**Assignment No.6**

|  |
| --- |
| **Title of Assignment:**  To switch from real mode to protected mode and display the values of GDTR, LDTR,  IDTR, TR and MSW register. |
| **Relevant Theory:**  **Global Descriptor Table Register (GDTR):**    The contents of the global table register define a table in the 80386DX's physical memory address space called the Global Descriptor Table (GDT). This global descriptor table is one important element of the 80386DX's memory management system. GDTR is a 48-bit register that is located inside the 80386DX. The lower two bytes of this register, which are identified as LIMIT, specify the size in byte of the GDT. The decimal value of LIMIT is one less than the actual size of the table. For instance, if LIMIT equals 00FFh the table is 256 bytes in length. Since LIMIT has 16 bits, the GDT can be up to 65,536 bytes long. The upper four bytes of the GDTR, which are labeled BASE, locate the beginning of the GDT in physical memory. This 32-bit base address allows the table to be positioned anywhere in the 80386DX's address space.  **Interrupt Descriptor Table Register (IDTR):**    The interrupt descriptor table register (IDTR) defines a table in physical memory. However, this  table contains what are called interrupt descriptors, not segment descriptors. Therefore it is  known as the Interrupt Descriptor Table (IDT).The IDTR is 48 bits in length.The lower two bytes of the register (LIMIT) define the table size. That is, the size of the table equals LIMIT+1 bytes. Since two bytes define the size, the IDT can also be up to 65,536 bytes long. But the 80386DX only supports up to 256 interrupts and exceptions; therefore, the size ofthe IDT should not be set to support more than 256 interrupts  **Local Descriptor Table Register (LDTR):**    The Local Descriptor Table Register (LDTR) is also part of the 80386DX's memorymanagement support mechanism. Each task can have access to its own private table descriptortable in addition to the global descriptor table.This private table is called the local descriptor table (LDT) and defines a local memory addressspace for use by the task. The LDT holds segment descriptors that provide access space for useby the task. The LDT holds segment descriptors that provide access to code and data in segmentsof memory that are reserved for the current task  **Control Registers:**    **Task Register (TR):**    **Instruction Description Mode**  **LGDT S:-** Load the global descriptor table register. S specifies both the memory location that  contains the first byte of the 6 bytes to be loaded into the GDTR.  **SGDT D:-** Store the global descriptor table register. D specifies both the memory location  that gets the first of the six bytes to be stored from the GDTR.  **LIDT S: -** Load the interrupt descriptor table register. S specifies both the memory location  that contains the first byte of the 6 bytes to be loaded into the IDTR.  **SIDT D:-** Store the interrupt descriptor table register. D specifies both the memory location  that gets the first of the six bytes to be stored from the IDTR.  **Algorithm :**  1) Start  2) Variable declaration in data section with initialization  3) Variable bss. section without initialization  4) Macro definition for display msg on screen  5) Read CRo  6) If PE beat =1  7) Store contains of GDT  8) Store contains of LDT  9) Store contains of IDT  10) Store contains of TR  11) Call display processor to display control of GDT  12) Call display processor to display contain of LDT  13) Call display processor to display contain of IDT  14) Call display processor to display control of TR  15) Call display processor to display control of MSW  16)Point to esi buffer  17)Load no. of digit to display  18) Rotate no. left by 4 bit  19) Move lower byte in DL  20) Mask upper digit of byte in DL  21) Add 30h to calculate ASCCI code  22) If DL < 39 , no add 7, yes Skip adding 07 more  23) Store ASCCI code in buffer  24) Point to next byte  25)Display the no. from buffer  26) END |
| **Conclusion:**  String manipulation operations like concatenation and finding no. of occurrences of substring is implemented and executed successfully. |