Page No.
Date

Devanshu Surana
PC-23, Batch C1
1032210755

MAIoT Lab Assignment 9

Problem statement:

Write an ALP to display the contents of GDTR, IDTR, LDTR, TR and MSW

Objectives

1. Understand the concept of protected mode.
2. Understand the values of GDTR, LDTR, IDTR, TR and MSW Registers.

Theory:

1. Explain the following instructions used to read the content of respective registers.

a. SGDT <mem>:

The contents of the global descriptor table register is copied by SGDT to the six bytes of memory specified by operand. The first word at the Effective address is assigned the limit field of register. SGDT is not used in application programs, they are used in OS.

b. SIDT <mem>:

The contents of the interrupt descriptor table register is copied to SIDT. The instruction stores 8-byte base and 2-byte limit values.

c. SLDT <mem>:

SLDT stores the local descriptor table register (LDTR) in 2-byte Register or memory location indicated by the effective address operand. The registor is a selector that points into Global Descriptor table. SLDT is only used in Operating System software.

d. STR <mem>:

STR instruction store the task registers value into a memory. The memory address to load from or store to is at an offset from the register.

e. SMSW <mem>:

This instruction can be used to switch to protected mode. stores machine status word into destination operand. The SMSW instruction is only useful in Operating System software.

2. Draw the structure of the each registers. Write use of the registers GDTR, IDTR, LDTR and TR.

→ System Table Registers.

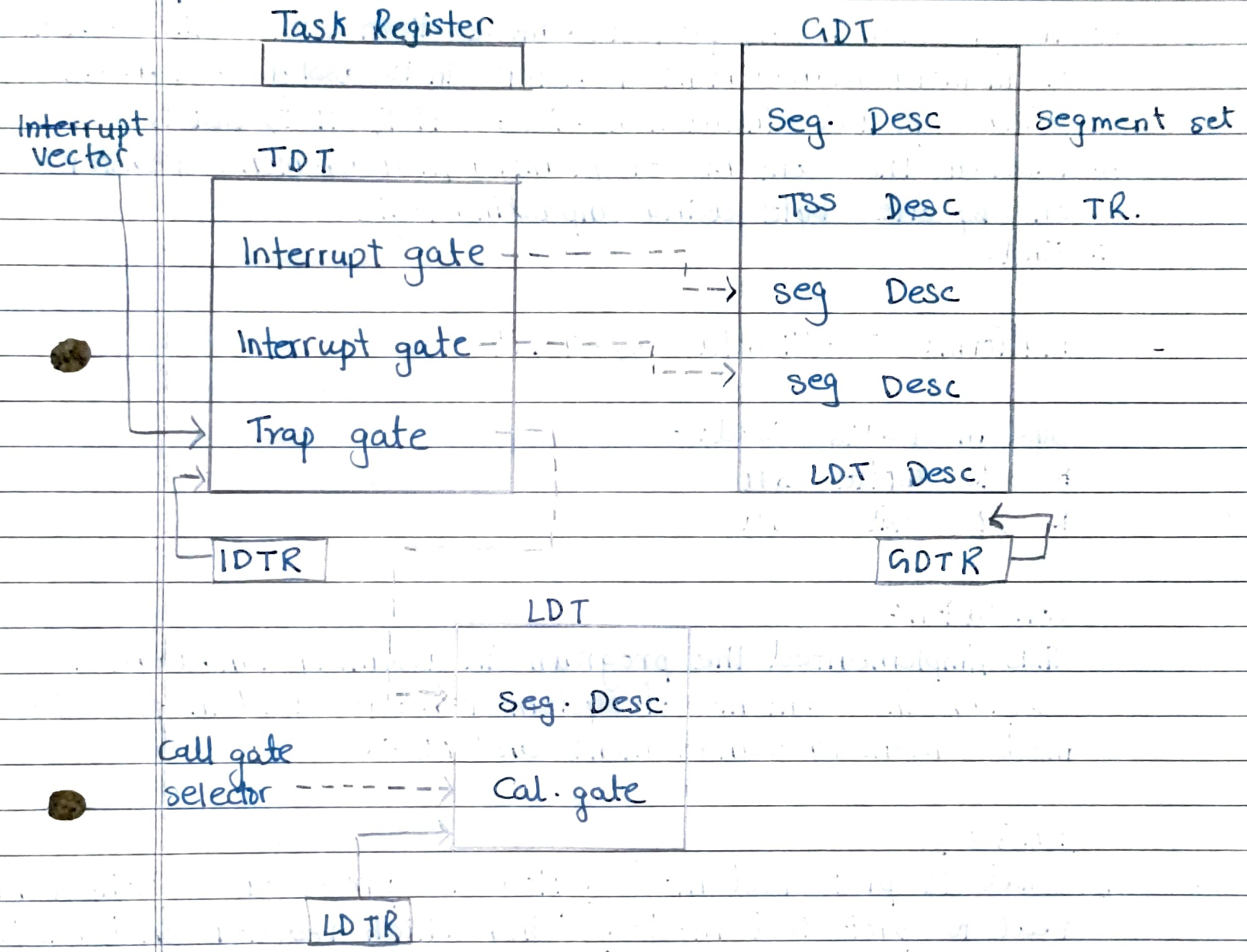
	47	16 15
GDTR	32-bit linear Base Address	16-bit Table limit
IDTR	32-bit linear Base Address	16-bit Table limit

	15	0
Task Register LDTR		Seg. sel seg. sel

Uses:

Holds the Base Address and loads new Base Address as parts of processor initialization process for protected mode operation.

3. Draw ~~dig~~ diagrammaticaly the tables where these registers pointed to.



Algorithm.

1. Initialize the gdtl and idtl variable by 6 byte and idtl and tl variable by 2 byte and mswl by 2 byte.
2. Store the msw or cr0 content in the accumulator.
3. Check the Bit0 of MSW or CR0 (PE = 0|1 - Protection Enable bit)

4. If it is set (1) then processor mode else it is reset (0) then processor is in real mode.
5. If processor is in protected mode then stored the contents of memory management register in to assigned variable using special instruction i.e: SGDT, SIDT, SLDT, STR.
6. Display the contents of memory management register one by one byte using unpacking logic
7. End.

Platform: OS - Ubuntu 16, 64-bit

System Calls Used:-
* SYS write call
* SYS exit call

Conclusion:
Thus, implemented the program in assembly language to display the contents of system register used in protected mode memory management and MSW.

FAQ's
1) What is protected mode? How is processor switched from real to protected mode? Using which register and which flag?
→ Protected mode is a mode of operation in modern X86 processors that provides hardware-enforced memory protection, virtual memory, and advanced features. The processor is switched from real to protected mode by loading the GDT address into the GDTR register, loading the code segment descriptor into the CR0 control register, and enabling interrupts.

Page No.
Date

2) What is MSW? Explain Bits present in MSW in brief.

→ The machine status word is stored by smsw in the two-byte register of memory location pointed to by the Effective Address operand.

8086 machines should use MOV---, CR0

Store the machine status word in the Effective Address SMSW s (ebx).

The machine status word consists of 4 flags - PE, MP, EM and TS of four lower order Bits. D19 to D16 of the upper word of the flag register.

PE - Protection Enable, switches processor between protected and real mode.

MP - Math present, Controls functions of wait.

EM - Emulation indicates whether co-processor function are emulated

TS - Task switched, Intrepreting co-processor instructions.

ET - Extension Type.

Next Bits are Reserved and the PG - Paging indicates whether processor uses page tables to translate linear addresses to physical Addresses.

3) Explain difference between Real Address mode and protected mode.

→ Real mode program uses BIOS subroutines along with OS subroutines whereas a 'protected mode' program uses only OS subroutines.

Instruction code differs since opcodes for registers are different and offset addresses of different ~~leg~~ length.

Name: Devasnhu Surana **Roll No.:** 23
Panel: C
Batch: C1

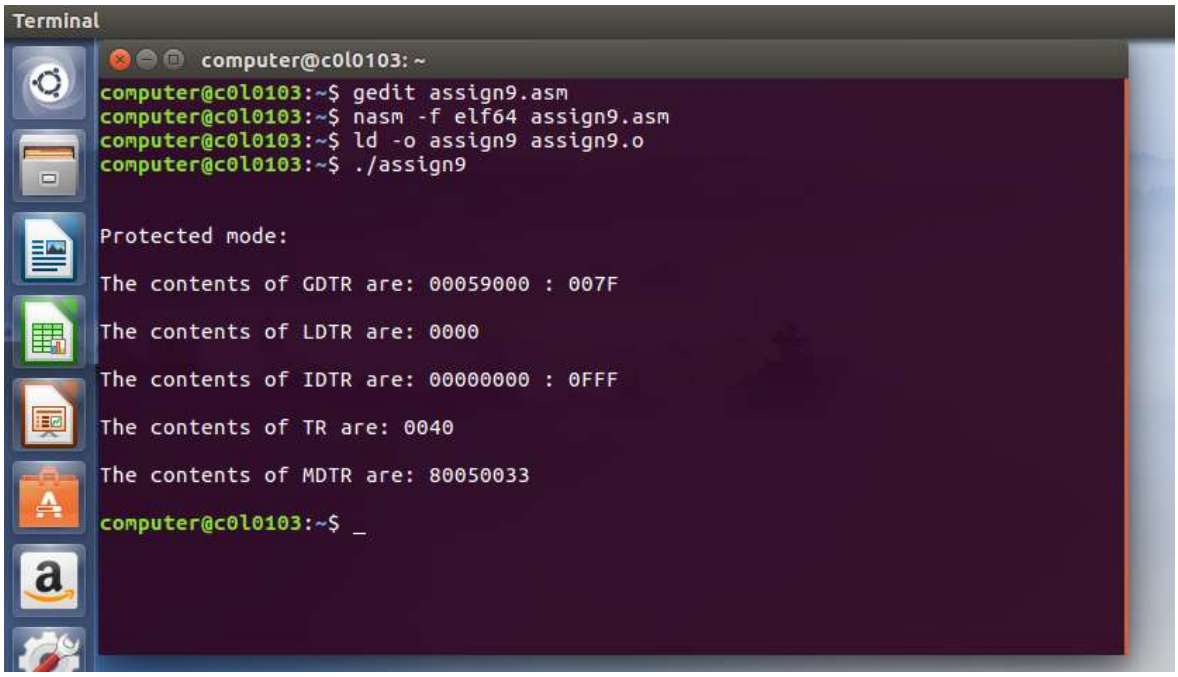
MAIoT Assignment 09

CODE:

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

section .data
gmsg db 10,10,"The contents of GDTR are: " gmsg_len equ $-gmsg

lmsg db 10,10,"The contents of LDTR are: " lmsg_len equ $-lmsg

imsg db 10,10,"The contents of IDTR are: " imsg_len equ $-imsg

tmsg db 10,10,"The contents of TR are: " tmsg_len equ $-tmsg

mmsg db 10,10,"The contents of MSW/CR0 are: " mmsg_len equ $-mmsg

pro db 10,10,"The processor is in protected mode " pro_len equ $-pro

real db 10,10,"The processor is in protected mode " real_len equ $-real

col db ":" col_len equ $-col

nline db 10,10 nlen equ $-nline

section .bss buff resb 4 gdt1 resb 6

idt1 resb 6 ldt1 resw 1 t1 resb 2

msw1 resb 4

section .text global _start _start:

smsw eax
mov [msw1],eax bt eax,0
jc protected
write real,real_len jmp end
```

```
protected:
write pro,pro_len sgdt [gdt1]
sldt [ldt1]
sidt [idt1]
str [t1]

write gmsg,gmsg_len mov bx,[gdt1+4]
call original_ascii mov bx,[gdt1+2]

call original_ascii write col,col_len mov bx,[gdt1] call original_ascii

write lmsg,lmsg_len mov bx,[ldt1]
call original_ascii

write lmsg,lmsg_len mov bx,[idt1+4] call original_ascii mov bx,[idt1+2]
call original_ascii write col,col_len mov bx,[idt1]

call original_ascii

write tmsg,tmsg_len mov bx,[t1]
call original_ascii

write mmsg,mmsg_len mov bx,[idt1+4]
call original_ascii
mov bx,[idt1+2]

call original_ascii

end:
write nline,nlen mov rax,60 mov rdi,0
mov rsi,0
mov rdx,0 syscall

original_ascii: mov rax,0 mov rcx,4

mov rdi,buff up2: rol bx,4 mov dl,bl

and dl,0fh cmp dl,09h jbe down2 add dl,07h

down2: add dl,30h mov [rdi],dl

inc rdi
loop up2 write buff,4
```

OUTPUT:


Terminal
computer@c0l0103: ~
computer@c0l0103:~$ gedit assign9.asm
computer@c0l0103:~$ nasm -f elf64 assign9.asm
computer@c0l0103:~$ ld -o assign9 assign9.o
computer@c0l0103:~$ ./assign9
Protected mode:
The contents of GDTR are: 00059000 : 007F
The contents of LDTR are: 0000
The contents of IDTR are: 00000000 : 0FFF
The contents of TR are: 0040
The contents of MDTR are: 80050033
computer@c0l0103:~$ _