Department of Computer Engineering.

D. Y. Patil College of Engineering, Akurdi, Pune-44

LABORATORY MANUAL

MICROPROCESSOR ARCHITECTURE LABORATORY

S.E. COMPUTER

SEMESTER – I

TEACHING SEHEME:	EXAMINATION SCHEME:
PRACTICAL: 02HRS \ WEEK \ BATCH	ORAL: 50 MARKS

(Mrs.Asmita Patil) **Prepared By**	**(Mrs.Shanthi.K.Guru)** **Checked By**	**(Mrs.M.A.Potey)** **HOD Comp**

Microprocessor Architecture Laboratory

List of Practical Assignments

Group A (Mandatory)

Group B

01	Write ALP to print “Hello World!” Program using 16, 32 and 64-bit model and segmentation
02	Write an ALP to accept ten 32-bit and 64 bit Hexadecimal numbers from user and then store in data segment table and display then numbers.
03	Write an ALP to accept a string and to display it’s length.
04	Write an ALP to perform arithmetic and logical operations using ‘n’, 32-bit and 64-bit numbers stored in an array using 64 bit register operations.
05	Write an ALP to perform memory segment and register load/store operations using different addressing modes.
06	Study of GDTR, LDTR and IDTR in Real Mode.

07	Write an ALP to fond the largest of given byte/Word/Dword/64-bit numbers.
08	Write a switch case driven ALP to perform 64-bit hexadecimal arithmetic operations (+,-,*, /) using suitable macros. Define procedure for each operation.
09	Write an ALP to read command line arguments passed to a program.
10	Write an ALP to count no. of positive and negative numbers from the array.
11	Write ALP to find average of n numbers stored in memory.
12	Write program to read & display contents of file.

Group C

13	Write ALP to switch from real mode to protected mode and display the values of GDTR, LDTR, IDTR, TR and MSW Registers.

Assignment No. 01

Print "Hello World!" using 32, 64 bit model and segmentation.

AIM: Write an ALP to print “Hello World!” using 32,64 bit model and segmentation.

THEORY:

The assembly program structure -

The assembly program using nasm assembler can be divided into three sections:

The .data section

This section is for "declaring initialized data", in other words defining "variables" that already contain stuff. However this data does not change at runtime so they're not really variables. The .data section is used for things like filenames and buffer sizes, and you can also define constants using the EQU instruction. Here we can use the DB, DW, DD, DQ and DT instructions. For example:

```
section .data
db 'Hello world!'
msglength equ $-msg
buffersize dw 1024
```

The .bss section

This section is where you declare your variables. You use the RESB, RESW, RESD, RESQ and REST instructions to reserve uninitialized space in memory for your variables, like this:

```
section .bss
filename resb 255      // reserve 255 bytes
number resb 1          // reserve 1 byte
bignum resw 10         // reserve 1 word (1 word= 2 bytes)
realarray resq 4       // reserve an array of 10 reals.
```

res stands for reserve that many byte/word/double word/quad type of memory locations

The .text section

This is where the actual assembly code is written. The .text section must begin with the declaration global _start, which just tells the kernel where the program execution begins. (It's like the main function in C or Java, only it's not a function, just a starting point.)

Eg.:

```
section .text
global _start
_start:
.
.
.
```

.
; Here is the where the program actually begins

Algorithm:

Steps to create and execute .asm program in nasm : -

1. Boot the machine with ubuntu
2. Select and click on <dash home> icon from the toolbar.
3. Start typing "terminal" . Different terminal windows available will be displayed.
4. Click on "terminal" icon. A terminal window will open showing command prompt.
5. Give the following command at the prompt to invoke the editor

   gedit hello.asm

6. Type in the program in gedit window, save and exit
7. To assemble the program write the command at the prompt as follows and press enter key

   nasm –f elf32 hello.asm –o hello.o (for 32 bit)

   nasm –f elf64 hello.asm –o hello.o (for 64 bit)

8. If the execution is error free, it implies hello.o object file has been created.
9. To link and create the executable give the command as ld –o hello hello.o
10. To execute the program write at the prompt ./hello
11. "hello world" will be displayed at the prompt

The Linux System Calls – (int 80h for 32bit execution)

Linux system calls are called in exactly the same way as DOS system calls:

1. write the system call number in eax.
2. set up the arguments to the system call in eax,ecx, etc.
3. call the interrupt 80h .
4. The result is usually returned in eax.

Interrupts:

- Interrupt to accept character/number:

32 bit	**64-bit**
mov eax,3	mov rax,0
mov ebx,0	mov rdi,0

int 80h	syscall

- Interrupt to display string/numbers:

32 bit	**64-bit**
mov eax,4	mov rax,1
mov ebx,1	mov rdi,1
int 80h	syscall

- Interrupt for termination/exit:

32 bit	**64-bit**
mov eax,4	mov rax,60h
mov ebx,1	mov rdi,0
int 80h	syscall

Registers in real mode in 80386dx:

Special purpose registers or Segment registers are:

CS-Code Segment

DS-Data Segment

SS-Stack Segment

ES-Extra Segment

FS and GS Segments are extra segments

General purpose registers:

32 bit	**64-bit**
EAX-Accumulator	RAX-Accumulator
EBX-Base	RBX-Base
ECX-Counter	RCX-Counter
EDX-Data	RDX-Data

Index,Pointer and Base Registers:

32 bit	**64-bit**
ESP-Stack Pointer	RSP-Stack Pointer
EBP-Base Pointer	RBP-Base Pointer
ESI-Source Index	RSI-Source Index
EDI-Destination Index	RDI-Destination Index

Flag Registers:

A flag is a filp-flop which indicates some condition produced by the execution of an instruction or controls certain operation of the EU(Execution Unit).It consists of Status flags, Control flags and System flags.

Conclusion:

Hence we have implemented and executed the ALP program to print “Hello World!”

Sample Program -

The “hello world!” assembly program (32bit execution)

section .data

hello db 'Hello world!',10 ; 'Hello world!' plus a linefeed character

helloLen: equ $-hello ; Length of the 'Hello world!' string

section .text

global _start

_start:

```
mov eax,4                  // The system call for write
mov ebx,1                  // File descriptor 1 - standard output
mov ecx,hello              // Put the offset of hello in ecx
mov edx,helloLen;          // helloLen is a constant,
mov edx,[helloLen]         // to get it's actual value
int 80h                    // Call the kernel
mov eax,1                  // The system call for exit (sys_exit)
mov ebx,0                  //  Exit with return code of 0 (no error)
int 80h
```

Sample output -:

```
$ nasm –f elf32 hello.asm -o hello.o
$ ld –o hello hello.o
$ ./hello
Hello world!
```

The "hello world!" assembly program (64 bit execution)

```
section .data
hello db Hello world!',10
helloLen: equ $-hello
Section .text
.global _start
_start:
mov rax,1                  //system call 1 is write
mov rdi,1                  //file handle 1 is stdout
mov rsi, hello             //address of string to output
```

```
mov rdx, hellolen        //number of bytes
syscall                  //invoke operating system
mov rax,60               //system call 60 is exit
mov rdi,0                // we want return code 0
syscall                  //invoke operating system
```

Sample Output:-

$ nasm –f elf64 hello.asm -o hello.o

$ ld –o hello hello.o

$ ./hello

Hello world!

Expected Oral Questions -:

1) Explain .bss/.data/.text section in nasm
2) What is meaning of ‘R’ in Rax register.
3) What if meaning of elf64.

Assignment No. 02

Accept Ten 32-bit and 64 bit Hexadecimal numbers from user and Display

Aim: Write an ALP To accept ten 32-bit and 64 bit hexadecimal numbers from user and store them in data segment table and display the number.

Theory:

Representation of 32-bit number 24ab3f89 -

32 bit number in ASCII form:
This requires 8 memory locations (8 bytes)

32	34	41	42	33	46	38	39

32 bit number in HEX form:
This requires 4 memory locations (4 bytes)

24	ab	3f	89

Representation of 64-bit number 24ab3f8912345678 -

64 bit number in ASCII form:
This requires 16 memory locations (16 bytes)

32	34	41	42	33	46	38	39	31	32	33	34	35	36	37	38

64 bit number in HEX form:

This requires 8 memory locations(8 bytes)

24	ab	3f	89	12	34	56	78

Consider that the numbers are stored in the array. We accept the numbers from the user, the numbers are stored in ASCII form. So to access the array of 32 bit we required to move the pointer by 8 to get the nest number. And to access the array of 64 bit we required to move the pointer by 16 to get the nest number.

Algorithm:
Step1. Start.
Step2. Initialize the data. Put count value =10.
Step3. Accept ten 32 bit the numbers.
Step4. Insert the number in the array one by one.
Step5. After one number is accepted increment pointer by 09, Step to get next number
Step6. check for counter.
Step7.when counter =0,it stops accepting number
Step8. Reload the pointer.
Step9. Display the number by decrementing the count.
Step10. End.

Repeat the same procedure for 64 bit numbers and increment the pointer by 17 to get the nest number.

Instructions:

1] jnz:
Description: Jump if above not zero to label specified
Flags: ZF = 0

2] jz:
Description: Jump if above zero to label specified
Flags: ZF = 1
short jump opcodes: 74

3] add:
Description: This instruction adds a number from source to number from destination and puts the result to specified destination.
destination=destination+source
Flags: CF, ZF, OF, PF
e.g. add eax, ebx

Conclusion: Hence we displayed the 32 bit and 64 bit array by using assembly language

Sample Program:

32 bit array representation

```
section .data
msg: db 'enter ten 32 bit nos' ,10        ; 10-for new line
len: equ $-msg
msg1: db 'entered nos are'                ; db=defined double word
len1: equ $-msg1
count db 0                                ;declare variables and initialize them.

section .bss
number: resb 200

section .text
global _start
_start:
mov eax,4                                 ;Interrupt to display message
mov ebx,1
mov ecx,msg
mov edx,len
int 80h
mov esi,number                            ; esi is my starting pointer
mov byte[count],10                        ; the contents of byte size count= 10
loop1:
mov eax,3                                 ; interrupt to accept numbers
mov ebx,0
mov ecx,esi
mov edx,09
int 80h
add esi,09
dec byte[count]
jnz loop1
mov eax,4
mov ebx,1
mov ecx,msg1
```

```
mov edx,len1
int 80h
mov esi,number                          ; number is the array and moved to esi to display it
mov byte[count],10
loop2:
mov eax,4
mov ebx,1
mov ecx,esi
mov edx,09
int 80h
add esi,09
dec byte[count]
jnz loop2
mov eax,1
mov ebx,0
int 80h
```

Sample Output :

;cl2@cl2-OptiPlex-390:~$ nasm -f elf arr.asm
;cl2@cl2-OptiPlex-390:~$ ld -s -o arr arr.o
;cl2@cl2-OptiPlex-390:~$ ./arr
;enter ten 32 bit nos
;1234abcd
;2345feda
;eac98762
;4592ace7
;5678eadf
;67abdef2
;aaaaaabb
;123456cf
;86754edc
;abcdef789

;entered nos are:
;1234abcd
;2345feda
;eac98762
;4592ace7
;5678eadf
;67abdef2
;aaaaaabb
;123456cf
;86754edc
;abcdef789

64 bit array representation

```
section .data
msg: db 'enter ten 64 bit nos' ,10
len: equ $-msg
msg1: db 'entered nos are'
len1: equ $-msg1
count db 0

section .bss
number: resb 200

section .text
global _start
_start:

mov eax,4
mov ebx,1
mov ecx,msg
mov edx,len
int 80h
mov esi,number                  ; esi is my starting pointer

mov byte[count],10              ; the contents of byte size count to 10
loop1:
mov eax,3                       ; interrupt to accept numbers
mov ebx,0
mov ecx,esi
mov edx,17
int 80h
add esi,17                      ;adds 17 to the contents of esi
dec byte[count]                 ;decrement the byte count
jnz loop1
mov eax,4                       ;system out function
mov ebx,1
mov ecx,msg1
mov edx,len1
int 80h
mov esi,number
mov byte[count],10
loop2:
mov eax,4
mov ebx,1
mov ecx,esi
mov edx,17
int 80h
```

```
add esi,17
dec byte[count]
jnz loop2
mov eax,1
mov ebx,0
int 80h
```

Sample Output

;cl2@cl2-OptiPlex-390:~$ nasm –f elf poo.asm
;cl2@cl2-OptiPlex-390:~$ ld -s -o poo poo.o
;cl2@cl2-OptiPlex-390:~$ ./poo
;enter 10 64 bit nos
;1234abcd12345678
;4567edbcfa871235
;eac98762987654ed
;4592ace7ffffffff
;5678eadfababaded
;67abdef245677889
;aaaaaabb234567eb
;abdef689123456cf
;86754fffea783edc
;abcdef78965432ae
;**entered nos are:**
;1234abcd12345678
;4567edbcfa871235
;eac98762987654ed
;4592ace7ffffffff
;5678eadfababaded
;67abdef245677889
;aaaaaabb234567eb
;abdef689123456cf
;86754fffea783edc
;abcdef78965432ae

Expected Oral Questions -:

1) How to access on array.
2) What byte [count] stands for?
3) What is ASCII value of 10.
4) Why to increment pointer of 17 for 64 bit numbers.

Assignment No. 03

Accept a string and display it's length.

Aim : Write an ALP to accept a string and to display its length.

Theory :
In this program we have to accept the string from the user. The string is accepted character by character and the count for the number of characters accepted is maintained. This count is length of the string. Also to detect the end of string we have to scan for enter i.e. ASCII 10

Instructions -
1) JNE -
Description: Jump if above not equal to label specified
Flags: CF = 0

2)PUSH-
Description: This is going to push the data defined in the source operand.
Stack pointer is first decremented and then data is pushed.
e. g. push eax

3)POP-
Description: This is going to pop the data defined in the destination operand. Stack pointer is first incremented and then data is poped.
e.g. pop eax

4)CMP -
Description: This is going to subtract source and destination operand and result is reflected in following flag registers.
Flags: CF, ZF
e.g. cmp eax, ecx

Algorithm :

1) Declare two variables temp and len.
2) Display the message to enter the string.
3) Put ebx pointer to the string. To get the proper ascii value of length initialize the len variable by 48 (DECIMAL to ASCII conversion)
4) Accept the sting character by character and maintain the count .
5) Check for enter (ASCII 10) to get end of string.
6) Then display the length in len.

Conclusion: Hence we found out length of string and displayed it using Assembly language.

Sample Program:

```
section .data                ; data section
msg1: db 'Enter string:'
```

```
size1: equ $-msg1

section .bss                ; code section
string: resb 50
temp: resb 1
len: resb 1

section .text
global _start

_start:
mov eax, 4                  ; Display interupt
mov ebx, 1
mov ecx, msg1
mov edx, size1
int 80h

mov ebx, string
mov byte[len], 0

;convert to ascii:

mov ebp, 48                 ; Decimal to ascii conversion
mov[len], ebp

reading:
push ebx
mov eax, 3
mov ebx, 0                  ; interrupt to accept the string
mov ecx, temp
mov edx, 1
int 80h

pop ebx
mov al, byte[temp]          ; read the string character by character
mov byte[ebx], al

inc byte[len]
inc ebx

cmp byte[temp], 10          ; check for end of string
jne reading

dec ebx                     ; stop reading
dec byte[ebx]
dec byte[len]
```

```
mov eax, 4                ; Interrupt to display length
mov ecx, len
mov edx, 1
int 80h

Exit:
mov eax, 1                ; termination interrupt
mov ebx, 0
int 80h
```

Sample Output:-

```
cl2@cl2-OptiPlex-390:~/Desktop$ nasm -f elf strng.asm
cl2@cl2-OptiPlex-390:~/Desktop$ ld -s -o strng strng.o
cl2@cl2-OptiPlex-390:~/Desktop$ ./strng

Enter string: MATLAB
Length of string is : 6
```

Expected oral questions

1) How to check end of string
2) What is meaning of POP,PUSH, CMP instructions.
3) How decimal to ASCII conversion is done .

Assignment No. 04

Arithmetic and logical operations using ‘64 bit register operations.

Aim: Write an ALP T to perform arithmetic and logical operations using 'n' 32-bit and 64-bit numbers.

Theory:

Consider that five numbers are stored in the array. In this program the addition arithmetic operation and OR logical operation is done. The result of both operation is in hex form. So we have to use conversion procedure from HEX to ASCII.

Instruction:
1)ADD-
Description: This is going to add the number in source with number in destination. It is going to affect the flag.
Flag: CF, ZF, OF, PF
e.g. add eax,ebx

2)OR-
Description: This is logically or the number in source with number in destination. The result is stored in destination.
e.g. or ax,bx

Algorithm:

1) Initialize array of five 32 bit numbers.
2) Put a pointer to the array.
3) Put a counter of 5
4) Add the numbers one by one till count =0.
5) Display the result.
6) Reload the pointer to the array
7) Reload the counter to 5.
8) Make logical or of 5 numbers.
9) Display the result.

Repeat the same procedure for 64 bit numbers.

Conclusion: Hence we done arithmetic and logical operation on 32 bit and 64 bit array by using assembly language.

Sample Program:

```
section . data
msg: db 'The addition numbers is:'
len: equ $-msg

msg1: db 'The logical or is:'
len1: equ $-msg1

arr:dq 1111111111111111h,0000h,000000011h,1111101010h,1111h

section .bss
count: resb 1
result: resb 16
result1: resb 16

section .text
global _start

_start:
mov rax,1
mov rdi,1
mov rsi,msg
mov rdx,len
syscall

;addi
mov byte[count],5
mov rax,0
mov rsi,arr

loop1:
add rax,[rsi]
add rsi,08
dec byte[count]
jnz loop1

mov rbx,rax
mov rsi,result
mov byte[count],10h
mov cl,04

call display
```

```
mov rax,1
mov rdi,1
mov rsi,result
mov rdx,17
syscall

;logical
mov rax,1
mov rdi,1
mov rsi,msg1
mov rdx,len1
syscall

mov byte[count],5
mov rax,0
mov rsi,arr

loop2:
or rax,[rsi]
add rsi,08
dec byte[count]
jnz loop2

mov rbx,rax
mov rsi,result1
mov byte[count],10h
mov cl,04

call display

mov rax,1
mov rdi,1
mov rsi,result1
mov rdx,17
syscall

display:
l1:
rol rbx,cl
mov dl,bl
and dl,0fh
cmp dl,09h
jbe l2
add dl,07h
```

```
l2:
add dl,30h
mov [rsi], dl
inc rsi
dec byte[count]
jnz l1
mov byte[rsi], 0ah
ret

mov rax, 60
xor rdi, rdi
syscall
```

Sample Output -

'The addition numbers is: 1111111122222334

The logical or is: 1111111111111111

Expected Oral Questions -:

1) How ADD, OR, ROL instructions works, and what flags are affected.
2) Explain display procedure.
3) What is SYSCALL.

Assignment No. 05

Memory segment and register load/store operations using different addressing modes.

Aim: Write an ALP to perform memory segment and register load/store operation using different addressing modes.

Theory:

Consider the array of five 32-bit numbers. In this program we perform the memory segment and register load/store operation using addressing modes. The commands used in the program are jmp, and, cmp, jbe, inc, dec, jnz.

Instructions:

1. **JMP:**

   Description: This instruction will cause the 80386 to fetch its next instruction from the location specified in the instruction rather than next instruction after JMP instruction.

   Flags: No flags are affected.

   Example: JMP WORD PTR [BX]

2. **AND:**

   Description: This instruction ANDs each bit in a source byte or word or double word with the same number bit in the destination byte or word or double word. The result is store at specified location.

   Flags: CF and OF are both 0 after the execution of the AND instruction.

   PF, SF and ZF are updated by AND instruction.

   AF is undefined.

   Example: AND AL, BL

3. **CMP:**

   Description: This instruction compares a double word/word/byte from source with the double word/word/byte form destination. The result is not stored in either of the destination or source. The result is stored in the CF ZF, SF flags.

   Flags: AF, OF, SF, ZF, PF and CF are updated according to the result.

   Example: CMP BH, CL

4. **JBE:**

   Description: This instruction jump to next instruction of the program when the condition if below or equal.

   Flags: The CF and ZF are affected.

   Example: JBE 12

5. **INC:**

   Description: This instruction adds 1 to the destination operand.

   Flags: SF, PF, OF, ZF, AF are affected.

   Example: INC CX

6. **DEC:**

    Description: This instruction subtracts 1 from the destination word, double word or byte.

    Flags: SF, ZF, OF, PF and AF are affected.

    Example: DEC AL

7. **JNZ:**

    Description: This instruction is used to jump to next instruction in the program when the zero flag is not equal to 0.

    Flags: Only the ZF is affected.

    Example: JNZ L1

Algorithm:

1. Enter the five 32-bit numbers in the array.
2. Put the pointer to the array.
3. Put the counter of the array as 5.
4. Find the different addressing modes in the array.

Repeat the same procedure for the 64-bit numbers.

Conclusion: We perform the memory segment and register load/store operation using different addressing modes in the assembly language.

Sample Program:

```
section .data
msg1 db 'register addressing : '
len1 equ $-msg1

msg2 db 'register indirect addressing : '
len2 equ $-msg2
```

```
msg3 db 'immediate addressing : '
len3 equ $-msg3

msg4 db 'indexed addressing : '
len4 equ $-msg4

msg5 db 'base addressing : '
len5 equ $-msg5

msg6 db 'base indexed addressing : '
len6 equ $-msg6

msg7 db 'base indexed addressing : '
len7 equ $-msg7

cnt dd 2bbbbbb2h
arr dd 11111111h,22222222h,44444444h,55555555h
arr1 dd 04h,08h,0ch,10h

section .bss
result resb 10
count resb 1

%macro disp 2
mov eax,4
mov ebx,1
mov ecx,%1
mov edx,%2
int 80h
%endmacro

section .text
global _start
_start:
disp msg1, len1
mov eax,1aaaaaa1h
mov ebx,eax
call display
disp msg2, len2
mov ebx,[cnt]

call display
disp msg3, len3
mov ebx,3cccccc3h

call display
```

```
disp msg4,len4
mov esi,arr
mov ebx,[esi+8]

call display
disp msg5,len5
mov ebx,arr
mov eax, [ebx+0ch]
mov ebx,eax

call display
disp msg6,len6
mov ebx,04h
mov esi,arr1
mov eax,[ebx+esi]
mov ebx,eax
call display
jmp exit

display:
mov byte[count],08
mov esi,result
mov cl,04
l1:rol ebx,cl
mov dl,bl
and dl,0fh
cmp dl,09h
jbe l2

add dl,07h
l2: add dl,30h
mov [esi],dl

inc esi
dec byte[count]
jnz l1
mov byte[esi],0ah

mov edx,9
mov ecx,result
mov ebx,1
mov eax,4
int 80h
ret

exit:
```

```
mov eax,1
mov ebx,0
int 80h
```

Sample Output –

register addressing : 1AAAAAA1

register indirect addressing : 2BBBBBB2

immediate addressing : 3CCCCCC3

indexed addressing : 44444444

base addressing : 55555555

base indexed addressing : 00000008

Expected Oral Questions –

1) Explain different addressing modes.
2) What is meaning of Offset and effective address.
3) What is Macro. Explain its format.

Assignment No. 06

Study of GDTR, LDTR and IDTR

Aim: Write an ALP to program to use GDTR , LDTR and IDTR in real mode.

Theory:

1. GDT:

    - This is the most important and main table of descriptors. It contains 8192 descriptors.
    - The same GDT can be used by all programs to refer to segment of memory.

- A 80386 processor in the protected mode can have many LDT's but only one GDT.

GDTR:

- The GDTR is the 48-bit register of the 80386 processor.
- The lower two bytes of the GDTR are called as the LIMIT. The LIMIT specifies the size of GDT in bytes.
- The upper 4-bytes of GDTR are called as BASE. The BASE gives beginning physical address of GDT in memory.
- Each descriptor in the GDTR is eight bytes long and the size of the GDT can be expanded to 65536 bytes simply by changing the value of LIMIT of the GDTR before the 80386 is switched from real-mode operation to the protected-mode operation.

2. IDT:

- IDT contains group of descriptors that define interrupts and exception handling routines.
- Its function is to hold the segment descriptors that define the interrupt or exception handling routines.
- For 80386 to work in protected mode , at least one IDT needs to be defined.
- In the IDT there are 256 gate descriptors.

IDTR:

- IDTR is a 48-bit register of the 80386 processor. Its functions is to search the IDT.
- The lower two bytes of the register that is LIMIT which gives the size of the IDT and upper three bytes that is BASE identifies the IDT is equal to LIMIT + 1 and IDT can be upto 65536 bytes long.
- The 80386 supports only 256 interrupts therefore the size of the IDT should not be support more than 256 interrupts.
- The default value that 80386 loads into IDTR defines a base address of 0 and limit of 03FFH to the IDT after reset.

3. LDT:

- A multitasking system is defined on a per task basis.
- The main purpose of an LDT would be to be combined with GDT in order to expand the total number of available descriptors.
- Each task can have its own LDT and can also be shared with other tasks.

LDTR:

- LDTR is a 16-bit register of the 80386 processor.
- The LDTR does not directly define the LDT. It gives a selrctor which points to an LDT descriptor in the GDTR.
- If a selector is loaded into LDTR then the corresponding descriptor is read from global memory and loaded into the local-descriptor table cache in the 80386.
- The LDT is also called as the “private table” which defines local memory address space for use by the task.
- Each task can have its own segment of local memory.

Conclusion : we successfully studied GDTR,LDTR,IDTR in real mode.

Expected Oral Questions –

1) What is meaning of GDT,LDT,IDT
2) How many GDT, LDT, IDT present in 80386dx.
3) What is descriptor?
4) What is selector?
5) What is a size of GDTR,LDTR,IDTR.

Assignment No. 07

Find Largest of given byte/Word/Dword/64-bit numbers

Aim: Write an ALP to found the largest of given byte/Word/Dword/64-bit numbers.

Theory:

Consider that the five numbers are stored in the array. In this program we find the largest number stored in the array. For this program we use the instructions like cmp , jnc , jnz , jbe.

Instructions:

1) **CMP:**

Description: This instruction compares a double word/word/byte from source with the double word/word/byte form destination. The result is not stored in either of the destination or source. The result is stored in the CF ZF, SF flags.

Flags: AF, OF, SF, ZF, PF and CF are updated according to the result.

Example: CMP BH, CL

2) **JNC:**

Description: This instruction is used to jump to next instruction when the carry flag is not equal to 0.

Flags: Only the CF is affected.

Example: JNC NEXT

3) **JNZ:**

Description: This instruction is used to jump to next instruction in the program when the zero flag is not equal to 0.

Flags: Only the ZF is affected.

Example: JNZ L1

4) **JBE:**

Description: This instruction jump to next instruction of the program when the condition if below or equal.

Flags: The CF and ZF are affected.

Example: JBE 12

Algorithm:

1. Take the five 32-bit numbers in the array.
2. Put pointer to the array.

3. Put counter to five.
4. Compare each number of the array with another numbers in the array.
5. Display the largest number in the array.

   Repeat same procedure for 64-bit number.

Conclusion: Here we count the largest numbers in the array.

Sample Program:

```
 Section .data
 Msg: db 'The largest number is:'
 Len: equ $-msg
Arr: dd 1aaaaaa1h, 22h, 11h, 3abcdee3h, 444h

Segment .bss
Count: resb 1
Result: rssb 15

Section .text
Global _start

_start:
Mov eax, 4
Mov ebx, 1
Mov ecx, msg
Mov edx, len
Int 80h

Mov byte [count], 5
Mov eax, 0
Mov esi, arr

Loop1:
    Cmp eax, [esi]
     Jnc next
    Mov eax, [esi]

Next:
    Add esi, 04
   Dec byte [count]
   Jnz loop1
```

```
  Mov ebx, eax
  Mov esi, result

  Mov byte [count], 08
  Mov cl, 04

L1:
    Rol ebx, cl
    Mov dl, bl
    And dl, 0fh
    Cmp dl, 09h
    Jbe l2

   Add dl, 07h

L2
    Add dl, 30h
    Mov [esi], dl
    Inc esi
    Dec byte [count]
    Jnz l1
    Mov byte [esi], 0ah

    Mov edx, 9
    Mov ecx, result
    Mov ebx, 1
    Mov eax, 4
    Int 80h

    Mov eax, 1
    Mov ebx, 0
     Int 80h
```

Sample output –

The largest number is : 3abcdee3h

Expected Oral Questions –

1) Explain logic of program.
2) Explain ADD, DEC instructions.
3) Explain procedure to convert the number from hex to ascii.

Assignment No. 08

Switch case driven ALP to perform 64-bit hexadecimal arithmetic operations (+,- ,*,%) using macros.

Aim: Write a switch case driven ALP to perform 64-bit hexadecimal arithmetic operations (+,- ,*,%) using macros. Define procedure for each operation.

Theory:

Consider that five numbers are stored in the array. The procedure is written for all operations. The format is

```
.procname
--- lines of code
---
ret
```

The macro is written for display interrupt. The format is

macroname macro <no. of arguments>
%arg1
%arg2
end
The switch case format is also written for scanning the choice of user.

Instruction:

1)ADD-
Description: This is going to add the number in source with numberindestination. The result is stored in destination.
Flag: CF, ZF, OF, PF
e.g. add eax,ebx

2)SUB-
Description: This is going to subtract the number in source from numberindestination. The result is stored in destination.
Flag: CF, ZF, OF, PF
e.g. sub eax,ebx

3)MUL-
Description: This is going to multiply the number in accumulator to the numberindestination. The result is stored in accumulator.
Flag: CF, ZF, OF, PF
e.g. mulebx

4)DIV-
Description: This is going to divide the number in accumulator by the number in destination. The quotient is in accumulator and reminder in edx/dx/dl
Flag: CF, ZF, OF, PF
e.g. div ebx

Algorithm:

1) Initialize array of five 64 bit numbers.
2) Put counter for the array
3) Ask user for its choice.
4) If choice is 1, call addition procedure.
5) If choice is 2, call subtraction procedure.
6) If choice is 3, call multiplication procedure.
7) If choice is 4, call division procedure.
8) If choice is different the exit from the program.
9) Display the result as per user choice

Conclusion: Hence we done arithmetic operations 64 bit numbers by using switch case, macro and procedure in assembly language.

Sample Program :

```
section .data

choice: db 'The enter your choice: 1.Addition 2.Subtraction 3.Multiplication 4.Division'
choicelen: equ $-choice

msg: db 'The ddition is:'
len: equ $-msg

msg1: db 'The subtraction is :'
len1: equ $-msg1

msg2: db 'The multiplication is :'
len2: equ $-msg2

msg3: db 'The division is :'
len3: equ $-msg3

arr:dq 2222222222222222h,11111h,1111h,1111h,1111h
temp dq 0
segment .bss
count: resb 1
result: resb 17
result1: resb 17
result2: resb 17
result3: resb 17


%macro disp 2
mov rax,1
 mov rdi,1
 mov rsi,%1
 mov rdx,%2
syscall
%endmacro

section .text
```

```
global _start

_start:

disp choice, choicelen
  mov rax, 0
  mov rdi, 0
mov rsi, temp
mov rdx, 8
syscall

mov rcx,[temp]

cmp cl,31h
je ch1
cmp cl,32h
je ch2
cmp cl,33h
je ch3
cmp cl,34h
je ch4
jmp exit


ch1:call addition
jmp exit
ch2:call subtraction
jmp exit
ch3:call multiplication
jmp exit
ch4:call division
jmp exit


addition:
disp msg,len
mov byte[count],5
mov rax,0
mov rsi,arr

loop1:
add rax,[rsi]
add rsi,08
dec byte[count]
jnz loop1
```

```
mov rbx,rax
 mov rsi,result
 mov byte[count],10h
 mov cl,04

 call display
 disp result,17
ret

subtraction:
disp msg1,len1
mov byte[count],4
mov rsi,arr
mov rax,[rsi]
add rsi,08

loop2:
sub rax,[rsi]
add rsi,08
dec byte[count]
jnz loop2

mov rbx,rax
 mov rsi,result1
 mov byte[count],10h
 mov cl,04

 call display
 disp result1,17
ret

multiplication:

disp msg2,len2
mov byte[count],4
mov rsi,arr
mov rax,[rsi]
add rsi,08

loop3:
mov rcx,[rsi]
mul rcx
add rsi,08
dec byte[count]
jnz loop3
```

```
mov rbx,rax
mov rsi,result2
 mov byte[count],10h
 mov cl,04
call display
disp result2,17
ret

division:
disp msg3,len3
mov byte[count],4
mov rsi,arr
mov rax,[rsi]
add rsi,08

loop4:
mov rcx,[rsi]
div rcx
add rsi,08
dec byte[count]
jnz loop4

mov rbx,rax
 mov rsi,result3
 mov byte[count],10h
 mov cl,04
call display
 disp result3,17
ret

display:
l1:
rol rbx,cl
mov dl,bl
and dl,0fh
cmp dl,09h
jbe l2
add dl,07h
l2:
add dl,30h
mov [rsi], dl
inc rsi
dec byte[count]
jnz l1
mov byte[rsi], 0ah
ret
```

```
exit:
mov rax, 60
xor rdi, rdi
syscall
```

Sample Output :

1.Addition 2.Subtraction 3.Multiplication 4.Division

The enter your choice: 1

The addition is: 2222222222256666

The enter your choice: 2

The subtraction is :222222222220DDDE

Enter your choice: 3

The multiplication is :AAA7E7B58E4A7EA2

The enter your choice: 4

The division is : F17800772BD3474F

Expected Oral Questions:

1) How to scan user choice.
2) Explain MUL, DIV instructions.
3) What is meaning of CALL.
4) How to declare array of 64 bit numbers.

Assignment No. 09

Read command line arguments passed to a program.

AIM : Write ALP to read command line arguments passed to a program.

Theory:

In this program the user is going to enter the input string on command line. Suppose user has enter the string MA Lab, which is present on stack memory in following form :

arguments	MA Lab
.asm program name	Abc.asm
no. of arguments in the program	2
Current SP	

By accessing this stack we can get the actual arguments in the program. Then for displaying the arguments we have to calculate length whole string.

Instruction:

1) JA –

Description - jump if above condition satisfies to lable specified.

e. g. ja up

2)CALL -

Description- This instruction is going to call a procedure. The control of a program is transfered to the calling procedure.

e.g. call display

3) TEST –

Description- This instruction is going to make logical and of source and destination operand. The result of operation is reflected in flag.

Flag: ZF, CF, SF

e. g. test edi, edi

4)**REPNE SCASB**-

Description- repeat scanning of a string byte till the byte pointed by di is equal to value mentioned in al. This instruction also going to decrement the value in cl. The string to be scanned should be pointed by di.

Flag: SF, CF,OF, ZF,AF, PF

Algorithm :

1) Enter the string on command line
2) put a pointer to the stack
3) Get the count of no. of arguments from the stack.
4) If no any argument is entered then exit from program.
5) If number of arguments are more than 4 then display the message that too many arguments.
6) If arguments are within the rang i.e. 1 to 4 then calculate the length of entered string/ arguments, by accessing stack.
7) Display the entered string by putting pointer to the stack.

Conclusion – Thus the average of numbers is calculated through ALP.

Sample Program:

```
Section .data
ErrMsg  db “Too many arguments The max number of args is 4”,10
ERRLEN equ $-ErrMsg
Line db 32
SECTION  .text
Global _start
 _start:
Push ebp
Mov ebp, esp
Cmp dword [ebp+4]
Je NOArg
 Cmp dword [ebp+4], 5
 Ja TooManyArgs
Mov ebx, 3

DoNextArg:
Mov edi, dword [ebp+4*ebx]
Test edi, edi
Jz Exit
```

```
    Call Getsbrlen
    Push ebx
    Mov ecx, dword [ebp+4*exit]
    Call Display Norm

    Pop edi
    Inc ebx
    Jmp DoNextArg
 NoArgs:
    Jmp exitDisplay Norm:
   Push ebx
   Mov eax, 4
   Mov ebx, 1
   Mov ecx, Line
   Mov edx, 1
   Int 80h
   Pop ebx
   Ret
GetStrlen:   Push ebx
  Xor ecx, ecx
  Not ecx
  Xor eax, eax
  Cld
  Repne scasb
  Mov byte [edi-1], 10
  Not ecx
  Pop ebx
```

```
 Lea edx, [ecx-1]
 Ret

TooManyArgs:
Mov eax, 4
Mov ebx, 1
Mov ecx, ErrMsg
Mov edx, ERRLEN
Int 80h
Exit:
 Mov esp, ebp
 Pop ebp
 Mov eax, 1
 Mov ebx, 0
 Int 80h
```

Sample Output –

```
cl2@cl2-OptiPlex-390:~/Desktop$ nasm -f elf commline.asm
cl2@cl2-OptiPlex-390:~/Desktop$ ld -s -o commline commline.o
cl2@cl2-OptiPlex-390:~/Desktop$ ./commline  This is MA lab

Entered arguments are : This is MA lab

cl2@cl2-OptiPlex-390:~/Desktop$ ./commline  This is Microprocessor Architecture
lab
Too many arguments The max number of args is 4.
```

Expected Oral Questions - :

1) What is command line argument.
2) Explain Repne scasb, rep movsb, cld , Lea instructions.
3) Where the command line arguments are stored.
4) What is execution result of Mov edi, dword [ebp+4*ebx] instruction.
5) What happens with program, if more than 4 arguments are entered?

<u>Assignment No. 10</u>

Count positive and negative numbers from the array.

Aim: Write an ALP to count numbers of positive and negative numbers from the array.

Theory:

Consider that five numbers are stored in the array. In this program we count the numbers of positive and negative numbers from the given array. For this we use the instructions like the bt, jmp, inc etc.

Instructions:

1) **BT:**

Description: This instruction tests the status the specified bit in the instruction. The status of that bit is copied to the carry flag.

Flags: CF is set to the value of selected bit and OF, ZF, SF, AF and PF are undefined.

Example: BT EAX, 05

2) **INC**

Description: This instruction adds 1to the destination operand.

Flags: SF, PF, OF, ZF, AF are affected.

Example: INC CX

3) **JMP:**

Description: This instruction will cause the 80386 to fetch its next instruction from the location specified in the instruction rather than next instruction after JMP instruction.

Flags: No flags are affected.

Example: JMP WORD PTR [BX]

Algorithm:

1) Initialize array of five 32 bit numbers.
2) Put the pointer to the array.
3) Put a counter to 5.
4) Check the numbers one by one to count the positive and negative numbers in the array.

5) Display the count of the positive and negative numbers in the array.

   Repeat the same procedure for 64 bit numbers.

Conclusion:

Here we count the 32-bit numbers of positive and negative numbers in the array in the assembly language.

Sample Program:

```
section .data
        msg db 10,'count +ve and -ve numbers in an array',10
        msg_len equ $-msg

        pmsg db 10,'Count of +ve numbers::'
        pmsg_len equ $-pmsg

        nmsg db 10,'Count of -ve numbers::'
        nmsg_len equ $-nmsg

        nwline db 10

        array dw 8505h,90ffh,87h,88h,8a9fh,0adh,02h
        arrcnt equ 7

        pcnt db 0
        ncnt db 0

section .bss
        dispbuff resb 2

%macro  print   2
        mov   eax, 4
        mov   ebx, 1
        mov   ecx, %1
        mov   edx, %2
        int   80h
%endmacro

section .text
        global _start
_start:
```

```
        print msg,msg_len
        mov esi,array
        mov ecx,arrcnt
up1:
        bt word[esi],15
        jnc pnxt
        inc byte[ncnt]
        jmp pskip

pnxt:   inc byte[pcnt]

pskip:  inc esi
        inc esi
        loop up1

        print pmsg,pmsg_len
        mov bl,[pcnt]
        call disp8num
        print nmsg,nmsg_len
        mov bl,[ncnt]
        call disp8num

        print nwline,1          ;New line char
exit:
        mov eax,01
        mov ebx,0
        int 80h

disp8num:
        mov ecx,2               ;Number digits to display
        mov edi,dispbuff        ;Temp buffer
dup1:
        rol bl,4        ;Rotate number from bl to get MS digit to LS digit
        mov al,bl               ;Move rotated number to AL
        and al,0fh              ;Mask upper digit
        cmp al,09               ;Compare with 9
        jbe dskip               ;If number below or equal to 9 go to add only 30h
        add al,07h              ;Else first add 07h
dskip:  add al,30h              ;Add 30h
        mov [edi],al            ;Store ASCII code in temp buff
        inc edi                 ;Increment pointer to next location in temp buff
        loop dup1               ;repeat till ecx becomes zero

        print dispbuff,2        ;display the value from temp buff
        ret                     ;return to calling program
```

Sample Output -:

count +ve and -ve numbers in an array

Count of +ve numbers::04

Count of -ve numbers::03

Expected Oral Questions –

1) What is logic of program.
2) What is meaning of BT instruction.
3) What is ascii for newline.

Assignment No. 11

Find average of n numbers stored in memory

Aim : Write ALP to find average of n numbers stored in memory

Theory:

This program finds average of 5 numbers. To find the average we have to add all 5 numbers and then divide it by 5. To display the result we have to convert the hex number into ASCII.

Instruction:

1) DIV –

Description - It is arithmetic instruction that divide the contents in accumulator by source specified. After division quotient will go in accumulator and reminder will go in edx/dx/dl as per format.

e. g. div ebx

2)ROL -

Description- This instruction rotates the bits in destination to the left by count mentioned in source.

e.g. rol ebx,5

3) AND –

Description- This instruction performs logical and opration of bit in a source to bit in destination .

e. g. and cx,[si]

Algorithm :

1) Define array of five 32 bit numbers
2) display the message
3) Take a counter of 5
4) put a pointer to the array
5) Add all 5 numbers
6) divide the numbers by 5
7) Convert the hex result into ASCII by using display procedure as given below.

HEX to ASCII conversion procedure (display procedure):

1) Take counter of 8 for 8 nibbles in number.
2) Take another counter of 4.
3) Rotate the number by 4.
4) Mask the upper nibble of lower 8 bit number.
5) Check whether the unmasked nibble is less than or greater than 9.
6) Then do corresponding addition of 30 or 37 respectively.
7) Put that ASCII converted number into result variable.
8) Repeat the procedure for all nibbles, and store all numbers in result variables.

Conclusion – Thus the average of numbers is calculated through ALP.

Sample Program:

```
Section .data
msg: db 'The result is' ,10
len: equ $-msg
arr: dd 111111111h,22222222h,33333333h,44444444h,55555555h

Section .bss
count: resb 1
result: resb 10
input: resb 1

Section .text
global_start
_start:
mov eax, 4                  ; interrupt to display message
mov ebx, 1
mov ecx, msg
mov edx, len
int 80h

mov byte[count], 5          ;take a counter
mov eax, 0
mov esi, arr

loop1:
add eax, [esi]
add esi, 04
dec byte[count]             ; decrement count by 1
jnz loop1

mov ebx, 05h
mov edx, 00h
div ebx

mov ebx, eax
mov byte[count], 08
mov esi, result
mov cl, 04              ; load number of bit to rotate in cl

; convert HEX to ASCII
L1:
rol ebx, cl             ; rotate ebx 4 bit position (swap nibbles)
mov dl, bl                  ; after rotation load lower bit in dl
```

```
and  dl, 0fh                    ; mask  upper nibble
cmp  dl, 09h                    ; compare whether  the dl is below 09h or not
jbe  L2                         ; if below then jump to L2
add  dl, 07h                    ; if greater then add  the value with dl
L2:
add dl, 30h                     ; add the value with dl
mov  [esi], dl                  ; load dl in esi
inc  esi                        ; increment  esi  by  1
dec  byte[count]                ; decrement  count  by 1
jnz  L1
mov  byte[esi], 0ah

; display interrupt
mov  eax, 04
mov  ebx, 01
mov  ecx, result
mov  edx, 09
int  80h

; exit  interrupt
mov  eax, 1
mov  ebx, 0
int  80h
```

Sample Output :

```
$nasm  –f elf  average.asm
$ld   -s  -o  average  average.o
$./average
```

The result is: 33333333

Expected Oral Questions –

1) How to find average of numbers.
2) Explain Hex to ascii conversion procedure.
3) Explain JNZ, JBE instructions.

Assignment No. 12

Read and display contents of a file.

Aim : Write ALP to read and display contents of a file.

Theory:

In this program the user is going to enter the text file name on command line. Suppose user has enter file named abc.txt, which is present on stack memory in following form :

arguments	abc.txt
.asm program name	file.asm
no. of arguments in the program	1
Current SP	

By accessing this stack we can get the file name in the program. Then by calling the interrupt for opening the file and closing the file we can access the contents of a file. The file contents are taken into buffer memory for display.

Instruction:

1) POP –

Description - This instruction going to pop the contents from stack into destination mentioned in the instruction. The stack pointer is first incremented and then the contents are popped.

e. g. pop ebx

2)JNS -

Description- This instruction is going jump on label mentioned if sign flag is not set.
Flag: SF
e.g. jns up

3) JS –

Description- This instruction is going jump on label mentioned if sign flag is set.
Flag: SF
e. g. js up

4)JZ-

Description- This instruction is going jump on label mentioned if zero flag is set.
Flag: ZF
e. g. jz up

Algorithm:

1) Enter the file name on command line.
2) put a pointer to the stack
3) Get file name from the stack.
4) Call interrupt to open the file.
5) The file descriptor is now available in eax. Test that descriptor.
6) If file descriptor is not valid then close the file and exit from program.
7) If valid file descriptor then copy the file contents into buffer.
8) Again test file descriptor. If it returns null then display the contents from the buffer.
9) Close the file.
10) Exit from the program

Conclusion – Thus the contents of .txt file are displayed.

Sample Program:

```
Section .data
Msg: db 'The file contents are', 10
Len: equ $-msg

Msg1: db 'error'
Len1: equ $-msg

Segment .bss
File: resb 10
```

```
Buf: resb 1000

Segment .text
Global _start
_start:
Mov eax, 4
Mov ebx, 1
Mov ecx, msg
Mov edx, len
Int 80h

Pop ebx
Pop ebx
Pop ebx

Mov eax, 5
Mov ecx, 0
Int 80h

Test eax, eax
Jns file _function
Mov ebx, eax
Mov eax, 1
Int 80h

File _function:
Mov ebx, eax
Mov eax, 3
Mov ecx, buf
Mov edx, 1000
Int 80h

Test eax, eax
Jz exit
Js error

Mov edx, eax
 Mov eax, 4
 Mov ebx, 1
 Int 80h
 Jmp exit

Error:
  Mov eax, 4
  Mov ebx, 1
  Mov ecx, msg1
```

```
 Mov edx, len1
 Int 80h

Exit:
Mov eax, 6
Int 80h
Mov eax, 1
Mov ebx, 0
Int 80h
```

Sample Output -:

```
$ nasm –f elf  file.asm
$ ld   -s  -o  file file.o
$ ./file  abc.txt
```

The file contains are :
"Welcome to MA LAB"

Expected Oral Questions:

1) What is meaning of file descriptor.
2) What is meaning of Test, Jns instruction.
3) How assembler reads the file name form command line.
4) What interrupt is used to open the file.

Assignment No. 13

Switch from real mode and display the values of GDTR, LDTR, IDTR, TR and MSW register.

Aim: Write ALP to switch from real mode and display the values of GDTR, LDTR, IDTR, TR and MSW register.

Theory:

1.Real Mode:

Real mode is an operating mode of 8086 and later x86-compatible CPUs. Real mode is characterized by a 20 bit segmented memory address space (meaning that only 1 MB of memory can be addressed),direct software access to BIOS routines and peripheral hardware and no concept of memory protection or multitasking at the hardware level. All x86 CPUs in the 80286 series and later start up in real mode at power-on ;80186 CPUs and earlier had only one operational mode, which is equivalent to real mode in later chips.

In order to use more than 64 kB of memory, the segment registers must be used. This created great complications for compiler implementers who introduced odd pointer modes such as "near", "far" and "huge" to leverage the implicit nature of segmented architecture to different degrees, with some pointers containing 16-bit offsets within implied segments and other pointers containing segment addresses and offsets within segments.

2.protected Mode:

In addition to real mode, the Intel 80286 supports protected mode, expanding addressable physical memory to 16MB and addressable virtual memory to 1 GB and providing protected memory, which prevents programs from corrupting one another. T his is done by using the segment registers only for the sorting an index to a segment table. There were two such tables ,the Global Descriptor Table(GDT) and the Local Descriptor Table(LDT),each holding up to 8192 segment descriptors, each segment giving access to 64 KB of memory. The segment table provided a 24-bit base address ,which can be added to the desired offset to create an absolute address. Each segment can be assigned one of four ring levels used for hardware-based computer security. The Intel 80386 introduced support in protected mode for paging, a mechanism making it possible to use paged virtual memory.

64-bit:Processor are now sign extended to 64 starting with the AMD Opteron processor the x86 architecture extended the 32-bit registers into 64-bit registers in a way similar to how the 16 to 32-bit extension took place. An R-prefix identifies the 64-bit registers (RAX,RBX,RCX,RDX,RSI,RDI,RBP,RSP,RFLAGS,RIP),and eight additional 64-bit general registers (R8-R15) were also introduced in the creation of x86-64.However these extensions are only usable in 64-bit mode, which is one of the two modes only available in long mode. The addressing modes were not dramatically changed from 32-bit mode, except that addressing was extended to 64 -bits virtual addresses bits (in order to disallow mode bits in virtual addresses) and another selector details were dramatically reduced. In addition, an addressing mode was added to allow memory references relative to RIP (the instruction pointer) to ease the implementation of position-independent code, used in shared libraries in some operating systems.

32-bit:With the advent of the 32-bit 80386 processor, the 16-bit general-purpose registers, base registers, index registers, instruction pointer and FLAGS registers, but not the segment registers, were expanded to 32-bits.This is represented by prefixing an "E" (for Extended) to the register names in x86 assembly language, Thus the AX register corresponds to the lowest 16 bits of the new 32-bit EAX register, SI corresponds to the lowest 16 bits of ESI and so on. The general-purpose registers, base registers and the index registers can all be used as the base in addressing modes and all of those registers except for the stack pointer can be used as the index in addressing modes.

Conclusion:

Hence we successfully convert switch from real mode to a protected mode.

Sample Program:-

Section .data rmodemsg db 10,’Processor is in real mode’

Rmsg_len:equ $-rmodemsg

Pmodemsg db 10,’processor is in protected mode’

Pmsg_len:equ $-pmodemsg

Gdtmsg db 10,’GDT contents are::’

Gmsg_len:equ $-gdtmsg

Ldtmsg db 10,’LDT contents are::’

Lmsg_len:equ $-ldtmsg

Idtmsg db 10,’IDT contents are::’

Imsg_len:equ $-idtmsg

Mswmsg db 10,’Machine Status Word::’

Mmsg_len:equ $-mswmsg

Colmsg db ‘:’

Nwline db 10

Section .bss

Gdt resb 1

Resw 1

ldt resw 1

idt resd 1

resw 1

```
tr  resw 1
cr0_data resd 1
dnum_buff resb 04
%macro disp 2
Mov eax,04
Mov ebx,01
Mov ecx,%1
Mov edx,%2
Int 80h
%endmacro
Section .text
Global _start
_start:
Smsw eax
Mov [cr0_data],eax
Dt eax ,0
Mode
Jc prmode
Disp rmodemsg,rmsg_len
Jmp nxt1
Prmode : disp pmodemsg,pmsg_len
Nxt1: sgdt [gdt]
        Sldt [ldt]
        Sidt [idt]
        Spr [pr]
        Disp gdtmsg,gmsg_len
        Mov bx,[gdt+4]
```

```
        Call disp_num
        Mov bx,[gdt+2]
        Call disp_num
        Disp colmsg,1

        Mov bx,[gdt]
        Call disp_num
        Disp ldtmsg,lmsg_len
        Mov bx,[ldt]
        Call disp_num
        Disp idtmsg,imsg_len
        Mov bx,[idt+4]
        Call disp_num
        Mov bx,[idt+2]
        Call disp_num
        Disp colmsg,1
        Mov bx,[idt]
        Call disp_num
        Disp prmsg,tmsg_len
        Mov bx,[pr]
        Call disp_num
        Disp mswemsg,mmsg_len
        Mov bx,[cr0_data+2]
        Call disp_num
        Mov bx,[cr0_data]
        Cal disp_num
Exit: disp nwline,1
```

```
    Mov eax,01
    Mov ebx,00
    Int 80h

Disp_num:
    Mov esi,dnum_buff
    Mov  ecx,04
    Up1:
        Rol bx,04
        Mov dl,bl
        And dl,0fh
        Add dl,30h
        Cmp dl,39h
        Jbe skip1
        Add dl,07h
    Skip1:
        Mov [esi],dl
        Inc esi
        Loop up1
        Disp dnum_buff,4
        Ret
```

Sample Output:-

```
$ nasm –f elf64 gli.asm
$ld –o gli gli.o
$ ./gli
Processor is in Protected Mode
```

GDT Contents are::3F604000:007F

LDT Contents are::0000

IDT Contents are::81BDD000:0FFF

Task Register Contents are::0040

Machine Status Word::8005FFFF

Expected oral questions-

1) How to switch from real mode to protected mode.
2) What are contains of Machine Status Word.
3) What is use of Task Register?
4) What is meaning of DT, SIDT, SLDT, SGDT instructions.