DECEMBER 2018/ENDSEM U218-123(ESE)

S. Y. B. TECH. (PROGRAM) (SEMESTER - I)

COURSE NAME: COMPUTER ORGANIZATION AND MICROPROCESSORS TECHNIQUES

COURSE CODE: CSUA21173

(PATTERN 2017)

Marking Scheme

Q.1) a) Booth's Algorithm Flowchart for Two's Complement. [6 marks]

OR

b) A =1111 Q=0010 4 steps 1.5 Marks for each step [6 marks]

Q.2) a) i) Functions of I/O Modules. [2 marks]

ii) Diagram [2 marks]

iii) Description [2 marks]

OR

b) i) Diagram [3 marks]

ii) Description [3 marks]

Q.3)a) i) Diagram and Description of Data Flow Fetch cycle [3 marks]

ii) Diagram and Description of Data Flow Indirect cycle [3 marks]

OR

Q.3) b) i) Control Registers: [3 marks]

- Program Counter
- Instruction Register
- Memory Address Register
- Memory Buffer Register

ii) Status Registers [3 marks]

- Sign
- Zero
- Carry
- Equal
- Overflow
- Interrupt Enable/Disable
- Supervisor

Q.4) a) Diagram		[1 Marks]
Explanation		[3 Marks]

OR

Q.4) b) Match the pairs		[4 marks]
1- D, 2-A, 3-B, 4-C		

Q.5) a) Explanation		[3 Marks]
Diagram		[3 Marks]
b) Explanation		[4 Marks]
c) Explanation		[4 Marks]

OR

Q.6) a)) Explanation		[3 Marks]
Diagram of Segmentation		[3 Marks]
b) List		[1 Marks]
Explanation		[3 Marks]
c) Explanation	2 Marks each	[4 Marks]

Q.7) a) Explanation	2 Marks each instruction	[6 Marks]
b) 3 steps and Description		[4 Marks]
c)Data section		[1 Mark]
text section		[3 Mark]

OR

Q.8) a) Explanation	3 Marks each Solution	[6 Marks]
b) 1 mark for 1 Difference		[4 Marks]
c) c)Data section		[1 Mark]
text section		[3 Mark

8)

page 2

DECEMBER 2018/ENDSEM

S. Y. B. TECH. (PROGRAM) (SEMESTER - I)

COURSE NAME: COMPUTER ORGANIZATION AND MICROPROCESSORS TECHNIQUES

COURSE CODE: CSUA21173

(PATTERN 2017)

[illegible] ~~Marking Scheme~~ Solution

Q.1) a) Draw flow chart of Booth's Algorithm for Two's Complement Multiplication. **[6 marks]**

Solution:

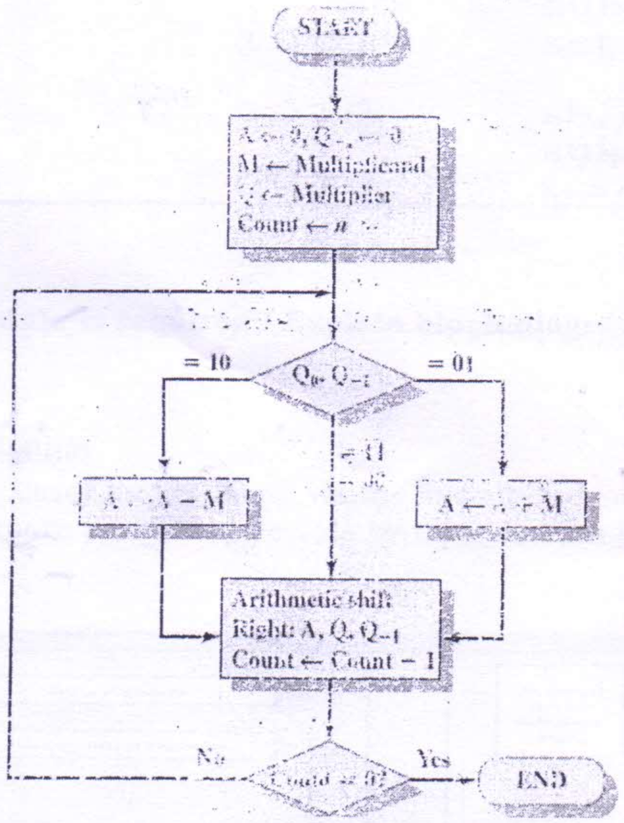


Figure 1a) Booth's Algorithm for Two's Complement Multiplication

OR

Q.1) a) Solve division of the following numbers using restoring division algorithm:

Dividend (A:Q) = 1111 : 1001 (-7)

Divisor (M)= 1101 (-3) **[6 marks]**

Solution:

A	Q	M = 1101
1111	1001	Initial value
1111	0010	shift
0010		subtract
1111	0010	restore
1110	0100	shift
0001		subtract
1110	0100	restore
1100	1000	shift
1111		subtract
1111	1001	set $Q_0 = 1$
1111	0010	shift
0010		subtract
1111	0010	restore

Q.2) a) Why I/O module is required? Explain block diagram of I/O Module. [6 marks]

Solution:

i) Functions of I/O Module

- Interface to the processor and memory via the system bus or central switch
- Interface to one or more peripheral devices by tailored data links

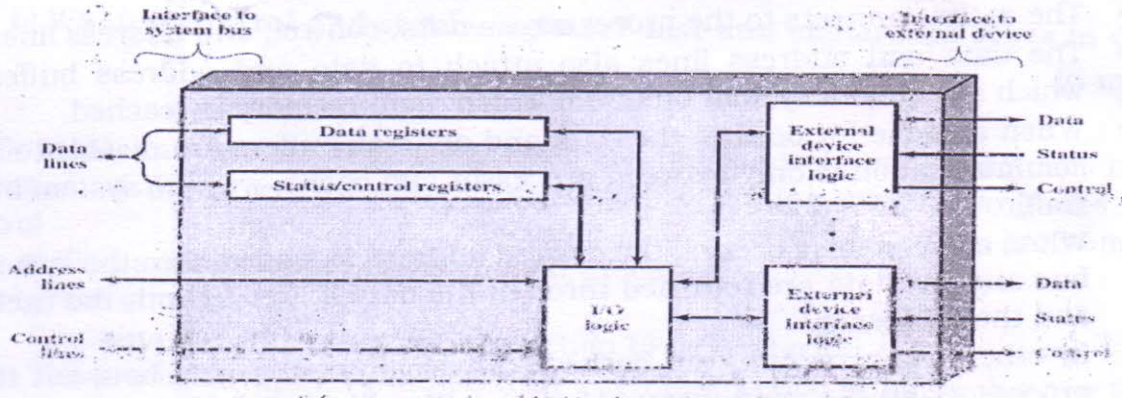


Figure. 2a) Block Diagram of an I/O Module

- The I/O module connects to the rest of the computer through a set of signal lines (e.g., system bus lines).
- Data transferred to and from the module are buffered in one or more data registers.
- There may also be one or more status registers that provide current status information.
- A status register may also function as a control register, to accept detailed control information from the processor.
- The logic within the module interacts with the processor via a set of control lines.
- The processor uses the control lines to issue commands to the I/O module.

- Some of the control lines may be used by the I/O module (e.g., for arbitration and status signals).
- The module must also be able to recognize and generate addresses associated with the devices it controls.
- Each I/O module has a unique address or, if it controls more than one external
  device, a unique set of addresses.
- Finally, the I/O module contains logic specific to the interface with each device
  that it controls.

OR

Q.2 b) Draw and Explain typical cache organization. [6 marks]

Solution :

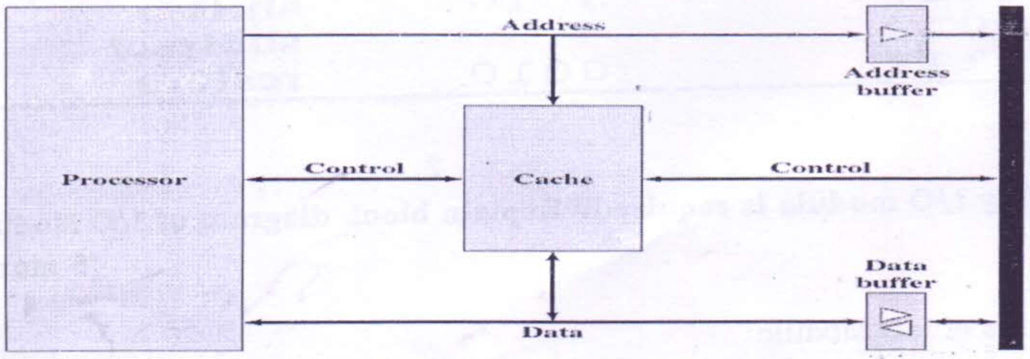
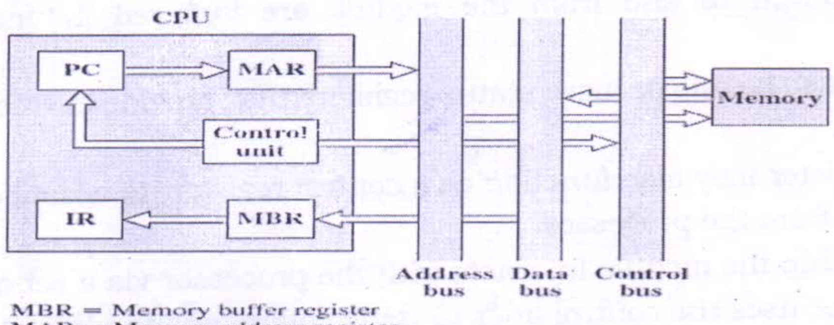


Figure 2b) Typical Cache Organization

- The cache connects to the processor via data, control, and address lines.
- The data and address lines also attach to data and address buffers, which attach to a system bus from which main memory is reached.
- When a cache hit occurs, the data and address buffers are disabled and communication is only between processor and cache, with no system bus traffic.
- When a cache miss occurs, the desired address is loaded onto the system bus and the data are returned through the data buffer to both the cache and the processor.
- In other organizations, the cache is physically interposed between the processor and the main memory for all data, address, and control lines.

Q.3) a) Draw and explain Data Flow Fetch Cycle and Data Flow Indirect Cycle. [6 marks]



MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Figure 3a)i) Data Flow Fetch Cycle

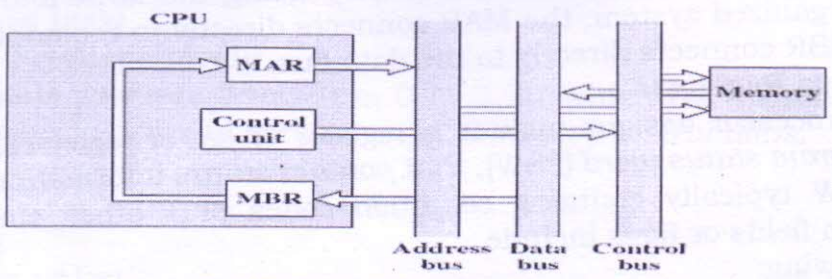


Figure 3a)ii) Data Flow Indirect Cycle

- During the *fetch cycle,* an instruction is read from memory.
- The PC contains the address of the next instruction to be fetched.
- This address is moved to the MAR and placed on the address bus.
- The control unit requests a memory read, and the result is placed on the data bus and copied into the MBR and then moved to the IR.
- Meanwhile, the PC is incremented by 1, preparatory for the next fetch. Once the fetch cycle is over, the control unit examines the contents of the IR to determine if it contains an operand specifier using indirect addressing.
- If so, an *indirect cycle* is performed.
- The rightmost N bits of the MBR, which contain the address reference, are transferred to the MAR.
- Then the control unit requests a memory read, to get the desired address of the operand into the MBR.

OR

Q.3 b) What is control and status register? List and explain its types in detail. [6 marks]

A) Control and status registers: Used by the control unit to control the operation of the processor and by privileged, operating system programs to control the execution of programs.
Four registers are essential to instruction execution:

1) **Program counter (PC):**
   - Contains the address of an instruction to be fetched.
   - The processor updates the PC after each instruction fetch so that the PC always points to the next instruction to be executed.
   - A branch or skip instruction will also modify the contents of the PC.
2) **Instruction register (IR):**
   - Contains the instruction most recently fetched.
   - The fetched instruction is loaded into an IR, where the opcode and operand specifiers are analyzed.
3) **Memory address register (MAR):** Contains the address of a location in memory
4) **Memory buffer register (MBR):**
   - Contains a word of data to be written to memory or the word most recently read

- Data are exchanged with memory using the MAR and MBR. In a bus-organized system, the MAR connects directly to the address bus, and the MBR connects directly to the data bus.

B) Status Registers :

Many processor designs include a register or set of registers, often known as the *program status word* (PSW), that contain status information.

The PSW typically contains condition codes plus other status information. Common fields or flags include
the following:

• **Sign:** Contains the sign bit of the result of the last arithmetic operation.

• **Zero:** Set when the result is 0.

• **Carry:** Set if an operation resulted in a carry (addition) into or borrow (subtraction)
out of a high-order bit. Used for multiword arithmetic operations.

• **Equal:** Set if a logical compare result is equality.

• **Overflow:** Used to indicate arithmetic overflow.

• **Interrupt Enable/Disable:** Used to enable or disable interrupts.

• **Supervisor:** Indicates whether the processor is executing in supervisor or user mode. Certain privileged instructions can be executed only in supervisor mode, and certain areas of memory can be accessed only in supervisor mode.

Q. 4) a) Draw with neat Diagram and Explain 80386 Flag register. [4marks]
Solution:

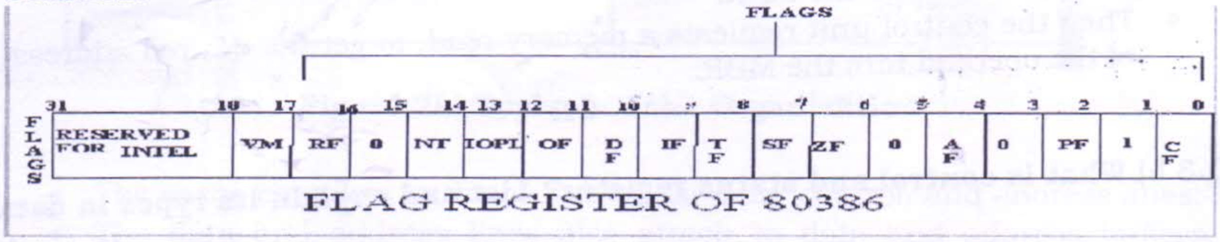


Figure 4) a) Flag Register of 80386

•<u>Flag Register of 80386</u>: The Flag register of 80386 is a 32 bit register. Out of the 32 bits, Intel has reserved bits D18 to D31, D5 and D3, while D1 is always set at 1.Two extra new flags are added to the 80286 flag to derive the flag register of 80386. They are VM and RF flags

- VM - Virtual Mode Flag:
  - ❑ If this flag is set, the 80386 enters the virtual 8086 mode within the protection mode.
  - ❑ This is to be set only when the 80386 is in protected mode.
  - ❑ In this mode, if any privileged instruction is executed an exception 13 is generated.
  - ❑ This bit can be set using IRET instruction or any task switch operation only in the protected mode.

•RF- Resume Flag:
- ❑ This flag is used with the debug register breakpoints.
- ❑ It is checked at the starting of every instruction cycle and if it is set, any debug fault is ignored during the instruction cycle.
- ❑ The RF is automatically reset after successful execution of every instruction, except for IRET and POPF instructions.

- **NT (Nested flag):**

   - This flag is set when one system task invokes another task. (i.e. nested task).
- **IOPL (I/O Privilege level):**
   - It holds privilege level, from 0 to 3, at which the current code is running in order to execute any I/O related instructions.

OR

Q.4)b) Match the pairs: **[4 marks]**
Solution:

1. Immediate Addressing	D. MOV AX,2387H
2. Based Indexed with Displacement Mode	A. MOV AX, [BX+DI+08]
3. Register Indirect Addressing Mode	B. MOV CX, [BX]
4. Direct Addressing Mode	C. ADD AX,[1592H]

Q. 5) a) Draw neat diagram to convert linear address to physical address using paging. **[6 marks]**
Solution:

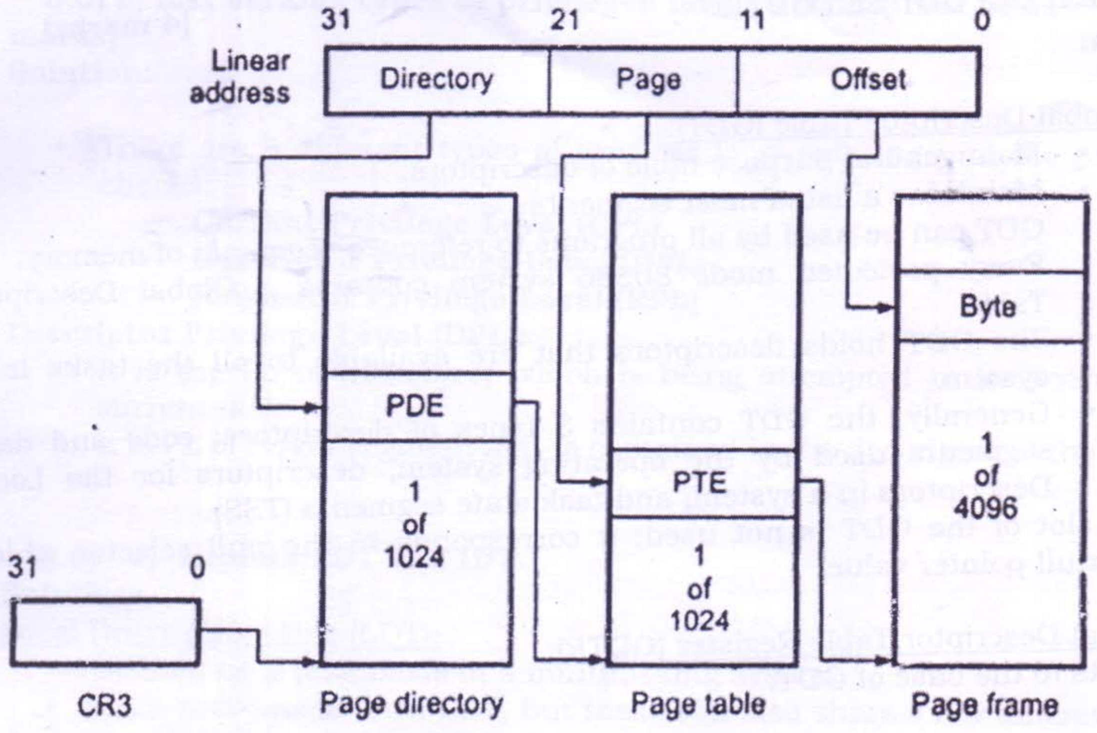


Figure 5)a) Paging

Q.5) b) Explain protection in 80386 in detail. **[4 marks]**
Solution:

- 80386 DX has four levels of protection which isolate and protect user programs from each other and the operating system.
- It offers an additional type of protection on a page basis, when paging is enabled(using U/S and R/W fields)
- The four-level hierarchical privilege system is illustrated as follows:

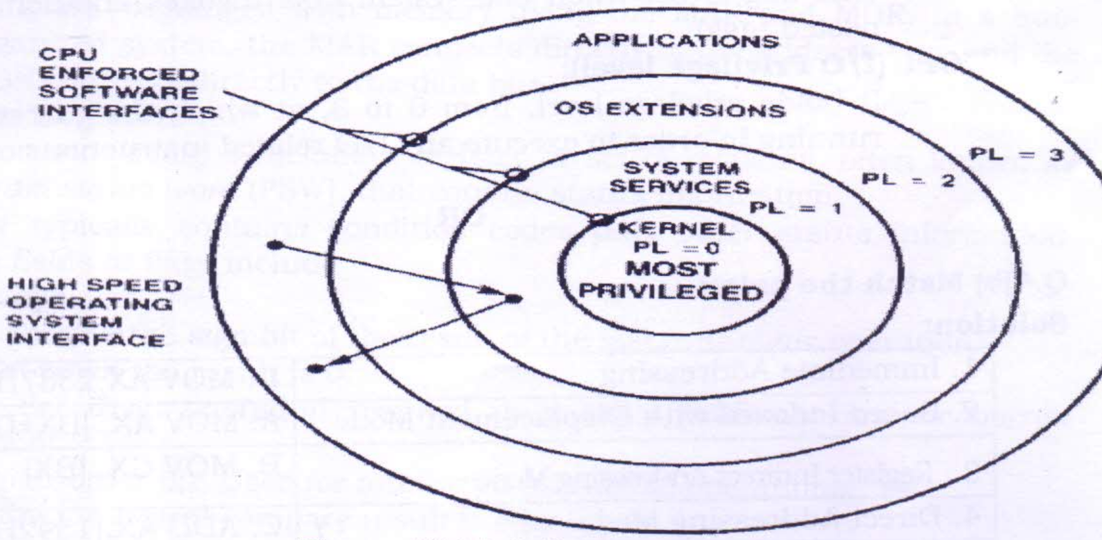


Figure 5)b) Privilege Levels

- The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged or trusted level

Q.5) c) Explain GDT and GDTR. **[4 marks]**
Solution:

1) Global Descriptor Table (GDT):
    - Main general purpose table of descriptors.
    - Maintains a list of most segments
    - GDT can be used by all programs to reference segments of memory.
    - Every protected mode 80386 system contains a Global Descriptor Table.
    - The GDT holds descriptors that are available to all the tasks in a system.
    - Generally, the GDT contains 3 types of descriptors: code and data segments used by the operating system, descriptors for the Local Descriptors in a system, and task state segments (TSS).

The first slot of the GDT is not used; it corresponds to the null selector which defines a null pointer value.

2)Global Descriptor Table Register (GDTR):
It points to the base of GDT.

OR

Q.6) a) Draw neat diagram to convert logical address to linear address using Segmentation. **[6 marks]**
Solution:

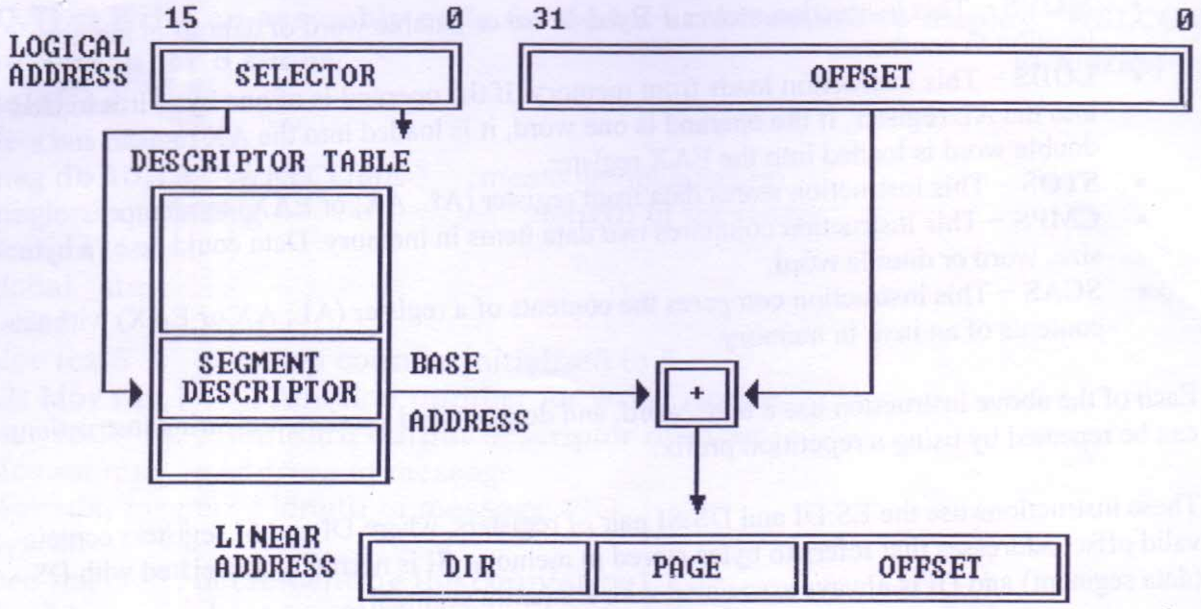


Figure 6)a) Segmentation

Q.6) b) List various types of privileged levels and Explain DPL. [4 marks]

Solution:

- There are 3 different types of privilege level entering into the privilege level checks:
  - **Current Privilege Level (CPL)**
  - **Descriptor Privilege Level (DPL)**
  - **Requestor Privilege Level (RPL)**

Descriptor Privilege Level (DPL):

- It is the PL of the object which is being attempted to be accessed by the current task
- It is PL of target segment and is contained in the descriptor of the segment

Q.6) c) Explain LDT and IDT. [4 marks]

Solution:

Local Descriptor Table (LDT):

- Defined on a task basis in a multitasking system.
- Each task has its own LDT, but tasks can also share a few different LDTs.

Interrupt Descriptor Table(IDT):

- It defines interrupt or exception handling routine.
- It is a direct replacement for the interrupt vector table used in 8086 systems.
- Exactly one GDT and one IDT must be defined for the 80386 to operate in protected mode.

Q.7) a) Explain Any 3 String Instructions with example. [6 marks]

Solution:

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

- **MOVS** – This instruction moves 1 Byte, Word or Double word 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 double word 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 double word.
- **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 double word 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.

The following table provides various versions of string instructions and the assumed space of the operands.

Basic Instruction	**Operands at**	**Byte Operation**	**Word Operation**	**Double word Operation**
MOVS	ES:DI, DS:SI	MOVSB	MOVSW	MOVSD
LODS	AX, DS:SI	LODSB	LODSW	LODSD
STOS	ES:DI, AX	STOSB	STOSW	STOSD
CMPS	DS:SI, ES: DI	CMPSB	CMPSW	CMPSD
SCAS	ES:DI, AX	SCASB	SCASW	SCASD

Q.7)b) Write the steps for executing an 64 bit assembly program on NASM. [4 marks]

Solution:

There are 3 steps

1) Assembling the code to remove syntax errors
   nasm -f elf64 prog.asm
2) Linking step to link all modules
   ld -o prog prog.o
3) Execution step to run the code
   ./prog

Q.7) c) Write an assembly code for 64 bit architecture to display "WELCOME" on Screen for 5 times. [4 marks]

Solution:

```
Section .data
msg db 10,13, " WELCOME "   ;message declaration
msglen equ $-msg              ; length of message
Section .text
global _start
_start:
Mov rcx,5         ; loop counter initialized to 5
L1: Mov rax,1       ; function number for write  device
Mov rdi,1        ; Standard output descriptor number
Mov rsi,msg   ; Address of message
Mov rdx, msglen  ; length of message
Syscall
Dec rcx        ; decrementing the counter by1
Jnz L1         ; checking for termination condition
Mov rax,60    ; function number to exit
Mov rdi,0       ; Standard input descriptor number
Syscall
```

OR other solution is without using loop

OR

Q.8) a) Assuming the following values in the register BX= 38 and CX= 5AH, show the contents of the register after following Operation ROL BX,4 and SHL CX,4. Justify your answer. [6 marks]

Solution:

BX= 38H	0011 1000	Initial value
	0111 0000	ROL by 1 bit
	1110 0000	ROL by next 1 bit
	1100 0001	ROL by next 1 bit
BX=83H	1000 0011	ROL by next 1 bit(final value)
CX= 5AH	0101 1010	Initial value
	1011 0100	SHL by 1 bit
	0110 1000	SHL by next 1
	1101 0000	SHL by next 1
CX=A0H	1010 0000	SHL by next 1 bit(final value)

Q.8) b) Differentiate between macro and procedure. [4marks]
Solution:

Procedure	Macro
Procedures are used for large group of instructions to be repeated.	Procedures are used for small group of instructions to be repeated.
Object code is generated only once in memory.	Object code is generated everytime the macro is called.
CALL & RET instructions are used to call procedure and return from procedure.	Macro can be called just by writing its name.
Length of the object file is less	Object file becomes lengthy.
Directives PROC & ENDP are used for defining procedure.	Directives MACRO and ENDM are used for defining MACRO
More time is required for it's execution	Less time is required for it's execution
Procedure can be defined as Procedure_name PROC ---- ------ Procedure_name ENDP	Macro can be defined as MACRO-name MACRO [ARGUMENT ,......... ARGUMENT N] ---- ------- ENDM
For Example Addition PROC near ------ Addition ENDP	For Example Display MACRO msg --------- ENDM

Q.8)c) Write an 64 bit assembly program to add two numbers assuming the numbers are in rax and rbx register (include comments in the code).[4marks]
Solution:

```
%macro scall 4
MOV rax,%1
MOV rdi,%2
MOV rsi,%3
MOV rdx,%4
syscall
%endmacro

section .data
msg2 db 10,13, "the sum is : "    ;message declaration
msg2len equ $-msg2      ; length of message

section .bss
res resb 16

section .text
global _start
_start :

mov rax,[num1]    ;Assuming rax and rbx has two nos. num1 and num2
mov rbx,[num2]
add rbx,rax              ; adding the nos n result in rbx
scall 1,1,msg2,msg2len    ; Display message
call display        ; calling display procedure
```

```
mov rax,60      ; exit
mov rdi,0
syscall
display :
        mov rdi,res    ; pointing to res
        MOV rcx,16     ; counter for looping
UP1 : ROL rbx,4             ; extracting a digit
             MOV al,bl
             AND al,0FH
             CMP al,09H      ; convert to ASCII HEX from HEX
             JBE L2
             ADD al, 07H
        L2: ADD al,30H
           mov [rdi],al        ; storing in res variable
           INC rdi
           DEC rcx
           JNZ UP1
         scall 1,1,res,16   ; calling the macro and displaying result
        ret                    ; return to procedure
```