

**Thadomal Shahani Engineering College**  
**Computer Engineering Department**  
**Experiment List**

Subject: MP Lab  
Semester: IV

Year: 2020-2021  
Class: C2

SR.No.	Experiments	Date
1.	TASM programming for 8-bit addition, subtraction, multiplication, division(menu driven)	02-02-21,09-02-21
2.	TASM programming for 16-bit addition, subtraction, multiplication, division. (menu driven)	23-02-21
3.	TASM program to accept a string, display a string, print a string, find the length of a string, check if given string is a palindrome.(Menu driven)	02-03-21
4.	TASM program to display the contents of the flag register.	16-03-21
5.	Assembly program to sort numbers in ascending/descending order.	23-03-21
6.	Assembly program to find minimum/maximum no. from a given array.	30-03-21
7.	Assembly program to find factorial of number using procedure.	06-04-21
8.	Mixed Language program to check if given year is leap year or not.	20-04-21
9.	Mixed Language program to find the GCD/LCM of two numbers.	20-04-21
10.	Mixed Language program to increment, decrement the size of the cursor and to disable it.	27-04-21
	<b>ASSIGNMENTS</b>	
1	i. Differentiate Minimum Mode and Maximum Mode of 8086 ii. What is ISR? Explain servicing of interrupts by 8086 Microprocessor. iii. Write a short note on Assembler Directives of 8086. iv. Design a 8086 based system with the following specifications a. 8086 working at 10MHz b. 128KB EPROM using 32KB devices c. 32KB RAM using 16KB devices Show the memory map and complete design.	16-03-2021

2	i. Explain P4 Net burst micro architecture with block diagram ii. Explain the cache organization of Pentium Processor. iii. Draw and Explain the block diagram of 8257 DMA controller iv. Explain the interfacing of 8259 with 8086 in maximum mode.	27-04-2021
---	---	------------

Name: Tushar Nankani

Roll No: 1902112

Batch: C23

## Microprocessor: Experiment 1

**Aim:** Write an assembly language program for 8-bit addition, subtraction, multiplication and division.

### Theory:

Assembly Language:-

In computer programming, assembly language (or assembler language), often abbreviated asm, is any low-level programming language in which there is a very strong correspondence between the instructions in the language and the architecture's machine code instructions. Because assembly depends on the machine code instructions, every assembly language is designed for exactly one specific computer architecture. Assembly language may also be called symbolic machine code. Assembly code is converted into executable machine code by a utility program referred to as an assembler.

ASSEMBLER DIRECTIVE:-

An assembler is a program used to convert an assembly language program into the equivalent machine code modules. The assembler decides the address of each label and substitutes the values for each of the constants and variables. It then forms the machine code for mnemonics and data in assembly language program. Assembler directives help the assembler to correctly understand assembly language programs to prepare the codes. Commonly used assembler directives are DB, DD, DW, DUP, ASSUME, BYTE, SEGMENT, MACRO, PROC, OFFSET, NEAR, FAR, EQU, STRUC, PTR, END, ENDM, ENDP etc. Some directives generate and store information in the memory, while others

do not. 1) DB :- Define byte directive stores bytes of data in memory.

2) BYTE PTR :- This directive indicates the size of data referenced by

pointer. 3) SEGMENT :- This directive is to indicate the start of the segment.

4) DUP (Duplicate) :- The DUP directive reserves memory locations given by the number preceding it, but stores no specific values in any of these locations.

5) ASSUME :- The ASSUME statement is only used with full segment definitions. This statement tells the assembler what names have been chosen for the code, data, extra and stack

segments.

6)EQU : -The equate directive equates a numeric ASCII or label to another label.

7)ORG : -The ORG (origin) statement changes the starting offset address in a segment.

8)PROC and ENDP : -The PROC and ENDP directives indicate start and end of a procedure (Sub routine). Both the PROC and ENDP directives require a label to indicate the name of the procedure. The PROC directive, must also be followed with the NEAR or FAR. A NEAR procedure is one that resides in the same code segment as the program. A FAR procedure may reside at any location in the memory system.

### Function 01h- Character input with echo

**Action:** Reads a character from the standard input device and echoes it to the standard output device.  
If no character is ready it waits until one is available.  
I/O can be re-directed, but prevents detection of EOF.

**On entry:** AH = 01h

**Returns:** AL = 8 bit data input

### Function 02h - Character output

**Action:** Outputs a character to the standard output device. I/O can be re-directed, but prevents detection of 'disc full'.

**On entry:** AH = 02h  
DL = 8 bit data (usually ASCII character)

**Returns:** Nothing

### Function 09h- Output character string

**Action:** Writes a string to the display.

**On entry:** AH = 09h  
DS:DX = segment:offset of string

**Returns:** Nothing

## Function 2lh - Random read

**Action:** Reads a selected record from an opened file.

**On entry:** AH = 21h

DS:DX = Segment:offset of previously opened FCB

**Returns:** AL = 0 if successful

AL = 1 if end of file reached

AL = 2 if segment wrap occurs

AL = 3 if partial record read at end of file

**ROL:** - Rotate operand1 left. The number of rotates is set by operand2.

Algorithm: shift all bits left, the bit that goes off is set to CF and the same bit is inserted to the right-most position.

Example: MOV AL, 1Ch ;

AL = 00011100b ROL AL, 1

;AL = 00111000b, CF=0.

RET

**ROR:** - Rotate operand1 right. The number of rotates is set by operand2.

Algorithm: shift all bits right, the bit that goes off is set to CF and the same bit is inserted to the left-most position.

Example: MOV AL, 1Ch ;

AL = 00011100b ROR AL, 1

;AL = 00001110b, CF=0.

RET

**SUB:** - Subtract.

Algorithm: operand1 = operand1 -

operand2Example: MOV AL, 5

SUB AL, 1

;AL = 4

RET

**ADD:** - Add.

Algorithm: operand1 = operand1 +

operand2  
Example: MOV AL, 5 ;

AL = 5 ADD AL, -3

;AL = 2 RET

**MOV:** - Copy operand2 to operand1.

The MOV instruction cannot:

-

z set the value of the CS and IP registers. Z

copy value of one segment register to another segment register (should copy to general register first). copy immediate value to segment register (should copy to general register first).

Algorithm: operand1 = operand2

Example: ORG 100h MOV AX, 0B800h;      set AX = B800h (VGA

memory). MOV DS, AX;      copy value of AX to DS.

MOV CL, 'A';      CL = 41h (ASCII code).

MOV CH, 01011111b;      CL = color attribute.

MOV BX, 15Eh;      BX = position on screen.

MOV [BX], CX;      w.[0B800h:015Eh] = CX.

RET;      returns to operating system.

**JC:** - Short Jump if Carry flag is set to

1. Algorithm: if CF = 1 then

jump Example: include

'emu8086.inc' ORG 100h

MOV AL, 255

ADD AL, 1

JC label1

PRINT 'no

carry.'JMP exit

label1:

PRINT 'has

carry.'exit:

RET

### Algorithm for 8 bit addition:

- 1) Start
- 2) Initialize data segment through AX register in the DS register.
- 3) Display the message as "Enter the first number"
- 4) Read first digit in AL register through keyboard (e.g. AL=31h)
- 5) Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=01h
- 6) Move contents of AL register to a BL. (BL=AL so BL=01h)
- 7) Rotate the contents of BL register by 4 positions at left side. (BL=10h)
- 8) Read a second digit in AL register through keyboard AL=35h
- 9) Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=05h
- 10) Add the contents of BL and AL store the result in BL (BL=BL+AL so BL=15h)
- 11) Display the message as "Enter the second number"
- 12) Read first digit in AL register through keyboard AL=32h
- 13) Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=02h
- 14) Move contents of AL register to a CL. (CL=AL so CL=02h)
- 15) Rotate the contents of CL register by 4 positions at left side. (CL=20h)
- 16) Read a second digit in AL register through keyboard (AL=33h)
- 17) Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=03h
- 18) Add the contents of CL and AL store the result in CL (CL=CL+AL so CL=23h)



(Now both numbers are accepted as 15h and 23h)

19) Add the contents of BL and CL and result gets stored in BL (E.g

ADD BL,CL so BL=38h) 20) Preserve the result of addition in some

temporary variable say temp from BL.

21) Mask the first nibble by AND operation with number F0h (AND BL,F0h so BL=30h)

22) Call Output procedure with BL register to make a digit back in ASCII

hexadecimal range (BL=33h) 23) Move the contents of BL to DL and display it on

the screen

24) Move result from temporary variable to BL again (So BL=38h)

25) Mask the second nibble by AND operation with number 0Fh (AND BL,0Fh so  
BL=08h)

26) Call Output procedure with BL register to make a digit back in ASCII

hexadecimal range (BL=38h) 27) Move the contents of BL to DL and display it on

the screen

28) Stop

Algorithm for Input procedure: (To accept input from 0 to F)

- 1) Compare the contents of AL with 41h.
- 2) Jump to step no 4 if carry flag is set (digit is in the range of 0 to 9 so add only 30h)
- 3) Sub 07h to AL register (If digit is in the range from A to F then add 30h and 7h both)
- 4) Sub 30h to AL register
- 5) Return

Algorithm for Output procedure:

Compare the contents of BL with 0Ah

Jump to step no 4 if carry flag is set (digit is in the range of 0 to 9 so add only 30h)

- 3) Add 07h to BL register (If digit is in the range from A to F then add 30h and 7h both)
- 4) Add 30h to BL register
- 5) Return

Algorithm for 8 bit subtraction:

Start

Initialize data segment through AX register in the DS register.

Display the message as "Enter the first number"

Read first digit in AL register through keyboard (e.g. AL=31h)

Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=01h

Move contents of AL register to a BL. (BL=AL so BL=01h)

Rotate the contents of BL register by 4 positions at left side. (BL=10h) 8. Read a second digit in AL register through keyboard AL=35h

Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=05h

Add the contents of BL and AL store the result in BL (BL=BL+AL so BL=15h)

Display the message as "Enter the second number"

Read first digit in AL register through keyboard AL=32h

Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=02h

Move contents of AL register to a CL. (CL=AL so CL=02h)

Rotate the contents of CL register by 4 positions at left side. (CL=20h) 16. Read a second digit in AL register through keyboard (AL=33h)

Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=03h

Add the contents of CL and AL store the result in CL (CL=CL+AL so CL=23h) (Now both numbers are accepted as 15h and 23h)

Subtract the contents of CL from BL and result gets stored in BL (E.g SUBBL,CL so BL=F2h) 20. Preserve the result in some temporary variable say temp from BL.

Mask the first nibble by AND operation with number F0h (AND BL,F0h so BL=30h)

Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BL=33h)

23. Move the contents of BL to DL and display it on the screen

Move result from temporary variable to BL again (So BL=38h)

Mask the second nibble by AND operation with number 0Fh (AND BL,0Fh so BL=08h)

Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BL=38h)

27. Move the contents of BL to DL and display it on the screen

28. Stop

Algorithm for Input procedure: (To accept input from 0 to f) 1. Compare the contents of AL with 41h

2. Jump to step no 4 if carry flag is set 3. Sub 07h to AL register

4. Sub 30h to AL register 5. Return

Algorithm for Output procedure: 1. Compare the contents of BL with 0Ah

2. Jump to step no 4 if carry flag is set

3. Add 07h to BL register

4. Add 30h to BL register

5) Return

## 8 BIT Addition:-

CODE:-

Data segment

```
msg db 0dh,0ah,"Enter first  
number: $" msg1 db  
0dh,0ah,"Enter second number: $"  
result db 0dh,0ah,"The Result  
is: $"
```

Data ends

Code

segment

```
assume
```

CS:Code,DS:Data

start:

```
mov  
ax,Data  
mov DS,ax  
mov dx,offset  
msg mov ah,09h  
int 21h  
mov  
ah,01h  
int 21h  
sub  
al,30h  
mov  
bl,al
```

```
rol bl,4
mov
ah,01h
int 21h
sub
al,30h
add
bl,al
mov dx,offset
msg1 mov ah,09h
int 21h
mov
ah,01h
int 21h
sub
al,30h
mov
cl,al
rol
cl,4
mov
ah,01h
int 21h
sub
al,30h
add
cl,al
add
bl,cl
```

```
mov dx,offset
result mov
ah,09h
int 21h
mov
cl,bl
and
bl,0f0h
ror bl,4
call AsciiConv
mov
dl,bl
mov
ah,02h
int 21h
mov
bl,cl
and
bl,0fh
call AsciiConv

mov
dl,bl
mov
ah,02h
int 21h
mov
ah,4ch
int 21h
AsciiConv
```

```
proc cmp
    bl,0ah jc
    skip
    add bl,07h
    skip: add
    bl,30h ret
endp
Code
ends end
start
```

OUTPUT:-

For Linking and debugging same as 32 bit : tlink,td

Complink,DPMIload and TasmX also available using 32bit commands

---

C:\TASM>tasm add.asm

Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International

Assembling file: add.asm

Error messages: None

Warning messages: None

Passes: 1

Remaining memory: 476k

C:\TASM>tlink add

Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

Warning: no stack

C:\TASM>add

Enter first number: 23

Enter second number: 34

The Result is: 57

C:\TASM>



## 8 BIT SUBTRACTION: -

CODE:-

Data segment

```
msg db 0dh,0ah,"Enter first number: $"
```

```
msg1 db 0dh,0ah,"Enter second number: $"
```

```
result db 0dh,0ah,"The Result is: $"
```

Data ends

Code segment

```
assume CS:Code,DS:Data
```

start:

```
mov ax,Data
```

```
mov DS,ax
```

```
mov dx,offset msg
```

```
mov ah,09h
```

```
int 21h
```

```
mov ah,01h
```

```
int 21h
```

```
sub al,30h
```

```
mov bl,al
```

```
rol bl,4
```

```
mov ah,01h
```

```
int 21h
```

```
sub al,30h
```

```
add bl,al
```

```
mov dx,offset msg1
```

```
mov ah,09h
```

```
int 21h
```

```
mov ah,01h
```

```
int 21h
```

```
sub al,30h
mov cl,al
rol cl,4
mov ah,01h
int 21h
sub al,30h
add cl,al
sub bl,cl
mov dx,offset result
mov ah,09h
int 21h
mov cl,bl
and bl,0f0h
ror bl,4
call AsciiConv
mov dl,bl
mov ah,02h
int 21h
mov bl,cl
and bl,0fh
call AsciiConv
mov dl,bl
mov ah,02h
int 21h
mov ah,4ch
int 21h
AsciiConv proc
cmp bl,0ah
jc skip
add bl,07h
```

skip: add bl,30h

ret

endp

Code ends

end start

OUTPUT: -

For Linking and debugging same as 32 bit : tlink,td

Complink,DPMIload and TasmX also available using 32bit commands

---

C:\TASM>tasm sub.asm

Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International

Assembling file: sub.asm

Error messages: None

Warning messages: None

Passes: 1

Remaining memory: 476k

C:\TASM>tlink sub

Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

Warning: no stack

C:\TASM>sub

Enter first number: 34

Enter second number: 23

The Result is: 11

C:\TASM>

## 8 BIT MULTIPLICATION: -

CODE:-

data segment

msg1 db 0dh,0ah,"PROGRAM ON 8 BIT Multiplication\$";

msg2 db 0dh,0ah,"Enter First Number: \$";

msg3 db 0dh,0ah,"Enter Second number: \$";

msg4 db 0dh,0ah,"Product :\$";

data ends

code segment

assume cs:code,ds:data

start:

mov ax,data

mov ds,ax

mov dx,offset msg1

mov ah,09h

int 21h

mov dx,offset msg2

mov ah,09h

int 21h

call input

rol al,04

mov bl,al

call input

add bl,al

```
mov dx,offset msg3
mov ah,09h
int 21h
```

```
call input
rol al,04
mov cl,al
call input
add cl,al
```

```
mov ah,00H
mov al,bl
mul cl
```

```
MOV BX,AX
MOV CX,BX
```

```
mov dx,offset msg4
mov ah,09h
int 21h
```

```
mov ax,bx
and ax,0F000h
ror ax,12
mov dl,al
call output
```

```
mov ax,bx
and ax,0F00h
ror ax,08
```

```
mov dl,al
call output
```

```
mov ax,bx
and ax,00F0h
ror ax,04
mov dl,al
call output
```

```
mov ax,bx
and ax,000Fh
mov dl,al
call output
```

```
mov ah,4ch
int 21h
```

```
input proc
mov ah,01h
int 21h
cmp al,41h
jc lb
sub al,07h
lb:sub al,30h
mov ah,00h
ret
endp
```

```
output proc
cmp dl,0Ah
```

```
jc lb1
add dl,07h
lb1:add dl,30h
mov ah,02h
int 21h
ret
endp
```

code ends

end start

OUTPUT: -

```
C:\TASM>tasm mul8.asm
Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International

Assembling file:   mul8.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k

C:\TASM>tlink mul8
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
Warning: no stack

C:\TASM>mul8

PROGRAM ON 8 BIT Multiplication
Enter First Number: 5

Enter Second number: 7

Product :0D89
C:\TASM>_
```

## 8 BIT DIVISION: -

CODE:-

data segment

msg1 db 0dh,0ah,"PROGRAM ON 8 BIT Division\$";

msg2 db 0dh,0ah,"Enter First Number: \$";

msg3 db 0dh,0ah,"Enter Second number: \$";

msg4 db 0dh,0ah,"Quotient :\$";

msg5 db 0dh,0ah,"Remainder :\$";

data ends

code segment

assume cs:code,ds:data

start:

mov ax,data

mov ds,ax

mov dx,offset msg1

mov ah,09h

int 21h

mov dx,offset msg2

mov ah,09h

int 21h

call input

rol al,04

mov bl,al

call input



add bl,al

mov dx,offset msg3

mov ah,09h

int 21h

call input

rol al,04

mov cl,al

call input

add cl,al

mov ah,00H

mov al,bl

DIV cl

MOV BX,AX

MOV CX,BX

mov dx,offset msg4

mov ah,09h

int 21h

mov ax,bx

and al,0F0h

ror al,04

mov dl,al

call output

mov ax,bx

```
and al,0Fh
mov dl,al
call output
```

```
mov dx,offset msg5
mov ah,09h
int 21h
```

```
mov ax,bx
and ah,0F0h
ror ah,04
mov dl,ah
call output
```

```
mov ax,bx
and ah,0Fh
mov dl,ah
call output
```

```
mov ah,4ch
int 21h
```

```
input proc
mov ah,01h
int 21h
cmp al,41h
jc lb
sub al,07h
lb:sub al,30h
mov ah,00h
```

ret

endp

output proc

cmp dl,0Ah

jc lb1

add dl,07h

lb1:add dl,30h

mov ah,02h

int 21h

ret

endp

code ends

end start

OUTPUT: -

```
C:\TASM>tasm div8.asm
Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International
```

```
Assembling file:   div8.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  475k
```

```
C:\TASM>tlink div8
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
Warning: no stack
```

```
C:\TASM>div8
```

```
PROGRAM ON 8 BIT Division
Enter First Number: 67
Enter Second number: 5
```

```
Quotient :02
Remainder :0D
C:\TASM>_
```

Name: **Tushar Nankani**

Roll No: **1902112**

Batch: **C23**

## Microprocessor: Experiment 2

**Aim:** Write an assembly language menu driven program for 16-bit addition, subtraction, multiplication and division.

### THEORY:-

**MOV** is used to load and store data.

**ADD** is used to add two numbers where their one number is in accumulator or not.

**JNC** is a 2-bit command which is used to check whether the carry is generated from accumulator or not.

**INC** is used to increment an register by 1.

**HLT** is used to stop the program.

**AX** is an accumulator which is used to load and store the data.

**BX, CX** are general purpose registers where BX is used for storing second number and CX is used to store carry.

The **CALL** instruction is used whenever we need to make a call to some procedure or a subprogram

The **RET** instruction stands for return. This instruction is used at the end of the procedures or the subprograms. This instruction transfers the execution to the caller program.

The **ADD** and **SUB** Instructions- The ADD and SUB instructions are used for performing simple addition/subtraction of binary data in byte, word and double word size, i.e., for adding or subtracting 8-bit, 16-bit or 32-bit operands, respectively.

The ADD and SUB instructions have the following syntax:- ADD/SUB destination, source

The **MUL/IMUL** Instruction- There are two instructions for multiplying binary data. The MUL (Multiply) instruction handles unsigned data and the IMUL (Integer Multiply) handles signed data. Both instructions affect the Carry and Overflow flag. Multiplicand in both cases will be in an accumulator, depending upon the size of the multiplicand and the multiplier and the generated product is also stored in two registers depending upon the size of the operands.

The syntax for the **MUL/IMUL** instructions is as follows:- MUL/IMUL multiplier

The **DIV/IDIV** InstructionsThe division operation generates two elements - a quotient and a remainder. In case of multiplication, overflow does not occur because double-length registers are used to keep the product. However, in case of division, overflow may occur. The processor generates an interrupt if overflow occurs. The DIV (Divide) instruction is used for unsigned data and the IDIV (Integer Divide) is used for signed data. The dividend is in an accumulator. Both the instructions can work with 8-bit, 16-bit or 32-bit operands.

The syntax for the DIV/IDIV instruction – DIV/IDIV divisor

### **ALGORITHM: -**

1. Start
2. Display the message as “Enter first 16 bit number”
3. Read first digit in AL register through keyboard (e.g. AL=32h)
4. Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number.AL=02h
5. Move contents of AH with 00h. (AH $\beta$  00h so AX=0002h)
6. Rotate AX contents in left directions by 12 bits. (AX=2000h)
7. Move the contents of AX to BX(BX $\beta$ AX so BX=2000h)
8. Read a second digit in AL register through keyboard AL=35h
9. Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=05h
- 10.Move contents of AH with 00h. (AH $\beta$  00h so AX=0005h)
- 11.Rotate AX contents in left directions by 8 bits. (AX=0500h)
- 12.Add the contents of AX and BX (BX=BX+AX so BX=2500h)
- 13.Read a third digit in AL register through keyboard AL=31h
- 14.Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=01h
- 15.Move contents of AH with 00h. (AH $\beta$  00h so AX=0001h)
- 16.Rotate AX contents in left directions by 4 bits. (AX=0010h)
- 17.Add the contents of AX and BX (BX=BX+AX so BX=2510h)

18. Read a fourth digit in AL register through keyboard AL=30h
19. Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=00h
20. Move contents of AH with 00h. (AH $\beta$  00h so AX=0000h)
21. Add the contents of AX and BX (BX=BX+AX so BX=2510h)
22. Display the message as "Enter second 16 bit number"
23. Read first digit in AL register through keyboard (e.g. AL=37h)
24. Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=07h
25. Move contents of AH with 00h. (AH $\beta$  00h so AX=0007h)
26. Rotate AX contents in left directions by 12 bits. (AX=7000h)
27. Move the contents of AX to CX (CX $\beta$  AX so CX=7000h)
28. Read a second digit in AL register through keyboard AL=35h
29. Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=05h
30. Move contents of AH with 00h. (AH $\beta$  00h so AX=0005h)
31. Rotate AX contents in left directions by 8 bits. (AX=0500h)
32. Add the contents of AX and CX (CX=CX+AX so CX=7500h)
33. Read a third digit in AL register through keyboard AL=31h
34. Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=01h
35. Move contents of AH with 00h. (AH $\beta$  00h so AX=0001h)
36. Rotate AX contents in left directions by 4 bits. (AX=0010h)
37. Add the contents of AX and CX (CX=CX+AX so CX=7510h)
38. Read a fourth digit in AL register through keyboard AL=34h
39. Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=04h
40. Move contents of AH with 00h. (AH $\beta$  00h so AX=0004h)
41. Add the contents of AX and BX (CX=CX+AX so CX=7514h)

42. Compare accepted choice from AL with 02h

43. If zero flag is set then goto step no 69 otherwise goto step no 49

44. Add the contents of BX and CX (BX=BX+CX so BX=9A24h)

45. Preserve the result in temporary variable as t of 16 bit (so t=9A24h)

46. Mask the first nibble by AND operation with number F000h (AND BX, f000h so BX=9000h)

47. Rotate the BX contents right by 12 (in decimal so BX=0009h)

48. Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BX=0039h)

49. Move the contents of BL to DL and display it on the screen

50. Move result from temporary variable t to BX again (Now BX=9A24h)

51. Mask the second nibble by AND operation with number 0F00h (AND BX, 0F00h so BX=0A00h)

52. Rotate the contents of BX to right by 8 (in decimal)

53. Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BX=0041h (ASCII hex value of 'A'))

54. Move the contents of BL to DL and display it on the screen.

55. Move result from temporary variable to BX again (Now BX=9A24h)

56. Mask the third nibble by AND operation with number 00F0h (AND BX, 00F0h so BX=0020h)

57. Rotate the contents of BX to right by 4 (in decimal)

58. Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BX=00032h)

59. Move the contents of BL to DL and display it on the screen

60. Move result back from temporary variable to BX again (Now BX=9A24h)

61. Mask the fourth nibble by AND operation with number 000Fh (AND BX, 000fh so BX=0004h)

62. Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BX=0004h)

63. Move the contents of BL to DL and display it on the screen.

64. Subtract the contents of CX from BX (BX=BX-CX so BX=AFFCh)

65. Preserve the result in temporary variable as t of 16 bit (so t=AFFCh)
66. Mask the first nibble by AND operation with number F000h (AND BX,F000h so BX=A000h)
67. Rotate the BX contents right by 12(in decimal so BX=000Ah)
68. Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BX=0041h (i.e. ASCII hex value for 'A' ))
69. Move the contents of BL to DL and display it on the screen
70. Move result from temporary variable t to BX again (Now BX=AFFCh)
71. Mask the second nibble by AND operation with number 0F00h (AND BX,0F00h so BX=0F00h)
72. Rotate the contents of BX to right by 8(in decimal)
73. Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BX=0046h(i.e ASCII hex value of 'F'))
74. Move the contents of BL to DL and display it on the screen.
75. Move result from temporary variable to BX again (Now BX=AFFCh)
76. Mask the third nibble by AND operation with number 00F0h (AND BX,00F0h so BX=00F0h)
77. Rotate the contents of BX to right by 4(in decimal)
78. Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BX=0046h (i.e ASCII hex value of 'F'))
79. Move the contents of BL to DL and display it on the screen
80. Move result back from temporary variable to BX again (Now BX=AFFCh)
81. Mask the fourth nibble by AND operation with number 000fh (AND AX,000Fh so AX=000Ch)
82. Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BX=0043h(i.e ASCII hex value of 'C'))
83. Move the contents of BL to DL and display it on the screen.
84. Stop



**Algorithm for Input procedure :( To accept input from 0 to f)**

Compare the contents of AL with 41h(Small case)

2. Jump to step no 4 if carry flag is set
3. Sub 07h to AL register
4. Sub 30h to AL register
5. Return.

**Algorithm for Output procedure:**

1. Compare the contents of BL with 0Ah
2. Jump to step no 4 if carry flag is set
3. Add 07h to AL register
4. Add 30h to AL register
5. Return.

## Code: -

data Segment

```
msg db 0dh, 0ah, "Enter a 16-bit number: $"
result db 0dh, 0ah, "The Result is: $"
newl db 0dh, 0ah, " $"
menu db 0dh, 0ah, "1: add", 0dh, 0ah, "2: sub", 0dh, 0ah, "3:
mul", 0dh, 0ah, "4: div", 0dh, 0ah, "$"
data ends
```

code Segment

assume cs:code, ds:data

Start:

```
mov ax, data
```

```
mov ds, ax
```

```
mov dx, offset msg
```

```
mov ah, 09h
```

```
int 21h
```

```
call AcceptNum
```

```
mov bh, bl
```

```
call AcceptNum
```

```
mov cx, bx
```

```
mov dx, offset msg
```

```
mov ah, 09h
```

```
int 21h
```

```
call AcceptNum
```

```
mov bh, bl
```

```
call AcceptNum
```

```
mov dx, offset menu
```

```
mov ah, 09h
```

```
int 21h
```

```
mov ah, 01h
```

```
int 21h
```

```
cmp al, '1'
```

```
jz Addition
```

```
cmp al, '2'
```

```
jz Subtraction
```

```
cmp al, '3'
```

```
jz Multiplication
```

```
cmp al, '4'
```

```
jz Division
```

Addition:

```
add cx, bx
```

```
jmp EndSwitch
```

Subtraction:

```
sub cx, bx
```

```
    jmp EndSwitch
```

Multiplication:

```
    mov ax, cx
    mul bl
    mov cx, ax
    jmp EndSwitch
```

Division:

```
    mov ah, 0
    mov al, cl
    div bl
    mov ch, 0
    mov cl, al
    jmp EndSwitch
```

EndSwitch:

```
    mov dx, offset result
    mov ah, 09h
    int 21h
```

```
    mov bl, ch
    call DispNum
```

```
    mov bl, cl
    call DispNum
```

```
mov dx, offset new1
```

```
mov ah, 09h
```

```
int 21h
```

```
mov ah, 4ch
```

```
int 21h
```

```
AcceptNum proc
```

```
mov ah, 01h
```

```
int 21h
```

```
call HexAccept
```

```
mov bl, al
```

```
rol bl, 4
```

```
mov ah, 01h
```

```
int 21h
```

```
call HexAccept
```

```
add bl, al
```

```
ret
```

```
endp
```

```
DispNum proc
```

```
mov al, bl
```

```

    and al, 0f0h
    ror al, 4

    mov dl, al
    call HexDisp
    mov ah, 02h
    int 21h

    mov al, bl
    and al, 0fh

    mov dl, al
    call HexDisp
    mov ah, 02h
    int 21h
endp

```

HexAccept proc ; Compare to 41 if it is less than A then we need to sub only 30

; If it is greater than or equal to 41 then we also need to sub 07

```

    cmp al, 41h
    jc norm
    sub al, 07h
    norm: sub al, 30h
    ret
endp

```

```
HexDisp proc      ; Compare to 0a if it is less than A then we  
need to add only 30
```

```
      ; If it is greater than or equal to 0a then we also need to  
add 07
```

```
      cmp dl, 0ah
```

```
      jc nothex
```

```
      add dl, 07h
```

```
nothex: add dl, 30h
```

```
      ret
```

```
endp
```

```
end Start
```

```
Code ends
```

#### OUTPUT: -

```
C:\TASM>tlink exp2
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
Warning: no stack

C:\TASM>exp2

Enter a 16-bit number: 3456
Enter a 16-bit number: 5678
1: add
2: sub
3: mul
4: div
1
The Result is: 8ACE

C:\TASM>_
```

```
C:\TASM>exp2  
Enter a 16-bit number: 3456  
Enter a 16-bit number: 6778  
1: add  
2: sub  
3: mul  
4: div  
2  
The Result is: CCDE  
C:\TASM>_
```

```
C:\TASM>exp2  
Enter a 16-bit number: 3478  
Enter a 16-bit number: 6790  
1: add  
2: sub  
3: mul  
4: div  
3  
The Result is: 4380  
C:\TASM>
```

```
C:\TASM>exp2  
Enter a 16-bit number: 6768  
Enter a 16-bit number: 9324  
1: add  
2: sub  
3: mul  
4: div  
4  
The Result is: 0002  
C:\TASM>_
```



Name: **Tushar Nankani**

Roll No: **1902112**

Batch: **C23**

## **Microprocessor: Experiment 3**

**Aim:** Write a menu driven assembly language program to accept ,display the string, find the length of string, reverse the string, check the string is palindrome or not and scan the string to check whether a accepted letter is present or not

### **Theory:**

A Macro is a set of instructions grouped under a single unit. It is another method for implementing modular programming in the 8086 microprocessors

A Macro can be defined in a program using the following assembler directives: MACRO (used after the name of Macro before starting the body of the Macro) and ENDM (at the end of the Macro). All the instructions that belong to the Macro lie within these two assembler directives

LEA (Load Effective Address) is a shift-and-add instruction. It was added to 8086 because hardware is there to decode and calculate addressing modes

string storage in memory,how is string declared,procedure near and far concept.

### **String Instructions**

Each string instruction may require a source operand, a destination operand or both. For 32-bit segments, string instructions use ESI and EDI registers to point to the source and destination operands, respectively.

For 16-bit segments, however, the SI and the DI registers are used to point to the source and destination, respectively.

There are five basic instructions for processing strings. They are –

- MOVS – This instruction moves 1 Byte, Word or Doubleword of data from memory location to another.
- LODS – This instruction loads from memory. If the operand is of one byte, it is loaded into the AL register, if the operand is one word, it is loaded into the AX register and a doubleword is loaded into the EAX register.

- STOS – This instruction stores data from register (AL, AX, or EAX) to memory.
- CMPS – This instruction compares two data items in memory. Data could be of a byte size, word or doubleword.
- SCAS – This instruction compares the contents of a register (AL, AX or EAX) with the contents of an item in memory.

Each of the above instruction has a byte, word, and doubleword version, and string instructions can be repeated by using a repetition prefix.

These instructions use the ES:DI and DS:SI pair of registers, where DI and SI registers contain valid offset addresses that refers to bytes stored in memory. SI is normally associated with DS (data segment) and DI is always associated with ES

(extra segment).

The DS:SI (or ESI) and ES:DI (or EDI) registers point to the source and destination operands, respectively. The source operand is assumed to be at DS:SI (or ESI) and the destination operand at ES:DI (or EDI) in memory.

For 16-bit addresses, the SI and DI registers are used, and for 32-bit addresses, the ESI and EDI registers are used.

procedure : A procedure is group of instructions that usually performs one task. It is a reusable section of a software program which is stored in memory once but can be used as often as necessary. A procedure can be of two types. 1) Near Procedure 2) Far Procedure

Near Procedure: A procedure is known as NEAR procedure if it is written (defined) in the same code segment which is calling that procedure. Only Instruction Pointer (IP register) contents will be changed in NEAR procedure.

FAR procedure : A procedure is known as FAR procedure if it is written (defined) in the different code segment than the calling segment. In this case both Instruction Pointer (IP) and the Code Segment (CS) register content will be changed.

Directives used for procedure:

PROC directive: The PROC directive is used to identify the start of a procedure. The PROC directive follows a name given to the procedure. After that the term FAR and NEAR is used to specify the type of the procedure.

ENDP Directive: This directive is used along with the name of the procedure to indicate the end of a procedure to the assembler. The PROC and ENDP directive are used to bracket a procedure.

CALL instruction and RET instruction:

The CALL instruction is used whenever we need to make a call to some procedure or a subprogram. Whenever a CALL is made, the following process takes place inside the microprocessor:

- The address of the next instruction that exists in the caller program (after the program CALL instruction) is stored in the stack.
- The instruction queue is emptied for accommodating the instructions of the procedure.
- Then, the contents of the instruction pointer (IP) is changed with the address of the first instruction of the procedure.
- The subsequent instructions of the procedure are stored in the instruction queue for execution.

The Syntax for the CALL instruction is as follows:

CALL subprogram\_name

CALL instruction : The CALL instruction is used to transfer execution to a procedure. It performs two operations. When it executes, first it stores the address of instruction after the CALL instruction on the stack. Second it changes the content of IP register in case of Near call and changes the content of IP register and CS register in case of FAR call.

There are two types of calls.

- 1) Near Call or Intra segment call.
- 2) Far call or Inter Segment call

Operation for Near Call : When 8086 executes a near CALL instruction, it decrements the stack pointer by 2 and copies the IP register contents on to the stack. Then it copies the address of the first instruction of the called procedure.

$SP \leftarrow SP - 2$

IP  $\leftarrow$  stores onto stack

IP  $\leftarrow$  starting address of a procedure.

Operation of FAR CALL: When 8086 executes a far call, it decrements the stack pointer by 2 and copies the contents of CS register to the stack. It then decrements the stack pointer by 2 again and copies the content of IP register to the stack. Finally it loads CS register with base address of segment having procedure and IP with address of first instruction in procedure.

$SP \leftarrow SP - 2$  CS contents stored on stack

$SP \leftarrow SP - 2$

IP  $\leftarrow$  contents stored on stack

CS  $\leftarrow$  Base address of segment having procedure IP  $\leftarrow$  address of first instruction in procedure

The RET instruction stands for return. This instruction is used at the end of the procedures or the subprograms. This instruction transfers the execution to the caller program. Whenever the RET instruction is called, the following process takes place inside the microprocessor:

- The address of the next instruction in the mainline program which was previously stored inside the stack is now again fetched and is placed inside the instruction pointer (IP).
- The instruction queue will now again be filled with the subsequent instructions of the mainline program.

The Syntax for the RET instruction is as follows:

RET

### Algorithm:

1. Start
2. Initialize data segment through AX register in the DS register.
3. If choice =1 then goto step no.4 (accept the string)else goto the step no.14
4. Display the message "Enter the string"
5. Initialize the SI with 1000h(source index)
6. Initialize the DI with 2000h(destination index)
7. Accept the character through keyboard(AL=52h i.e ASCII hex value of 'm')
8. Move the AL contents to a location pointed by SI and DI.
9. Increment the CL register contents by 1
10. Increment SI and DI by 1
11. Repeat the step 7 to 10 till Enter key get pressed(i.e AL=0Dh ASCII hex value for enter key used to detect the end of the string.)
12. Decrement the CL by 1 ( to avoid the enter key as a part of the string to obtain correct length value)
13. Preserve the length in temporary variable say count from CL
14. If choice = 2 then goto step no.15 (display the string)else goto step no. 21
15. Initialize SI again with 1000h (string source)
16. Move the content of location pointed by SI to DI
17. Display the character on the screen
18. Increment SI by 1 and decrement the CL by 1
19. Repeat the step from 15 to 18 till Zero flag will come set (i.e CL reaches to zero).
20. Load the String length from count to CL register back. 21. If choice=3 (display the length of string) then goto step no. 22 else goto step no. 24
22. Move the contents of CL to AL and add 30h to AL

23. Move the AL contents to DL and display the length of string.
24. If choice=4 (Reverse the string) then goto step no. 24 else goto step no.30
25. Add CX (e.g CX=0005)to SI to make SI to point to the last letter of String.
26. Decrement SI by 1 as SI will start from 0 index (e.x1000 to 10004 are the locations if string is of 5 letters)
27. Move the letter pointed by SI to DL and display it on the screen
28. Decrement the SI by 1 and decrement the CL by 1
29. Repeat step 25 to 29 till CL reaches to zero if zero flag get set.
30. If choice = 5 then goto step no. 31 else goto the step no.43
31. Initialize SI with 1000h again
32. Initialize DI with 2000h again
33. Load CL with original length of string from count
34. Add DI with CX (e.x CX=0005)so DI will point to the last letter of string
35. Move the letter from location pointed by SI to AL
36. Move the letter from location pointed by DI to BL
37. Compare the AL and BL if zero flag is not set then goto step no. 38 else goto step no 39
38. Display the string is not palindrome.
39. Increment the SI by 1 and Decrement DI by 1
40. Decrement CL by 1.
41. Repeat step no. 34 to 38 till CL reaches to zero
42. If zero flag is set then display the string is palindrome.
43. If choice=6 then goto step no. 44 else goto step no 49
44. Accept the letter to be searched in AL.
45. Load CL with original length of string from count
46. Initialize the DI with 2000h

47. Use REPNE SCASB instruction this instruction scans the string for a letter stops when it finds the first occurrence of a letter. It decrements CX also by 1 for every scan. When this instruction stops CX will contain value as position-1 value.
48. Obtain the position of a letter by subtracting total length-CX value and display it on the screen.
49. Stop.

## Programs and output

```
;macro for printing a string
print macro m
mov ah,09h
mov dx,offset m
int 21h
endm

.model small
.data
menu db 10,13, "Menu: "
      db 10,13, "1.Enter the string to find length "
      db 10,13, "2.Check for palindrome "
      db 10,13, "3.Display Entered String"
      db 10,13, "4.Exit  "
      db 10,13, "  "
      db 10,13, "Your choice: $"
msg1 db 10,13, "Your choice is: $"
mc1 db 10,13, "case 1 $"
mc2 db 10,13, "case 2 $"
mc3 db 10,13, "case 3 $"
mc4 db 10,13, "Exiting the program $"
mc6 db 10,13, "Invalid choice....try again $"
mc7 db 10,13, "Exiting the program $"
empty db 10,13, "  $"
str1 db 25,?,25 dup('$')
str2 db 25,?,25 dup('$')
len db ?
str3 db 0ah,0dh,"enter name: $"
;str4 db 25,?,25 dup('$')
mstring db 10,13, "Enter the string: $"
```



```

notpalin db 10,13, "String is not a palindrome. $"
palin db 10,13, "String is a palindrome. $"
mstring2 db 10,13, "Enter second string: $"
mlength db 10,13, "Length is: $"
scount db ?
;.code
start:
mov ax,@data
mov ds,ax
again:
print menu
call accept ;accept user choice
mov bl,al
case1:
    cmp bl,"1" ;compare user choice with '1'
    jne case2 ;if not equal,check for case 2
    print mc1
    print empty
    print mstring
    call accept_string ;function call to accept a string
    mov cl,str1+1 ;storing length in cl
    mov bl,cl
    print mlength
    call display1 ;printing the length
    print empty
    jmp again ;printing the menu again
case2: cmp bl,"2" ;checking for case 2
    jne case3 ;if not equal jump to case 3
    print mc2
    print empty

```

```

    print mstring
    call accept_string
    mov si,offset str1
    mov cl,str1+1          ;store the length
    mov ch,00h
    mov len,cl
    inc si
    add si,cx              ;si points to last
    mov di,offset str1    ;di to start of string
    add di,0002h          ;di to actual start of string;
                          ;in string 0th byte->size
                          ;1st byte->length of string
                          ;from 2nd byte->actual string
    mov cl,len ;load counter
cmpagain: mov al,[si]
    mov ah,[di]
    inc di
    dec si
    cmp al,ah
    jne nopalin
    dec cl
    jnz cmpagain
    print palin
    print empty
    jmp again
nopalin: print notpalin
    print empty
    jmp again

```

```

case3:
cmp bl,"3"      ;check for case 4
    jne case4
print mc3
lea dx,str3
mov ah,09h
int 21h
call accept_string
print empty
lea dx,str1+2
mov ah,09h
int 21h

case4: cmp bl,"4"      ;check for case 6
    jne case5      ;if not equal,default to case 7 and print the
error message
    print mc6
    jmp exit

case5: print mc7      ;print error message
    jmp again      ;display the menu again

exit:
mov ah,4ch      ;exit the program
int 21h

;accept procedure
accept proc near
mov ah,01

```

```
int 21h
ret
accept endp
```

```
display1 proc near
```

```
    mov al,bl
    mov bl,al
    and al,0f0h
    mov cl,04
    rol al,cl
    cmp al,09
    jbe number
    add al,07
```

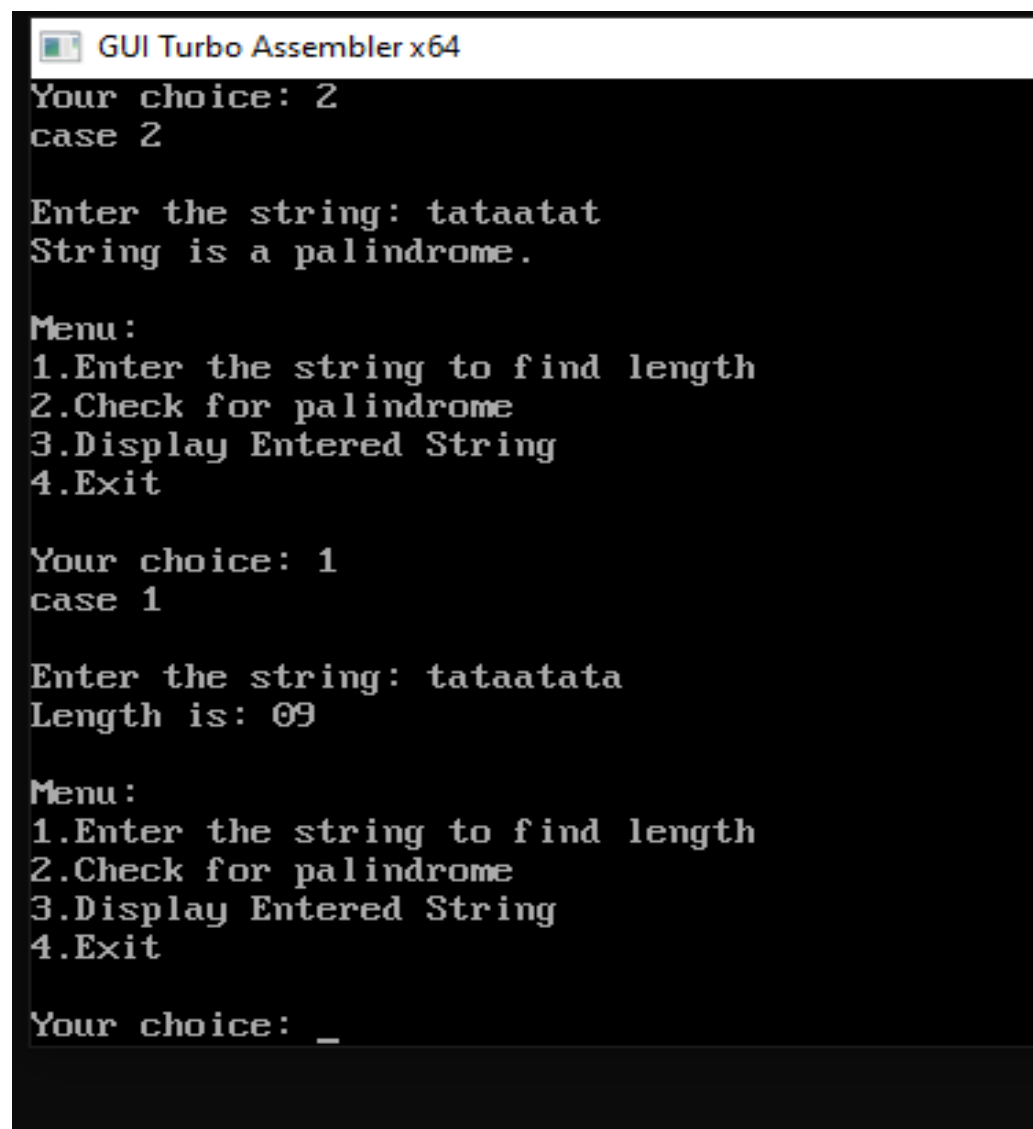
```
number:  add al,30h
        mov dl,al
        mov ah,02
        int 21h
```

```
        mov al,bl
        and al,00fh
        cmp al,09
        jbe number2
        add al,07
```

```
number2: add al,30h
        mov dl,al
        mov ah,02
        int 21h
```

```
ret
display1 endp
```

```
accept_string proc near
mov ah,0ah          ;accept string from user function
mov dx,offset str1  ; store the string in memory pointed by "DX"
int 21h
ret
accept_string endp
end start
end
```



The screenshot shows the GUI Turbo Assembler x64 window with a black background and white text. The program is running, and the user has interacted with it twice. In the first run, the user chose option 2 (Check for palindrome) and entered the string "tataatat", which the program correctly identified as a palindrome. In the second run, the user chose option 1 (Enter the string to find length) and entered the string "tataatata", which the program identified as having a length of 09. The menu is displayed after each run, and the user is prompted for their next choice.

```
GUI Turbo Assembler x64
Your choice: 2
case 2

Enter the string: tataatat
String is a palindrome.

Menu:
1.Enter the string to find length
2.Check for palindrome
3.Display Entered String
4.Exit

Your choice: 1
case 1

Enter the string: tataatata
Length is: 09

Menu:
1.Enter the string to find length
2.Check for palindrome
3.Display Entered String
4.Exit

Your choice: _
```

Name: **Tushar Nankani**

Roll No: **1902112**

Batch: **C23**

## **Microprocessor: Experiment 4**

**Aim:** Write an assembly language program to display the contents of 16 bit flag register.

### **Theory:**

The Flag register is a Special Purpose Register. Depending upon the value of result after any arithmetic and logical operation the flag bits become set (1) or reset (0). In 8085 microprocessor, flag register consists of 8 bits and only 5 of them are useful.

#### **1 Sign (S) flag: –**

This flag is set, when MSB (Most Significant Bit) of the result is 1.

Since negative binary numbers are represented in the 8085 CPU in standard two's complement notation, SF indicates sign of the result.

1-MSB is 1 (negative)

0-MSB is 0 (positive)

If the MSB of the result of an operation is 1, this flag is set, otherwise it is reset.

#### **2 Zero (Z) flag: –**

This flag is set, when the result of operation is zero, else it is reset. 1-zero result

0-non-zero result

#### **3 Auxiliary Carry (AC) flag: –**

This flag is set whenever there has been a carry out of the lower nibble into the higher nibble or a borrow from higher nibble into the lower nibble of an 8 bit quantity, else AF is reset. This flag is used by decimal arithmetic instructions.

1-carry out from bit 3 on addition or borrow into bit 3 on subtraction

0-otherwise

If there is a carry out of bit 3 and into bit 4 resulting from the execution of an arithmetic operation, it is set otherwise reset. This flag is used for BCD operation and is not available to the programmer to change the sequence of an instruction.

**4 Carry (CY) flag:** – If an instruction results in a carry (for addition operation) or borrow (for subtraction or comparison) out of bit D7, then this flag is set, otherwise reset.

This flag is set whenever there has been a carry out of, or a borrow into, the higher order bit of the result. The flag is used by the instructions that add and subtract multibyte numbers.

1-carry out from MSB bit on addition or borrow into MSB bit on subtraction

0-no carry out or borrow into MSB bit

**5 Parity (P) flag:–**

This flag is set whenever the result has even parity, an even number of 1 bits. If parity is odd, PF is cleared. 1-low byte has even number of 1 bits

0-low byte has odd parity

This flag is set when the result of an operation contains an even number of 1's and is reset otherwise.

**Control Flags** – The control flags enable or disable certain operations of the microprocessor. There are 3 control flags in 8086 microprocessor and these are:

**Directional Flag (D)** – This flag is specifically used in string instructions.

If directional flag is set (1), then access the string data from higher memory location towards lower memory location.

If directional flag is reset (0), then access the string data from lower memory location towards higher memory location.

**Interrupt Flag (I)** – This flag is for interrupts.

If interrupt flag is set (1), the microprocessor will recognize interrupt requests from the peripherals.

If interrupt flag is reset (0), the microprocessor will not recognize any interrupt requests and will ignore them.

**Trap Flag (T)** – This flag is used for on-chip debugging. Setting trap flag puts the microprocessor into single step mode for debugging. In single stepping, the microprocessor executes a instruction and enters into single step ISR.

If trap flag is set (1), the CPU automatically generates an internal interrupt after each instruction, allowing a program to be inspected as it executes instruction by instruction. If trap flag is reset (0), no function is performed.

The control and status signals are ALE, RD, WR, IO/M, S0, and S1 and READY, The interrupt signals are TRAP, RST 7.5, RST 6.5, RST 5.5, INTR. INTA is an interrupt acknowledgement signal indicating that the processor has acknowledged an INTR interrupt. Serial I/O signals are SID and SOD, DMA signals are HOLD and HLDA, and Reset signals are RESET IN and RESET OUT.



The PUSHF (push flags) and PUSHFD (push flags double) mnemonics reference the same opcode. The PUSHF instruction is intended for use when the operand-size attribute is 16 and the PUSHFD instruction for when the operand-size attribute is 32. Some assemblers may force the operand size to 16 when PUSHF is used and to 32 when PUSHFD is used. Others may treat these mnemonics as synonyms (PUSHF/PUSHFD) and use the current setting of the operand-size attribute to determine the size of values to be pushed from the stack, regardless of the mnemonic used

When in virtual-8086 mode and the I/O privilege level (IOPL) is less than 3, the PUSHF/PUSHFD instruction causes a general protection exception (#GP).

In the real-address mode, if the ESP or SP register is 1, 3, or 5 when the PUSHA/PUSHAD instruction is executed, the processor shuts down due to a lack of stack space. No exception is generated to indicate this condition

## Algorithm:

- 1.Start
- 2.Initialize data segment through AX register in the DS register.
- 3.Display the flag bit names as "X X X X O D I T SF ZF x AF X PF X CF "
- 4.Push the contents of flag register to a stack
- 5.Pop the contents of stack to register to any 16 bit register (say BX =0000 0100 1000 1001)
- 6.Move the contents of BX to temporary variable say t
- 7.Move the 8000h number to AX.(AX8000h)
- 8.Move the count as 16(in decimal) to CX register (as 16 bit flag register)
- 9.Move the contents of temporary variable t to BX.
- 10.And the contents of BX and AX.
- 11.If zero flag is set then goto the step no 14 otherwise goto step no. 12
- 12.Move the 31h to DL register.
- 13.Make the unconditional jump to a step no. 15
- 14.Move the 30h to DL register.
- 15.Preserve the (8000h )number from AX in t1 temporary variable.(As while displaying 30h or 31 h AH register get modified as 02h function is moved of INT 21h).
- 16.Display the contents of DL register.
- 17.Move the contents of t1 to AX register back (As while displaying 30h or 31 h AH register get modified as 02h function is moved of INT 21h).
- 18.Rotate the contents of AX by 1 positions in right direction.
- 19.Repeat step no 5 to 17 till count CX reaches to 0.
- 20.Stop.

CODE:

data segment

msg1 db 0dh,0ah, "DISPLAY FLAG CONTENTS\$";

msg2 db 0dh,0ah, "1.BEFORE SETTING DF AND RESETTNG IF :  
\$";

msg3 db 0dh,0ah, "2.AFTER SETTING DF AND RESETTNG IF :  
\$";

msg4 db 0dh,0ah, "XX XX XX XX OF DF IF TF SF ZF XX AF XX  
PF XX CF\$";

msg5 db 0dh,0ah, "\$";

flag dw ?

rotate dw ?

data ends

code segment

assume cs:code,ds:data

start:

mov ax,data

mov ds,ax

mov dx,offset msg1

mov ah,09h

int 21h

mov dx,offset msg2

mov ah,09h

int 21h

call disp

std

cli

mov dx,offset msg3

mov ah,09h

int 21h

call disp

mov ah,4ch

int 21h

disp proc

mov dx,offset msg4

mov ah,09h

int 21h

mov dx,offset msg5

mov ah,09h

int 21h

pushf

pop bx

mov flag,bx

mov dx,8000h

mov rotate,dx

mov cx,10h

next:

and bx,rotate

jz zero

mov dl,31h

mov ah,02h

int 21h

jmp space

zero:

mov dl,30h

mov ah,02h

int 21h

space:

mov dl,""

mov ah,02h

int 21h

mov dl,""

mov ah,02h

int 21h

mov bx,flag

ror rotate,01h

loop next

endp

code ends

end start

OUTPUT:

### DISPLAY FLAG CONTENTS

1.BEFORE SETTING DF AND RESETTNG IF :

XX	XX	XX	XX	OF	DF	IF	TF	SF	ZF	XX	AF	XX	PF	XX	CF
0	1	1	1	0	0	1	0	0	0	0	0	0	0	1	0

2.AFTER SETTING DF AND RESETTNG IF :

XX	XX	XX	XX	OF	DF	IF	TF	SF	ZF	XX	AF	XX	PF	XX	CF
0	1	1	1	0	1	0	0	0	0	0	0	0	1	1	0

Name: Tushar Nankani

Roll No: 1902112

Batch: C23

## Microprocessor: Experiment 5

Aim: Assembly program to sort numbers in ascending/descending order.

### Theory:

A Sorting Algorithm is used to rearrange a given elements according to a comparison operator on the elements.

Sorted elements can be easy to interact with and it is very easy to perform various operations on them like adding more elements, adding more elements at a specific position, deleting elements at a specific position, deleting from start or end, etc.

### ALGORITHM: -

- 1.Start
- 2.Initialize data segment through AX register in the DS register.
- 3.Display Array Limit
- 4.Initialize Si to 4000 and Cx to 10
- 5.Display message to enter Number
- 6.Take input of 1 digit into AL Register
- 7.Rotate AL Register to left by 4 bits
- 8.Copy contents of AL to BL
- 9.Take input of second digit into AL
- 10.Add AL to BL
- 11.Copy BL to Stack
- 12.Increment SI 13.Repeat from step 5 to step 12 10 Times
- 14.Initialise CX to 9
- 15.Initialise BX to 0.
- 16.Set SI to 4000.
- 17.Copy Data at location of SI to BL

- 18.Increment SI
- 19.Compare BL and contents at SI
- 20.IF Contents at SI are greater than BL Jump to 27
- 21.Move contents BL to DL
- 22.Move contents at SI to BL
- 23.Decrement SI
- 24.Move BL into Contents at SI
- 25.Increment SI
- 26.Move contents in DL to contents at SI
- 27.Move contents at SI to BL
- 28.Increment SI
- 29.Repeat from step 17 to step 28 9 Times
- 30.Set SI to 4000 and Cx to 10
- 31.Print Output Message
- 32.Copy contents at SI to BL
- 33.Print first Digit of BL
- 34.Print Second Digit of BL
- 35.Increment SI
- 36.Repeat from step 31 to step 35 10 Times
- 37.Stop



## Code:

1. Ascending order.

DATA SEGMENT

MSG1 DB 10,13,"ARRAY LIMIT IS 10 \$"

MSG2 DB 10,13,"Enter the number: \$"

MSG3 DB 10,13,"Sorted Array: \$"

MSG4 DB 10,13,"\$"

COUNT DB ?

TEMP DB ?

DATA ENDS

CODE SEGMENT

ASSUME DS:DATA,CS:CODE

START:

MOV AX,DATA

MOV DS,AX

LEA DX,MSG1

MOV AH,09H

INT 21H

MOV SI,4000H

MOV CX,10

LABEL1:

LEA DX,MSG2

MOV AH,09H

INT 21H

MOV AH,01H

INT 21H

CALL INPUT

ROL AL,04H

MOV BL,AL

MOV AH,01H

INT 21H

CALL INPUT

MOV AH,00H

ADD BL,AL

MOV [SI],BL

INC SI

LEA DX,MSG4

MOV AH,09H

INT 21H

LOOP LABEL1

```
MOV CX,9  
mov bx,0000h
```

```
LABEL2:
```

```
MOV SI,4000H  
MOV BL,[SI]  
INC SI  
MOV BH,9
```

```
LABEL3:
```

```
DEC BH  
CMP BL,[SI]  
JC LABEL4  
MOV DL,BL  
MOV BL,[SI]  
DEC SI  
MOV [SI],BL  
INC SI  
MOV [SI],DL
```

```
LABEL4:
```

```
MOV BL,[SI]  
INC SI  
CMP BH,00H  
JNZ LABEL3
```

LOOP LABEL2

MOV SI,4000H

MOV CX,10

d:

LEA DX,MSG3

MOV AH,09H

INT 21H

MOV BL,00H

MOV BL,[SI]

AND BL,0F0H

ROR BL,04H

CALL OUTPUT

MOV BL,[SI]

AND BL,0FH

CALL OUTPUT

INC SI

LOOP d

MOV AH,4CH

INT 21H

INPUT PROC

CMP AL,41H

JC LABEL6

SUB AL,07H

LABEL6:

SUB AL,30H

RET

ENDP

OUTPUT PROC

CMP BL,0AH

JC LABEL7

ADD BL,07H

LABEL7:

ADD BL,30H

MOV DL,BL

MOV AH,02H

INT 21H

RET

ENDP

CODE ENDS

END START

## OUTPUT:

```
Sorted Array: 85
C:\TASM>

C:\TASM>ascend

ARRAY LIMIT IS 10
Enter the number: 33

Enter the number: 95

Enter the number: 47

Enter the number: 12

Enter the number: 25

Enter the number: 73

Enter the number: 69

Enter the number: 55

Enter the number: 83

Enter the number:
Enter the number: 12

Enter the number: 25

Enter the number: 73

Enter the number: 69

Enter the number: 55

Enter the number: 83

Enter the number: 52

Sorted Array: 12
Sorted Array: 25
Sorted Array: 33
Sorted Array: 47
Sorted Array: 52
Sorted Array: 55
Sorted Array: 69
Sorted Array: 73
Sorted Array: 83
Sorted Array: 95
C:\TASM>_
```

2. Descending order.

DATA SEGMENT

MSG1 DB 10,13,"ARRAY LIMIT IS 10 \$"

MSG2 DB 10,13,"Enter the number: \$"

MSG3 DB 10,13,"Sorted Array: \$"

MSG4 DB 10,13,"\$"

COUNT DB ?

TEMP DB ?

DATA ENDS

CODE SEGMENT

ASSUME DS:DATA,CS:CODE

START:

MOV AX,DATA

MOV DS,AX

LEA DX,MSG1

MOV AH,09H

INT 21H

MOV SI,4000H

MOV CX,10

LABEL1:

LEA DX,MSG2

MOV AH,09H

INT 21H

MOV AH,01H

INT 21H

CALL INPUT

ROL AL,04H

MOV BL,AL

MOV AH,01H

INT 21H

CALL INPUT

MOV AH,00H

ADD BL,AL

MOV [SI],BL

INC SI

LEA DX,MSG4

MOV AH,09H

INT 21H

LOOP LABEL1

MOV CX,9



```
mov bx,0000h
```

```
LABEL2:
```

```
MOV SI,4000H
```

```
MOV BL,[SI]
```

```
INC SI
```

```
MOV BH,9
```

```
LABEL3:
```

```
DEC BH
```

```
CMP BL,[SI]
```

```
JNC LABEL4
```

```
MOV DL,BL
```

```
MOV BL,[SI]
```

```
DEC SI
```

```
MOV [SI],BL
```

```
INC SI
```

```
MOV [SI],DL
```

```
LABEL4:
```

```
MOV BL,[SI]
```

```
INC SI
```

```
CMP BH,00H
```

```
JNZ LABEL3
```

LOOP LABEL2

MOV SI,4000H

MOV CX,10

d:

LEA DX,MSG3

MOV AH,09H

INT 21H

MOV BL,00H

MOV BL,[SI]

AND BL,0F0H

ROR BL,04H

CALL OUTPUT

MOV BL,[SI]

AND BL,0FH

CALL OUTPUT

INC SI

LOOP d

MOV AH,4CH

INT 21H

INPUT PROC

CMP AL,41H

JC LABEL6

SUB AL,07H

LABEL6:

SUB AL,30H

RET

ENDP

OUTPUT PROC

CMP BL,0AH

JC LABEL7

ADD BL,07H

LABEL7:

ADD BL,30H

MOV DL,BL

MOV AH,02H

INT 21H

RET

ENDP

CODE ENDS

END START

## OUTPUT :

```
Sorted Array: BA
Sorted Array: BA
Sorted Array: BA
C:\TASM>descend

ARRAY LIMIT IS 10
Enter the number: 12

Enter the number: 43

Enter the number: 25

Enter the number: 37

Enter the number: 94

Enter the number: 69

Enter the number: 73

Enter the number: 84

Enter the number: 55

Enter the number:
Enter the number: 37

Enter the number: 94

Enter the number: 69

Enter the number: 73

Enter the number: 84

Enter the number: 55

Enter the number: 28

Sorted Array: 94
Sorted Array: 84
Sorted Array: 73
Sorted Array: 69
Sorted Array: 55
Sorted Array: 43
Sorted Array: 37
Sorted Array: 28
Sorted Array: 25
Sorted Array: 12
C:\TASM>
```

Name: Tushar Nankani

Roll No: 1902112

Batch: C23

## Microprocessor: Experiment 6

Aim: Assembly program to find minimum and maximum of a given array.

### Theory:

A naive solution is to compare each array element for minimum and maximum elements by considering a single item at a time. The time complexity of this solution would be linear.

Another solution can be to first sort the array and pick out the first element and the last element which will be the minimum element and maximum elements respectively.

### ALGORITHM:

#### **For max**

- 1.Start2.Initialize data segment through AX register in the DS register.
- 3.Initialize the SI to 2000h
- 4.Initialize total elements of array as a count in CX(e.g 0005h)
- 5.Preserve the above count in c temporary variable.
- 6.Display the message as “Enter an array elements”
- 7.Read first digit in AL register through keyboard(e.g. AL=31h)
- 8.Call Input procedure to make anumber from ASCII hexadecimal to a normal hexadecimal number.AL=01h
- 9.Move AL contents to BL
- 10.Rotate BL contents by 4 in left direction.
- 11.Read second digit in AL register through keyboard (e.g AL=32h)

12. Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=02h
13. Add BL and AL contents (BL←-BL+AL)
14. Store the BL (current accepted number) to location pointed by SI
15. Increment SI by 1 to point to next location for the next number
16. Repeat step no. 7 to 15 till CX count reaches to 0.
17. Initialize SI again to 2000h and CX also with total number of elements.
18. Initialize AL with first element pointed by SI for the next comparison

Lab Manual Microprocessor Ms. Urvi Kore 27

19. Compare number pointed by SI from an array with AL register
20. If carry is not generated (i.e. if number in AL < number pointed by SI) then goto step no. 22 else goto step no. 21
21. Make a unconditional jump to step no. 23
22. Move number pointed by SI to AL
23. Incremented SI by 1
24. Decrement CX by 1
25. Compare CX with 0000h (i.e. Repeat step no. 19 to 25 till all numbers of array are not covered for the comparison)
26. If Zero flag is not set then jump to step no. 19
27. Finally maximum number will be available in AL register.
28. Display the contents of AL register.
29. Stop.

## **For min**

- 1.Start
- 2.Initialize data segment through AX register in the DS register.
- 3.Initialize the SI to 5000h
- 4.Initialize total elements of array as a count in CX(e.g 0005h)
- 5.Preserve the above count in c temporary variable.
- 6.Display the message as “Enter an array elements”
- 7.Read first digit in AL register through keyboard(e.g. AL=31h)
- 8.Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number.AL=01h
- 9.Move AL contents to BL
- 10.Rotate BL contents by 4 in left direction.
- 11.Read second digit in AL register through keyboard (e.g AL=32h)
- 12.Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number.AL=02h
- 13.Add BL and ALcontents (BL<-BL+AL)
- 14.Store the BL (current accepted number) to location pointed by SI

15.Increment SI by 1 to point to next location for the next number

16.Repeat step no. 7 to 15 till CX count reaches to 0.

17.Initialize SI again to 5000h and CX also with total number of elements.

18.Initialize AL with first element pointed by SI for the next comparison

19.Compare number pointed by SI from an array with AL register

20.If carry is generated (i.e. if number in AL > number pointed by SI) then goto step no. 22 else goto step no. 21

21.Make a unconditional jump to step no. 23

22.Move number pointed by SI to AL

23.Incremented SI by 1

24.Decrement CX by 1

25.Compare CX with 0000h (i.e. Repeat step no.19 to 25 till all numbers of array are not covered for the comparison)

26.If Zero flag is not set then jump to step no.19

27.Finally minimum number will be available in AL register.

28.Display the contents of AL register.

29.Stop.



Code:

1. For max

Data Segment

```
msg db 0dh,0ah,"Please enter the length of the array: $"
```

```
msg1 db 0dh,0ah,"Enter a number: $"
```

```
new1 db 0dh,0ah," $"
```

```
res db 0dh,0ah,"The maximum is: $"
```

```
len db ?
```

```
max db ?
```

Data ends

Code Segment

```
assume CS:Code,DS:Data
```

Start:

```
mov ax,Data
```

```
mov DS,ax
```

```
mov dx,offset msg
```

```
mov ah,09h
```

```
int 21h
```

```
call Accept
```

```
mov len,bl
```

```
mov cl,bl
```

```
    mov ch,00h
    mov di,1000h
back: mov dx,offset msg1
    mov ah,09h
    int 21h
    call Accept
    mov [di],bl
    inc di
    loop back
    mov di,1000h
    mov cl,len
    mov ch,00h
    mov dx,offset newl
    mov ah,09h
    int 21h
    mov al,[di]
    mov max,al
chk:  mov bl,max
    mov al,[di]
    cmp bl,al
    jnc a
    mov max,al
    jmp b
a:   mov max,bl
b:   inc di
    loop chk
    mov dx,offset res
```

```
mov ah,09h
int 21h
mov bl,max
call DispNum
mov ah,4ch
int 21h
```

Accept proc

```
mov ah,01h
int 21h
call AsciiToHex
rol al,4
mov bl,al
mov ah,01h
int 21h
call AsciiToHex
add bl,al
ret
```

endp

DispNum proc

```
mov dl,bl
and dl,0f0h
ror dl,4
call HexToAscii
mov ah,02h
int 21h
mov dl,bl
and dl,0fh
```

```

    call HexToAscii
    mov ah,02h
    int 21h
endp
AsciiToHex proc
    cmp al,41h
    jc sk
    sub al,07h
sk: sub al,30h
    ret
endp
HexToAscii proc
    cmp dl,0ah
    jc sk2
    add dl,07h
sk2: add dl,30h
    ret
endp
Code ends
end Start

```

## OUTPUT :

1. For max

```
C:\TASM>max
```

```
Please enter the length of the array: 05
```

```
Enter a number: 23
```

```
Enter a number: 12
```

```
Enter a number: 59
```

```
Enter a number: 47
```

```
Enter a number: 32
```

```
The maximum is: 59
```

```
C:\TASM>_
```

2. For min

Data Segment

```
msg db 0dh,0ah,"Please enter the length of the array: $"
```

```
msg1 db 0dh,0ah,"Enter a number: $"
```

```
newl db 0dh,0ah," $"
```

```
res db 0dh,0ah,"The minimum is: $"
```

```
len db ?
```

```
min db ?
```

Data ends

Code Segment

```
assume CS:Code,DS:Data
```

Start:

```
mov ax,Data
```

```
mov DS,ax
```

```
mov dx,offset msg
```

```
mov ah,09h
```

```
int 21h
```

```
    call Accept
    mov len,b1
    mov cl,b1
    mov ch,00h
    mov di,1000h
back: mov dx,offset msg1
    mov ah,09h
    int 21h
    call Accept
    mov [di],b1
    inc di
    loop back
    mov di,1000h
    mov cl,len
    mov ch,00h
    mov dx,offset newl
    mov ah,09h
    int 21h
    mov al,[di]
    mov min,al
chk:  mov bl,min
    mov al,[di]
    cmp bl,al
    jc a
    mov min,al
    jmp b
a:   mov min,bl
```

```
b: inc di
    loop chk
    mov dx,offset res
    mov ah,09h
    int 21h
    mov bl,min
    call DispNum
    mov ah,4ch
    int 21h
```

Accept proc

```
    mov ah,01h
    int 21h
    call AsciiToHex
    rol al,4
    mov bl,al
    mov ah,01h
    int 21h
    call AsciiToHex
    add bl,al
    ret
```

endp

DispNum proc

```
    mov dl,bl
    and dl,0f0h
    ror dl,4
    call HexToAscii
    mov ah,02h
```

```

    int 21h
    mov dl,bl
    and dl,0fh
    call HexToAscii
    mov ah,02h
    int 21h
endp
AsciiToHex proc
    cmp al,41h
    jc sk
    sub al,07h
sk: sub al,30h
    ret
endp
HexToAscii proc
    cmp dl,0ah
    jc sk2
    add dl,07h
sk2: add dl,30h
    ret
endp
Code ends
end Start

```

## OUTPUT:

For min



```
C:\TASM>min
```

```
Please enter the length of the array: 05
```

```
Enter a number: 69
```

```
Enter a number: 42
```

```
Enter a number: 12
```

```
Enter a number: 55
```

```
Enter a number: 34
```

```
The minimum is: 12
```

```
C:\TASM>
```

Name: **Tushar Nankani**

Roll No: **1902112**

Batch: **C23**

## **Microprocessor: Experiment 7**

**Aim:** Assembly program to find factorial of number using procedure.

### **Theory:**

Factorial

Factorial of a non-negative integer, is multiplication of all integers smaller than or equal to n. For example, factorial of 6 is  $6*5*4*3*2*1$  which is 720.

AAM - ASCII Adjust after Multiplication. Corrects the result of multiplication of two

BCD values.

Algorithm:

AH = AL / 10

AL = remainder

AAM - No operands

Example:

MOV AL, 15

AL = 0Fh

AAM

AH = 01

AL = 05

RET

IMUL - Signed multiply.

Algorithm: when operand is a byte:

AX = AL \* operand. when operand is a word: (DX AX) = AX \* operand.

Example:

MOV AL, -2

MOV BL, -4

IMUL BL

AX = 8 RET

### **ALGORITHM:**

Input the Number whose factorial is to be find and Store that Number in CX Register (Condition for LOOP Instruction)

Insert 0001 in AX(Condition for MUL Instruction) and 0000 in DX

Multiply CX with AX until CX become Zero(0) using LOOP Instruction

Copy the content of AX to memory location 0600

Copy the content of DX to memory location 0601

Stop Execution

### **Code:**

DATA SEGMENT

NUM DB ?

FACT DB 1H

RES DB 10 DUP ('\$')

MSG1 DB "ENTER NUMBER : \$"

MSG2 DB 10,13,"RESULT : \$"

DATA ENDS

CODE SEGMENT

ASSUME DS:DATA,CS:CODE

START:

MOV AX,DATA

```
MOV DS,AX
LEA DX,MSG1
MOV AH,9
INT 21H
MOV AH,1
INT 21H
SUB AL,30H
MOV NUM,AL
MOV AH,0
MOV AL,FACT
MOV CH,0
MOV CL,NUM
CALL FACTO
FACTO PROC ;define FACTORIAL Procedure
LABEL1: MUL CL
LOOP LABEL1
LEA SI,RES
FACTO ENDP ;here factorial procedure ends
CALL HEX2DEC
LEA DX,MSG2
MOV AH,9
INT 21H
LEA DX,RES
MOV AH,9
INT 21H
MOV AH,4CH
INT 21H
```

HEX2DEC PROC NEAR

MOV CX,0

MOV BX,10

LOOP1: MOV DX,0

DIV BX

ADD DL,30H

PUSH DX

INC CX

CMP AX,9

JG LOOP1

ADD AL,30H

MOV [SI],AL

LOOP2:

POP AX

INC SI

MOV [SI],AL

LOOP LOOP2

RET

HEX2DEC ENDP

CODE ENDS

END START

OUTPUT :

```
C:\TASM>facto
ENTER NUMBER : 5
RESULT : 120
C:\TASM>facto
ENTER NUMBER : 3
RESULT : 06
```

Name: **Tushar Nankani**

Roll No: **1902112**

Batch: **C23**

## **Microprocessor: Experiment 8**

**Aim:** Mixed Language program to check if given year is a leap year or not.

### **Theory:**

Mixed-language programming is the process of building programs in which the source code is written in two or more languages.

It allows you to:

- Call existing code that is written in another language
- Use procedures that may be difficult to implement in a particular language
- Gain advantages in processing speeds

Mixed-language programming is possible between Intel Fortran and other languages.

Although other languages (such as assembly language) are discussed, the primary focus of this section is programming using Intel Fortran and C/C++ .

Mixed language programming between these two languages is relatively straightforward because each language implements functions, subroutines, and procedures in approximately the same way.

### **What is leap year?**

Leap year is the year having 1 extra day in the calendar, i.e. a leap year has 366 days instead of 365, which are there in an ordinary year. (February 29 is added in a leap year which has 28 days in an ordinary year). Mathematically, years divisible by 4 are considered leap years except for the century years as it occurs after every 4 years.

### Logic to find Leap Year

1. In general, as the leap year occurs after every 4 years, so a leap year is the one that should be evenly divisible by 4.
2. Since after every 100 years, we skip a leap year unless it is divisible by 400. So, for a year to be a leap year, it should be divisible by 100.
3. If the year is divisible by 100, it should also be divisible by 400; then, it is considered a leap year.
4. If the year is divisible by 100 but not by 400, it is not considered to be a leap year.

### ALGORITHM:

1. START
2. Take integer variable year
3. Assign value to the variable
4. Check if year is divisible by 4 but not 100, DISPLAY "leap year"
5. Check if year is divisible by 400, DISPLAY "leap year"
6. Otherwise, DISPLAY "not leap year"
7. STOP



## CODE:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int y,num;
    clrscr();
    printf("\nEnter year : ");
    scanf("%d",&y);
    asm mov ax,y
    asm mov bx,04h
    asm mov num,ax
    asm mov dx,00h
    asm div bx
    asm cmp dx,00h
    asm jz label1
    printf("\n%d is not a leap year");
    asm jmp exit

    label1:
    asm mov ax,num
    asm mov dx,00h
    asm mov bx,400h
    asm div bx
    asm cmp dx,00h
```

```
asm jz label2
printf("\n%d is a leap year",y);
asm jmp exit
```

```
label2:
```

```
asm mov ax,num
asm mov dx,00h
asm mov bx,100h
```

```
asm div bx
asm cmp dx,00h
```

```
asm jz label3:
printf("\n%d is a leap year",y);
asm jmp exit
```

```
label3:
printf("\n%d is not a leap year",y);
```

```
exit:
getch();
}
```

## OUTPUT :

```
Enter year : 2024  
2024 is a leap year_
```

```
Enter year : 2017  
2017 is not a leap year_
```

Name: **Tushar Nankani**

Roll No: **1902112**

Batch: **C23**

## **Microprocessor: Experiment 9**

**Aim:** Mixed Language program to find the GCD/LCM of two numbers.

### **Theory:**

Mixed-language programming is the process of building programs in which the source code is written in two or more languages.

It allows you to:

- Call existing code that is written in another language
- Use procedures that may be difficult to implement in a particular language
- Gain advantages in processing speeds

Mixed-language programming is possible between Intel Fortran and other languages.

Although other languages (such as assembly language) are discussed, the primary focus of this section is programming using Intel Fortran and C/C++ .

Mixed language programming between these two languages is relatively straightforward because each language implements functions, subroutines, and procedures in approximately the same way.

### **What is GCD?**

In mathematics, the greatest common divisor (GCD) of two or more integers, which are not all zero, is the largest positive integer that divides each of the integers.

For eg.

$$36 = 2 \times 2 \times 3 \times 3$$

$$60 = 2 \times 2 \times 3 \times 5$$

Multiplication of common factors:  $2 \times 2 \times 3 = 12$

### What is LCM ?

The least common multiple (LCM) of two numbers is the smallest number (not zero)

that is a multiple of both

For eg:

$$15 = 5 \times 3$$

$$25 = 5 \times 5$$

Union of all the factors =  $5 \times 5 \times 3$

$$= 75$$

## Algorithm:

### GCD

- Load value d1 in ax and d2 in bx
- Call the gcd function
- If value in bx is zero
- Then set the value of gcd ( CX ) as ax
- Else set the value of ax as bx and value of bx as ax % bx
- Call the gcd function recursively
- Load the value of cx into ax
- Call the print function to print the gcd of two numbers

### LCM

- Start
- Store first number(num1) in a register
- Store second number(num2) in another register
- Initialize a counter register(Rd) to 01h
- Compare both the values num1 and num2
- If num1 = num2 : Store num1 or num2 as result and jump to step 8
- If num1 < num2 : Swap the register values so that num1 > num2
- Multiply num2 and Rd and divide the product with num1
- Check the remainder
- If remainder is zero then store product obtained from multiplication in step 6 as result and jump to step 8
- Else increment Rd and repeat steps 6 and 7
- Stop

## CODE:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int gcd,a,b;
    clrscr();

    printf("enter two numbers\n");
    scanf("%d%d",&a,&b);

    asm mov ax,a;
    asm mov bx,b;

l1:
    asm cmp ax,bx;
    asm jz l;
    asm jc m;
    asm jmp n;

m:
    asm sub bx,ax;
    asm loop l1;

n:
    asm sub ax,bx;
    asm loop l1;
```

l:

asm mov gcd,ax;

printf("GCD of the two numbers is %d\n",gcd);

int a1,b1,temp;

printf("Please enter first number: ");

scanf("%d",&a1);

printf("Please enter second number: ");

scanf("%d",&b1);

temp = a1;

asm mov ax,a1

asm mov bx,b1

bck:

asm cmp ax,0000h

asm jz ex

asm cmp bx,0000h

asm jz ex

asm div bl

asm cmp ah,00h

asm jz ex

temp = temp + a1;

asm mov ax,temp

asm mov bx,b1

asm jmp bck

ex:



```
printf("The LCM is: %d\n",temp);
```

```
getch();
```

```
}
```

## OUTPUT :

```
enter two numbers
12
18
GCD of the two numbers is 6
Please enter first number: 5
Please enter second number: 25
The LCM is: 25
```

Name: **Tushar Nankani**

Roll No: **1902112**

Batch: **C23**

## **Microprocessor: Experiment 10**

**Aim:** Mixed Language program to increment, decrement the size of the cursor and to disable it.

### **Theory:**

Mixed-language programming is the process of building programs in which the source code is written in two or more languages.

It allows you to:

- Call existing code that is written in another language
- Use procedures that may be difficult to implement in a particular language
- Gain advantages in processing speeds

Mixed-language programming is possible between Intel Fortran and other languages.

Although other languages (such as assembly language) are discussed, the primary focus of this section is programming using Intel Fortran and C/C++ . Mixed language programming between these two languages is relatively straightforward because each language implements functions, subroutines, and procedures in approximately the same way.

INT 10h / AH = 01h - set text-mode cursor shape.

### **input:**

CH = cursor start line (bits 0-4) and options (bits 5-7).

CL = bottom cursor line (bits 0-4).

When bits 6-5 of CH are set to 00, the cursor is visible, to hide a cursor set these bits to 01 (this CH value will hide a cursor: 28h - 00101000b). Bit 7 should always be zero.

INT 20h - "exit program" system call

CODE:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int choice, count=1;
    clrscr();
    while(choice != 4)
    {
        printf("\nEnter choice");
        printf("\n1. Increment cursor size");
        printf("\n2. Decrement cursor size");
        printf("\n3. Disable cursor");
        printf("\n4. Exit\nYou chose: ");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1 :
                count++;
                asm mov cx, count;
                asm mov ah, 01h;
                asm INT 10h;
                break;
            case 2:
                count--;
                asm mov cx, count;
                asm mov ah, 01h;
```

```
    asm INT 10h;
    break;
case 3:
    asm mov cl, 20h;
    asm mov ah, 01h;
    asm INT 10h;
    break;
}
}
}
```

## OUTPUT:

### Increase size

```
Enter choice
1. Increment cursor size
2. Decrement cursor size
3. Disable cursor
4. Exit
You chose: 1

Enter choice
1. Increment cursor size
2. Decrement cursor size
3. Disable cursor
4. Exit
You chose: 1

Enter choice
1. Increment cursor size
2. Decrement cursor size
3. Disable cursor
4. Exit
You chose: -
```

### Decrease size

```
2. Decrement cursor size
3. Disable cursor
4. Exit
You chose: 1

Enter choice
1. Increment cursor size
2. Decrement cursor size
3. Disable cursor
4. Exit
You chose: 2

Enter choice
1. Increment cursor size
2. Decrement cursor size
3. Disable cursor
4. Exit
You chose: 2

Enter choice
1. Increment cursor size
2. Decrement cursor size
3. Disable cursor
4. Exit
You chose: -
```

## Disable cursor

```
2. Decrement cursor size
3. Disable cursor
4. Exit
You chose: 2
```

```
Enter choice
1. Increment cursor size
2. Decrement cursor size
3. Disable cursor
4. Exit
You chose: 2
```

```
Enter choice
1. Increment cursor size
2. Decrement cursor size
3. Disable cursor
4. Exit
You chose: 3
```

```
Enter choice
1. Increment cursor size
2. Decrement cursor size
3. Disable cursor
4. Exit
You chose:
```

## MP ASSIGNMENT - I

Q1. Differentiate minimum mode & maximum mode of 8086.

## Minimum Mode

- uniprocessor mode - 8086 is the only processor
- $\overline{MN}/\overline{MX}$  is connected to  $V_{cc}$ .
- ALE for the latch is given by 8086 itself.
- $\overline{DEN}$  and  $DT/\overline{R}$  for the transceivers by 8086 itself.
- Direct control signals  $\overline{M}/\overline{IO}$ ,  $\overline{RD}$  and  $\overline{WR}$  are decoded by a 3:8 decoder.

## Maximum Mode.

- multiprocessor mode - 8087, 8089 along with 8086.
- $\overline{MN}/\overline{MX}$  is connected to ground.
- As there are multi-processor,  $\overline{DEN}$  &  $DT/\overline{R}$  for the transceivers are given by 8288 bus controller.
- Status signals  $S_2, S_1$  and  $S_0$  require special decoding are decoded by 8288 bus controller.

Q2. What is ISR? Explain servicing of interrupt by 8086 microprocessor.

An interrupt is a special condition that arises during the working of  $\mu P$ . The service it gives by executing a subroutine called Interrupt Service Routine (ISR).



In 8086, microprocessor, following tasks are performed when it encounters an interrupt:

1. The value of flag register is pushed into the stack. It means the first the value of SP is decr. by 2, then the value of flag register is pushed to the memory add. of stack segment.
2. The value of starting memory add. of CS (Code Segment) is pushed into the stack.
3. The value of IP (Instruction Pointer) is pushed into the stack.
4. IP is loaded from word location (Interrupt type).
5. CS is loaded from the next word location.
6. Interrupt and Trap flag are reset to 0.

The diff. types of interrupts are given by:

- ① Hardware Interrupts: are those which are caused by any peripheral device by sending a signal through a specified pin. There are 2 kinds:
  - (a) NMI (Non Maskable interrupts)
  - (b) INTR (Interrupt Requests)
- ② Software Interrupts: Some important software interrupts are:
  - (a) Type 0 corresponds to division by zero.
  - (b) Type 1 is used for single step exec<sup>n</sup> for debugging.
  - (c) Type 2 represents NMI and is used in power failure condition.
  - (d) Type 3 represents a breakpoint interrupt.
  - (e) Type 4 is the overflow interrupt.

Q3. Write a short note on assembler directives of 8086.

Assembly language has 2 types of statements:

1. Executable: Instructions that are translated into machine code by the assembler.
2. Assembler directives:
  - statements that direct the assembler to do some task
  - No M/C language code is produced for these statements.
  - Their main task is to inform the assembler about the start/end of a segment/procedure.

Some of the assembler directives are:

- ① DB (define byte): used to define a byte type var.
- ② DW (define word): used to define a word type var.
- ③ DD (double word): to define a double word (4 bytes).
- ④ DQ (quad word): to define a quad word var (8 bytes).
- ⑤ DT (Ten bytes): to define ten bytes to a variable.
- ⑥ DUP ( ): copies the content of the brackets followed by this keyword.
- ⑦ SEGMENT: used to indicate the beg of a procedure.
- ⑧ ENDS: used to indicate the end of the segment.
- ⑨ ASSUME: Associates a logical statement with a processor segment.
- ⑩ PROC: used to indicate the beg of the procedure.
- ⑪ ENDP: used to indicate the end of the procedure.
- ⑫ END: used to indicate end of the program.
- ⑬ EQU: define a const.
- ⑭ EVEN/ALIGN: ensures that the data will be stored by assembler in the memory aligned.

- (15) **OFFSET**: can be used to tell the assembler to simply substitute the offset address of the
- (16) **Start**: It is the label from where the  $\mu p$  to start executing the prog.
- (17) **Model Directives**:
- **MODEL SMALL**; All data fits in one 64 KB seg. All code fits in one 64 KB seg.
  - **MODEL MEDIUM**; All data fits in one 64 KB seg. Code may be greater than 64 KB.
  - **MODEL LARGE**; Both data & code may be greater than 64 KB.

Q4. Design a 8086 based system with the following specifications:

- 8086 working at 10 MHz in minimum mode.
- 128 KB EPROM using 32 KB devices.
- 32 KB RAM using 16 KB devices.

Show the memory map and complete design.

**Memory calculations:**

→ **EPROM:**

Reqd. Memory = 128 KB

Available memory = 32 KB

Number of chip required = 4

Number of address lines = 15 lines ( $A_{15} - A_1$ )

→ **RAM:**

reqd. memory = 64 KB

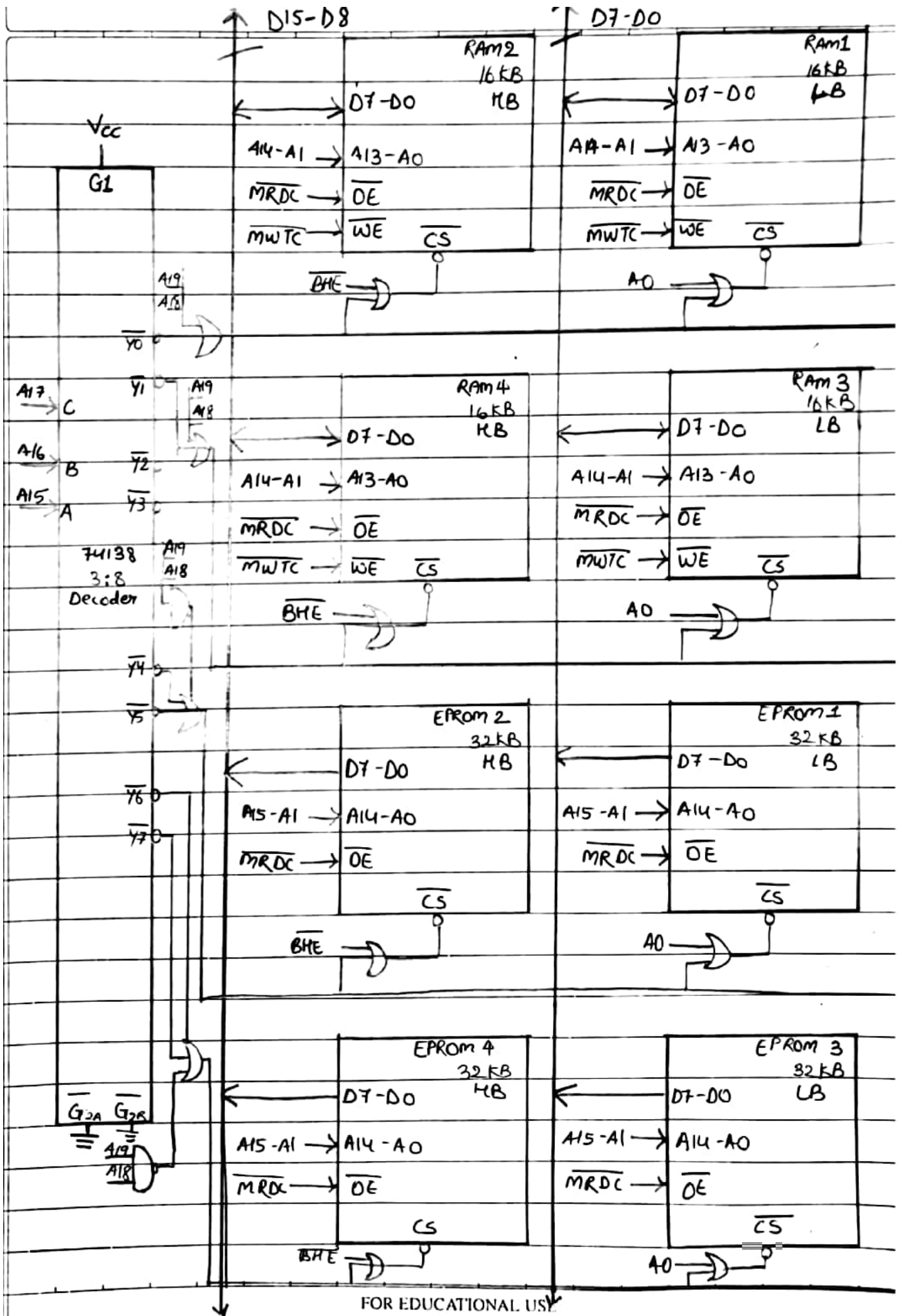
Available memory = 16 KB

number of chips reqd = 4.

number of address lines = 14 lines ( $A_{14} - A_1$ )

**MEMORY MAP:**

Memory Chip	Chip Select					Address Lines															Memory Address
	A19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	A0	
RAM 1 (LB)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00000H
	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	07FFE H
RAM 2 (HB)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	00001 H
	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	07FFF H
RAM 3 (LB)	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	08000 H
	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0FFFF H
RAM 4 (HB)	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	08001 H
	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0FFFF H
	Chip Select					Address Lines															
	A19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	A0	
ROM 1 (LB)	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	E0000 H
	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	E0001 H
ROM 2 (HB)	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	E0001 H
	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	E0001 H
ROM 3 (LB)	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	F0000 H
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	FFFFF H
ROM 4 (HB)	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	F0001 H
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFFF H





## MP ASSIGNMENT-II

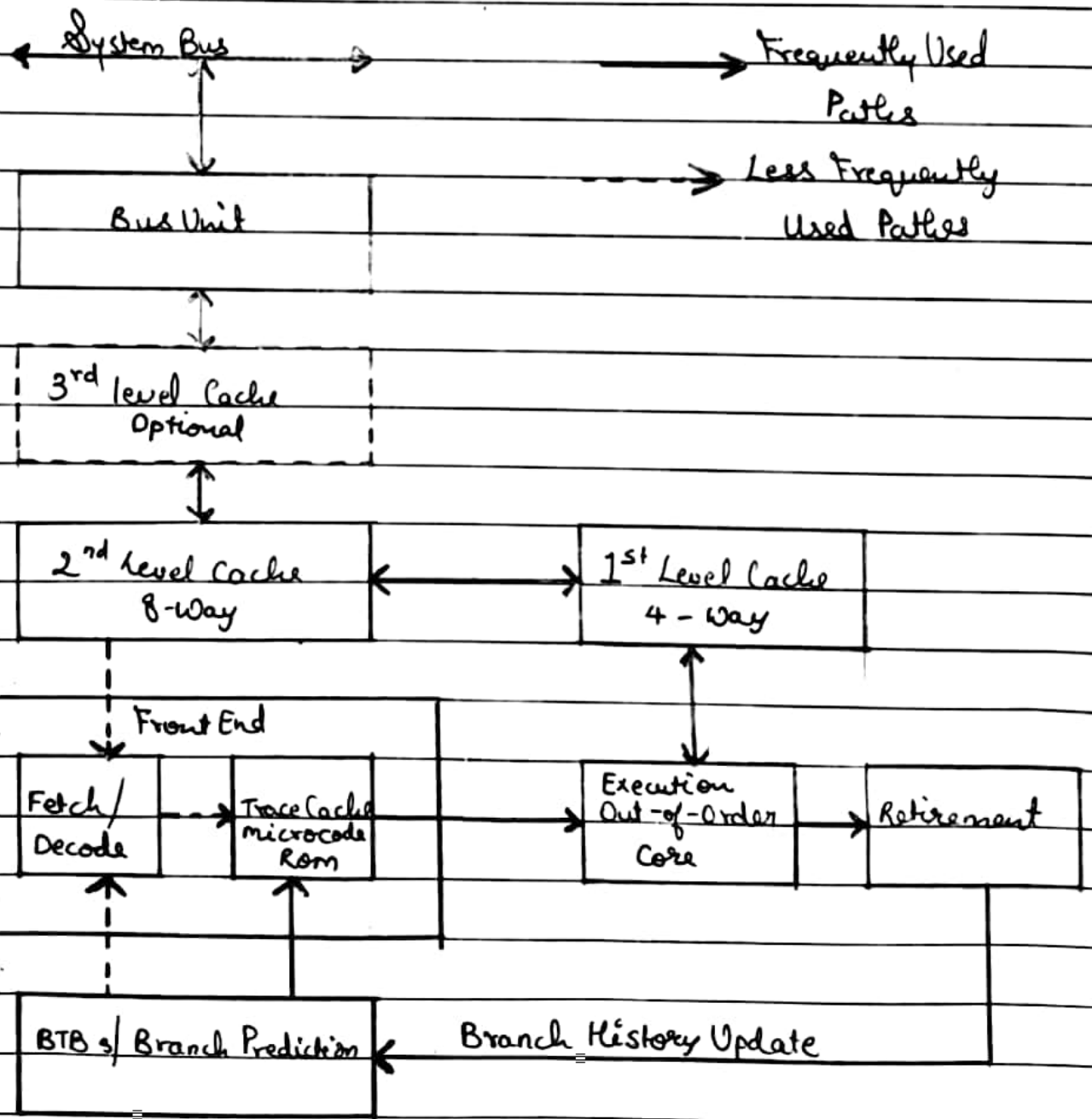
Q1. Explain P4 net burst micro architecture with block diagram.

- Ans.
- Specially designed for high end performance on band width - hungry internet and advanced application.
  - Ideal solution for sophisticated end users working with complex internet, imaging, video and multimedia applications.
  - Based on Intel NetBurst microarchitecture.
  - 400 MHz. System bus.
  - Hyper-pipelined technology & advanced dynamic execution.
  - Rapid execution engine.
  - Advanced transfer codecache.
  - Execution trace cache.
  - Streaming SIMD (Single Instruction, multiple data), Extensions 2 (SSE2)

① 400 MHz System bus :- The intel pentium 4 processor features a 400 MHz system bus (the 100 MHz clock is quad-pumped) that provides 3X bandwidth over the 133 MHz system bus in the Intel Pentium III (1.06 GB/s)

② Hyper pipelined technology :- with pentium 4 processor, intel has doubled the pipeline depth to 20 stages, enabling a higher clock frequency. The additional pipeline stages establish a new baseline for processor speed.

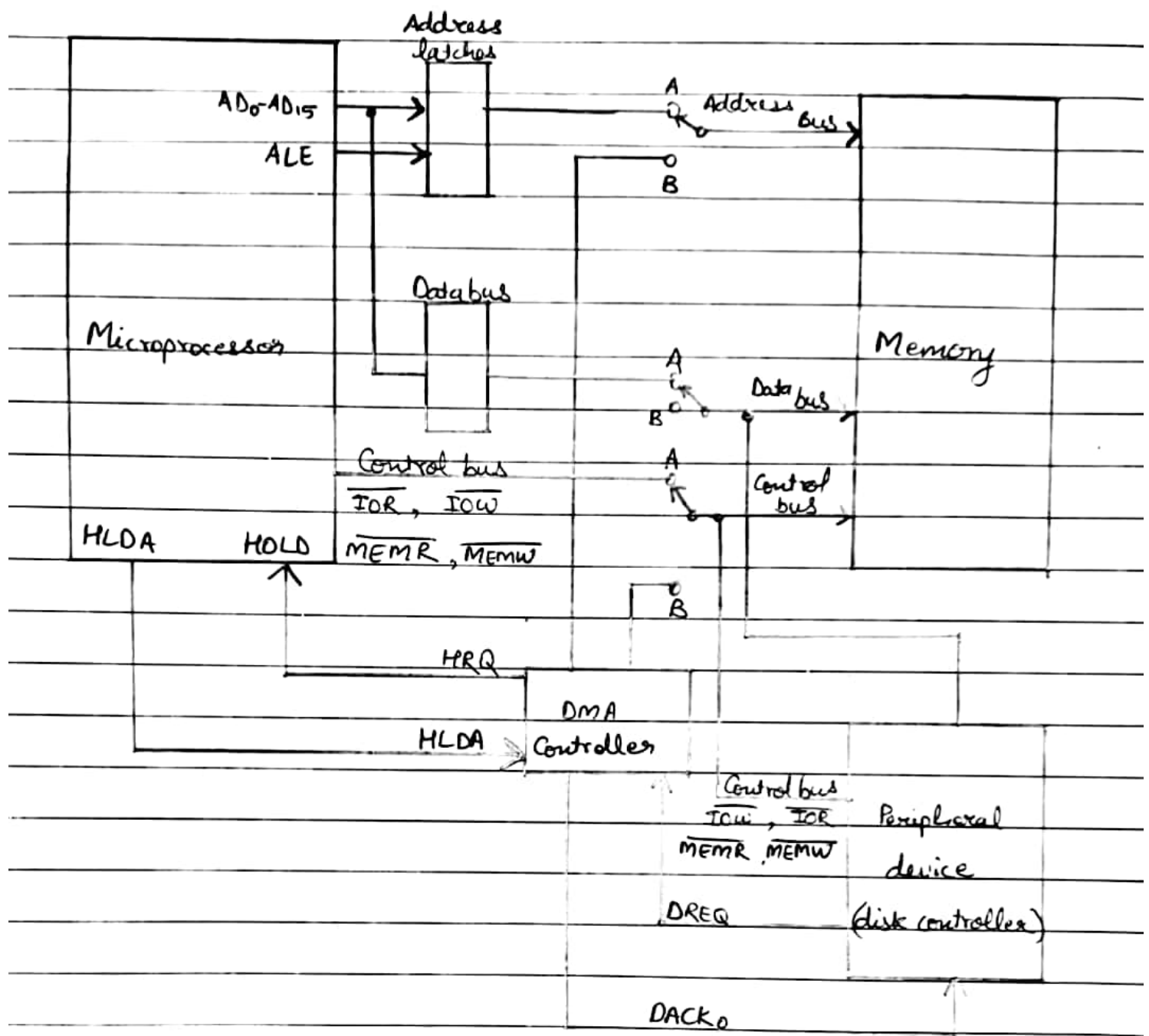
- (iii) Hyper Threading (HT) Technology :- was developed to improved the performance of IA-32 processors when executing multi-threading OS.
- (iv) Advanced Dynamic Execution - Engine is a very deep, out of order speculative execution engine that keeps the execution of unit instructions.
- (v) Rapid Execution Engine: The two ALUs in the intel pentium 4 processor run at twice the core frequency of the processor. This makes it possible to execute instructions.
- (vi) Revolutionary cache subsystem: - In order to increase performance and scalability, the Intel Pentium 4 processor features an innovative new cache subsystem designed to optimize data transfer to the core.
- (vii) Streaming SIMD extensions (SSE 2) :- to make the SIMD instruction set even more powerful; the processor provides 144 new performance improving instructions.
- (viii) Balanced platform solution: - As part of a complete platform solution, the intel pentium 4 processor was designed in tandem with the Intel 880 chipset to create a powerful new platform.





Q2. Explain the cache organization of Pentium processor.

- Ans:
1. The Pentium processor has a separated code and data cache each of 8K byte.
  2. The cache line size is 32 - bytes.
  3. Since, the Pentium processor has data bus of 8 bytes (64 bits), it requires four consecutive transfers to fill the cache line of 32 bytes.
  4. Each cache is organized as 2-way set-associative.
  5. The data cache can be configured as a write through or a write back cache on a line by line basis.
  6. The code cache does not require a write policy, as it is a read only cache.
  7. Each cache has a dedicated translation look aside buffer (TLB) to translate linear addresses to physical addresses.
  8. The data cache tags are triple ported to support 2 data transfers on a snoop cycle in the same clock.
  9. The code cache tags are also triple ported to support snooping.
  10. Individual pages in the main memory can be configured as cacheable or non-cacheable by software or hardware.
  11. The cache can be enabled or disabled by software or hardware.



### \* Steps for performing a DMA transfers:

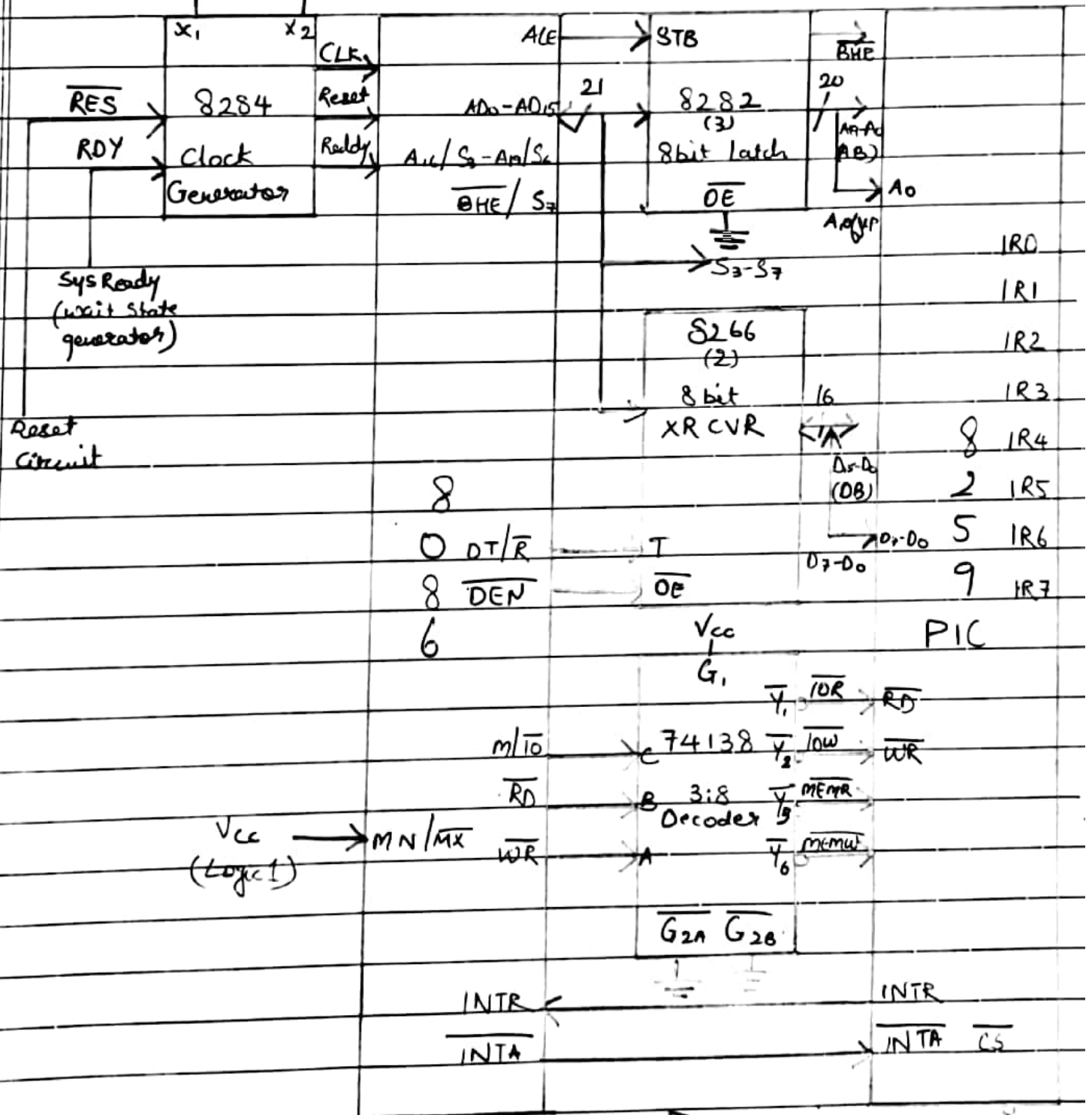
1.  $\mu P$  initializes the DMAC: this is done by giving the starting address and the number types to be transferred.
2. I/O device requests the DMAC: I/O makes the DREQ signal = 1.
3. DMAC requests the  $\mu P$  for control of the system bus:  $\mu P$  finishes the current machine cycle.  $\mu P$  enters the HOLD state.
4.  $\mu P$  releases control of the system bus:  $\mu P$  finishes the current machine cycle.
5. DMAC becomes the bus master: on getting HCLDA from  $\mu P$ , DMAC becomes the bus master.
6. DMA transfer begins: DMAC transfers data b/w memory & I/O, one byte in one cycle. Now, the DMA transfer is completed.
7. DMA transfer ends: DMAC releases control of the system bus. It makes HOLD = 0.  $\mu P$  takes control of the system bus and continues its operations.

Q4. Explain the interfacing of 8259 with 8086 in maximum mode.

A single 8259 can accept 8 interrupts, whenever a device interrupts 8259, it will interrupt the  $\mu P$  on INTR pin. Hence, first the INTR signal of the  $\mu P$  should be enabled using the STI instruction. 8259 is initialized, the following sequence takes place

- ① The corresponding bit for interrupt is set in IRR.
- ② The priority resolver checks the 3 registers: IRR (for highest req.), IMR (for masking status) and InSR (for the current level serviced) and thus highest priority is defined.
- ③ If the interrupt that has occurred is of the highest priority amongst those that have occurred, is unmasked and is higher priority than the one currently being serviced, only then the interrupt will be sent to the INT signal of  $\mu P$ .
- ④  $\mu P$  finishes the current instruction and acknowledge the interrupt by sending the first  $\overline{INTA}$ .
- ⑤ 1<sup>st</sup>  $\overline{INTA}$  is a confirmation by the  $\mu P$  that the interrupt will be serviced.
- ⑥ The  $\mu P$  sends the second  $\overline{INTA}$  pulse to 8259.
- ⑦ In response, 8259 sends the 1 byte vector number  $N$  to  $\mu P$ .
- ⑧ Now, the  $\mu P$  multiplies  $N \times 4$ , to get the values of CS & IP from the IVT. The ISR thus begins.
- ⑨ At the end of the ISR, the programmer gives an EOI command (assuming the Normal EOI mode). This makes the corresponding bit '0' in register of 8259.

"3x"



10map of 8259 at 80H

	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>			
ICW1	1	0	0	0	0	0	0	0	80H	A <sub>7</sub>	74138
ICW2	1	0	0	0	0	0	1	0	82H	A <sub>6</sub>	3:8 Decoder
	chip selection						Internal Selection			A <sub>5</sub>	G <sub>2A</sub> G <sub>2B</sub>

FOR EDUCATIONAL USE