119 0443          ADD     AL,43
073F:011B 4F            DEC     DI
073F:011C 44            INC     SP
073F:011D 45            INC     BP
073F:011E 21980700      AND     [BX+SI+0007],BX
```

SAMPLE I/O:

```
-d 076a:0000
076A:0000  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-g

Program terminated normally
-d 076a:0000
076A:0000  14 0A E4 07 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

B) System time

a. START: Move the starting address of data segment to AX register and move the data from AX register to DS register.
b. Move 2ch to AH register.
c. Calling int 21H with 2c in AH register will return hour in CH register, minute in CL register and second in DH register.
d. Move the offset of the variable HOUR in SI register.

e. Move the contents stored in CH register to the location in SI register.

f. Move the offset of the variable MINUTE in SI register.

g. Move the contents stored in CL register to the location in SI register.

h. Move the offset of the variable SECOND in SI register.

i. Move the contents stored in DH register to the location in SI register.

j. Move the hexadecimal value 4C into AH register. INT 21H means invoke the interrupt identified by the hexadecimal number 21. In MS-DOS, invoking interrupt 21h while AH = 4Ch causes the current process to terminate and uses the value of register AL as the exit code of the process.

	Program	**Comments**
START:	MOV AX, DATA MOV DS, AX	Transferring the data from DATA to AX register and from AX register to DS register.
	MOV AH, 2CH	AH <- 2CH
	INT 21H	INT 21h /AH=2Ch - get system date
	MOV SI, OFFSET HOUR	SI <- HOUR
	MOV [SI], DL	[SI] <- CH
	MOV SI, OFFSET MINUTE	SI <- MINUTE
	MOV [SI], DH	[SI] <- CL
	MOV SI, OFFSET SECOND	SI <- SECOND

	MOV [SI], CX	[SI] <- DH
	MOV AH,4CH	Setup function-4C of the int21.
	INT 21H	Call BIOS int21 to return to DOS.

Unassembled code:

```
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 B42C          MOV     AH,2C
076B:0107 CD21          INT     21
076B:0109 BE0000        MOV     SI,0000
076B:010C 882C          MOV     [SI],CH
076B:010E BE0100        MOV     SI,0001
076B:0111 880C          MOV     [SI],CL
076B:0113 BE0200        MOV     SI,0002
076B:0116 8834          MOV     [SI],DH
076B:0118 B44C          MOV     AH,4C
076B:011A CD21          INT     21
076B:011C FF7701        PUSH    [BX+01]
076B:011F 40            INC     AX
```

Sample I/O:

```
-d 076a:0000
076A:0000  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-g

Program terminated normally
-d 076a:0000
076A:0000  14 0A 32 00 00 00 00 00-00 00 00 00 00 00 00 00   ..2.............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

Result:

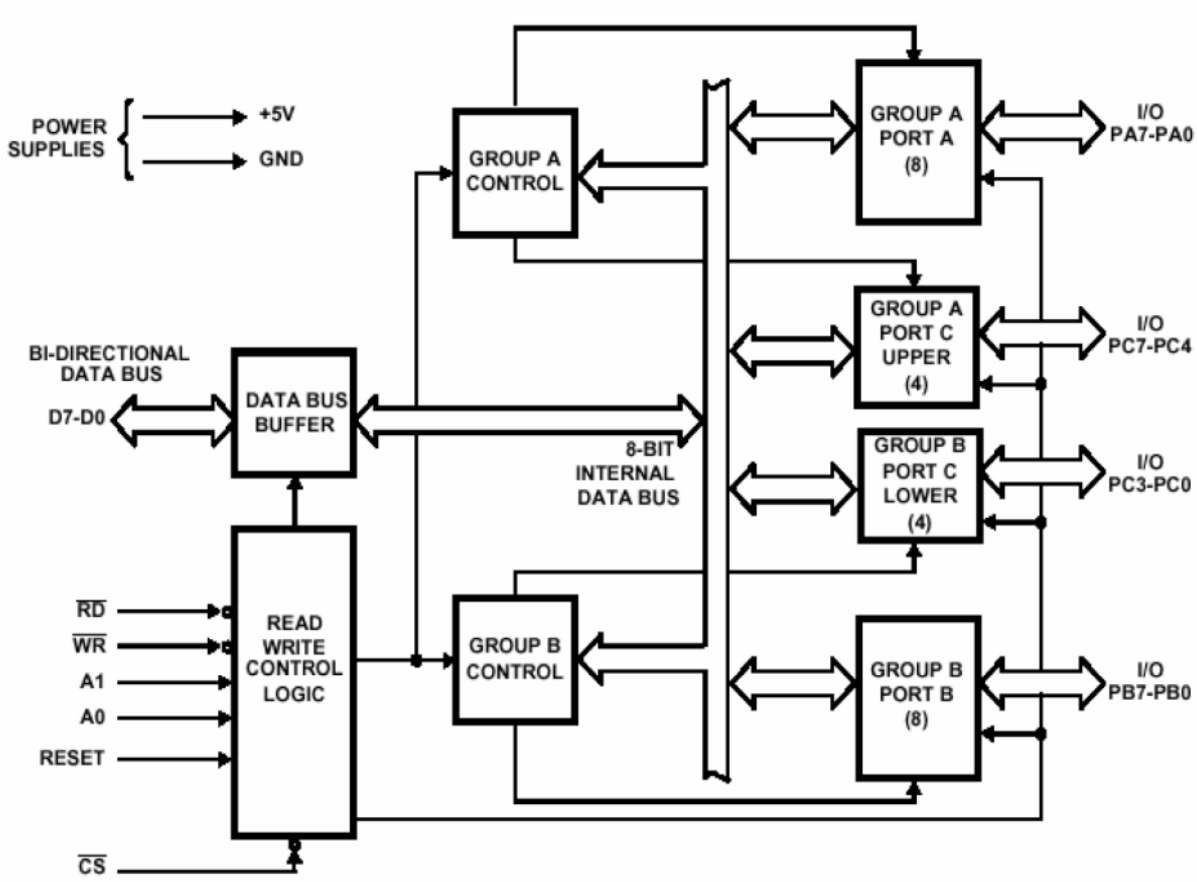
Thus the 8086 programs for displaying system date and time are executed successfully using DOS BOX.

Exp No:12

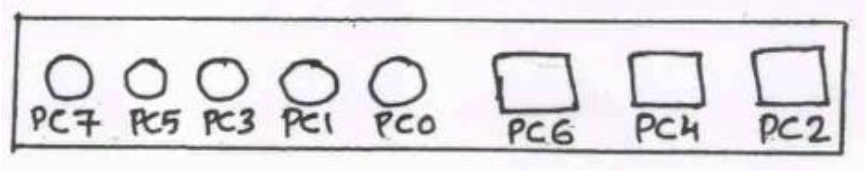
8255 Parallel Interface (PPI)

Name: Shreya Sriram
Register Number:195001106
Date: 30/09/21

ARCHITECTURE DIAGRAM:

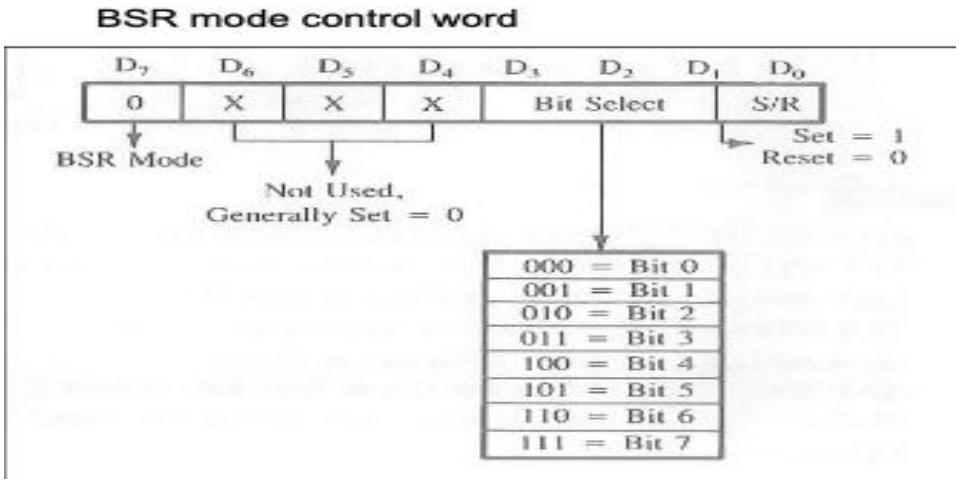


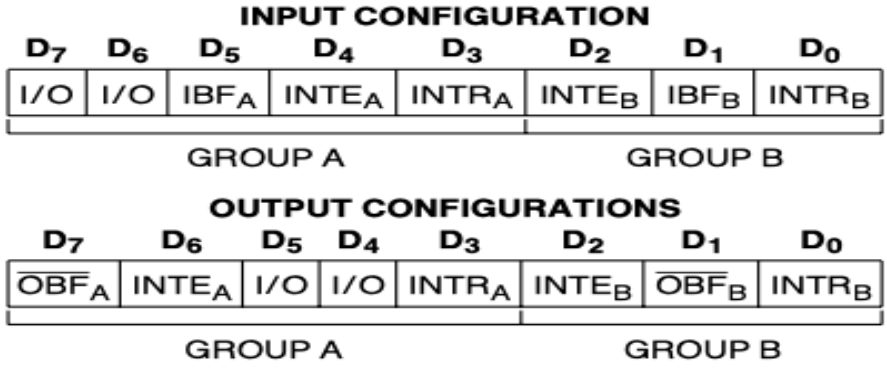
PORT C PATTERN IN BOARD:



PORT ADDRESSES:

PORT ADDRESS	
CONTROL REGISTER	C6
PORT A	C0
PORT B	C2
PORT C	C4





1. **AIM:**

To initialize Port A as an input port in mode-0

ALGORITHM:

a) Move 1500 to SI
b) Move 90 to AL

c) Move AL to C6(output)
d) Move AL to C0 (input)
e) Move AL to [SI]
f) End code

PROGRAM:

CODE	COMMENTS
MOV SI,1500H	; Move 1500 to SI
MOV AL,90	; Move 90 to AL
OUT C6,AL	; Move AL to C6(output)
IN AL,C0	; Move AL to C0 (input)
MOV [SI],AL	; Move AL to [SI]
HLT	; End code

2. AIM:

To initialize Port A as input port and Port B as output port in mode-0

ALGORITHM:

a) Move 90 to AL
b) Move AL to output port C6(control register)
c) Move C0 to AL(Port A)
d) Move AL to C2 (Port B)
e) End code

PROGRAM:

CODE	COMMENTS
MOV AL,90H	; Move 90 to AL
OUT C6,AL	; Move AL to output port C6(control register)
IN AL,C0	; Move C0 to AL(Port A)
OUT C2,AL	; Move AL to C2 (Port B)
HLT	; End code

3. AIM:

To initialize port C as output port in mode-0

ALGORITHM:

a) Move 90 to AL
b) Move AL to output port C6(control register)
c) Move 80 to AL(Port A)
d) Move AL to C2 (Port C)

e) End code

PROGRAM:

CODE	COMMENTS
MOV AL,90H	; Move 90 to AL
OUT C6,AL	; Move AL to output port C6(control register)
IN AL,80	; Move 80 to AL(Port A)
OUT C4,AL	; Move AL to C2 (Port C)
HLT	; End code

4. AIM:

To initialize Port C as an output port in mode-0 and to explain the bit set and reset feature of Port C

ALGORITHM:

a) Move 80 to AL
b) Move AL to output port C6(control register)
c) Move 01 to AL
d) Move AL to C4 (Port C)
e) Move 07 to AL
f) Move AL to Port C
g) End code

PROGRAM:

CODE	COMMENTS
MOV AL,80H	; Move 80 to AL
OUT C6,AL	; Move AL to output port C6(control register)
MOV AL,01	; Move 01 to AL
OUT C4,AL	; Move AL to C4 (Port C)
MOV AL,07	; Move 07 to AL
OUT C4,AL	; Move AL to Port C
HLT	; End code

5. AIM:

To initialize Port C as an input port in Mode-0

ALGORITHM:

a) Move 1200 to SI

b) Move 99 to AL
c) Move AL to C6(control register)
d) Move AL to C4 (Port C)
e) Move AL to [SI]
f) End code

PROGRAM:

CODE	COMMENTS
MOV SI,1200H	; Move 1200 to SI
MOV AL,99	; Move 99 to AL
OUT C6,AL	; Move AL to C6(control register)
IN AL,C4	; Move AL to C4 (Port C)
MOV [SI],AL	; Move AL to [SI]
HLT	; End code

6. AIM:

To verify the working of 8255 in mode 1

ALGORITHM:

a) Move 80 to AL
b) Move AL to output port C6(control register)
c) Move 01 to AL
d) Move AL to C4 (Port C)
e) Move 07 to AL
f) Move AL to Port C
g) End code

PROGRAM:

CODE	COMMENTS
MOV AL,B4	; Move 80 to AL
OUT C6,AL	; Move AL to output port C6(control register)
READ: IN AL,C4	; Move 01 to AL
AND AL,20	; Move AL to C4 (Port C)
JZ READ	; Move 07 to AL
IN AL,C0	; Move AL to Port C
OUT C2,AL	; End code
HLT	

RESULT:

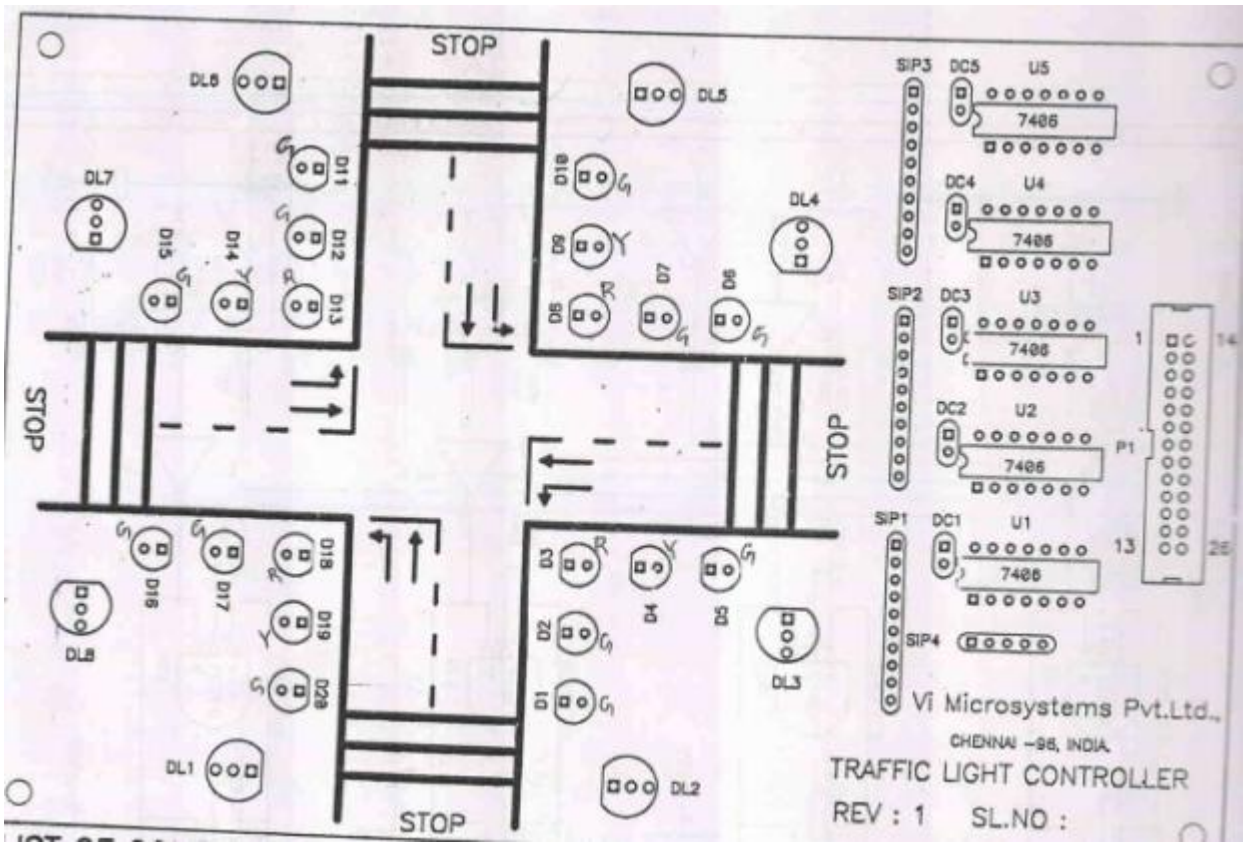
Parallel interface experiments are successfully implemented using 8255 interface

Exp No:13

Traffic Light Controller

Name: Shreya Sriram
Register Number:195001106
Date: 30/09/21

BLOCK DIAGRAM:



PORTS:

PORT A

BITS	A7	A6	A5	A4	A3	A2	A1	A0
LEDS	D8	D7	D6	D5	D4	D3	D2	D1

PORT B

BITS	B7	B6	B5	B4	B3	B2	B1	B0
LEDS	D20	D19	D18	D17	DL7 DL8	DL5 DL6	DL3 DL4	DL1 DL2

PORT C:

BITS	C7	C6	C5	C4	C3	C2	C1	C0
LEDS	D16	D15	D14	D13	D12	D11	D10	D9

CONTROL WORD LOOKUP TABLE

12 > Port C 27 > Port B 44 > Port A Call Delay	Fig 1
48 >Port A Call Delay1	Fig 2
10 > Port C 2B > Port B 92 > Port A Call Delay	Fig 3
4B > Port B Call Delay1	Fig 4
10 > Port C 9D > Port B 84 > Port A Call Delay	Fig 5
20 > Port C Call Delay1	Fig 6
48 > Port C 2E > Port B 84 > Port A Call Delay	Fig 7
49 > Port C 04 > Port A Call Delay1	Fig 8

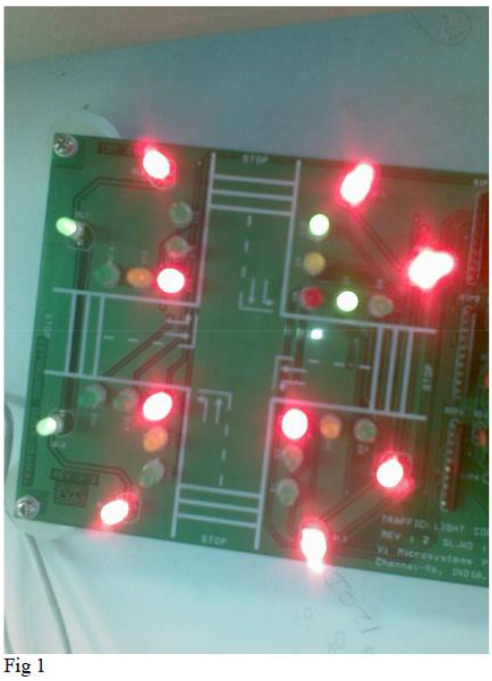


Fig 1

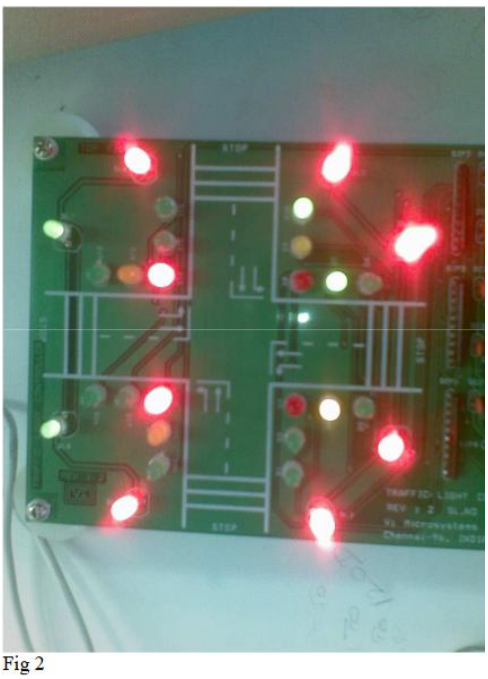


Fig 2

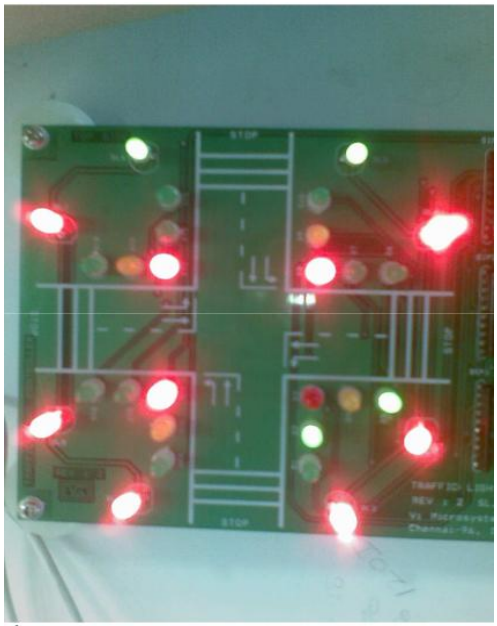


Fig 3

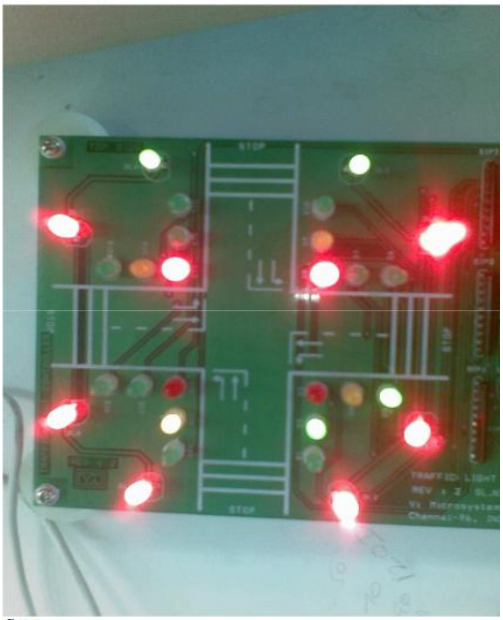


fig 4

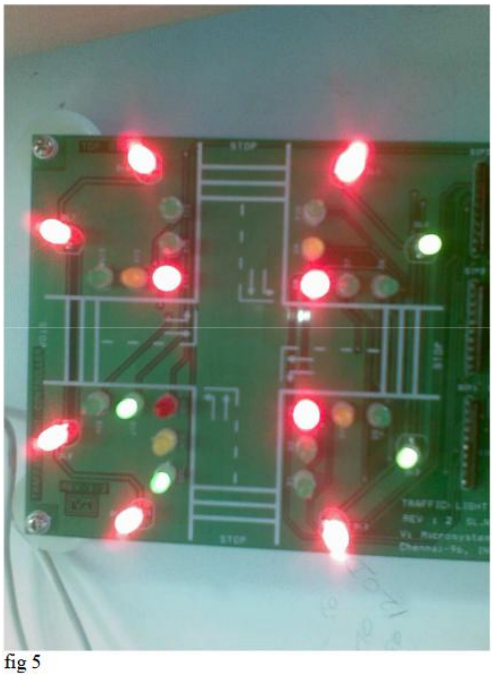


fig 5

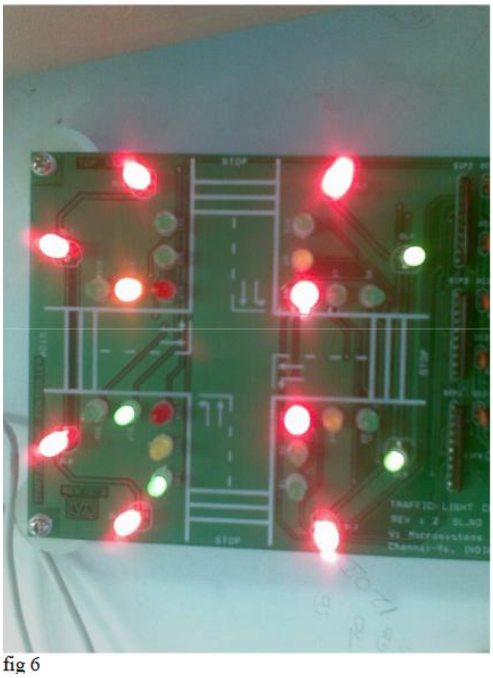


fig 6

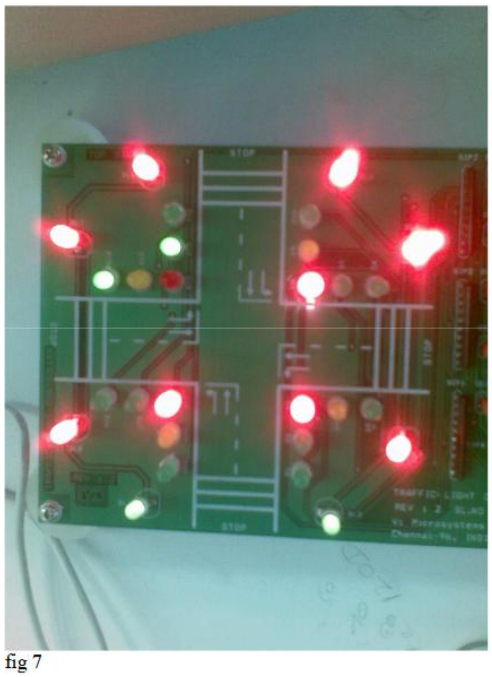


fig 7

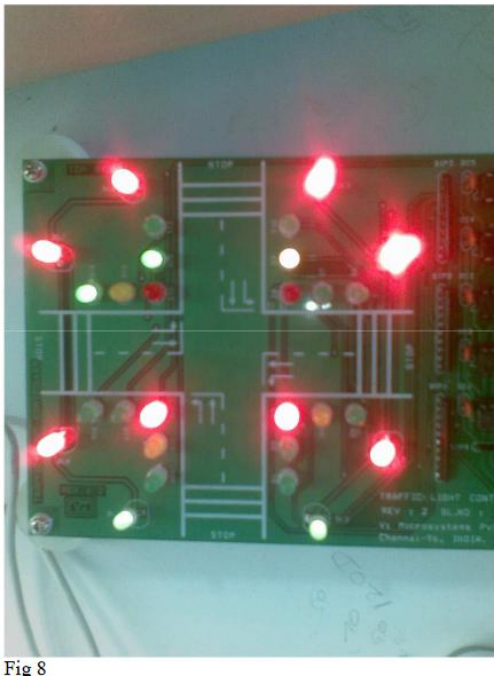


Fig 8

AIM:

To implement a traffic light controller by interfacing with an 8086 microprocessor.

ALGORITHM:

1) EQU Directive used to assign values to CNTL ,PORT A ,PORT B and PORT C
2) Move control word into AL
3) Output the content of AL into CNTL
4) Load the lookup table with values into BX
5) Move Label into SI
6) Jump to 13
7) Move SI ro AL
8) Output AL contents to PORT A
9) Call DELAY 1 routine
10) Increment SI and BX values
11) Repeat 7-10 for PORT B AND PORT C
12) Jump to 6
13) Output AL content to PORT C and increment BX
14) Move BX to AL
15) Output AL content to PORT B and increment BX
16) Move BX to AL
17) Output AL content to PORT A and increment BX
18) Return to after calling routine
19) Delay and Delay 1 defined
20) 40H move to DI
21) Move FFFH to DX
22) Decrement DX till DX=0
23) Decrement till DI =0
24) Return to after calling routine
25) Load lookup table with the required values
26) STOP

PROGRAM:

```
                    ORG   1000H

        CNTRL       EQU   26H
        PORTA       EQU   20H
        PORTB       EQU   22H
        PORTC       EQU   24H

1000  C6 C0 80      START:  MOV   AL,80H
1003  E6 26                 OUT   CNTRL,AL
1005  C7 C3 73 10   REPEAT: MOV   BX,LOOK UP
1009  C7 C6 7F 10          MOV   SI,LABEL
100D  E8 33 00            CALL  OUT
1010  8A 04              MOV   AL,[SI]
1012  E6 20              OUT   PORTA,AL
1014  E8 4D 00           CALL  DELAY1
1017  46                INC   SI
1018  43                INC   BX
1019  E8 27 00           CALL  OUT
101C  8A 04              MOV   AL,[SI]
101E  E6 22              OUT   PORTB,AL
1020  E8 41 00           CALL  DELAY1
1023  46                INC   SI
1024  43                INC   BX
1025  E8 1B 00           CALL  OUT
1028  8A 04              MOV   AL,[SI]
102A  E6 24              OUT   PORTC,AL
102C  E8 35 00           CALL  DELAY1
102F  46                INC   SI
1030  43                INC   BX
1031  E8 0F 00           CALL  OUT
1034  8A 04              MOV   AL,[SI]
1036  E6 24              OUT   PORTC,AL
1038  46                INC   SI
1039  8A 04              MOV   AL,[SI]
103B  E6 20              OUT   PORTA,AL
103D  E8 24 00           CALL  DELAY1
1040  E9 C2 FF           JMP   REPEAT
1043  8A 07              OUT:  MOV   AL,[BX]
1045  E6 24              OUT   PORTC,AL
1047  43                INC   BX
1048  8A 07              MOV   AL,[BX]
```

```
104A  E6 22                          OUT     PORTB,AL
104C  43                             INC     BX
104D  8A 07                          MOV     AL,[BX]
104F  E6 20                          OUT     PORTA,AL
1051  E8 01 00                       CALL    DELAY
1054  C3                             RET
1055  C7 C7 40 00          DELAY:    MOV     DI,00040H
1059  C7 C2 FF FF          A:        MOV     DX,0FFFFH
105D  4A                   A1:       DEC     DX
105E  75 FD                          JNZ     A1
1060  4F                             DEC     DI
1061  75 F6                          JNZ     A
1063  C3                             RET
1064  C7 C7 15 00          DELAY1:   MOV     DI,00015H
1068  C7 C2 FF FF          B:        MOV     DX,0FFFFH
106C  4A                   B1:       DEC     DX
106D  75 FD                          JNZ     B1
106F  4F                             DEC     DI
1070  75 F6                          JNZ     B
1072  C3                             RET
1073  12 27 44 10          LOOK UP:  DB      12H,27H,44H,10H
1077  2B 92 10 9D                              2BH,92H,10H,9DH
107B  84 48 2E 84                              84H,48H,2EH,84H

107F  48 4B 20 49          LABEL :   DB      48H,4BH,20H,49H
1083  04                                      04H
107C                                 END
```

RESULT:

Traffic light controller has been successfully implemented

Exp No:14

Stepper Motor Interfacing

Name: Shreya Sriram
Register Number:195001106
Date: 07/10/21

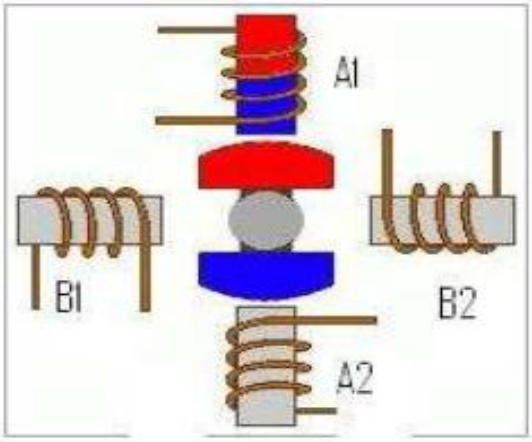
AIM:

To interface the stepper motor with 8086 and run in clockwise and anticlockwise directions

ALGORITHM:

a) Move 1018 to D1
b) Move 04 to CL
c) Move address of DI to AL
d) Move AL to C0
e) Move 1010 to DX
f) Decrement DX
g) Jump to 100F if DX not zero
h) Increment DI
i) Loop to 1007
j) Jump to start

DIAGRAM:



LOOKUP TABLE:

Address	For Clockwise Rotation	For Anticlockwise Rotation
1018	08	08
1019	01	02
101A	04	04
101B	02	01

Clock wise						Anticlock wise					
Step	A1	A2	B1	B2		Step	A1	A2	B1	B2	
1	1	0	0	0	8	1	1	0	0	0	8
2	0	0	0	1	1	2	0	0	1	0	2
3	0	1	0	0	4	3	0	1	0	0	4
4	0	0	1	0	2	4	0	0	0	1	1

PROGRAM:

CODE	COMMENTS
MOV DI,1018	; Move 1018 to D1
MOV CL,04	; Move 04 to CL
MOV AL,[DI]	; Move address of DI to AL
OUT C0,AL	; Move AL to C0
MOV DX,1010	; Move 1010 to DX
DEC DX	; Decrement DX
JNZ 100F	; Jump to 100F if DX not zero
INC DI	; Increment DI
LOOP 1007	; Loop to 1007
JMP 1000	; Jump to start

RESULT:

The stepper motor has been successfully interfaced with 8086 and made to run in clockwise and anticlockwise directions

Exp No:15

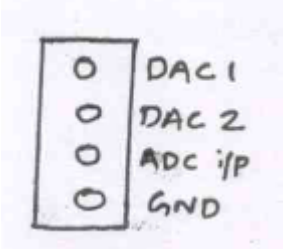
Wave Generation using DAC interface

Name: Shreya Sriram
Register Number:195001106
Date: 07/10/21

INPUT OUTPUT TABLE

INPUT DATA IN HEX	O/P VOLTAGE(V)
00	-5
01	-4.96
.	.
.	.
.	.
7F	0
.	.
.	.
.	.
FE	4.96
FF	5

PIN CONNECTION IN BOARD



DATA ADDRESS REGISTER

DAC 1	C0
DAC 2	C8

a) **AIM:**

To generate a square wave at the output of DAC 2

ALGORITHM:

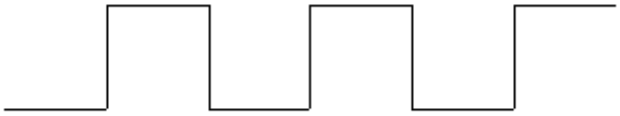
a) Move 0 to AL
b) Move AL to C8 (DAC2)

c) Call line stored at 1010
d) Move FF to AL
e) Move AL to C8
f) Call mov cx,05ff
g) Jump to start
h) Move 05FF to CX
i) Loop statement 1013
j) Return to main

PROGRAM:

CODE	COMMENTS
MOV AL,00	; Move 0 to AL
OUT C8,AL	; Move AL to C8 (DAC2)
CALL 1010	; Call line stored at 1010
MOV AL,FF	; Move FF to AL
OUT C8,AL	; Move AL to C8
CALL 1010	; Call mov cx,05ff
JMP 1000	; Jump to start
MOV CX,05FF	; Move 05FF to CX
LOOP 1013	; Loop statement 1013
RET	; Return

OUTPUT:



b) AIM:

To create a saw-tooth wave at the output of DAC 2

ALGORITHM:

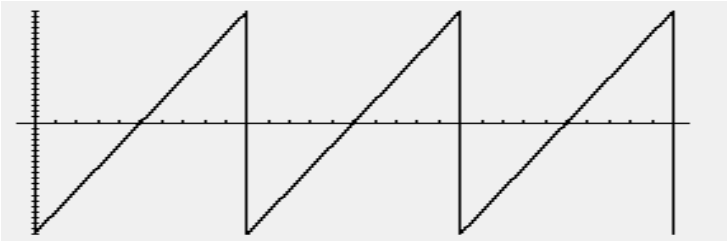
a) Move 0 to AL
b) Move AL to C8 (DAC2)
c) Increment AL
d) Jump to 1002 if not 0
e) Jump to location 1000

PROGRAM:

CODE	COMMENTS
MOV AL,00	; Move 0 to AL
OUT C8,AL	; Move AL to C8 (DAC2)

INC AL	; Increment AL
JNZ 1002	; Jump to 1002 if not 0
JMP 1000	; Jump to location 1000

OUTPUT:



c) AIM:

To generate triangular waveform at DAC 2 output

ALGORITHM:

a) Move 0 to BL
b) Move BL to AL (DAC2)
c) Move AL to C8
d) Increment BL
e) Jump to location 1002 from BL
f) Move FF to BL
g) Move BL to AL
h) Move AL to C8
i) Decrement BL
j) Jump to 100C if BL is not zero
k) Jump to location 1000

PROGRAM:

CODE	COMMENTS
MOV BL,00	; Move 0 to BL
MOV AL,BL	; Move BL to AL (DAC2)
OUT C8,AL	; Move AL to C8
INC BL	; Increment BL
JNZ 1002	; Jump to location 1002 from BL
MOV BL,FF	; Move FF to BL
MOV AL,BL	; Move BL to AL
OUT C8,AL	; Move AL to C8
DEC BL	; Decrement BL
JNZ 100C	; Jump to 100C if BL is not zero
JMP 1000	; Jump to location 1000

OUTPUT:

RESULT:

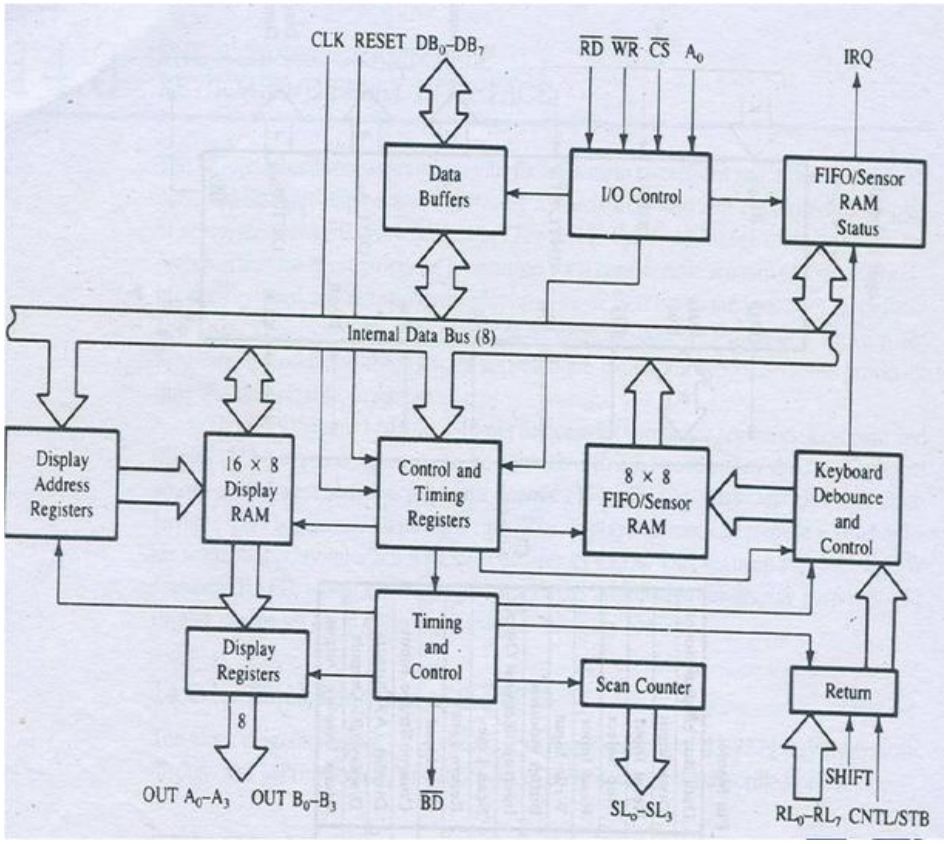
The waves are successfully generated using DAC interface

Exp No:16

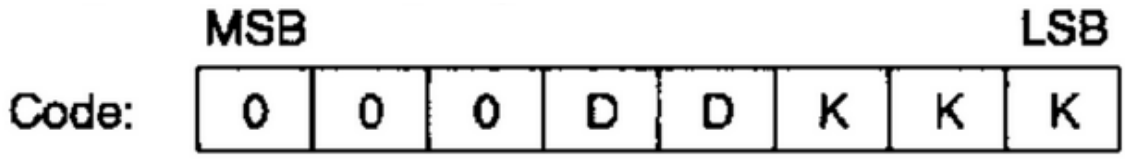
Keyboard and Display Interface - 8279

Name: Shreya Sriram
Register Number:195001106
Date: 07/10/21

ARCHITECTURE DIAGRAM OF 8279:



KEYBOARD/DISPLAY MODE SET



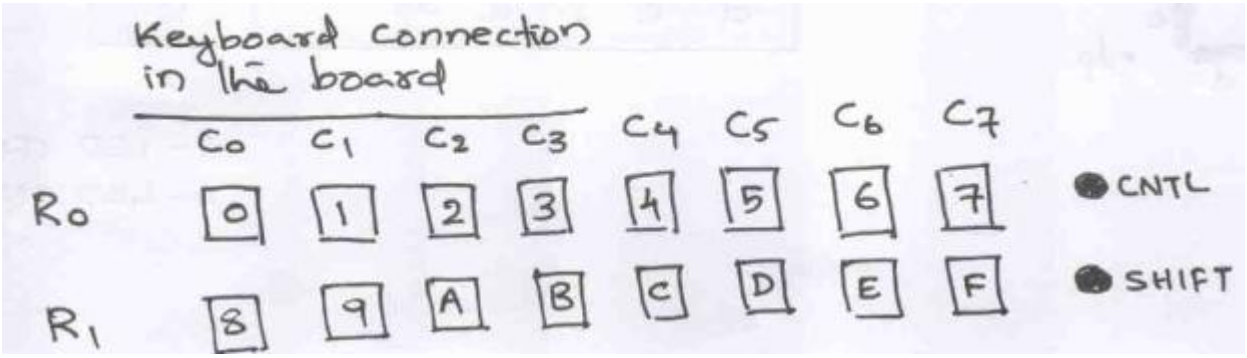
DISPLAY MODE

DD		
0	0	8 8-bit character display—Left entry
0	1	16 8-bit character display—Left entry*
1	0	8 8-bit character display—Right entry
1	1	16 8-bit character display—Right entry

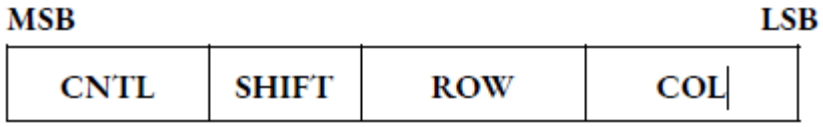
KEYBOARD MODE

KKK			
0	0	0	Encoded Scan Keyboard—2 Key Lock-out*
0	0	1	Decoded Scan Keyboard—2-Key Lock-out
0	1	0	Encoded Scan Keyboard—N-Key Roll-over
0	1	1	Decoded Scan Keyboard—N-Key Roll-over
1	0	0	Encoded Scan Sensor Matrix
1	0	1	Decoded Scan Sensor Matrix
1	1	0	Strobed Input, Encoded Display Scan
1	1	1	Strobed Input, Decoded Display Scan

KEYBOARD CONNECTION IN THE BOARD



KEYBOARD DATA FORMAT



CONTROL REGISTER	-	C2
DATA REGISTER	-	C0

AIM:

a) Read a key from the keyboard and store in a memory location

ALGORITHM:

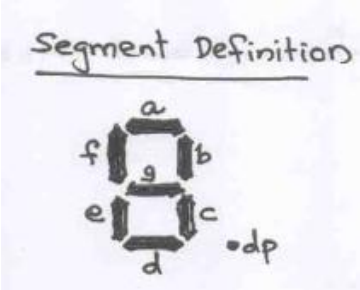
a) Move 1100 to BX
b) Move control register contents to AL
c) Check if AL is equal to 7
d) Unless it is 7, goto loop
e) Move 40 to AL
f) Move AL to control register
g) Move contents of data register to AL
h) Move AL to address of BX
i) End

PROGRAM:

CODE	COMMENTS
MOV BX,1100	; Move 1100 to BX
LOOP: IN AL,C2	; Move control register contents to AL
TEST AL,07	; Check if AL is equal to 7
JZ LOOP	; Unless it is 7, goto loop
MOV AL,40	; Move 40 to AL
OUT C2,AL	; Move AL to control register
IN AL,C0	; Move contents of data register to AL
MOV [BX],AL	; Move AL to address of BX
HLT	; End

AIM:

b) Roll the message “HELP US” in the display



DATA FORMAT:

D7	D6	D5	D4	D3	D2	D1	D0
d	c	b	a	dp	g	f	e

H	1001 1000	98
E	0110 1000	68
L	0111 1100	7C
P	1100 1000	C8
U	0001 1100	1C
S	0110 1001	29

ALGORITHM:

MAIN CODE:

a) Set origin to 1000
b) Move 1200 to SI
c) Move 0F to CX
d) Move AL to C2 port
e) Move CC to AL
f) Move AL to C2
g) Move 90 to AL
h) Move AL to C2
i) Move SI to AL
j) Move AL to C0
k) Call delay function
l) Increment SI
m) Loop statement

DELAY FUNCTION:

a) Move 03 to D1
b) Move FFFF to DX
c) Decrement DX
d) If DX to not zero, jump to B1
e) Decrement DI
f) Jump to B if not zero
g) Return to main

PROGRAM:

CODE	**COMMENTS**
ORG 1000H	; Set origin to 1000
START: MOV SI,1200	; Move 1200 to SI
MOV CX,000F	; Move 0F to CX
OUT C2,AL	; Move AL to C2 port
MOV AL,CC	; Move CC to AL
OUT C2,AL	; Move AL to C2
MOV AL,90	; Move 90 to AL
OUT C2,AL	; Move AL to C2
NEXT: MOV AL,[SI]	; Move SI to AL
OUT C0,AL	; Move AL to C0
CALL DELAY	; Call delay function
INC SI	; Increment SI
LOOP NEXT	; Loop statement
JMP	
DELAY: MOV D1,0003	Move 03 to D1
B: MOV DX,FFFF	Move FFFF to DX
B1: DEC DX	Decrement DX
JNZ B1	If DX to not zero, jump to B1
DEC DI	Decrement DI
JNZ B	Jump to B if not zero
RET	Return to main

RESULT:

The keyboard and display operations have been implemented successfully using 8279

Exp No:17

8 bit arithmetic operations using 8051

Name: Shreya Sriram
Register Number:195001106
Date: 19/10/21

AIM:

To perform 8 bit operations – addition, subtraction, multiplication, division using 8051

1) **Addition**

ALGORITHM:

a) Move 0 to R0(represents carry)
b) Move num1 to a
c) Add num2 to a
d) If no carry, move to label
e) Else, increment carry
f) Move a to dptr
g) Mov R0 to a
h) Increment dptr and move a to dptr

CODE:

MOV R0, #00	Move 0 to R0
MOV A, #FE	Move num1 to A
ADD A, #01	Move num2+num1 to A
JNC LABEL	Move to label if no carry
INC R0	Increment R0
LABEL: MOV DPTR, #4150	Move a location to dptr
MOVX @DPTR, A	Move a to the location
MOV A, R0	Move R0 to a
INC DPTR	Increment dptr
MOVX @DPTR, A	Move a to dptr
HERE: SJMP HERE	Jump here

OUTPUT:

A = FE

DPTR = 4150

[4150] = FF

[4151] = 0

2) Subtraction

ALGORITHM:

a) Move 0 to R0(represents carry)
b) Move num1 to a
c) Subtract num2 from a and store in a
d) If no carry, move to label
e) Else, increment carry
f) Complement a and increment a
g) Move a to dptr
h) Move R0 to a
i) Increment dptr and move a to dptr

CODE:

MOV R0, #00	Move 0 to R0
MOV A, #FE	Move num1 to A
SUBB A, #01	Move num2-num1 to A
JNC LABEL	Move to label if no carry
INC R0	Increment R0
CPL A	Complement A
INC A	Increment A
LABEL: MOV DPTR, #4150	Move a location to dptr
MOVX @DPTR, A	Move a to the location
MOV A, R0	Move R0 to a
INC DPTR	Increment dptr
MOVX @DPTR, A	Move a to dptr
HERE: SJMP HERE	Jump here

OUTPUT:

A = FE

DPTR = 4150

[4150] = FD

[4151] = 0

3) Multiplication

ALGORITHM:

a) Move num1 to a
b) Move num2 to b
c) Multiply ab
d) Move a to dptr
e) Increment dptr
f) Move b to a
g) Move a to dptr

CODE:

MOV R0, #00	Move 0 to R0
MOV A, #FE	Move num1 to A
MUL AB	Move num2*num1 to A
MOV DPTR, #4150	Move a location to dptr
MOVX @DPTR, A	Move a to the location
MOV A, R0	Move R0 to a
INC DPTR	Increment dptr
MOV A,B	Move b to a
MOVX @DPTR, A	Move a to dptr
HERE: SJMP HERE	Jump here

OUTPUT:

A = 03

B = 05

DPTR = 4150

[4150] = 0F

[4151] = 00

4) Division

ALGORITHM:

a) Move num1 to a
b) Move num2 to b
c) Divide a and b and store in a
d) Move a to dptr

e) Increment dptr
f) Move b to a
g) Move a to dptr

CODE:

MOV R0, #00	Move 0 to R0
MOV A, #FE	Move num1 to A
DIV AB	Move num2/num1 to A
MOV DPTR, #4150	Move a location to dptr
MOVX @DPTR, A	Move a to the location
MOV A, R0	Move R0 to a
INC DPTR	Increment dptr
MOV A,B	Move b to a
MOVX @DPTR, A	Move a to dptr
HERE: SJMP HERE	Jump here

OUTPUT:

A = 0F

B = 07

DPTR = 4150

[4150] = 01

[4151] = 08

RESULT:

The 8 bit arithmetic operations are successfully implemented in 8051

Exp No:18

Cube of a number using 8051

Name: Shreya Sriram
Register Number:195001106
Date: 26/10/21

AIM:

To perform cube of a 8 bit number

ALGORITHM:

a) Move num to A
b) Move num to B
c) Move the number to R0 (three copies of the number)
d) Multiply A*B and store in A
e) Clear B
f) Move R0 to B
g) Multiply A*B and store in A
h) Move A to a location in external memory using DPTR
i) Increment DPTR
j) Move B to A (higher bits)
k) Move A to DPTR

CODE:

```
MOV A, #03              Move num to A
MOV B, #03              Move num to B
MOV R0, #3              Move num to R0
MUL AB                  Multiply A*B and store in A
CLR B                   Clear B(higher bits)
MOV B, R0               Move R0 to B
MUL AB                  Multiply A*B and store in A
MOV DPTR, #4150         Move a location to dptr
MOVX @DPTR, A           Move A to the location
INC DPTR                Increment dptr
MOV A,B                 Move B to A
MOVX @DPTR, A           Move A to dptr
HERE: SJMP HERE         Jump here
```

OUTPUT:

```
A = 03
DPTR = 4150
[4150] = 1B
[4151] = 00
```

RESULT:

The cube of a number is performed and evaluated in 8051

Exp No:19

BCD to ASCII conversion using 8051

Name: Shreya Sriram
Register Number:195001106
Date: 26/10/21

AIM:

Perform 8 bit BCD to ASCII conversion using 8051

ALGORITHM:

a) Move BCD number to A
b) Move A to R0
c) Perform A and F0, swap A to obtain lower nibble
d) Move A to R1
e) Move R0 to A
f) Move 0 to R0
g) Perform A and 0F to obtain the lower nibble
h) Add 30 to A
i) Move A to a external location using DPTR
j) Move R1 to A
k) Add 30 to A
l) Increment DPTR
m) Move A to DPTR
n) Move A to DPTR

CODE:

```
MOV A,#23          Move bcd to A
MOV R0,A           Move the contents of A to R0
ANL A, #F0         Perform A and F0
SWAP A             Swap the upper and lower bits of A
MOV R1, A          Move A to R1
MOV A, R0          Move R0 to A
MOV R0, #00        Move 0 to R0
ANL A, #0F         Perform A and 0F
MOV R0, A          Move A to R0
MOV A, R0          Move R0 to A
ADD A,#30          Add 30 to A
MOV DPTR, #4150    Move a location to dptr
MOVX @DPTR, A      Move A to the location
MOV A, R1          Move R1 to A
ADD A,#30          Move 30 to A
INC DPTR           Increment dptr
MOVX @DPTR, A      Move A to dptr
HERE: SJMP HERE    Jump here
```

OUTPUT:

BCD = 23

DPTR = 4150

[4150] = 33

[4151] = 32

RESULT:

8 bit BCD to ASCII conversion is performed using 8051