easy-solutions®

New Edition SPPU

As per the New Revised Syllabus (2019 course)
of Savitribai Phule Pune University w.e.f. academic year 2020-2021

MICROPROCESSOR

(Code : 210254)

"QUICK READ SERIES"

Semester IV
Computer Engineering

Chapterwise Solved University Paper Solution
For End Semester Examination

TechKnowledge™ Publications

easy-solutions

easy – solutions

Savitribai Phule Pune University

As per New Credit System Syllabus(Rev. 2019) of Savitribai Phule Pune University with effective from Academic Year 2020-2021

Microprocessor

(Code : 210254)

"Quick Read Series"

Semester IV - Computer Engineering

TechKnowledge™ Publications

EPE124A Price ₹ 90/-

SYLLABUS

In-Sem. Exam

Unit I : Introduction to 80386 — 07 Hours

Brief History of Intel Processors, 80386 DX Features and Architecture, Programmers Model, Operating modes, Addressing modes and data types.

Applications Instruction Set : Data Movement Instructions, Binary Arithmetic Instructions, Decimal Arithmetic Instructions, Logical Instructions, Control Transfer Instructions, String and Character Transfer Instructions, Instructions for Block Structured Language, Flag Control Instructions, Coprocessor Interface Instructions, Segment Register Instructions, Miscellaneous Instructions.

Unit II : Bus Cycles and System Architecture — 07 Hours

Initialization - Processor State after Reset. Functional pin Diagram, functionality of various pins, I/O Organization, Memory Organization (Memory banks), Basic memory read and writes cycles with timing diagram.

Systems Architecture - Systems Registers (Systems flags, Memory Management registers, Control registers, Debug registers, Test registers), System Instructions.

End-Sem. Exam

Unit III : Memory Management — 08 Hours

Global Descriptor Table, Local Descriptor Table, Interrupt Descriptor Table, GDTR, LDTR, IDTR. Formats of Descriptors and Selector, Segment Translation, Page Translation, Combining Segment and Page Translation. Home

Unit IV : Protection — 08 Hours

Need of Protection, Overview of 80386DX Protection Mechanisms: Protection rings and levels, Privileged Instructions, Concept of DPL, CPL, RPL, EPL.

Inter privilege level transfers using Call gates, Conforming code segment, Privilege levels and stacks. Page Level Protection, Combining Segment and Page Level Protection.

Unit V : Multitasking and Virtual 8086 Mode — 08 Hours

Multitasking - Task State Segment, TSS Descriptor, Task Register, Task Gate Descriptor, Task Switching, Task Linking, Task Address Space. Virtual Mode – Features, Memory management in Virtual Mode , Entering and leaving Virtual mode

Unit VI : Interrupts, Exceptions, and Introduction to Microcontrollers — 07 Hours

Interrupts and Exceptions: Identifying Interrupts, Enabling and Disabling Interrupts, Priority among Simultaneous Interrupts and Exceptions, Interrupt Descriptor Table (IDT), IDT Descriptors, Interrupt Tasks and Interrupt Procedures, Error Code, and Exception Conditions.

Introduction to Microcontrollers: Architecture of typical Microcontroller, Difference between Microprocessor and Microcontroller, Characteristics of microcontrollers, Application of Microcontrollers.

❑❑❑

Table of Contents

❑❑❑

Savitribai Phule Pune University
Semester IV (Computer Engineering)

Microprocessor

Unit III : Memory Management

Chapter 5 : Memory Management

Q. 1 Draw and explain how 80386 Microprocessor translates Logical address into Linear address.

SPPU - Nov. 12, 10 Marks

OR With the help of diagram explain the 80386 mechanism to translate logical address to linear address.

SPPU - May 19, Dec. 19, 3 Marks

Ans. : The logical (virtual) address is converted to linear address with the help of segmentation mechanism; while the linear address is converted to physical address with the help of paging mechanism. This address conversion flow can be as shown in the Fig. 5.1.

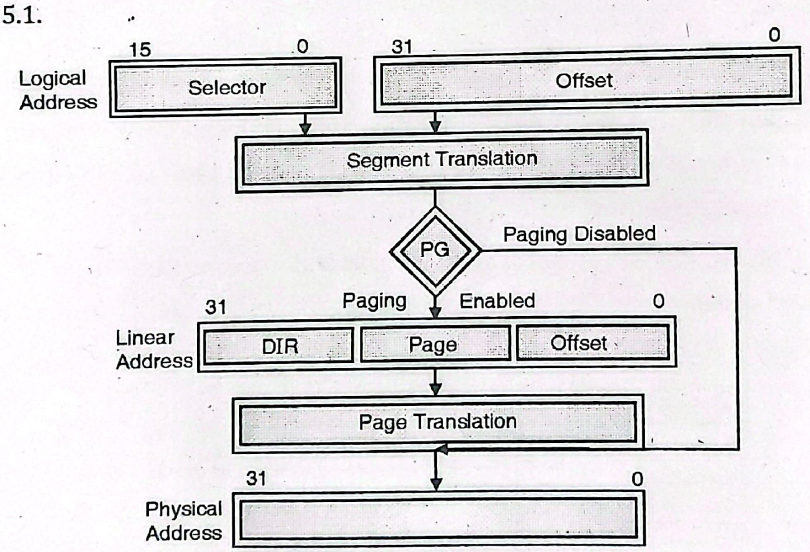

Fig. 5.1 : Virtual to physical address translation in 80386

Hence memory management includes segmentation and banking in 80386.

Segmentation in Protection Mode Logical to Linear Address Translation)

The segment size in the protection mode for x86 can be anything between 1 byte to 4 GB (beyond 4MB, it can be of different sizes in multiples of 4KB). Also the segment base address can start from anywhere removing the constraint (which was there in 8086) of last hex. digit to be '0'. Hence the entire memory is accessible in protection mode and the memory beyond 1MB is called as extended memory. The logical address in protected mode architecture is formed out of two components :

(a) The 16-bit selector (segment register) that is used to determine the descriptor.

(b) 32-bit offset to give the internal address within the segment.

SPPU - Dec. 17, May 18, 4 Marks

SPPU - May 18, 6 Marks

Q. 2 Explain paging mechanism.

OR Draw and explain structure of the TLB.

OR Explain how linear address is converted into physical address by 80386 memory management.

SPPU - Dec. 18, 6 Marks

OR With the help of diagram explain the 80386 mechanism to translate linear to physical address.

SPPU - Nov. 12, Dec. 19, 3 Marks

Ans. :

- 80386 / Pentium processor use the virtual memory system with the help of on-chip Memory Management Unit (MMU) i.e. paging mechanism.
- The MMU considers the memory divided into pages of 4KB each. Hence the 4GB memory is divided into 1 M pages of 4 KB each. To select one of these 1 M pages, 20 bits are required.
- CR3 consists of 20-bit page directory base address that selects a page, called as page directory. This page is also of 4 KB that has 32-bit (i.e. 4 Byte) entry, and hence has 1 K entries, called as page directory entries.
- To select one of these 1 K entries, 10 bits are required. Hence the first 10-bits of the linear address i.e. bit 31 to bit 22, select one of the 1 K entries. This entry in page directory (32-bit entry), has 20-bit that selects another page, called as page table.
- Since there are 1 K entries in the page directory each of them selecting a page table, there can be 1 K page tables.
- The page table size is also 4 KB that has 32-bit (i.e. 4 Byte) entry, and hence has 1 K entries. To select one of these 1 K entries, 10 bits are required. Hence the next 10-bits of the linear address i.e. bit 21 to bit 12, select one of the 1 K entries, called as page table entries.
- This entry in a page table (32-bit entry), has 20-bit that selects the user required page. This page is also of 4 KB. The required memory operand from this page is selected by the last 12-bits of the linear address i.e. bit 11 to bit 0. This paging mechanism is shown in the Fig. 5.2(a).

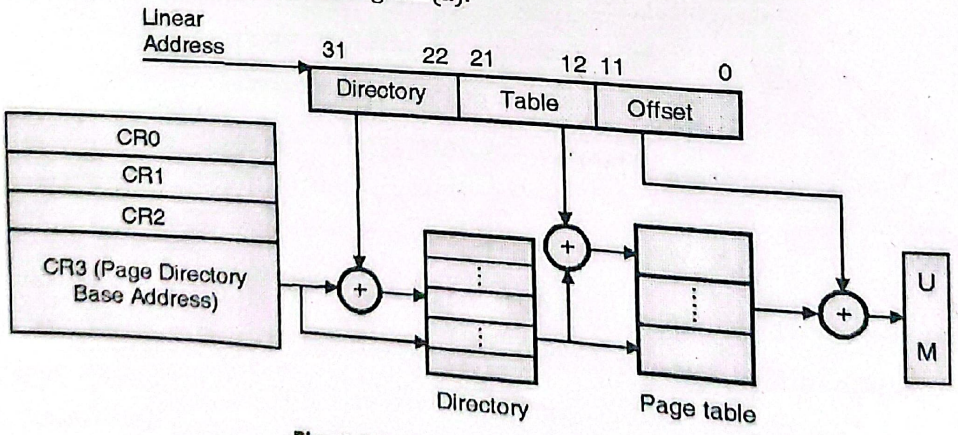

Fig. 5.2(a): Paging mechanism

The page table or page directory entry is shown in Fig. 5.2(a). The different bits in these entries are as follows:

- Bit 31-12, Page frame address for page table entry, while Page table address for page directory entry.
- Bit 11-9, are reserved for Operating System (OS). The OS can use it for its purpose, storing some information related to this page.

- Bit 8 and 7, are zeroes, are unused.
- Bit 6, D (Dirty); It is set before anything is written page directory. It indicates that this page has some updated data in the physical memory, while the virtual memory has this page not updated or stale data.
- Bit 5, A (Accessed); It indicates that this page is accessed previously or not.
- Bit 4, PCD (Page Cache Disable); As in CR0
- Bit 3, PWT (Page Write Through); As in CR0
- Bit 2, U/S (User/Supervisor); This bit differentiates between user and supervisor privileges.
- Bit 1, R/W (Read/Write); It indicates read or write protection for the page.
- Bit 0, P (Present); It indicates whether entry in the PDE and PTE (page Directory Entry and Page Table Entry) can be used. It indicates if the entry is initialized or not.

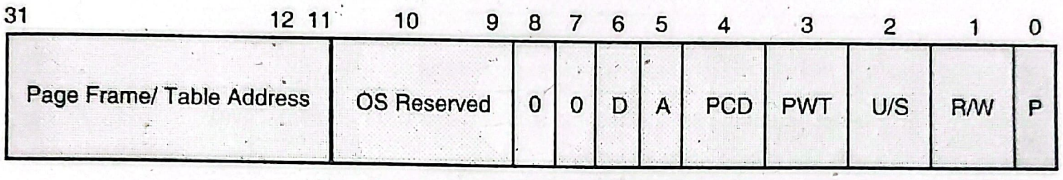

Fig. 5.2(b) : Page table/ directory entry

Translational Lookaside Buffer (TLB) is implemented in the memory management system, which reduces the memory access time, by giving the physical address without undergoing the paging mechanism. TLB is implemented in the 80386 processor whose structure is as shown in Fig. 5.2(c).

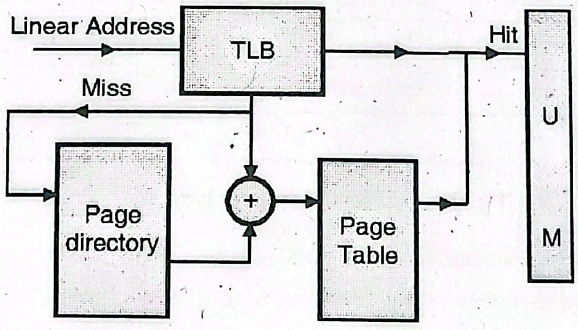

Fig. 5.2(c) : Translational lookaside buffer implementation

Q. 3 Explain how linear address 0080400 H will be translated into physical address using paging mechanism. Whether the address generated will be the same to linear address ? **SPPU - Dec. 19, 6 Marks**

Ans. :

$$00080400H = (\underbrace{0000\ 0000\ 0000}_{\text{Page direction entry}}\ \underbrace{1000\ 0000}_{\text{Page table entry}}\ \underbrace{0100\ 0000\ 0000}_{\text{Data Operand}})_2$$

- Thus this linear address will select the (000)16 entry of the page directory to select corresponding page table.
- In the page table, it will select (080)16 entry and hence select corresponding user page.
- In the selected user page it will select (400)16 entry.
- The linear address and physical address will not be same.

SPPU - May 19, 2 Marks

Q. 4 Draw and Explain the format of a selector.

Ans. :

- Selects one of the 8192 descriptors from one of the tables viz. GDT and LDT (Global Descriptor Table and Local Descriptor Table).
- GDTR (Global Descriptor Table Register) points to segment (of 64KB in size) common for all programs while LDTR points to segment (of 64KB is size) containing programs that are unique to an application
- Each segment (GDT and LDT) consists of 8192 descriptors of 8 bytes each and hence a total of 16K segments in all can be addressed by any task.
- Hence to select a descriptor out of 8K descriptors, selector (segment register) uses 13 MSB as shown in the Fig. 5.3 One bit to select the descriptor table (GDT or LDT) called as Table Identity (TI) bit. Remaining two bits are used to pass on the Requested Privilege Level (RPL).

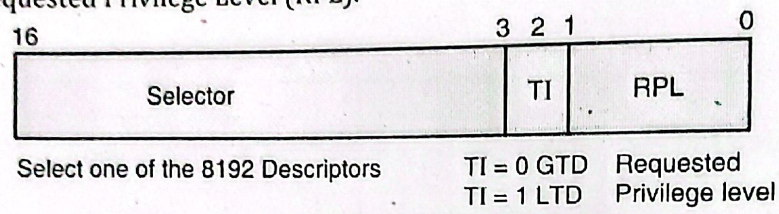

Fig. 5.3 : Structure of the selector

Q. 5 Draw and explain segment descriptor.

SPPU - May 17, May 18, 6 Marks

Ans. :

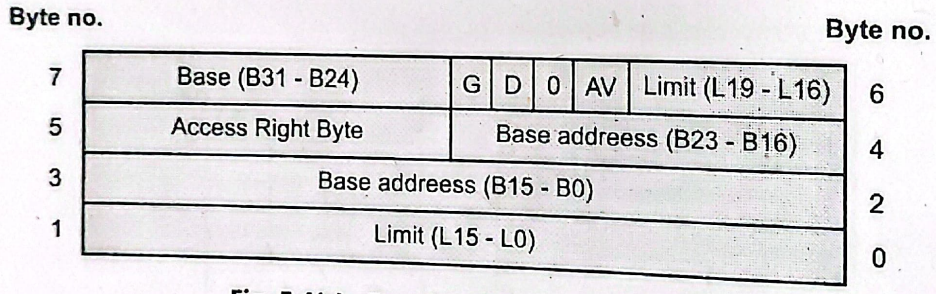

Fig. 5.4(a) : Descriptor register format

- **Limit address (L0 to L19)** when added with the base address gives the last address of the segment. In case of 286 there is a 24 bit base address and 16-bit limit while in 386 we have 32 bits base address and 20 bit limit address.
- Granularity bit (G)

  When G = 0, 20-bit Limit specifies the segment size from 1 byte to 1MB

  When G = 1, 20-bit Limit is to be multiplied by 4K hence segment size varies from 4KB to 4GB
- **AV** indicates the availability of segment

  When AV = 1, it indicates the corresponding segment is not used by any other task and hence is available

  When AV = 0, it indicates the corresponding segment is used by some other task and hence is not available
- **D** indicates data size

  When D = 0, it indicates 16-bit OS instructions (required in DOS, Windows).

  When D = 1, it indicates 32-bit OS instructions (required in OS/2, Windows NT etc).

- Access Right byte (ARB)

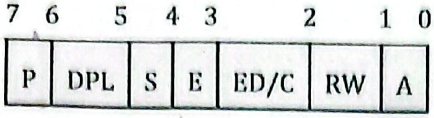

Fig. 5.4(b) : ARB Structure

7. The bits of the Access Right Byte of a descriptor are discussed the Table 5.1

Table 5.1 : Access Right Byte (ARB) bits and their significance

Bit position	Name	Function	
7	Present (P)	P = 1	The entry of the descriptor is initialized.
		P = 0	The entry of the descriptor is not initialized.
6-5	Descriptor Privilege Level (DPL)	Segment privilege level, used in privilege tests.	
4	Segment Descriptor (S)	S = 1	Code or Data (includes stack) segment descriptor.
		S = 0	System segment descriptor or Gate descriptor. In case it is a system descriptor, the remaining four bits (bit 3 to bit 0) indicates the type of system descriptor.
3	Executable (E)	E = 0	Descriptor type is data (or stack) segment.
If E=0, bit 2 and 1 of ARB, have the following meaning			
2	Expansion Direction (ED)	ED = 0	Expand up segment. (data segment)
		ED = 1	Expand down segment. (Stack segment)
1	Writeable (W)	W = 0	Data segment cannot be written into i.e. read only.
		W = 1	Data segment may be written into.
3	Executable (E)	E = 1	Descriptor type is code segment.
If E=1, bit 2 and 1 of ARB, have the following meaning			
2	Conforming (C)	C = 1	Code segment may only be executed when CPL >= DPL. If C=0, no privilege check will be made.
1	Readable (R)	R = 0	Code segment cannot be read.
		R = 1	Code Segment may be read.
0	Accessed (A)	A = 0	Segment has not been accessed.
		A = 1	Segment has been accessed earlier.

Q. 6 Explain segment address translation in detail.

Ans. :

- Global memory location is one which is accessible to all the tasks. The Global Descriptor Table (GDT) is a common table accessible to all the tasks. The GDT is selected by a 48-bit register called as GDT Register (GDTR). GDTR has the following two fields :
  1. 32-bit base address that provides the starting address of the descriptor table
  2. 16-bit Limit address that when added to the 32-bit base address gives the last address of the GDT.
- Hence the maximum value of the 16-bit limit address can be 64 KB which means that the maximum size of the GDT can be 64 KB. Each descriptor, as already discussed, is of 8 bytes.
- Hence the total number of descriptors in GDT is 8K (64 KB / 8 B). 13-bit selector field of the segment register selects one of the descriptor which gives the base address, limit address and the ARB of the corresponding segment.
- The base address of the segment when added with the offset address provided in the instruction generates the physical address.
- The offset address must not be more than the limit address or else it will generate a exception interrupt 5 i.e. bound exception. This access of global memory or segment translation mechanism for global data is as shown in Fig. 5.5.

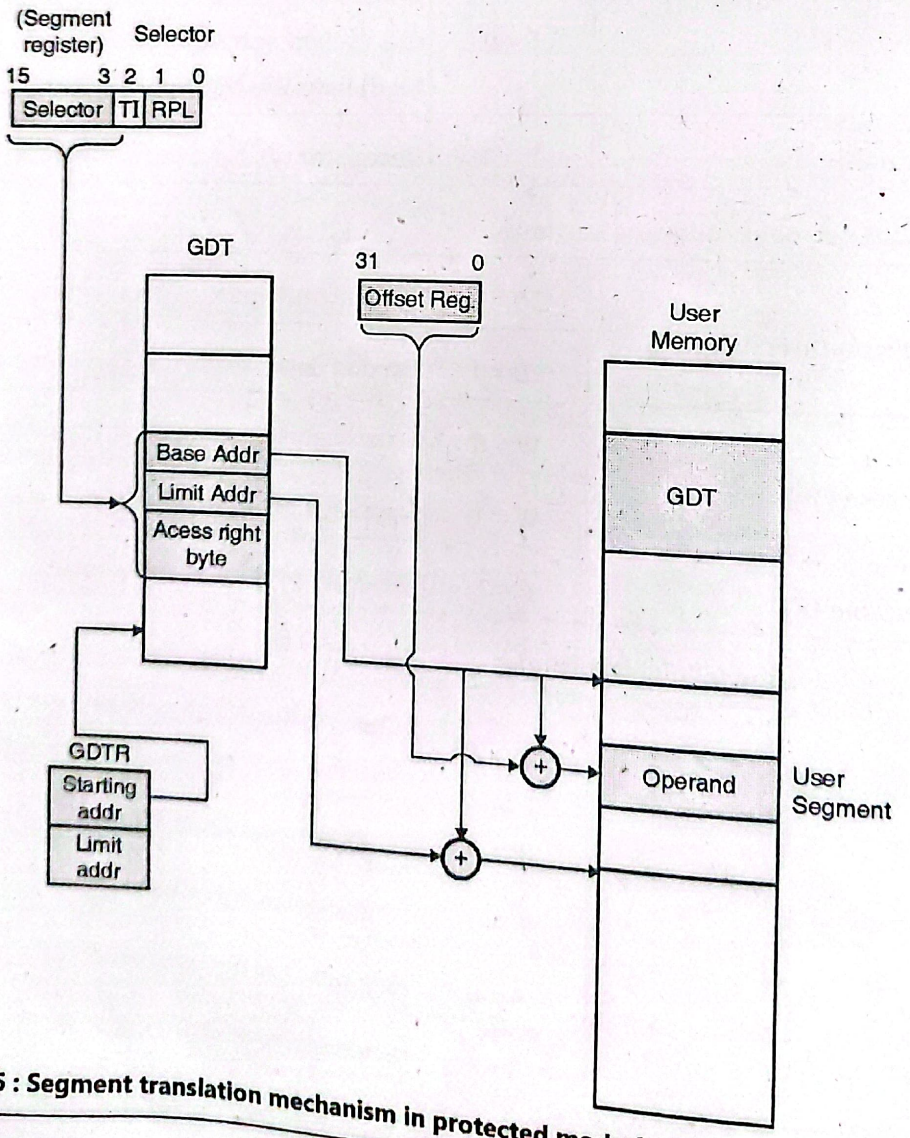

Fig. 5.5 : Segment translation mechanism in protected mode for global memory access

Q. 7 What is the significance of Local Descriptor Table Registers ? Explain with diagram. **SPPU - May 13, 4 Marks**

Ans. :

- Local memory location is one which is separately accessible to each task. The Local descriptor Table (LDT) is a separate for each task. The LDT is selected by a descriptor from the GDT, which in turn is selected by the LDT Register (LDTR). LDTR is 16-bit register that works as a selector similar to the segment register.
- This descriptor is called as a system descriptor, already discussed. The maximum value of the 16-bit limit address can be 64 KB which means that the maximum size of the LDT can be 64 KB. Each descriptor, as already discussed, is of 8 bytes.
- Hence the total number of descriptors in LDT is 8K (64 KB / 8 B). 13-bit selector field of the segment register selects one of the descriptor which gives the base address, limit address and the ARB of the corresponding segment.
- The base address of the segment when added with the offset address provided in the instruction generates the physical address.
- The offset address must not be more than the limit address or else it will generate a exception interrupt 5 i.e. bound exception.
- This access of local memory or segment translation mechanism for local data is as shown in Fig. 5.6.

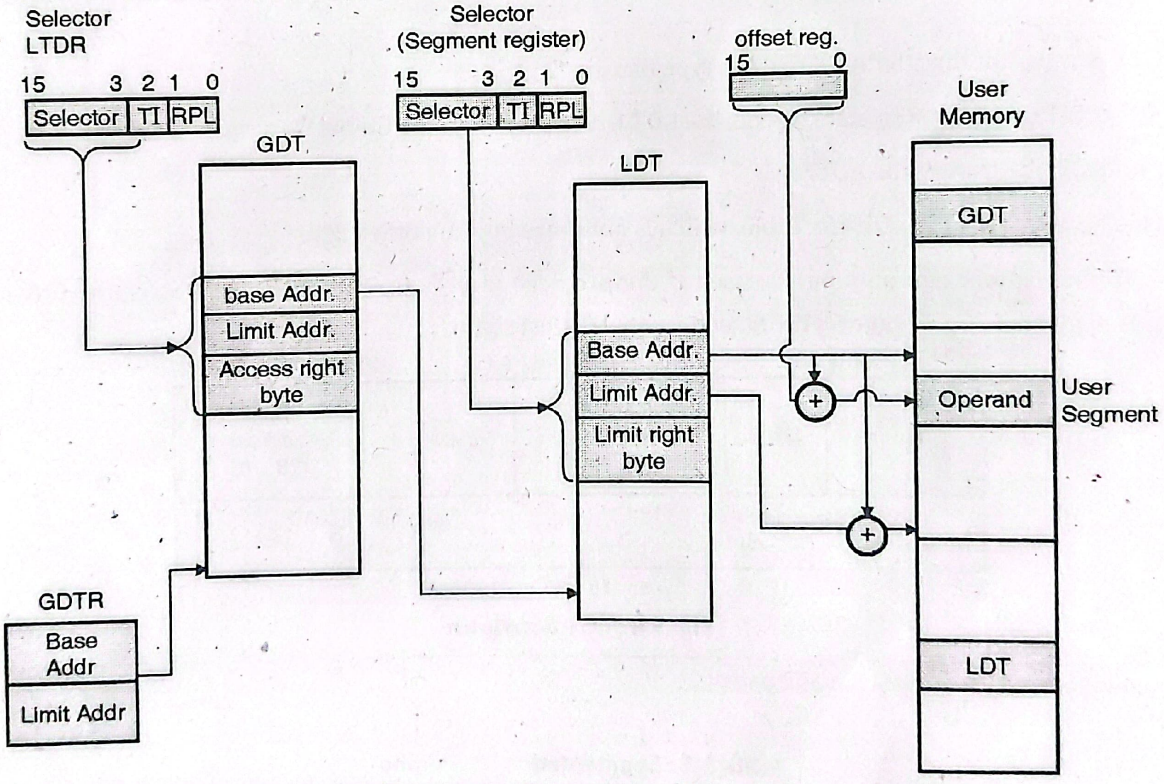

Fig. 5.6 : Segment translation mechanism in protected mode for local memory access

- Hence any task can access a maximum of 16K descriptors i.e. 8K descriptors from GDT and 8K descriptors from LDT. The GDT is common for all tasks but the LDT for individual task is different. Hence it helps in maintaining local and global data separately.
- The IDTR (Interrupt Descriptor Table Register) works similar to GDTR, but selects the descriptor that work as pointers to ISRs. While the TR (Task Register) works similar to LDTR, but selects the current Task State Segment (TSS).

SPPU - Dec. 13, 2 Marks

Q. 8 What are various types of descriptors?

Ans. :

Fig. 5.7 shows the types of segment descriptors. As shown in the Fig. 5.7 there are two main categories of segment descriptors : **"System Segment Descriptors"** and **"Non-system Segments Descriptors"**. They are further categorised into five types.

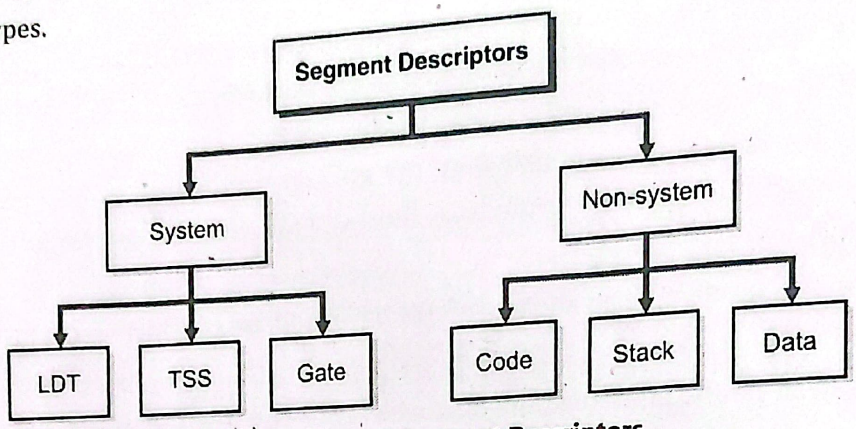

Fig. 5.7 : Types of Segment Descriptors

SPPU - May 14, 3 Marks

Q. 9 What is LDT descriptor ?

Ans. :

- For LDT descriptor, the S bit is '0' and the type bits are '2'.
- LDTR (Local Descriptor Register) selects the LDT Descriptor from the Global Descriptor Table (GDT).
- They point to the base of the LDT.
- Each task has its Local Descriptor Table, which is not shared by any other task
- The LDT descriptors can only be accessed if the privilege level is 0. Otherwise the Descriptor Privilege Level (DPL) is neglected. Fig. 5.8 shows the structure of LDT Descriptor

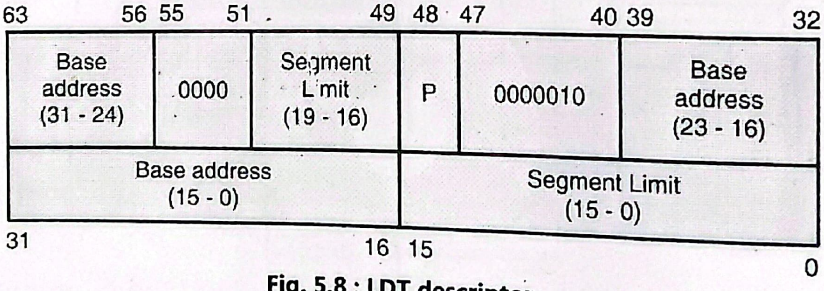

Fig. 5.8 : LDT descriptor

Q. 10 Compare between segmentation and paging.

Ans. : **(4 Marks)**

Table 5.2 : Segmentation Vs. Paging

Sr. No.	Segmentation	Paging
1.	The size of a segment is variable i.e. the size can vary from 1 byte to 4 GB	The page size is fixed and is 4KB
2.	Segmentation is a very efficient use of memory with less fragmentation.	Paging is comparatively less efficient use of memory with more fragmentation.
3.	Segmentation converts the logical address to the linear address	Paging converts the linear address to physical address

Q. 11 Draw and explain the 80386 address translation mechanism considering PG bit in CR0 in set.

SPPU - May 18, 6 Marks

OR With the help of neat diagram explain how logical address is converted into physical address ? Assume paging mechanism is disabled.

SPPU - Dec. 18, 6 Marks

Ans. :

- Segmentation coverts the logical address provided by the instruction (Selector or segment register and the offset register or pointer register) into the logical address.
- Paging converts the logical address obtained from the segmentation unit into the physical address used to access the physical memory
- The combination of the two is shown in the Fig. 5.9

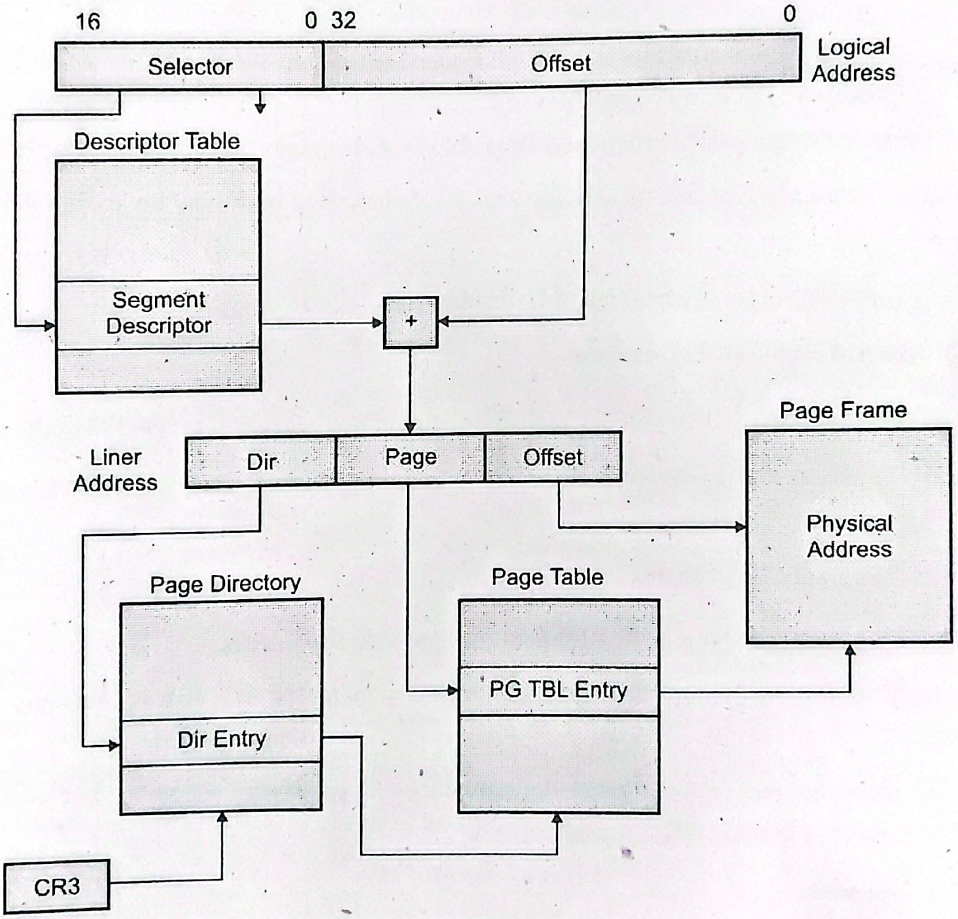

Fig. 5.9 : Combining Segment and Page Translation

- We can use different styles of memory management when paging translation is enabled.
- The different styles of memory management are explained below.

1. "Flat" Architecture

- If a particular software doesn't uses segments for its operation, then it would have been convenient to disable the segmentation.
- But such a feature of disabling segmentation is not available in 80386.

- But this can be achieved by initially loading the segment registers and the offset registers such that they give the required linear address of 32-bits
- Thereafter, once the segment register is loaded, they need not be changed. Only the offset registers can be changed to access the required memory location

2. Segments Spanning Several Pages

- Since the segments size is variable 1 bytes to 4GB, while the page size is fixed, segments can span over multiple pages
- For example if there is a segment of 45KB, it will span over 12 pages
- But these complications need not be known to the application programs. This is taken care by the memory management unit

3. Pages Scanning Several Segments

- Also, since the segments can be as small as 1 Byte, there can be pages that contain multiple segments
- For example if there are 2 segments of sizes 2KB each, then they both may be accommodated in the same page.
- But again the application programs need not consider that

4. Non-aligned Page and Segment Boundaries

- There is no correspondence between the page and segment boundaries in the 80386 architecture.
- So, a page may reside at the beginning of a segment while another page may be beginning at the end of a segment or vice-versa.

5. Aligned Page and Segment Boundaries

- There may be some pages that may be completely aligned with segments.
- Thus if a segment is of 4KB and is also having a boundary matching with that of the page, then the two will be aligned.
- We can also make it compulsory to align the boundaries of pages and segments, but in that case, we may need to leave some pages partially unused.

6. Page Table Per Segment

- In 80386 memory management implements a one-to-one correspondence between the segment descriptors and page directory entries.
- The base address of the descriptor is mapped to the first entry of page table. The segment can have size between 1 to 4 MB.
- The segment can reside in 1 to 1 KB page frames depending on the limit, restricting the task to these 1 KB segments. Each task can contain upto 4 MB.
- The allocation and deallocation of the descriptors corresponding to page table and page directory entry can be done.

Q. 12 Explain the flag register of x86 family. **(6 Marks)**

Ans. :

The flag register of the x86 family is as shown in the Fig. 5.10.

1. **VM Flag :** When the processor enters in virtual 8086 mode, which is an emulation of the programming environment of the 8086 microprocessor, this bit is set to '1'. When the processor is in protected mode and this bit is set, the processor moves to virtual 8086 mode. In this mode processor handles the segment loads as in 8086.

2. **RF Flag :** When RF = 1, it ignores the debug exception on execution of the next instruction. It is automatically reset at the successful completion of every instruction.

3. **NT Flag :** If NT = 1, it indicates that the currently executing task is nested within another task and it has a valid link to caller task i.e. this task is executed using the call instruction.

4. **IOPL Flag :** The IOPL encoded values indicates the privilege level at which the task should be executed to access the I/O device. Privilege levels are used in protected mode to maintain multiple tasks being executed at different privileges and hence can access things accordingly. Protection mode will be studied in the further sections of this chapter.

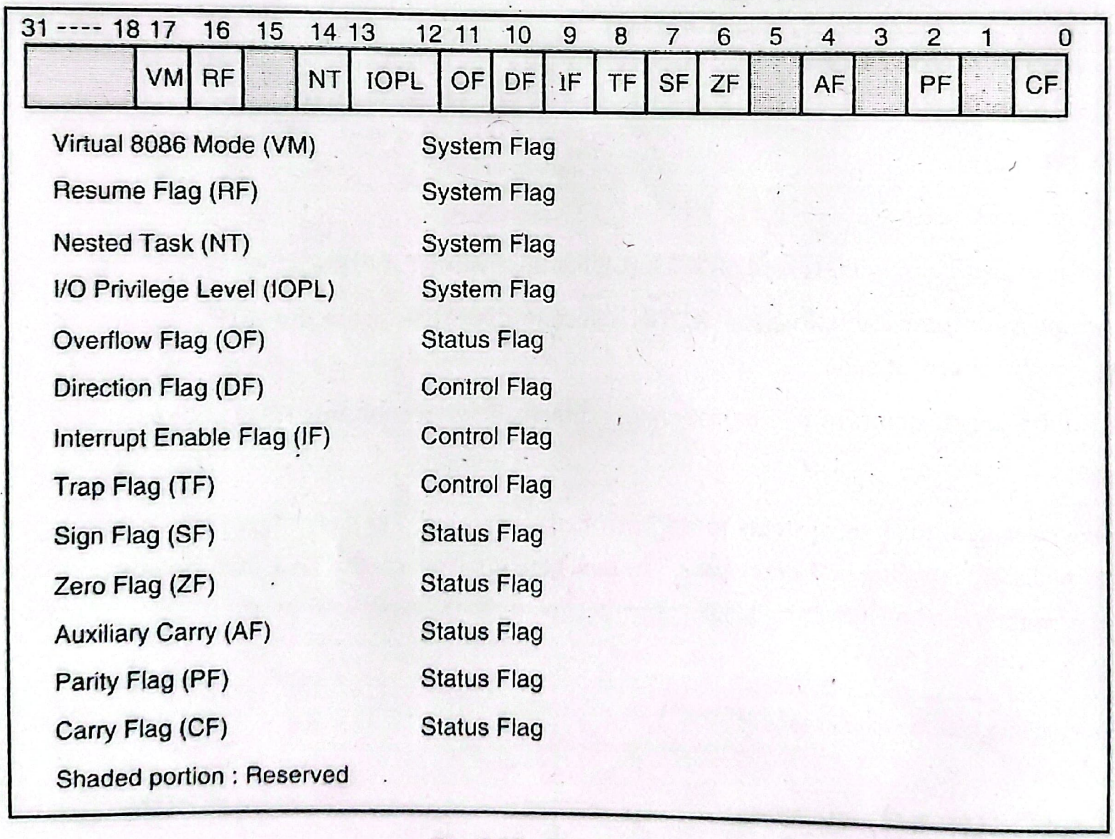

Fig. 5.10 : Flag register

5. **OF Flag :** The OF = 1 indicates that the operation resulted in signed overflow. Sign overflow occurs when a operation results in carry / borrow in the sign bit but doesn't result in a carry / borrow out of the high order bit or vice-versa.

6. **DF Flag :** DF defines whether ESI and/or EDI registers are auto-incremented or auto-decremented during the execution of string instructions. Auto-increment occurs if DF = 0; auto-decrement occurs if DF = 1.

7. **IF Flag :** When IF = 1, it allows recognition of external maskable interrupt INTR pin. When IF = 0, external maskable interrupt on INTR are not recognized.
8. **TF Flag :** When TF = 1, the processor is put into single-step mode used for debugging. In this mode, processor generates a single stepping interrupt after each instruction, which allows a program to be inspected as it executes.
9. **SF Flag :** SF = 1, if the MSB of the result is 1, i.e. the result is negative in case of a signed operation. SF copies the MSB i.e. the bit 7, 15, 31, for 8-, 16-, and 32-bit operations respectively.
10. **ZF Flag :** The ZF = 1 only if all bits of the result are zero; else ZF = 0.
11. **AF** Auxiliary Carry Flag: Also called as half way carry and is used for BCD operations. For 8-, 16-, or 32-bit operations, AF is set according to the carryout of bit 3 in each case.
12. **PF Flag :** The PF = 1, if the lower 8 bits of the operation contain an even number of 1s (i.e. even parity). The PF = 0, if the lower 8 bits have odd parity.
13. **CF Flag :** The CF = 1, if the operation resulted in a carryout of the MSB; else CF = 0. For 8-, 16-, or 32-bit operations, CF is set according to the carryout of bit 7, 15, or 31, respectively.

Q. 13 What is an interrupt and task gate? **(4 Marks)**

Ans. :

1. **Interrupt gates**
   - It is a system descriptor.
   - It accesses memory similar to that of GDTR (Global Descriptor Table Register).
   - Interrupt Descriptor Table Register (IDTR) selects a descriptor from the GDT. This descriptor selected is called as the interrupt gate.
   - The Interrupt gate points to the corresponding Interrupt Service Routine (ISR) to be executed
2. **Task gates**

   They are used in multitasking systems to perform task switching. The task register (TR) selects the task gate, similar to the LDTR selecting LDT descriptor. The task gate then selects the Task State Segment (TSS) that is used to copy the context of the current task. On return to this task, the context from the TSS is again loaded into the registers.

Q. 14 Draw and explain gate descriptor format. **(6 Marks)**

Ans. :

Function and Use

- A gate descriptor is a special type of descriptor. The 80386 processors uses this descriptor for performing protection checks. They also control the access to the entry points within the target code segment when control transfer instructions are executed.
- The gate with type numbers 4, 5, 6 and 7 are specified in type field for the call gate, task gate, interrupt gate and trap gate respectively.
- There are four types of gate descriptors.

They are :

(i) Call gates

Call gates are used to call the function/procedure which is at another privilege level. Hence it can change the privilege levels and transfer the program control to a more privileged level.

(ii) Interrupt gates

- It is a system descriptor.
- It accesses memory similar to that of GDTR (Global Descriptor Table Register).
- Interrupt Descriptor Table Register (IDTR) selects a descriptor from the GDT. This descriptor selected is called as the interrupt gate.
- The Interrupt gate points to the corresponding Interrupt Service Routine (ISR) to be executed

(iii) Task gates

They are used in multitasking systems to perform task switching. The task register (TR) selects the task gate, similar to the LDTR selecting LDT descriptor. The task gate then selects the Task State Segment (TSS) that is used to copy the context of the current task. On return to this task, the context from the TSS is again loaded into the registers.

(iv) Trap gates

- Trap gate is used for handling the single stepping interrupt
- Fig. 5.11 shows the format of the four types of gate descriptors.

Selector	Offset (15-0)							
Offset (31-16)	P	DPL	0	Type	0	0	0	Word count 4 ... 0

Value	Description
4	80286 call gate.
5	Task gate (for 80286 or intel 80386DX task)
6	80286 interrupt gate.
7	80286 trap gate.
C	Intel 80386DX call gate
E	Intel 80386DX interrupt gate
F	Intel 80386DX trap gate.
0	Description contents are not valid.
1	Description contents are valid.
DPL :	Least privilege level at which task may access the gate.
Selector :	16 bit selector - selection to target code segment or selector to the target task state segment for task gate.
Word count :	Number of parameters to copy from callers stack to the called procedure stack. The parameters are 32 bit for 80386 processor gates and 16 bit for 80286 processor gates.
Offset (0-15) : 80286 Offset (0-31) : 80386Dx	} entry point within target code segment.

Fig. 5.11 : Gate descriptor format

Unit IV : Protection

Chapter 6 : Protection

Q. 1 Explain four level of hierarchical protection in 80386 DX microprocessor. SPPU - Dec. 15, 3 Marks

OR List five aspects of protection in the 80386 SPPU - May 18, 2 Marks

OR List aspects of protection related to pages. SPPU - Dec. 19, 2 Marks

Ans. :

- The protection is implemented internally in the memory management system; both in segment translation mechanism as well as page translation mechanism
- Protection in the 80386 has five aspects:
  1. Type checking
  2. Limit checking
  3. Restriction of address able domain
  4. Restriction of procedure entry points
  5. Restriction of instruction set
- The restriction of addressable domain, procedure entry points and instruction set is also called as Privilege check. With respect to the procedures, privilege is the degree to which the procedure can be trusted not to make a mistake that might affect other procedures or data.
- With respect to the data, privilege is the degree of protection that a data structure should have from less trusted procedures.
- Each reference to memory is checked by the hardware to verify that it satisfies the protection criteria. All these checks are made before the memory cycle is started; any violation prevents that cycle from starting and results in an exception. Since the checks are performed concurrently with address formation, there is no performance penalty. Invalid attempts to access memory result in an exception.
- The concept of privilege applies both to segment protection and to page protection.

Q. 2 Explain CPL, RPL and DPL concepts. SPPU - May 12, 3 Marks

OR What is CPL and RPL ? SPPU - Dec. 17, May 18, Dec. 19, 2 Marks

OR With appropriate diagram explain the concept of privilege level in 80386. SPPU - May 19, 4 Marks

Ans. :

1. Most processors have only two protection levels (user and supervisor), but x86 architecture features four levels of protection, called **privilege levels (PL).**
2. They are designed to support the needs of multitasking OS to isolate and protect user programs from each other and the OS from unauthorized access.

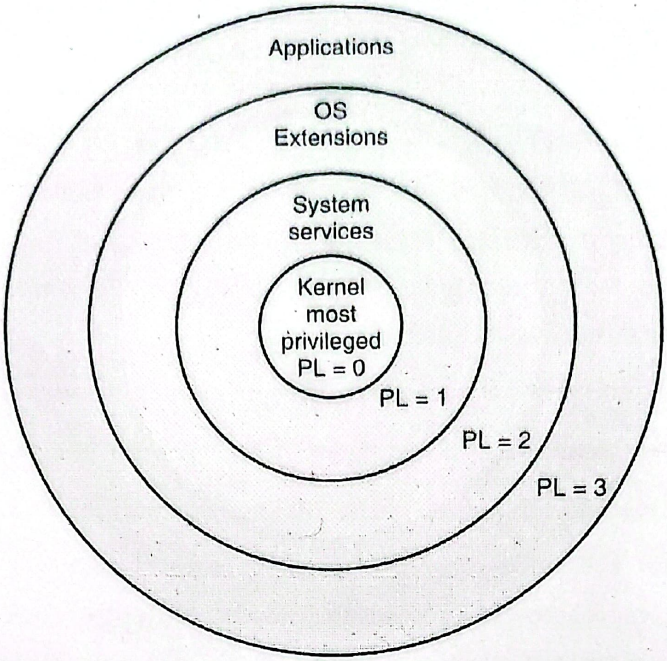

Fig. 6.1 : Privilege levels of the tasks in x86 architecture

3. The privilege levels control the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. The x86 architecture offers an additional type of protection on a page basis, when paging is enabled.

   Requested PL ⇒ RPL

   Descriptor PL ⇒ DPL

   Current PL ⇒ CPL

   Effective PL ⇒ EPL

   EPL = max (RPL, CPL)

4. The PLs are numbered 0, 1, 2, and 3. Level 0 is the most privileged level. Level 3 is the least privileged. As shown in Fig. 6.1, level 3 is used for user application, level 2 is used for OS extensions, level 1 for system services, and the most privileged level 0 is used for kernel.

5. The x86 architecture controls access to both data and code between levels of task, according to the following rules of privilege:

   (a) Data stored in a segment with PL = p can be accessed only by code executing at a PL, numerically, at least as privileged as p.

   (b) A code segment (a procedure) with PL = p can be called only by a task executing at, numerically, the same or lower PL than p.

   (c) A stack segment with PL = p can be used only by a task executing at the same PL.

6. The following PLs are used to maintain privilege level check :

   (a) Requestor PL, RPL, the PL of the original task that supplies the selector. RPL is determined by the two LSBs of the selector.

(b) Descriptor PL, DPL, the PL (according to the above rules) at which a task may access that descriptor and the segment associated with that descriptor. Bits 6 and 5 of the access rights byte (ARB) of a descriptor determine the DPL.

(c) Current PL, CPL, the PL at which a task is currently executing, i.e. at which the code segment is being executed. CPL is stored in the processor, but not accessible to the programmer.

(d) Effective PL, EPL, the least privileged of the RPL and the CPL. Since smaller PL values indicate greater privilege, EPL is the numerical maximum of RPL and CPL, i.e. EPL = max (RPL, CPL). EPL is not stored anywhere, but is immediately copied into CPL.

7. System segments describe information about tasks, interrupts, subroutines etc.

The different types (the lower four bits of ARB for a system descriptor) of system descriptor are as listed below:

Type 2 – It refers to LDT descriptor. It is located in the GDT and points to the base of the LDT.

TYPE 4, 7, C, E, F – These refer to gate descriptors. The gates are used to control access to entry points within the target code segment for control transfer instructions. This also allows processor to perform protection checks automatically. There are four types of gates:

(a) Call gates that serve as an intermediary between code segments at different PLs. Call gates are used to change the PLs.

(b) Task gates that are used to perform task switch. The Task Register (TR) selects this gate which in turn selects the current Task State Segment (TSS). The TSS holds the context of the current task.

(c) Interrupt gates that are used to specify interrupt service routines (ISRs).

(d) Trap gates that are used to specify trap-handling routines.

8. Call gates are used to transfer program control to a more privileged level. The call gate descriptor consists of three fields.

(a) The access rights byte (ARB),

(b) The pointer (selector and offset), which points to the start of the target routine, and

(c) The word count (5 bit long), which specifies how many parameters are to be copied from the caller's stack to the stack of the called subroutine. This field is used by call gates only when there is a change in the PL. Other gates (Task gate, Interrupt gate and Trap gate) ignore this field.

9. Call gates are accessed via a CALL instruction and are syntactically identical to calling a normal subroutine. When an inter level (different privilege level) call gate is activated, the following actions occur:

(a) CS:EIP are loaded from the selector and offset fields of the gate, respectively, and are checked for validity.

(b) SS is pushed on stack with zero-extended to 32 bits.

(c) ESP is pushed on stack.

(d) The word count (32-bit parameters) is copied from the old stack to the new stack.

(e) The return address is pushed on stack.

10. Accessing different types of programs using gates is shown in Fig. 6.2.

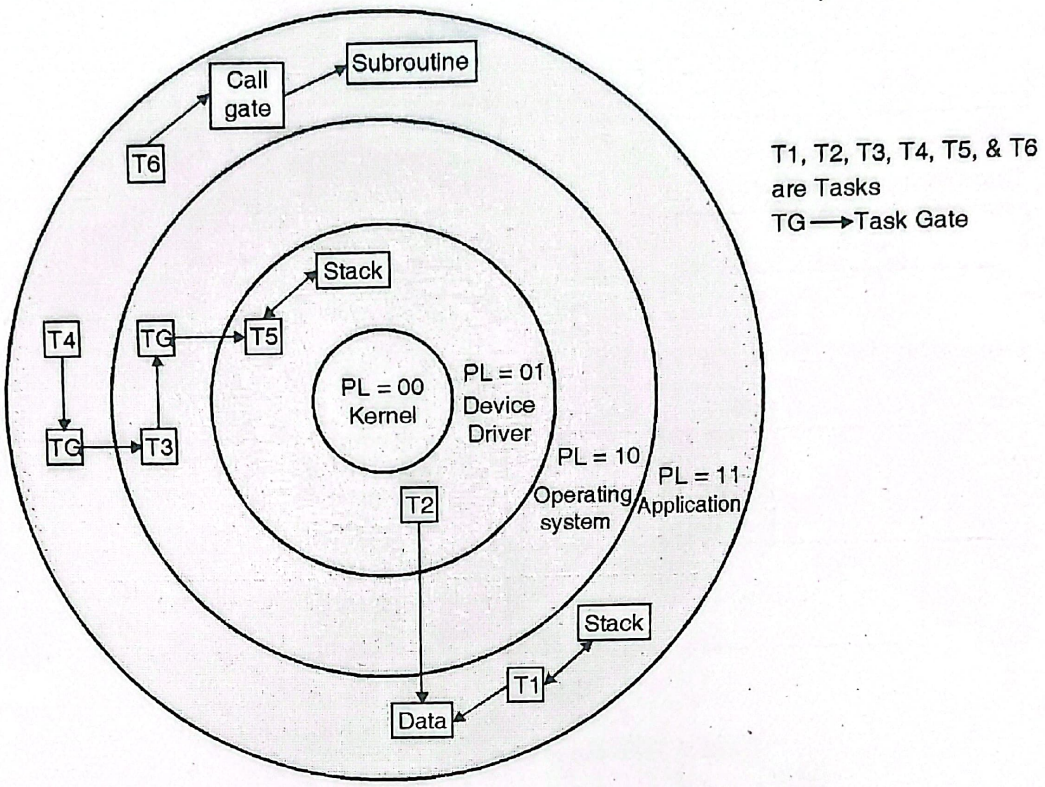

Fig. 6.2 : Rules for accessing data, code and stack

11. Gate descriptors follow all the access rules of privilege; i.e., gates can be accessed by a task if its EPL is equal to, or more privileged than the gate descriptor's DPL.
12. A task can access the code of higher PL using a gate (task gate, call gate, trap gate or interrupt gate)
13. A task can access data of lower or equal PL.
14. A task can access stack of same PL only.
15. The PL of the task (CPL) is compared with the PL of code /data /stack (DPL) for the above decisions.

Q. 3 Write privilege checks performed by 80386 while accessing code or data with protection mechanism.

SPPU - May 12, 5 Marks

Ans. :

- To address data operands in memory, an 80386 program must load the selector of a data segment into a data-segment register (DS, ES, FS, GS, SS).
- Instructions may load a data-segment register only if the DPL of the target segment is numerically greater than or equal to the maximum of the CPL and the selector's RPL. In other words, a procedure can only access data that is at the same or less privileged level.
- The addressable domain of a task varies as CPL changes. When CPL is zero, data segments at all privilege levels are accessible; when CPL is one, only data segments at privilege levels one through three are accessible; when CPL is three, only data segments at privilege level three are accessible. This property of the 80386 can be used, for example, to prevent applications procedures from reading or changing tables of the operating system.

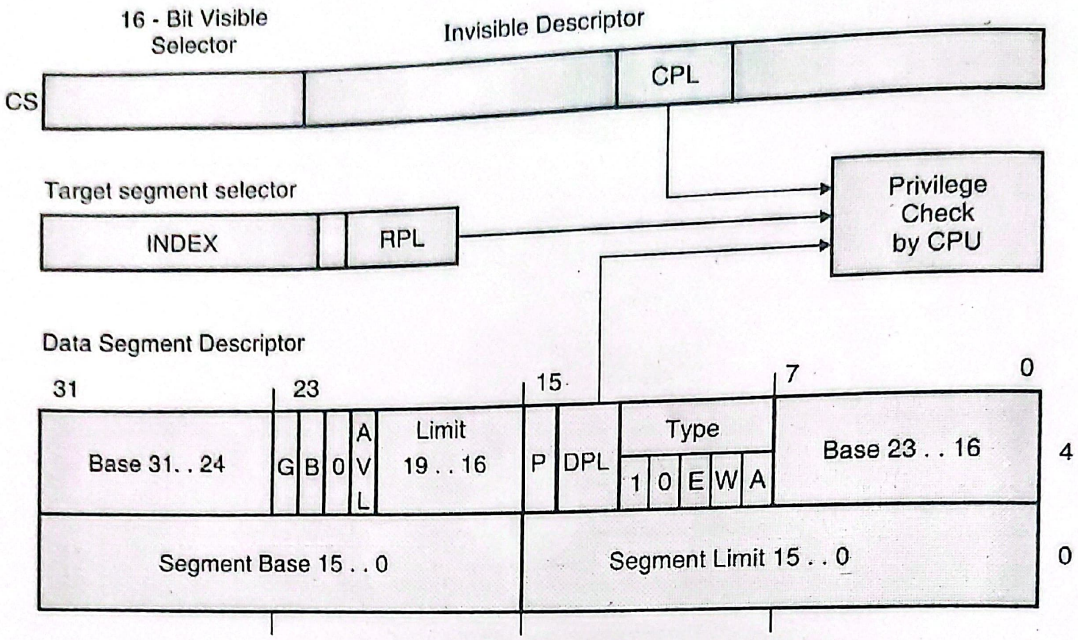

Fig. 6.3 :Privilege Check for Data Access

Q. 4 Write privilege checks performed by 80386 while accessing code or data with protection mechanism.

SPPU - May 12, 5 Marks

Ans. :

- Less common than the use of data segments is the use of code segments to store data.
- Code segments may legitimately hold constants; it is not possible to write to a segment described as a code segment. The following methods of accessing data in code segments are possible:
  1. Load a data-segment register with a selector of a nonconforming, readable, executable segment.
  2. Load a data-segment register with a selector of a conforming, readable, executable segment.
  3. Use a CS override prefix to read a readable, executable segment whose selector is already loaded in the CS register.
- The same rules as for access to data segments apply to case 1. Case 2 is always valid because the privilege level of a segment whose conforming bit is set is effectively the same as CPL regardless of its DPL.
- Case 3 always valid because the DPL of the code segment in CS is, by definition, equal to CPL.

Conforming and non-conforming code

- For conforming code the access to the code will be only if the privilege level rules match.
- For certain code, there is no need to check the privilege level. Such code is called as non-conforming code.
- The code is confirming code or non-conforming is decided by the conforming but in the description.

Q. 5 What is call gate ? Explain how it is used in calling a function with higher privilege level. **SPPU - May 12, 8 Marks**

OR Explain CALL Gate mechanism in detail. **SPPU - Nov. 12, 6 Marks**

OR What is privileged instruction ? Explain its significance with examples. **SPPU - May 13, 6 Marks**

OR How call gate descriptor is used to locate the procedure in another code segment ? How protection is provided. **SPPU - May 18, 6 Marks**

OR With the help of suitable diagram, explain how call gate descriptor is used to change the privilege levels in protected mode ? **SPPU - Dec. 18, 6 Marks**

OR How call gate descriptor is used to locate the procedure in another code segment ? How protection is provided ? **SPPU - May 19, 6 Marks**

Ans. :

- To provide protection for control transfers among executable segments at different privilege levels, the 80386 uses gate descriptors. There are four kinds of gate descriptors :
  - Call gates
  - Trap gates
  - Interrupt gates
  - Task gates
- Fig. 6.4(a) illustrates the format of a call gate. A call gate descriptor may reside in the GDT or in an LDT.

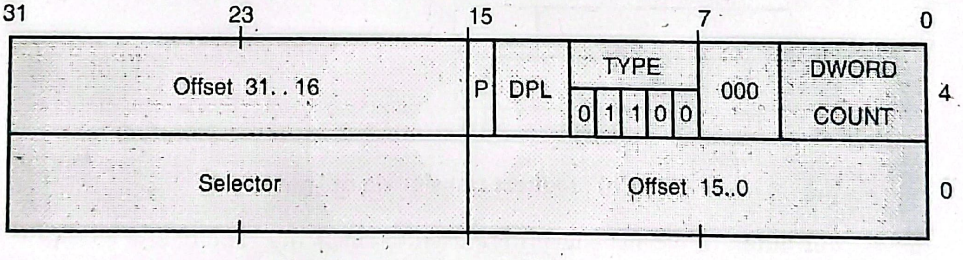

Fig. 6.4 (a): Format of 80386 call gate

- A call gate has two primary functions :
  1. To define an entry point of a procedure.
  2. To specify the privilege level of the entry point.
- Call gate descriptors are used by call and jump instructions in the same manner as code segment descriptors. When the hardware recognizes that the destination selector refers to a gate descriptor, the operation of the instruction is expanded as determined by the contents of the call gate.
- The selector and offset fields of a gate form a pointer to the entry point of a procedure. A call gate guarantees that all transitions to another segment go to a valid entry point, rather than possibly into the middle of a procedure (or worse, into the middle of an instruction). The far pointer operand of the control transfer instruction does not point to the segment and offset of the target instruction; rather, the selector part of the pointer selects a gate, and the offset is not used. Fig. 6.4(b) illustrates this style of addressing.

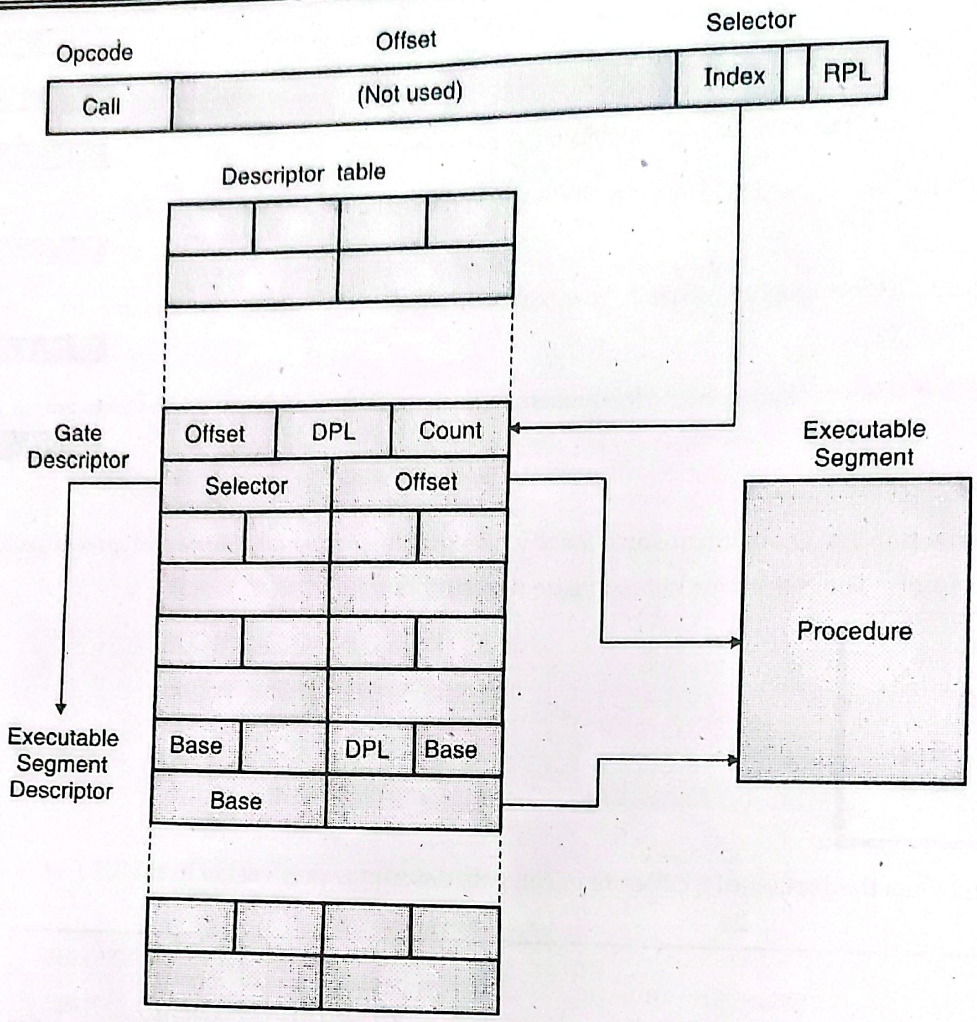

Fig. 6.4(b) : Indirect transfer via call gate

- As Fig. 6.4(c) shows, four different privilege levels are used to check the validity of a control transfer via a call gate:
  1. The CPL (current privilege level).
  2. The RPL (requestor's privilege level) of the selector used to specify the call gate.
  3. The DPL of the gate descriptor.
  4. The DPL of the descriptor of the target executable segment.
- The DPL field of the gate descriptor determines what privilege levels can use the gate. One code segment can have several procedures that are intended for use by different privilege levels. For example, an operating system may have some services that are intended to be used by applications, whereas others may be intended only for use by other systems software.
- Gates can be used for control transfers to numerically smaller privilege levels or to the same privilege level (though they are not necessary for transfers to the same level). Only CALL instructions can use gates to transfer to smaller privilege levels. A gate may be used by a JMP instruction only to transfer to an executable segment with the same privilege level or to a conforming segment.

- For a JMP instruction to a nonconforming segment, both of the following privilege rules must be satisfied; otherwise, a general protection exception results.

  MAX (CPL,RPL) ≤ gate DPL

  target segment DPL = CPL

- For a CALL instruction (or for a JMP instruction to a conforming segment), both of the following privilege rules must be satisfied; otherwise, a general protection exception results.

  MAX (CPL,RPL) ≤ gate DPL

  target segment DPL ≤ CPL

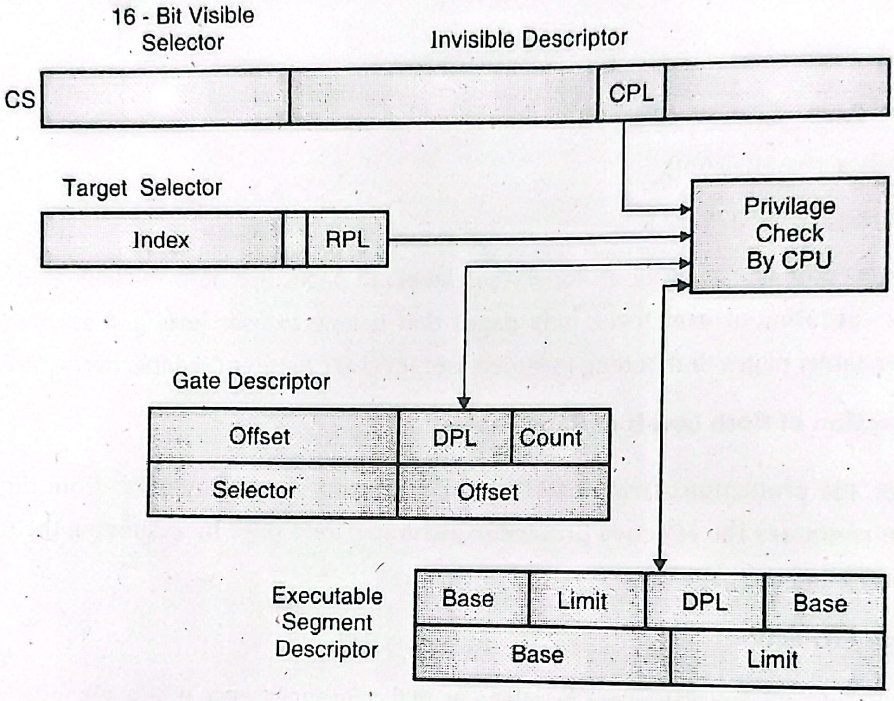

Fig. 6.4(c) : Privilege Check via Call Gate

Q. 6 List aspects of protection related to pages. SPPU - May 19, 2 Marks

Ans. :

The protection mechanisms applicable to the pages are:

1. Restriction of addressable domain.
2. Type checking.

Page-Table Entries Hold Protection Parameters

Fig. 6.5 highlights the fields of PDEs and PTEs that control access to pages.

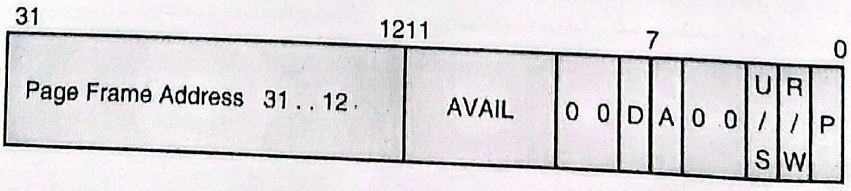

Fig. 6.5 :Protection fields of page table entries

1. Restricting Addressable Domain

- The concept of privilege for pages is implemented by assigning each page to one of two levels:
  1. Supervisor level (U/S=0) - for the operating system and other systems software and related data.
  2. User level (U/S=1) - for applications procedures and data.
- The current level (U or S) is related to CPL. If CPL is 0, 1, or2, the processor is executing at supervisor level. If CPL is 3, the processor is executing at user level.
- When the processor is executing at supervisor level, all pages are addressable, but, when the processor is executing at user level, only pages that belong to the user level are addressable.

2. Type Checking

- At the level of page addressing, two types are defined:
  a. Read-only access(R/W=0)
  b. Read/write access(R/W=1)
- When the processor is executing at supervisor level, all pages are both readable and writable. When the processor is executing at user level, only pages that belong to user level and are marked for read/write access are writable; pages that belong to supervisor level are neither readable nor writable from user level.

3. Combining Protection of Both Levels of Page Tables

For any one page, the protection attributes of its page directory entry may differ from those of its page table entry. The 80386 computes the effective protection attributes for a page by examining the protection attributes in both the directory and the page table.

4. Overrides to Page Protection

Certain accesses are checked as if they are privilege-level 0 references, even if CPL = 3:

1. LDT, GDT, TSS, IDT references.
2. Access to inner stack during ring-crossing CALL/INT.

Q. 7 What is need of protection in 80386.

Ans. : **(4 Marks)**

- The main purpose of the protection features of the 80386 is to have multitasking with each task having its protected memory. Protection ensures that the multiple tasks do not access the data or code of other tasks.
- In case of multiple tasks (multi-tasking), there is a possibility that a task accesses the code or data of another task and changes the same inadvertently.
- Hence to avoid such cases, protection is implemented in 80386.

Q. 8 Write a note on Type checking.

Ans. : **(6 Marks)**

- The Type field of a descriptor has two functions:
  1. It distinguishes among different descriptor formats.
  2. It specifies the intended usage of a segment.

- The type check basically checks the type of the descriptor (and hence the segment) with the selector used to access it.
- A descriptor may be executable or non-executable and hence accordingly the code segment register or the data segment register must be used for accessing.
- Also the descriptor could be a system descriptor.
- The type fields of data and executable segment descriptors include bits which further define the purpose of the segment
- The writable bit in a data-segment descriptor specifies whether instructions can write into the segment.
- The readable bit in an executable-segment descriptor specifies whether instructions are allowed to read from the segment (for example, to access constants that are stored with instructions).

Table 6.1 : System and Gate Descriptor Types

Code	Type of Segment or Gate	
0	-reserved	
1	Available 286	TSS
2	LDT	
3	Busy 286 TSS	
4	Call Gate	
5	Task Gate	
6	286 Interrupt	Gate
7	286 Trap Gate	
8	-reserved	
9	Available 386	TSS
A	-reserved	
B	Busy 386 TSS	
C	386 Call Gate	
D	-reserved	
E	386 Interrupt	Gate
F	386 Trap Gate	

Q. 9 What is stack switching in 80386. **(4 Marks)**

Ans. :

- If the destination code segment of the call gate is at a different privilege level than the CPL, an inter level transfer is being requested.

- To maintain system integrity, each privilege level has a separate stack.
- These stacks assure sufficient stack space to process calls from less privileged levels. Without them, a trusted procedure would not work correctly if the calling procedure did not provide sufficient space on the caller's stack.
- The processor locates these stacks via the task state segment (TSS).
- Each task has a separate TSS, thereby permitting tasks to have separate stacks.
- Systems software is responsible for creating TSSs and placing correct stack pointers in them.
- The initial stack pointers in the TSS are strictly read-only values. The processor never changes them during the course of execution.
- When a call gate is used to change privilege levels, a new stack is selected by loading a pointer value from the Task State Segment (TSS). The processor uses the DPL of the target code segment (the new CPL) to index the initial stack pointer for PL 0, PL 1, or PL 2.
- The DPL of the new stack data segment must equal the new CPL; if it does not, a stack exception occurs. It is the responsibility of systems software to create stacks and stack-segment descriptors for all privilege levels that are used.
- Each stack must contain enough space to hold the old SS:ESP, the return address, and all parameters and local variables that may be required to process a call.
- As with intra level calls, parameters for the subroutine are placed on the stack.
- To make privilege transitions transparent to the called procedure, the processor copies the parameters to the new stack.
- The count field of a call gate tells the processor how many double-words (up to 31) to copy from the caller's stack to the new stack. If the count is zero, no parameters are copied.
- The processor performs the following stack-related steps in executing an inter-level CALL.
  1. The new stack is checked to assure that it is large enough to hold the parameters and linkages; if it is not, a stack fault occurs with an error code of 0.
  2. The old value of the stack registers SS:ESP is pushed onto the new stack as two double-words.
  3. The parameters are copied.
  4. A pointer to the instruction after the CALL instruction (the former value of CS:EIP) is pushed onto the new stack. The final value of SS:ESP points to this return pointer on the new stack.

□□□

Unit V : Multitasking and Virtual 8086 Mode

Chapter 7 : Multitasking

Q. 1 What is Multitasking ? SPPU - May 13, 3 Marks

OR Write a short note on "Multitasking" feature of 80386. SPPU - Dec. 19, 4 Marks

Ans. :

- The hardware support for multitasking is given by 80386. To implement this multitasking, 80386 has some special components. They ensure rapid switching between tasks. These are:
  - Task state segment
  - Task state segment descriptor
  - Task register
  - Task gate descriptor

  In addition to the simple task switch, the 80386 also offers two other task-management features:

1. Interrupts and exceptions can cause task switches. The processor can switch automatically to the task that handles the interrupt or exception, and then automatically switch back to the interrupted task when the interrupt or exception has been serviced.
2. Every time a task switch takes place, the LDT as well as the page directory can be changed. Hence, the tasks can have a different logical-to-linear mapping and a different linear-to-physical mapping. This is yet another protection feature, because tasks can be isolated and prevented from interfering with one another.

Q. 2 What is a Task State Segment (TSS) ? SPPU - Nov. 12, 2 Marks

OR Discuss the use of TSS in Multitasking. SPPU - Nov. 12, 8 Marks

OR Explain Task State Segment (TSS) with the help of diagram in detail. SPPU - May 13, 3 Marks

OR Draw and briefly explain Task State Segment. SPPU - Dec. 17, May 18, 6 Marks

OR What is the role of TSS in multitasking ? SPPU - Dec. 18, 3 Marks

OR Write a short note on "Multitasking" feature of 80386. SPPU - Dec. 19, 4 Marks

Ans. :

- Task State Segment stores the entire context of a task when task switching takes place. Each task is given a separate Task State Segment (TSS). Fig. 7.1 shows the format of a TSS for executing 80386 tasks.
- The fields of a TSS belong to two classes:

1. A dynamic set that the processor updates with each switch from the task. This set includes the fields that store:
   - The general registers (EAX, ECX, EDX, EBX, ESP, EBP, ESI,EDI).
   - The segment registers (ES, CS, SS, DS, FS,GS).
   - The flags register (EFLAGS).
   - The Extended Instruction Pointer (EIP).
   - The selector of the TSS of the previously executing task (updated only when a return is expected).

2. A static set that the processor reads but does not change. This set includes the fields that store:
   - The selector of the task's LDT.
   - The register (PDBR) that contains the base address of the task's page directory (read only when paging is enabled).
   - Pointers to the stacks for privilege levels 0-2.
   - The T-bit (debug trap bit) which causes the processor to raise a debug exception when a task switch occurs.

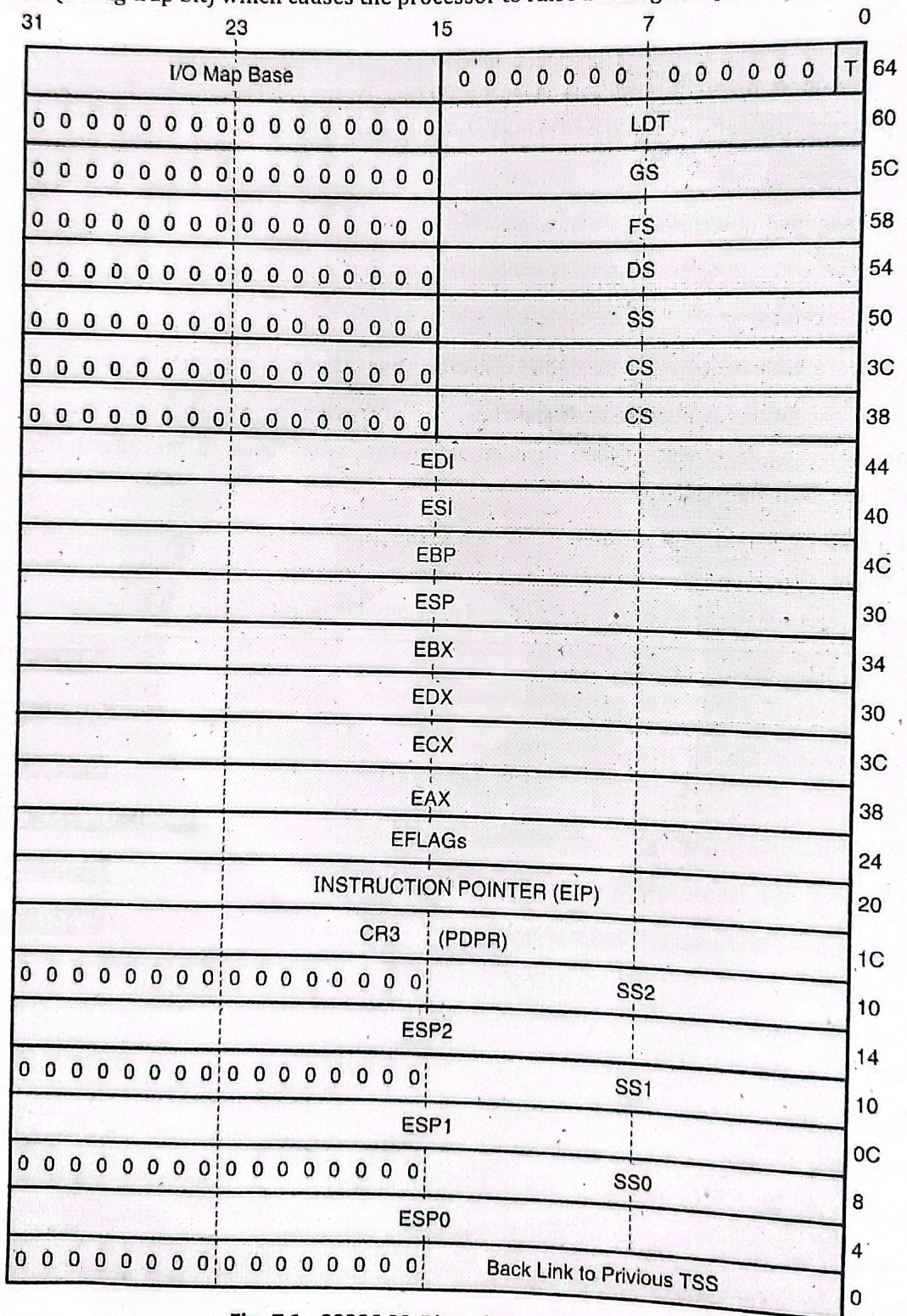

Fig. 7.1 : 80386 32-Bit task state segment

Q. 3 Explain the significance and format of the TSS descriptor. SPPU - May 12, 5 Marks

OR Draw neat diagram to explain TSS descriptor. SPPU - May 13, 3 Marks

OR What is TSS descriptor? SPPU - Dec. 13, 3 Marks

Ans. :

- The task state segment, like all other segments, is defined by a descriptor. Fig. 7.2 shows the format of a TSS descriptor.
- The BASE, LIMIT, and DPL fields and the G-bit and P-bit have functions similar to their counterparts in data-segment descriptors. The LIMIT field, however, must have a value equal to or greater than 103. An attempt to switch to a task whose TSS descriptor has a limit less than 103 causes an exception. A larger limit is permissible, and a larger limit is required if an I/O permission map is present. A larger limit may also be convenient for systems software if additional data is stored in the same segment as the TSS.
- A procedure that has access to a TSS descriptor can cause a task switch. In most systems the DPL fields of TSS descriptors should be set to zero, so that only trusted software has the right to perform task switching.
- Having access to a TSS-descriptor does not give a procedure the right to read or modify a TSS. Reading and modification can be accomplished only with another descriptor that redefines the TSS as a data segment. An attempt to load a TSS descriptor into any of the segment registers (CS, SS, DS, ES,FS, GS) causes an exception.
- TSS descriptors may reside only in the GDT. An attempt to identify a TSS with a selector that has TI = 1 (indicating the current LDT) results in an exception.

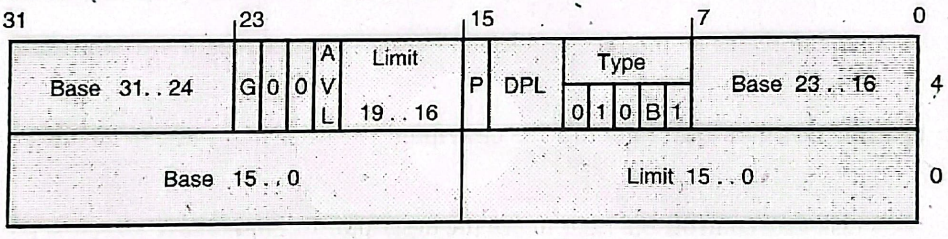

Fig. 7.2 : TSS descriptor

Q. 4 Draw neat diagram to explain task register. SPPU - May 13, 3 Marks

OR List the registers and data structures that are used in multitasking. SPPU - May 17, 2 Marks

OR Explain the role of Task Register in multitasking and the instructions used to modify and read TR. SPPU - May 18, 6 Marks

Ans. :

- The Task Register (TR) identifies the currently executing task by pointing to the TSS. Fig. 7.3 shows the path by which the processor accesses the current TSS.
- The task register has both a "visible" portion (i.e., can be read and changed by instructions) and an "invisible" portion (maintained by the processor to correspond to the visible portion; cannot be read by any instruction). The selector in the visible portion selects a TSS descriptor in the GDT. The processor uses the invisible portion to cache the base and limit values from the TSS descriptor.
- LTR (Load task register) loads the visible portion of the task register with the selector operand, which must select a TSS descriptor in the GDT. LTR also loads the invisible portion with information from the TSS descriptor selected by the operand. LTR is a privileged instruction; it may be executed only when CPL is zero. LTR is generally used during system initialization to give an initial value to the task register; thereafter, the contents of TR are changed by task switch operations.

- STR (Store task register) stores the visible portion of the task register in a general register or memory word. STR is not privileged.

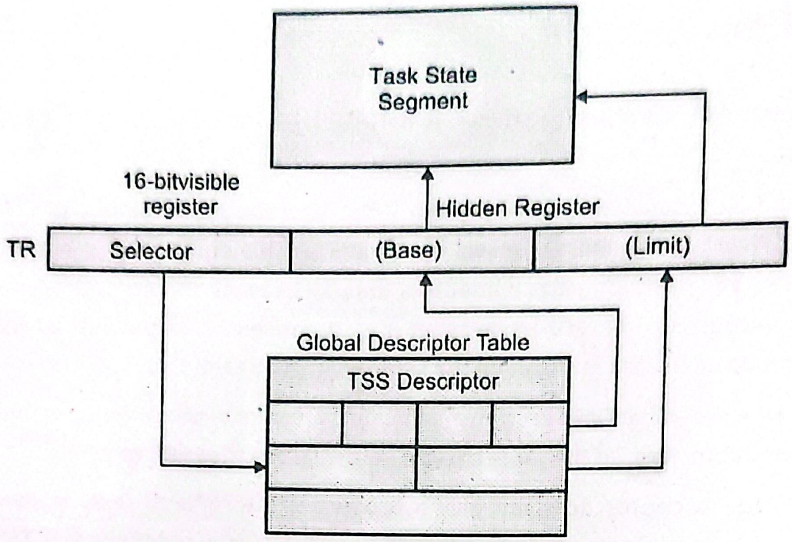

Fig. 7.3 : Task register

Q. 5 What is the use of task gate ?

SPPU - May 14, 3 Marks

Ans. :

- A task gate descriptor provides an indirect, protected reference to a TSS. Fig. 7.4(a) illustrates the format of a task gate.
- The Selector field of a task gate must refer to a TSS descriptor. The value of the RPL in this selector is not used by the processor.
- The DPL field of a task gate controls the right to use the descriptor to cause a task switch. A procedure may not select a task gate descriptor unless the maximum of the selector's RPL and the CPL of the procedure is numerically less than or equal to the DPL of the descriptor. This constraint prevents untrusted procedures from causing a task switch.

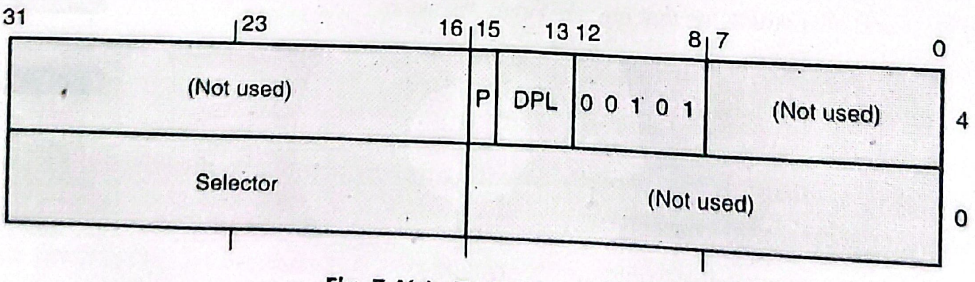

Fig. 7.4(a) : Task gate descriptor

- A procedure that has access to a task gate has the power to cause a task switch, just as a procedure that has access to a TSS descriptor.
- The advantages of task gates over TSS descriptors are:
  1. There may be several task gates that select the single TSS descriptor.
  2. They provide selective access to tasks.
  3. They support an interrupt or exception to cause a task switch.

- Fig. 7.4(b) illustrates how both a task gate in an LDT and a task gate in the IDT can identify the same task.

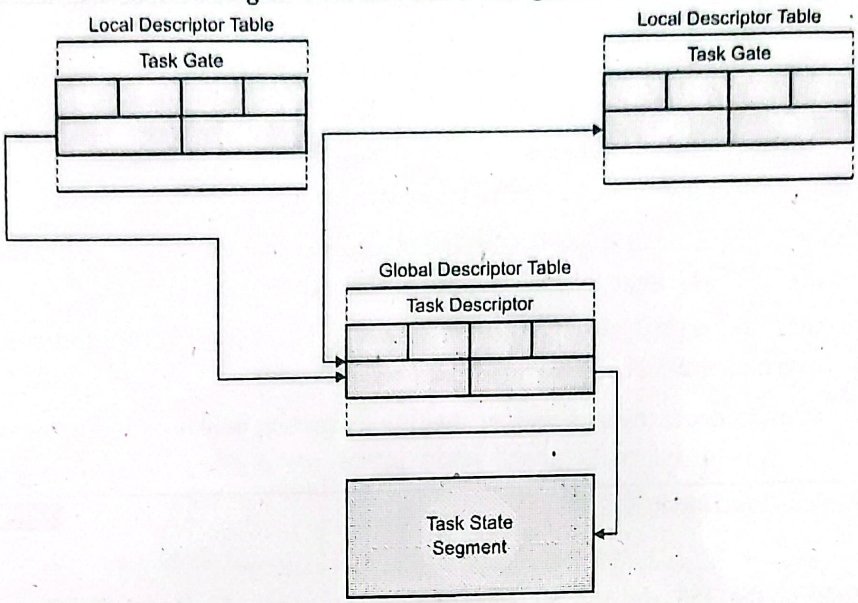

Fig. 7.4(b) : Task gate indirectly identifies task

Q. 6 How does 80386 do a Task Switch ? SPPU - May 12, 4 Marks

Ans. :

- The 80386 switches execution to another task in any of four cases:
  1. The current task executes a JMP or CALL that refers to a TSS descriptor.
  2. The current task executes a JMP or CALL that refers to a task gate.
  3. An interrupt or exception vectors to a task gate in the IDT.
  4. The current task executes an IRET when the NT flag is set.
- To cause a task switch, a JMP or CALL instruction can refer either to a TSS descriptor or to a task gate. The effect is the same in either case: the 80386 switches to the indicated task.
- An exception or interrupt causes a task switch when it vectors to a task gate in the IDT.
- If it vectors to an interrupt or trap gate in the IDT, a task switch does not occur.
- Whether invoked as a task or as a procedure of the interrupted task, an interrupt handler always returns control to the interrupted procedure in the interrupted task.
- If the NT flag is set, however, the handler is an interrupt task, and the IRET switches back to the interrupted task.
- A task switching operation involves the following steps:
  1. Checking that the current task is allowed to switch to the designated task. Data-access privilege rules apply in the case of JMP or CALL instructions.
  2. Checking that the TSS descriptor of the new task is marked present and has a valid limit.
  3. Saving the state of the current task. The processor finds the base address of the current TSS cached in the task register.

     It copies the registers into the current TSS (EAX, ECX, EDX, EBX, ESP, EBP, ESI,EDI, ES, CS, SS, DS, FS, GS, and the flag register).

4. Loading the task register with the selector of the called task's TSS descriptor, marking the incoming task's TSS descriptor as busy, and setting the TS (task switched) bit of the CR0.
5. Loading the called task's state from its TSS and resuming execution. The registers loaded are the LDT register; the flag register; the general registers EIP, EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI; the segment registers ES, CS, SS, DS, FS, and GS; and PDBR.

- The state of the caller task is always saved when a task switch occurs.
- If execution of that task is resumed, it starts after the instruction that caused the task switch. The registers are restored to the values they held when the task stopped executing.
- Every task switch sets the TS (task switched) bit in the CR0. The TS flag is useful to systems software when a coprocessor (such as a numeric coprocessor 80387) is present.
- The TS bit when set indicates to the coprocessor that the its context may not correspond to the current 80386 task.

Q. 7 Write a short note on "Tast Linking" **SPPU - Dec. 19, 4 Marks**

Ans. :

- The back-link field of the TSS and the NT (nested task) bit of the flag word together allow the 80386 to automatically return to a task that CALLed another task or was interrupted by another task.
- When a CALL instruction, an interrupt instruction, an external interrupt, or an exception causes a switch to a new task, the 80386 automatically fills the back-link of the new TSS with the selector of the outgoing task's TSS and sets the NT bit in the new task's flag register.
- The NT flag indicates whether the back-link field is valid.
- The new task releases control by executing an IRET instruction. When executing the IRET instruction, the 80386 checks the NT flag. If NT is set, the 80386 switches back to the task selected by the back-link field.

Busy Bit

- The B-bit (busy bit) of the TSS descriptor ensures the integrity of the back-link. A chain of back-links may grow to any length as interrupt tasks interrupt other interrupt tasks or as called tasks call other tasks. The busy bit ensures that the CPU can detect any attempt to create a loop. A loop would indicate an attempt to reenter a task that is already busy; however, the TSS is not a reentrable resource.
- The processor uses the busy bit as follows:
  1. When switching to a task, the processor automatically sets the busy bit of the new task.
  2. When switching from a task, the processor automatically clears the busy bit of the old task if that task is not to be placed on the back-link chain (i.e., the instruction causing the task switch is JMP or IRET). If the task is placed on the back-link chain, its busy bit remains set.
  3. When switching to a task, the processor signals an exception if the busy bit of the new task is already set.
- By these actions, the processor prevents a task from switching to itself or to any task that is on a back-link chain, thereby preventing invalid reentry into a task.
- The busy bit is effective even in multiprocessor configurations, because the processor automatically asserts a bus lock when it sets or clears the busy bit. This action ensures that two processors do not invoke the same task at the same time.

Chapter 8 : Virtual 8086 Mode

Q.1 What is Virtual mode ? **SPPU - May 15, 2 Marks**

OR What is V-86 mode? Explain in detail. **SPPU - May 16, 6 Marks**

OR Which bit of EFLAGs indicate V86 mode? Explain , how hardware and software cooperate with each other to emulate V86 mode? **SPPU - May 17, 4 Marks**

OR Explain features of "Virtual 8086 mode" . **SPPU - May 19 , 3 Marks**

Ans. :

- The 80386 supports execution of one or more 8086, 8088, 80186, or 80188 programs in an 80386 protected-mode environment.
- An 8086 program runs in this environment as part of a V86 (virtual 8086) task.
- V86 tasks take advantage of the hardware support of multitasking offered by the protected mode.
- Not only can there be multiple V86 tasks, each one executing an 8086 program, but V86 tasks can be multi-programmed with other 80386 tasks
- The Virtual 8086 mode is the last of the three main operating modes of the 80386 microprocessor.
- The purpose of a V86 task is to form a "virtual machine" with which to execute an 8086 program.
- A complete virtual machine consists not only of 80386 hardware but also of systems software.
- Thus, the emulation of an 8086 is the result of cooperation between hardware and software:
- The hardware provides a virtual set of registers (via the TSS), a virtual memory space (the first megabyte of the linear address space of the task), and directly executes all instructions that deal with these registers and with this address space.
- The software controls the external interfaces of the virtual machine (I/O, interrupts, and exceptions) in a manner consistent with the larger environment in which it executes.
- In the case of I/O, software can choose either to emulate I/O instructions or to let the hardware execute them directly without software intervention.
- The processor executes in V86 mode when the VM (virtual machine) bit in the EFLAGS register is set. The processor tests this flag under two general conditions:
  1. When loading segment registers to know whether to use 8086-style address formation
  2. When decoding instructions to determine which instructions are sensitive to IOPL.
- Except for these two modifications to its normal operations, the 80386 in V86 mode operated much as in protected mode.

Q. 2 Write a short note on "Virtual 8086 mode". SPPU - Dec. 19 , 3 Marks

Ans. :

- In case of 80386, the tasks that require real mode operation, become very difficult to manage.
- The 80386 needs to be restarted to return in real mode from protected mode.
- VM86 or Virtual 86 mode makes it easy for 80386 to execute tasks that are 8086 based operation.
- VM86 mode operation also supports Multiple 8086 real-mode software applications execution at one time.
- We know that once the 80386 processor operates in protected mode, it cannot return easily back to the real mode. For returning back to real mode it needs a reset operation.
- Hence, virtual mode provides the advantage of running 8086 real mode programs while the processor is in protected mode in multitasking environment.

Q. 3 Write a short note on "Virtual 8086 mode". SPPU - Dec. 19, 3 Marks

Ans. :

- The register set available in V86 mode includes all the registers defined for the 8086 plus the new registers introduced by the 80386: FS, GS, debug registers, control registers, and test registers.
- New instructions that explicitly operate on the segment registers FS and GS are available, and the new segment-override prefixes can be used to cause instructions to utilize FS and GS for address calculations.
- Instructions can utilize 32-bit operands through the use of the operand size prefix.
- 8086 programs running as V86 tasks are able to take advantage of the new applications-oriented instructions added to the architecture by the introduction of the 80186/80188, 80286 and 80386:
  1. PUSH immediate data
  2. Push all and pop all (PUSHA and POPA)
  3. Multiply immediate data
  4. Shift and rotate by immediate count
  5. String I/O
  6. ENTER and LEAVE
  7. BOUND
  8. LSS, LFS, LGS instructions
  9. Long-displacement conditional jumps
  10. Single-bit instructions
  11. Bit scan
  12. Double-shift instructions
  13. Byte set on condition
  14. Move with sign/zero extension
  15. Generalized multiply

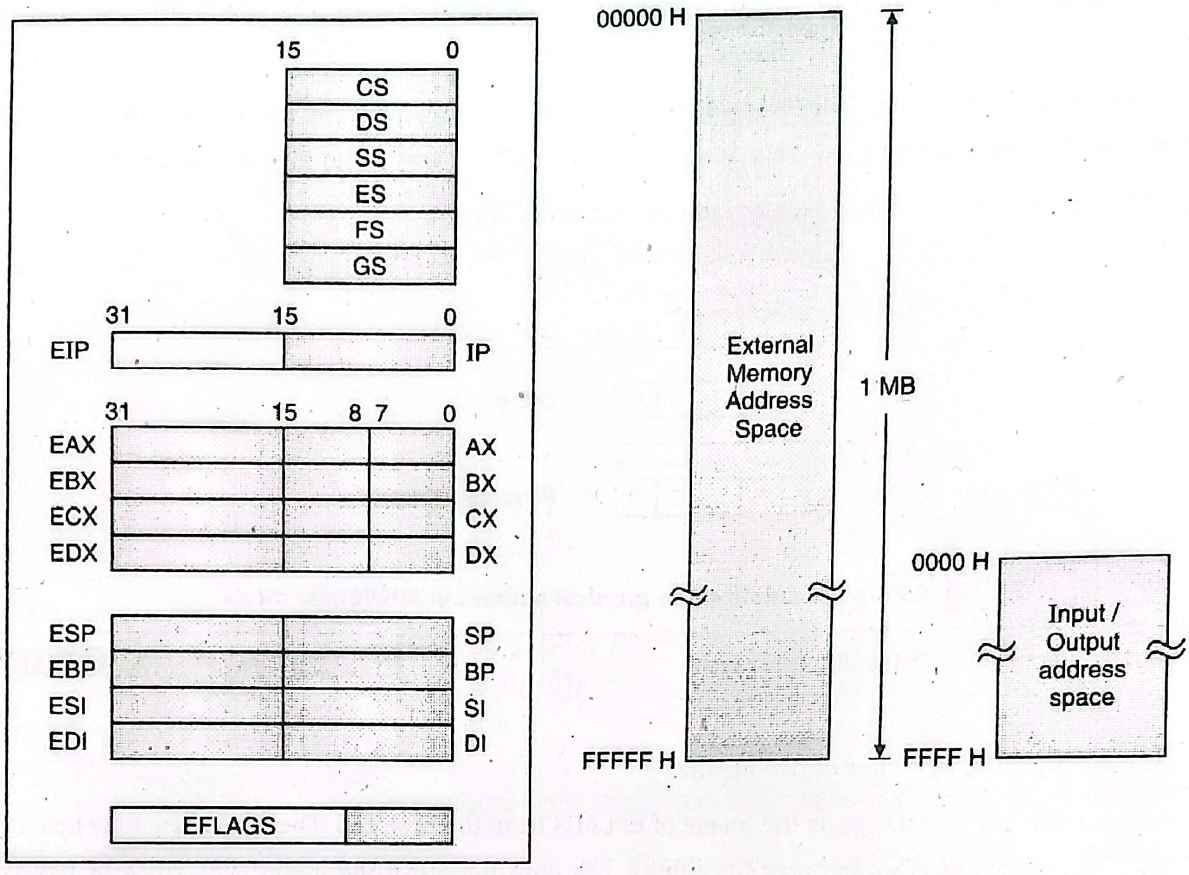

Fig. 8.1 : Programmers model of 80386 in virtual mode

Q. 4 With neat diagram explain the process of linear address formation in V86 mode. **SPPU - Dec. 17, 6 Marks**

OR Write a short note on "Virtual 8086 mode". **SPPU - Dec. 19 , 3 Marks**

Ans. :

- In V86 mode, the 80386 processor does not interpret 8086 selectors by referring to descriptors; instead, it forms linear addresses as an 8086 would.
- It shifts the selector left by four bits to form a 20-bit base address. The effective address is extended with four high-order zeros and added to the base address to create a linear address as Fig. 8.2(a) illustrates.

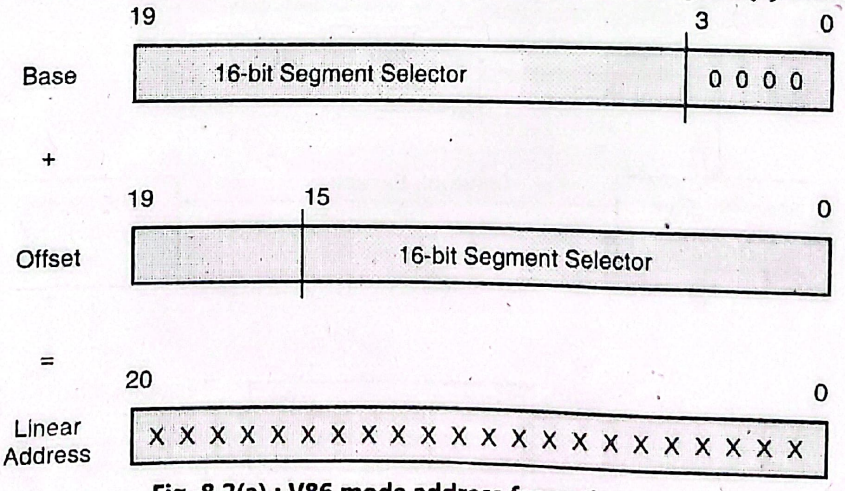

Fig. 8.2(a) : V86 mode address formation

- Because of the possibility of a carry, the resulting linear address may contain up to 21 significant bits.

- An 8086 program may generate linear addresses anywhere in the range 0 to 10FFEFH (one megabyte plus approximately 64 Kbytes) of the task's linear address space.
- V86 tasks generate 32-bit linear addresses. While an 8086 program can only utilize the low-order 21 bits of a linear address, the linear address can be mapped via page tables to any 32-bit physical address
- Let DS = B000H and offset = 5F00H then the physical address will be,

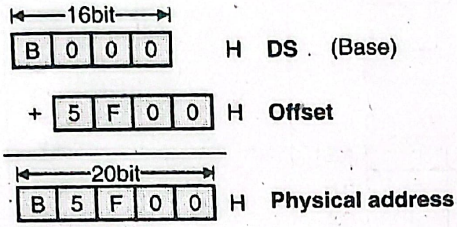

Fig. 8.2(b) : Calculating the physical address in 80386 real mode

Q. 5 With neat diagram explain "Entering V86 mode". SPPU - Dec. 17 , May 18, May 19 ,6 Marks

Ans. :

The processor can enter V86 by either of two means:

1. A task switch to an 80386 task loads the image of EFLAGS from the new TSS. The TSS of the new task must be an 80386 TSS, not an 80286 TSS, because the 80286 TSS does not store the high-order word of EFLAGS, which contains the VM flag. A value of one in the VM bit of the new EFLAGS indicates that the new task is executing 8086 instructions; therefore, while loading the segment registers from the TSS, the processor forms base addresses as the 8086 would.
2. An IRET from a procedure of an 80386 task loads the image of EFLAGS from the stack. A value of one in VM in this case indicates that the procedure to which control is being returned is an 8086 procedure. The CPL at the time the IRET is executed must be zero, else the processor does not change VM.

- Fig. 8.3 shows entering and leaving VM86 mode.

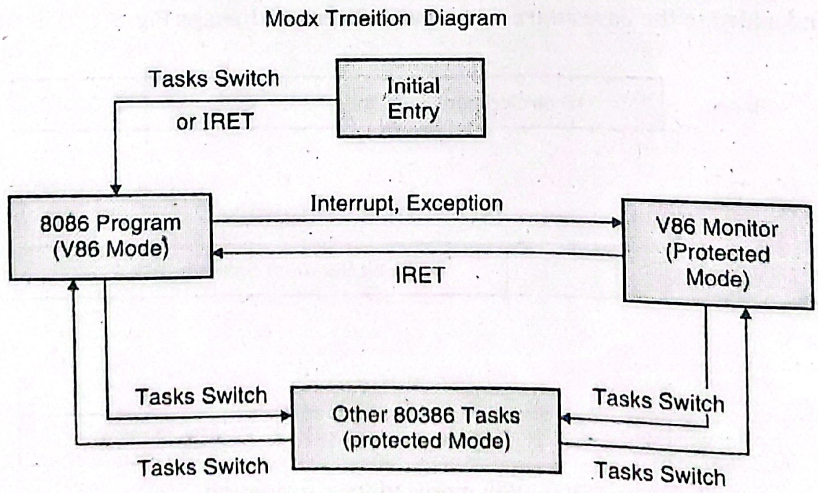

Fig. 8.3 : Entering and leaving virtual 8086 mode

Q. 6 With neat diagram explain "leaving V86 mode" . **SPPU - Dec. 17, May 18, May 19 ,6 Marks**

Ans. :

The processor leaves V86 mode when an interrupt or exception occurs. There are two cases:

1. The interrupt or exception causes a task switch. A task switch from a V86 task to any other task loads EFLAGS from the TSS of the new task. If the new TSS is an 80386 TSS and the VM bit in the EFLAGS image is zero or if the new TSS is an 80286 TSS, then the processor clears the VM bit of EFLAGS, loads the segment registers from the new TSS using 80386-style address formation, and begins executing the instructions of the new task according to 80386 protected-mode semantics.
2. The interrupt or exception vectors to a privilege-level zero procedure. The processor stores the current setting of EFLAGS on the stack, then clears the VM bit. The interrupt or exception handler, therefore, executes as "native" 80386 protected-mode code. If an interrupt or exception vectors to a conforming segment or to a privilege level other than three, the processor causes a Systems software does not manipulate the VM flag directly, but rather manipulates the image of the EFLAGS register that is stored on the stack or in the TSS. The V86 monitor sets the VM flag in the EFLAGS image on the stack or in the TSS when first creating a V86 task. Exception and interrupt handlers can examine the VM flag on the stack. If the interrupted procedure was executing in V86 mode, the handler may need to invoke the V86 monitor.

Q. 7 State and explain the difference between all operating modes of 80386. **SPPU - Dec. 14, May 15, Dec. 15, 6 Marks.**

Ans. :

Table 8.1

Parameter	Real mode	Protected Mode	Virtual mode
General	It is the default mode of the processor when it is switched ON. It is similar to 8086 and hence is called as real mode or real addressing mode	To support multi-tasking 80386 has a special mode called as protected mode. The main reason of the name as "protected", is that the tasks are to be protected from others. It should not happen that a task accesses the code or data of another task, causing problems to the other task.	Switching between real and protected mode is quite complicated, as it requires a restart. Virtual mode allows 8086 tasks to be executed without restarting the processor.
Use	It is used to execute the 8086 tasks	It is used to execute multi-tasking based tasks of 80386	It is used to execute 8086 tasks without restarting the processor i.e. virtually going into real addressing mode
Register Access	All general purpose as well as memory management registers can be accessed	In protected mode also all the registers are accessible.	Some registers such as memory management registers, test registers, debug registers are not accessible in Virtual mode. The general purpose registers are accessible in virtual mode

Parameter	Real mode	Protected Mode	Virtual mode
Memory addressing	In real mode memory upto 1MB + 64KB -16 bytes can be accessed	In protected mode entire 4GB memory is accessible	In real mode memory upto 1MB + 64KB -16 bytes can be accessed
Entering the mode	On restart or RESET, 80386 enters in this mode	Enabling of Protection Enable (PE) bit of CR0 makes the processor enter into protected mode	Enabling of VM bit in the EFLAG register makes the processor enter into Virtual mode
Leaving the mode	When PE bit is set, 80386 enters Protected mode and exits from the real mode	Entering into real mode requires disabling of PE bit and restarting the processor; and hence exiting the protected mode	Exiting from the virtual mode is via an interrupt or exception

Q. 8 Draw and explainstructure of V86 task. **(6 Marks)**

Ans. :

- A V86 task consists partly of the 8086 program to be executed and partly of 80386 "native mode" code that serves as the virtual-machine monitor.
- The task must be represented by an 80386 TSS (not an 80286 TSS).
- The processor enters V86 mode to execute the 8086 program and returns to protected mode to execute the monitor or other 80386 tasks.
- To run successfully in V86 mode, an existing 8086 program needs the following:
  - A V86 monitor.
  - Operating-system services.
- The V86 monitor is 80386 protected-mode code that executes at privilege-level zero.
- The monitor consists primarily of initialization and exception-handling procedures.
- As for any other 80386 program, executable-segment descriptors for the monitor must exist in the GDT or in the task's LDT.
- The linear addresses above 10FFEFH are available for the V86 monitor, the operating system, and other systems software.
- The monitor may also need data-segment descriptors so that it can examine the interrupt vector table or other parts of the 8086 program in the first megabyte of the address space
- In general, there are two options for implementing the 8086 operating system:

1. The 8086 operating system may run as part of the 8086 code. This approach is desirable for any of the following reasons :
   - The 8086 applications code modifies the operating system.
   - There is not sufficient development time to reimplement the 8086 operating system as 80386 code.

2. The 8086 operating system may be implemented or emulated in the V86 monitor. This approach is desirable for any of the following reasons:
   - Operating system functions can be more easily coordinated among several V86 tasks.
   - The functions of the 8086 operating system can be easily emulated by calls to the 80386 operating system.

- Regardless of the approach chosen for implementing the 8086 operating system, different V86 tasks may use different 8086 operating systems.
- The I/O port permission map can be used to control the access to I/O ports.

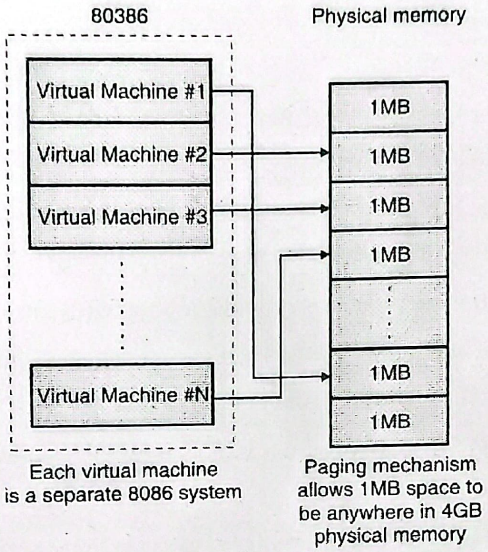

Fig. 8.4 : Concept of virtual machines running on 80386

Q. 9 How interrupts and exceptions are handled in virtual 86 mode. **(4 Marks)**

Ans. :

- When the processor is executing 8086 code in a V86 task, the instructions PUSHF, POPF, and IRET are sensitive to IOPL so that the V86 monitor can control changes to the interrupt-enable flag (IF).
- Other instructions that affect IF (STI and CLI) are IOPL sensitive both in 8086 code and in 80386/80386 code.
- Many 8086 programs that were designed to execute on single-task systems set and clear IF to control interrupts. However, when these same programs are executed in a multitasking environment, such control of IF can be disruptive.
- If IOPL is less than three, all instructions that change or interrogate IF will trap to the V86 monitor. The V86 monitor can then control IF in a manner that both suits the needs of the larger environment and is transparent to the 8086 program.
- The 8086 virtual mode handles hardware, software interrupts, aborts, traps, faults in a different way than they are handled in the protected mode.
- By examining the VM bit of EFLAGs register, the 80386 processor determines whether the interrupt comes from a protected mode or a real mode.
- In the virtual 8086 mode exceptions and interrupts process in the 8086 ISR or the protected mode ISR.

Unit VI : Interrupts, Exceptions, and Introduction to Microcontrollers

Chapter 9 : Exceptions and Interrupts

Q. 1 Explain "How 80386 indentifies interrupts". SPPU - May 19, 4 Marks

OR List different sources of interrupts. SPPU - Dec. 19, 3 Marks

Ans. :

- When many I/O devices are connected to a microprocessor based system, one or more than one of the I/O devices may request for service at any time.
- When a device requests, the microprocessor suspends the execution of the current program and gives service to the I/O devices. This feature is called as **interrupt**.
- It is an external asynchronous input or an instruction that informs the microprocessor to complete the instruction that it is currently executing and fetch a new routine in order to offer service to the I/O device.
- Once the I/O device is serviced, the microprocessor will continue with the execution of its normal program.
- **Definition :** It is a mechanism by which an I/O device (Hardware interrupt) or an instruction (Software interrupt) can suspend the normal execution of the processor and get itself serviced.
- **Interrupt service routine (ISR) :** A small program or a routine that when executed services the corresponding interrupting source is called as an ISR
- **Vectored/Non-vectored interrupt :** If the ISR address or some information related to the ISR address, of an interrupt is to be taken from the interrupting source itself, it is called as a non-vectored interrupt; else it is a vectored interrupt.
- **Maskable/Non-maskable interrupt :** Interrupt that can be masked (disabled) and unmasked (enabled) by the programmer is called as maskable interrupt else it is a non maskable interrupt.

Interrupt sources

- In 80386, we have three sources of interrupt.

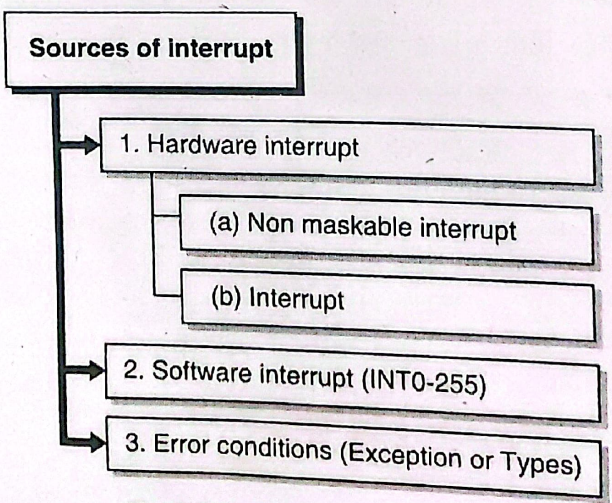

Fig. 9.1(a) : Sources of interrupt

1. Hardware interrupt

- In this type of interrupt, physical pins are provided in the chip. This interrupts are generated by changing the logic levels on the 80386's interrupt pins.
- In 80386 we have two interrupt pins :
  a. NMI (Non Maskable Interrupt).
  b. INTR.
- To interrupt the 80386 processor we have to apply signal to these pins.

(a) Non maskable interrupt

- **NMI** is **non maskable** interrupt i.e. microprocessor has to service this interrupt, it cannot avoid it. It can be edge triggered.
- This interrupt cannot be disabled.
- This interrupt is mainly used in critical situations.
- This interrupt always causes an interrupt sequence to be executed.

(b) Interrupt

- The **INTR** is a maskable interrupt.
- This interrupt can be enabled and disabled using the CLI and STI instruction.
- This interrupt is level sensitive.
- If we want more number of interrupts to be processed then we assign priority to the interrupts.
- If IE flag in Eflag register is '0', microprocessor will not recognize interrupt available on the INTR pin.

2. Software interrupt

- Software interrupts are generated directly by executing the program.
- The INT instruction of the 80386 can be used to do one of the 256 interrupts (Type 0-255).
- The interrupt type is specified by the number as a part of the instruction. e.g. INT 0 instruction can be used to send execution to a divide by zero interrupt service routine.
- With the help of these software interrupts we can call the routines from different programs in the system e.g. BIOS. The BIOS routines are called with INT instructions.

3. Error conditions (Exception or Types)

- We know that 80386 supports division, multiplication, addition etc. Suppose by mistake if user asks microprocessor to divide any number by ZERO, then you know that dividing any number by ZERO produces answer '∞ ***(infinity)***'. So in this case microprocessor will generate an interrupt ***"Automatically"*** and interrupt current execution.
- Thus, internally generated errors produces an interrupt for microprocessor, normally referred as ***"TYPE"*** by Intel engineer and referred as ***"Exception"*** by the engineer. Thus we conclude that 80386 have a simple and versatile interrupt system.
- Every interrupt is assigned a ***"interrupt type"*** that identifies it to the CPU.

- The 80386 can handle upto 256 different interrupt types. Interrupts may be initiated by devices external to the CPU; in addition, they also may be triggered by software interrupt introductions and under certain condition, by the CPU itself.
- Fig. 9.1(b) shows interrupt sources for 80386.
- As shown in Fig. 9.1(b) 80386 have two lines that external device may use to signal interrupts.
- The INTR line is usually driven by an Intel 8259A (PIC), which in turn connected to the devices that need interrupt services.

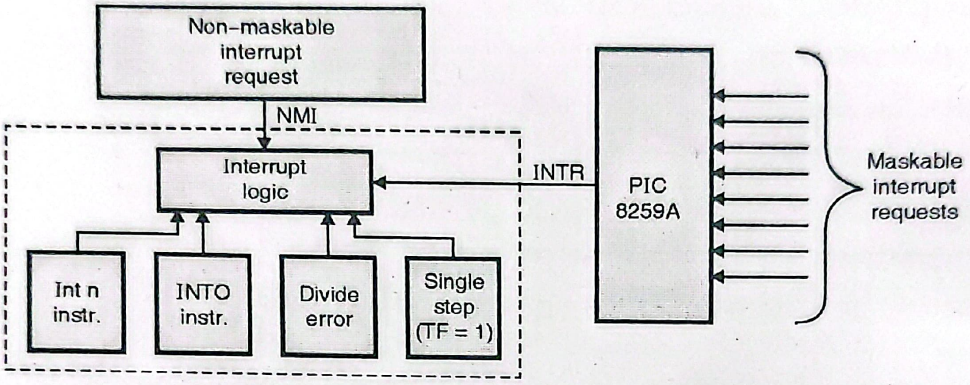

Fig. 9.1(b) : Interrupt sources

Q. 2 Explain different way by which 80386 can enable and disable interrupts. **SPPU - Dec. 19, 3 Marks**

Ans. :

- The 80386 microprocessor services interrupts and exceptions after the completion of an instruction and before the execution of the next instruction begins.
- In case of string instructions, the interrupts and exceptions can be between repetitions.
- At the instruction boundaries some flag settings or conditions can enable/disable interrupts and exceptions.
- These conditions are :

1. NMI masks NMI

While an NMI handler is executing, the processor ignores further interrupt signals at the NMI pin until the next IRET instruction is executed.

2. IE mask INTR

- The external interrupts that are controlled through the INTR pin are controlled by the IE (Interrupt Enable) flag.
- If IE = 1, the INTR pin is enabled and if IE = 0, the INTR pin is disabled.
- STI and CLI instructions can be used to set or clear the Interrupt Enable Flag. However, these instructions can be executed only if the CPL ≤ IOPL, otherwise an exception is generated.

3. RF masks debug faults

- The RF bit in EFLAGS controls the recognition of debug faults. This permits debug faults to be raised for a given instruction at most once, no matter how many times the instruction is restarted.

4. MOV or POP to SS Masks some interrupts and Exceptions

- The instructions,

  MOV SS, AX

  MOV ESP, stacktop

  are used for modifying the stack segments.

- If an interrupt or exception is processed after SS has been changed but before ESP has received the corresponding change, the two parts of the stack pointer SS:ESP are inconsistent for the duration of the interrupt handler or exception handler.
- To prevent this situation, the 80386, after both a MOV to SS and a POP to SS instruction, inhibits NMI, INTR, debug exceptions, and single-step traps at the instruction boundary following the instruction that changes SS. Some exceptions may still occur; namely, page fault and general protection fault. Always use the 80386 LSS instruction, and the problem will not occur.

Q. 3 Explain different types of exceptions in 80386 with suitable example of each. **SPPU - May 16, 6 Marks**

OR Define "Faults" **SPPU -May 19, 2 Marks**

Ans. :

- Exceptions are generated by internal events while interrupts are generated by external events.
- In protected mode of the 80386 processor, more internal conditions can initiate an internal interrupt or exception.
- Exceptions are divided into three groups depending on the way they are reported, and whether or not restart of the instruction causing an exception is supported.
- These groups are as follows :

(i) Faults

What is fault ?

- Faults are exceptions that are reported "before" the instruction causing the exception.
- Faults are either detected before the instruction begins to execute, or during execution of the instruction. If detected during the instruction, the fault is reported with the machine restored to a state that permits the instruction to be restarted.

(ii) Traps

What is Trap ?

- A trap is an exception that is reported at the instruction boundary immediately after the instruction in which the exception was detected.

(iii) Aborts

What are aborts ?

- An abort is an exception that permits neither precise location of the instruction causing the exception nor restart of the program that caused the exception.
- Aborts are used to report severe errors, such as hardware errors and inconsistent or illegal values in system tables.

Q. 4 Explain the different exception conditions-Faults, Traps and Aborts. **SPPU - May 17, 6 Marks**

Ans. :

Table 9.1 : Difference between interrupt, fault, trap and abort

Interrupt	Fault	Trap	Abort
Interrupts are externally generated event	Faults are internally generated event	Faults are internally generated event	Faults are internally generated event
It is a mechanism by which an external device can suspend the normal execution of the processor and get itself serviced	Faults are exceptions that are reported "before" the instruction causing the exception.	A trap is an exception that is reported at the instruction boundary immediately after the instruction in which the exception was detected	An abort is an exception that permits neither precise location of the instruction causing the exception nor restart of the program that caused the exception
After servicing the interrupt the 80386 will execute the next sequential instruction, before witch the interrupt occurred.	Faults are either detected before the instruction begins to execute, or during execution of the instruction. If detected during the instruction, the fault is reported with the machine restored to a state that permits the instruction to be restarted.	The instruction that caused the Trap cannot be restarted.	Aborts are used to report severe errors, such as hardware errors and inconsistent or illegal values in system tables.
Example : INT 32-255.	**Example :** INT 0, INT 5, INT 6.	**Example :** INT 2, INT 3, INT 4.	**Example :** INT 8, INT9.

Q. 5 How are interrupts handled in protected mode ? Explain with the help of neat diagram. **SPPU - Nov. 12, May 14, 8 Marks**

OR Explain the procedure of handling interrupts in protected mode. **SPPU - Dec. 18, 6 Marks**

Ans. :

- Real mode uses a 1 KB **Interrupt Vector Table (IVT)** starting at address 00000H. Each 4 byte entry in the IVT consists of a CS : IP pair that specifies the address of the first instruction in the interrupt service routine (ISR).
- An 8 bit vector number is shifted two bits to the left to form an index into the IVT.
- The protected mode depends on the **Interrupt Descriptor Table (IDT)** to support the interrupts and exceptions.
- The IDT comprises 8 byte gate descriptors for task, trap or interrupt gates.
- The IDT has a maximum size of 256 descriptors. The size of the IDT is controlled by a 16 bit limit value stored in the **Interrupt Descriptor Table Register (IDTR)**

- The interrupt descriptor table contains gate descriptors and not vectors. This table can be located anywhere in the memory. It is an array of 8 byte descriptors like the GDT and LDT.
- There are 256 gate descriptors in interrupt descriptor table as shown in Fig. 9.2 However, in case of protected mode, there are 8 bytes for each interrupt, e.g. Gate-0 is located at address (IDT + 0) to (IDT + 7) and Gate-255 is located at address (IDT + 7F8) to (IDT + 7FF).
- If all 256 gates are not needed for an application, the limit can be set to a value lower than 7FF H to minimize the amount of memory for the table.
- The protected mode interrupt descriptor table can reside anywhere in the physical address space.
- The location and size of the interrupt descriptor table are again defined by the contents of IDTR as shown in Fig. 9.2.
- The Interrupt Descriptor Table Register (IDTR) is a 48 bit register.
- It consists of 32-bit BASE gives the starting point of the table in memory and 16-bit LIMIT gives the size of the interrupt descriptor table.
- Whenever an interrupt or exception occurs, an identifier or vector number is multiplied by 8 and is added to the IDT base address that is stored in the IDTR.
- The instructions LIDT (Load Interrupt Descriptor Table Register) and SIDT (Store Interrupt Descriptor Table Register) are used with the IDTR. The LIDT is a privileged instruction and is executed whenever the CPL is zero. The SIDT is a non-privileged instruction. It can be executed any time.
- Each instruction uses a single operand that specifies the address of the 48 bit memory word. The 6 byte word is used to change the location of the IDT or find its current address.

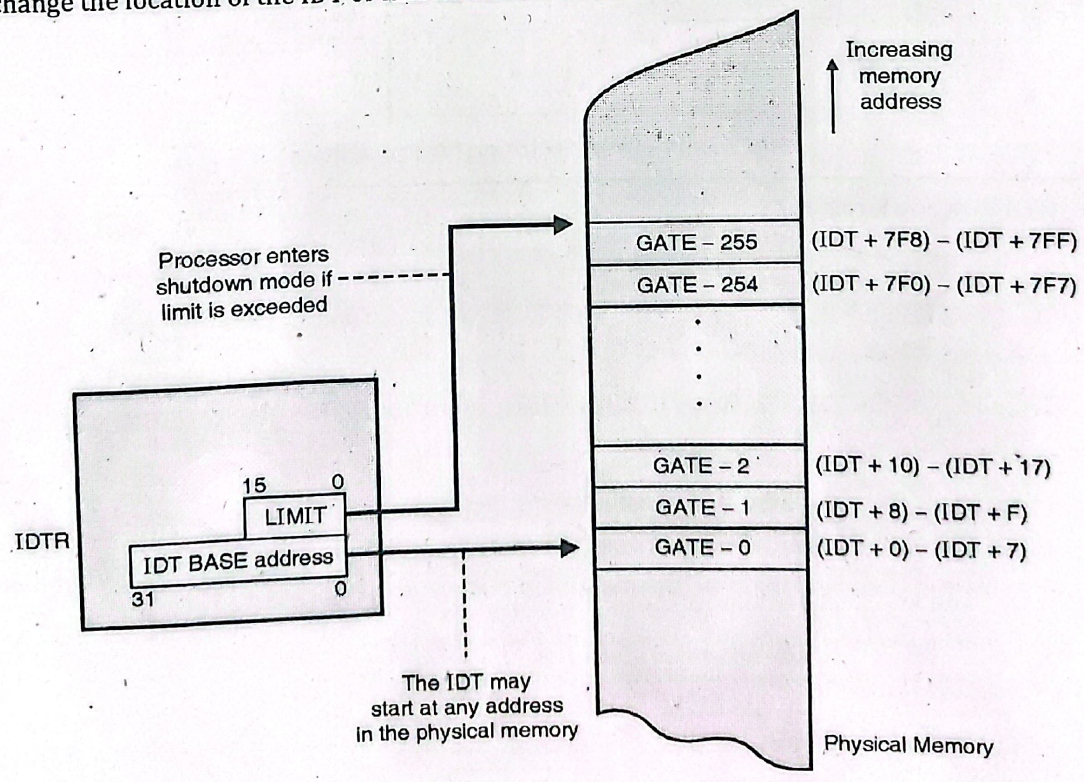

Fig. 9.2 : The protected mode interrupt descriptor table

Q. 6 Explain what happens when an interrupt calls a procedure as an interrupt handler. **SPPU - May 17, Dec. 17, 6 Marks**

Ans. :

- Just as a CALL instruction can call either a procedure or a task, so an interrupt or exception can "call" an interrupt handler that is either a procedure or a task.
- When responding to an interrupt or exception, the processor uses the interrupt or exception identifier to index a descriptor in the IDT.
- If the processor indexes to an interrupt gate or trap gate, it invokes the handler in a manner similar to a CALL to a call gate.
- If the processor finds a task gate, it causes a task switch in a manner similar to a CALL to a task gate
- An interrupt gate or trap gate points indirectly to a procedure which will execute in the context of the currently executing task as illustrated by Fig. 9.3.
- The selector of the gate points to an executable-segment descriptor in either the GDT or the current LDT.
- The offset field of the gate points to the beginning of the interrupt or exception handling procedure.

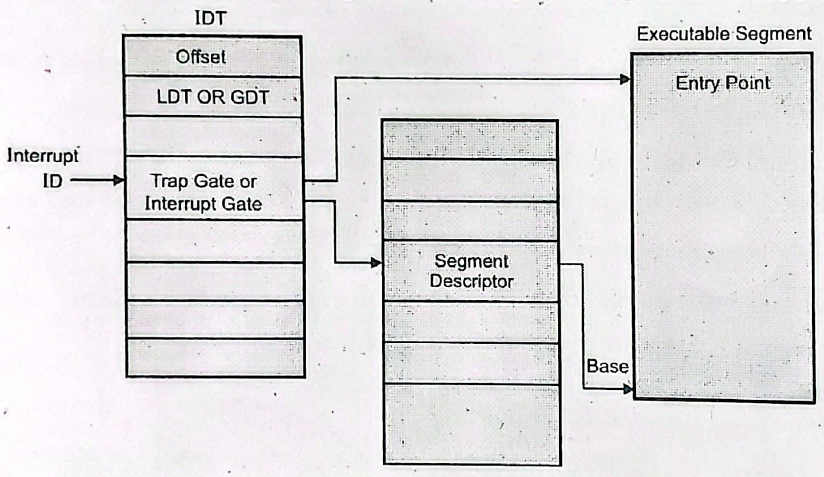

Fig. 9.3 : Interrupt vectoring for procedures

Q. 7 What is IDT and how to locate IDT? **SPPU - May 17, 4 Marks**

Ans. :

- The interrupt descriptor table (IDT) associates each interrupt or exception identifier with a descriptor for the instructions that service the associated event.
- Like the GDT and LDTs, the IDT is an array of 8-byte descriptors. Unlike the GDT and LDTs, the first entry of the IDT maycontain a descriptor.
- To form an index into the IDT, the processor multiplies the interrupt or exception identifier by eight. Because there are only 256 identifiers, the IDT need not contain more than 256 descriptors. It can contain fewer than 256 entries; entries are required only for interrupt identifiers that are actually used.
- The IDT may reside anywhere in physical memory. As Fig. 9.4(a) shows, the processor locates the IDT by means of the IDT register (IDTR). The instructions LIDT and SIDT operate on the IDTR.
- Both instructions have one explicit operand: the address in memory of a 6-byte area. Fig. 9.4(b) shows the format of this area.

- LIDT (Load IDT register) loads the IDT register with the linear base address and limit values contained in the memory operand.
- This instruction can be executed only when the CPL is zero. It is normally used by the initialization logic of an operating system when creating an IDT. An operating system may also use it to change from one IDT to another.
- SIDT (Store IDT register) copies the base and limit value stored in IDTR to a memory location. This instruction can be executed at any privilege level.

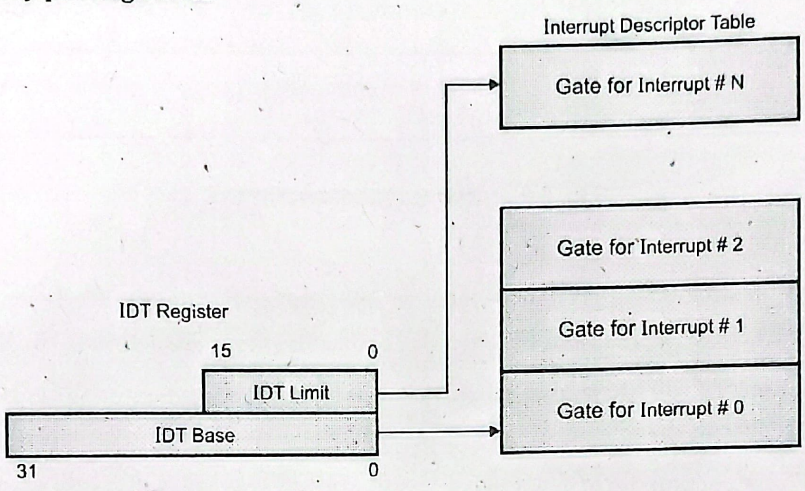

Fig. 9.4(a) : IDT register and table

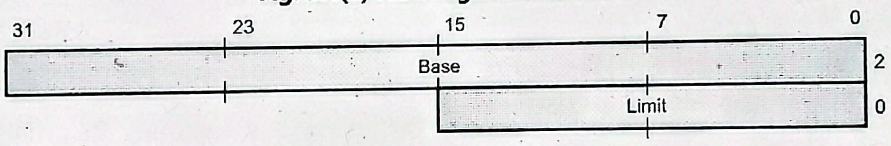

Fig. 9.4(b) : Pseudo Descriptor format for LIDT and SIDT

Q. 8 Draw the format of trap gate descriptor. **SPPU - Dec. 18, 2 Marks**

Ans. :

- **Interrupt procedure :** Whenever an exception occurs and the handler selects a trap gate descriptor in the IDT, 80386 saves its state onto the stack and then an intersegment JMP is taken to the selector : offset specified in the trap gate.

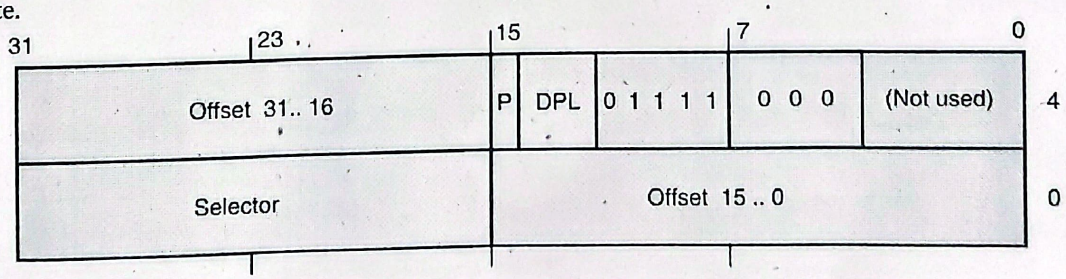

Fig. 9.5 : Trap gates descriptors

- Fig. 9.5 shows the format of trap gate descriptor.
- Also the access rights byte is verified. The selector : offset mentioned in the trap gate should point to an executable code segment only. When the selector indicates a code descriptor in LDT then that descriptor should be present in current task's LDT.
- Whatever scheme we use, we must be sure that the exception handler's code segment descriptor has been marked "present". And if it is marked not present then a not-present fault (exception 11 or 14) is generated.

Q. 9 Draw the format of interrupt gate. SPPU - Dec. 18, 2 Marks

Ans. :

- Fig. 9.6 shows the format of interrupt gate descriptor.
- This gate is same as the trap gate but has one significant difference.

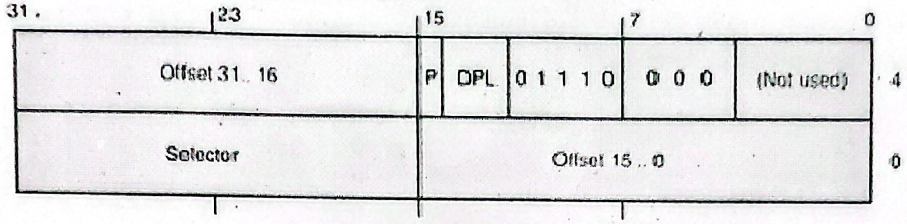

Fig. 9.6 : Interrupt gate descriptors

Interrupt procedure

- Whenever an exception occurs and handler selects an interrupt gate, the 80386 clears the Interrupt Flag (IF) after it has pushed the return address and EFLAGs. But before the first instruction of the ISR is executed.
- When the IF flag is cleared, 80386 masks all further hardware initiated interrupts. Until the ISR is finished executing and IRET is executed.
- On execution of IRET instruction the original image of EFLAGs register is popped off stack.

Q. 10 Draw the format of task gate. SPPU - Dec. 18, 2 Marks

Ans. :

- We know that the Task State Segment Descriptors give the information regarding the privilege level of the task through the 2 DPL bits.
- Fig. 9.7 shows the format of task gate descriptor.
- If a task of lower priority is being presently executed then it may result in privilege violation.
- The use of **Task gates** is another way of achieving task switching.
- Task Gate is very much like the call gate. The task gate contains a selector to a TSS of new task as well as the access rights byte.

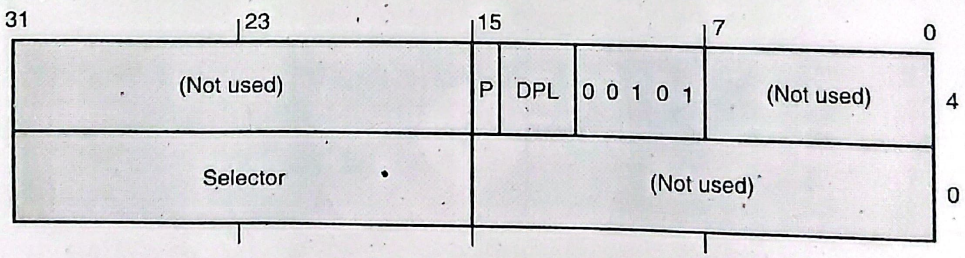

Fig. 9.7 : Task gates descriptors

- Interrupt procedure : In this method of task switching, the task can be invoked indirectly by jumping or calling a task gate.
- This method is used to transfer control to a task at the RPL that is higher than CPL. The application running currently should be privileges enough to give a CALL to task gate.

- The rule is :

Max (CPL, RPL) ≤ task gate DPL

- In this case, the instruction includes a selector that points to the task gate, which is in LDT or IDT instead of a task state selector.
- And the TSS selector held in this gate is loaded into TR to select the TSS and start the task.
- The TSS selector of current task that now becomes dormant, will be copied into the Backlink field of the new task. The NT bit in the EFLAGs register will be set. An IRET instruction is executed on completion of the exception handling task. Depending on the Backlink information, the 80386 processor will activate the interrupt task again.

Q. 11 What is the difference between them interrupt gate and trap gate? **SPPU - Dec. 18, 6 Marks**

Ans. :

Table 9.1

Interrupt Gate Descriptor	Trap Gate Descriptor
Whenever an exception occurs and handler selects an interrupt gate descriptor the 80386 clears the Interrupt Flag (IF) after it has pushed the return address and EFLAGs. But before the first instruction of the ISR is executed. When the IF flag is cleared, 80386 masks all further hardware initiated interrupts. Until the ISR is finished executing and IRET is executed. On execution of IRET instruction the original image of EFLAGs register is popped off stack.	Whenever an exception occurs and the handler selects a trap gate descriptor in the IDT, there is no change in the flags. The condition of flags is as they were before the exception.

- Code segment descriptors are used for referring code segment. Data segment descriptors are used for referring data segment. Stack segment descriptors are used for referring stack segment.
- Interrupt and trap descriptor are special types of gate descriptor used for interrupt and exception handling.

Q. 12 Compare between Real and Protected mode interrupts. **(6 Marks)**

Ans. :

Table 9.2

Sr. No.	Real mode IVT	Protected mode IDT
(i)	The IVT has the address of all 256 interrupts in 4 byte format i.e. CS:IP	The IDT has the address of all 256 interrupts in 8 byte format as base address, limit address and the access right byte.
(ii)	In real mode 80386 loads a 0000H as base address and 03FFH as the limit address for IVT. Although, IVT can be relocated	IDT gives the address of IDTR in the protected mode operation. Hence IDT may be placed anywhere in the memory
(iii)	In real mode IVT is used as the IDT.	In protected mode, IDT is used and not IVT. IDT contains descriptors.

Q. 13 Write short note on "General Detect Fault". **SPPU - May 17, 3 Marks**

OR When does a page fault occur ? **SPPU - Dec. 17, 2 Marks**

OR Explain Interrupt no.0 and 4. **SPPU - May 18, 4 Marks**

Ans. :

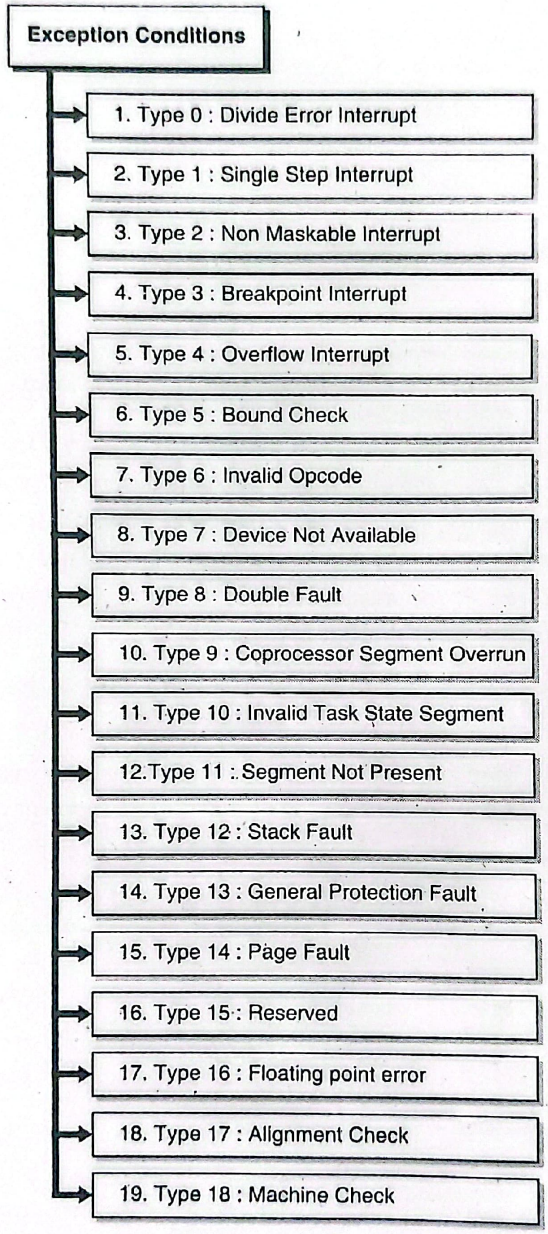

Fig. 9.8 : Exception Conditions

(1) Type 0 : Divide Error Interrupt

The divide-error fault occurs during a DIV or an IDIV instruction when the divisor is zero.

(2) Type 1 : Single Step Interrupt

- In the single step mode, the system will execute one instruction and wait for the user. The user can observe the contents of registers, memory locations. If they are correct then user can proceed with the next instruction.

- This is a trap (Single Step) debug exception. This exception occurs if the trap flag is set.
- The 80386 processor does not have any instruction to set or reset the trap flag.

(3) Type 2 : Non Maskable Interrupt

- This is a hardware interrupt using non maskable interrupt pin.
- This interrupt cannot be disabled by any software instruction.
- This interrupt can be activated by a low to high transition on the NMI pin.

(4) Type 3 : Breakpoint Interrupt

- INT-3 is a special software interrupt designed to be used as a breakpoint.
- The difference between this INT-3 interrupt and other INT interrupts is, INT-3 is one byte instruction while the others are 2-byte instructions.
- This function is used as a debugging help in cases where single stepping provides more detail than wanted. The user can insert the number of breakpoints he wishes.

(5) Type 4 : Overflow Interrupt

- Interrupt on Overflow (INTO) is a conditional software interrupt which tests the overflow (O) flag. If O = 0, the INTO performs no operation, but if O = 1, an INTO instruction executes.

(6) Type 5 : Bound Check

- This fault occurs when the processor, while executing a BOUND instruction, finds that the operand exceeds the specified limits.
- A program can use the BOUND instruction to check a signed array index against signed limits defined in a block of memory.

(7) Type 6 : Invalid Opcode

- This fault occurs when an invalid opcode is detected by the execution unit.
- (The exception is not detected until an attempt is made to execute the invalid opcode; i.e., prefetching an invalid opcode does not cause this exception.) No error code is pushed on the stack. The exception can be handled within the same task.
- This exception also occurs when the type of operand is invalid for the given opcode. Examples include an intersegment JMP referencing a register operand, or an LES instruction with a register source operand.

(8) Type 7 : Co-processor Not Available

- This exception occurs in either of two conditions :
- The processor encounters an ESC (escape) instruction, and the EM (emulate) bit ofCR0 (control register zero) is set
- The processor encounters either the WAIT instruction or an ESC instruction, and both the MP (monitor coprocessor) and TS (task switched) bits of CR0 are set

(9) Type 8 : Double Fault

- Normally, when the processor detects an exception while trying to invoke the handler for a prior exception, the two exceptions can be handled serially.

- If, however, the processor cannot handle them serially, it signals the double-fault exception instead. To determine when two faults are to be signalled as a double fault, the 80386 divides the exceptions into three classes: benign exceptions, contributory exceptions, and page faults

(10) Type 9 : Coprocessor Segment Overrun

- This exception is raised in protected mode if the 80386 detects a page or segment violation while transferring the middle portion of a coprocessor operand to the NPX. This exception is avoidable.

(11) Type 10 : Invalid Task State Segment

- Interrupt 10 occurs if during a task switch the new TSS is invalid
- An error code is pushed onto the stack to help identify the cause of the fault.
- The EXT bit indicates whether the exception was caused by a condition outside the control of the program; e.g., an external interrupt via a task gate triggered a switch to an invalid TSS.
- This fault can occur either in the context of the original task or in the context of the new task.
- Until the processor has completely verified the presence of the new TSS, the exception occurs in the context of the original task.
- Once the existence of the new TSS is verified, the task switch is considered complete; i.e., TR is updated and, if the switch is due to a CALL or interrupt, the backlink of the new TSS is set to the old TSS. Any errors discovered by the processor after this point are handled in the context of the new task

(12) Type 11 : Segment Not Present

Exception 11 occurs when the processor detects that the present bit of a descriptor is zero. The processor can trigger this fault in any of these cases

1. While attempting to load the CS, DS, ES, FS, or GS registers; loading the SS register, however, causes a stack fault
2. While attempting loading the LDT register with an LLDT instruction; loading the LDT register during a task switch operation, however, causes the "invalid TSS" exception
3. While attempting to use a gate descriptor that is marked not-presen

(13) Type 12 : Stack Fault

This fault occurs if the SS descriptor accesses a segment that is not present or if any stack operation causes a limit violation.

(14) Type 13 : General Protection Fault

This fault occurs in many cases whenever protection is violated. Some of those reasons could be as follows:

(i) Exceeding segment limit when using CS, DS, ES, FS, or GS

(ii) Exceeding segment limit when referencing a descriptor table

(iii) Transferring control to a segment that is not executable

(iv) Writing to a read-only data segment or code segment

(v) Read from execute only code segment

(vi) Loading selector with system descriptor

(v) Switching to a busy task

(vi) Violating privilege rules.

(vii) Leaving the virtual mode with a wrong interrupt handler.

(15) Type 14 : Page Fault

- This fault occurs if PG = 1 and the privilege level is incorrect or the page tables or directory contains a zero.

(16) Type 15

Reserved.

(17) Type 16

- Floating point error
- This is a floating point (coprocessor) error. The $\overline{\text{ERROR}}$ pin on math coprocessor causes this fault.

Q. 14 Explain the interrupts priority structure of 80386. **(6 Marks)**

Ans. :

- If multiple interrupts occur simultaneously, the one with higher priority is serviced first. The lower priority interrupt remains pending until the higher priority interrupt is serviced.
- Table 9.3 shows the priorities of 80386 interrupt.

Table 9.3

Interrupt	Priority
Divide Error, INT n, INTO, Faults, Trap instructions, Debug faults and traps	Highest
NMI	↓
INTR	↓
Single Step	Lowest

When considering the precedence of interrupts for multiple simultaneous interrupts, the following guideline apply :

1. INTR is the only maskable interrupt and if detected simultaneously with other interrupts, resetting of IF by the other interrupts will mask INTR. These causes the INTR to be the lowest priority interrupt serviced after all other interrupts unless the other interrupts service routine re-enable interrupts.
2. Of the non-maskable interrupts (NMI, single step and software generated i.e. INT n, INTO), in general, single step has lowest priority (will be serviced last) INT n has the highest priority and will be serviced first, followed by NMI, followed by single step.

Q. 15 What is the sequence of processing an interrupt in 80386 ? **(6 Marks)**

Ans. : Following sequence is followed to process an interrupt in 80386

Step I : Whenever an INT instruction is sensed, the 80386 microprocessor will first push the flags on the Stack. The flags are pushed so that they will provide us the internal status of the 80386 processor when it was interrupted.

Step II : The processor will clear the trap flag and the interrupt enable flag.

Step III : The contents of CS and IP are then pushed onto the Stack.

Step IV : In real mode, the interrupt number is multiplied by 4 to get the address within the interrupt vector table (IVT) that contains the interrupt service routine address. All the ISR addresses are of 4 bytes in real mode. These addresses are in the Standard CS : IP format. In case of protected mode, the IDT (Interrupt Descriptor Table) is used to give the base address along with limit address and the access right byte.

Step V : Once the vector address is computed, the 80386 microprocessor reads the ISR address from the table. It then places the ISR address onto the CS : IP to begin the program execution from this address.

Step VI : After completion of the ISR routine the program execution has to be continued from the place where the program was interrupted. So, the values of the CS : IP and flags which specify the status of the 80386 microprocessor have to be restored. i.e. this information is to be popped from the Stack using IRET instruction.

Q. 16 Explain the IVT in the Real mode of 80386. **(6 Marks)**

Ans. :

- The interrupt vector (or interrupt pointer) table is the link between an interrupt type code and the procedure that has been designated to service interrupts associated with that code.
- 80386 supports total 256 types i.e. 00H to FFH. For each type it has to reserve four bytes i.e. double word. This double word pointer contains the address of the procedure that is to service interrupts of that type.
- The higher addressed word of the pointer contains the base address of the segment containing the procedure.

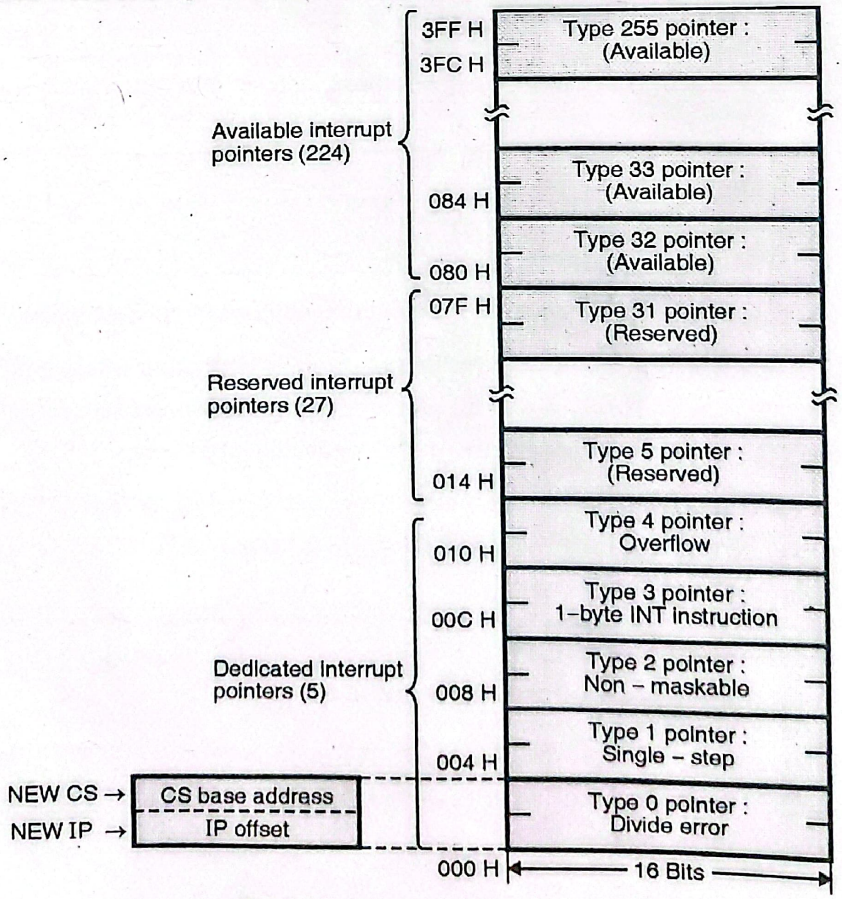

Fig. 9.9(a) : Interrupt vector table

- This base address of the segment is normally referred as NEW CS. The lower addressed word contains the procedure's offset from the beginning of the segment. This offset we normally refer as NEW IP.
- Thus NEW CS : NEW IP provides NEW physical address from where user ISR routine will start.
- As for each type, four bytes (2 for NEW CS and 2 for NEW IP) are required, therefore interrupt pointer table occupies up to the first 1KBytes (i.e. $256 \times 4 = 1024$ bytes), of low memory. Thus 00000 H to 003FF H, these locations of 80386 microprocessor are reserved for interrupt vector table.
- The microprocessor receives NEW CS and NEW IP value from vector table, after calling internal service routine.
- We know that for each "type" we have 4 locations reserved. So call routine, will simply sense the type number, multiply the same with 4 and will get double word pointer from that location.
- Suppose for example type code is '3'. Then $3 \times 4 = (12)_{10} = 0C$ H.
- Then 0CH/0DH location will provide OFFSET IP (NEW IP) and 0EH/0FH will provide base address of the segment (NEW CS).
- The same is graphically presented in Fig. 9.9(b)

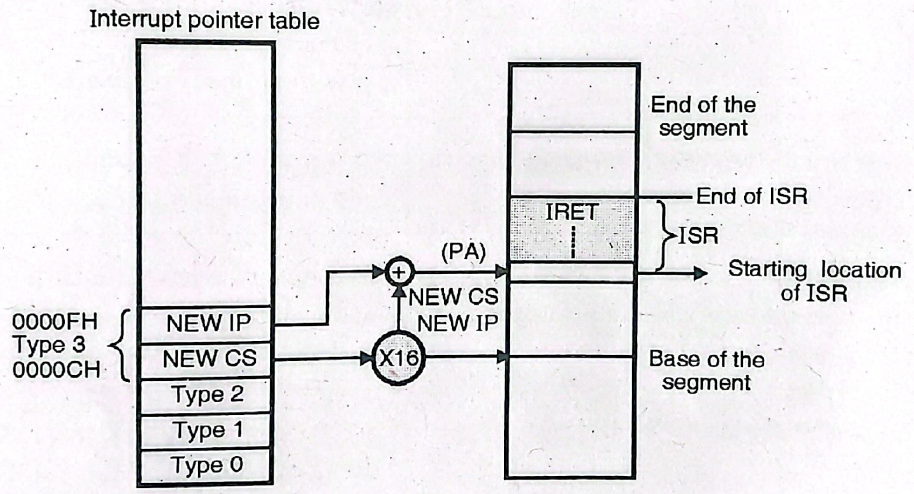

Fig. 9.9(b) : Pointing to ISR routine via interrupt vector table

- As shown in Fig. 9.9(b), total interrupt vector table is divided into three groups :
  (1) Dedicated interrupt pointers (Type 0/1/2/3/4)
  (2) Reserved interrupt pointers (Type 5 to Type 31)
  (3) Available interrupt pointers (Type 32 to Type 255).
- **Type 0 to 4** are dedicated interrupt pointers by Intel for divide by zero error, single step, NMI, 1-byte INT instruction and overflow.
- Interrupt **types from 5 to 31** are reserved by Intel for use in more complex microprocessors, such as 80286, 80386, 80486.
- Finally the **types from 32 to 255** i.e. total 224, are available to user to use for either hardware or software interrupt.

❑❑❑

Chapter 10 : Introduction to Microcontrollers

Q. 1 Compare between microprocessors and microcontrollers. **(6 Marks)**

Ans. :

Sr. No.	Microprocessor	Microcontroller
1.	Microprocessor is suited to processing information in computer systems.	Microcontroller is suited to control of I/O devices requiring a minimum component count.
2.	Microprocessor is processing intensive, has powerful addressing modes, instructions to perform complex operations and manipulate large volumes of data.	Processing capability of Microcontrollers never approaches those of Microprocessors. Microcontroller are to control of inputs and outputs. Hence instructions are to set/clear bits, Boolean operations (AND, OR, XOR, NOT, jump if a bit is set/cleared), etc. It has extremely compact instructions, many implemented in one byte.
3.	Microprocessor usually requires external circuitry to interface I/O devices.	Microcontroller has built-in I/O control module, event timing, counting etc.
4.	Microprocessor have very wide bus widths. It has large memory address spaces (> 4 Gbytes) and lots of data (Data bus: 32, 64, 128 bits wide)	Microcontroller has narrow buses. It has relatively small memory address spaces (typically Kbytes) and less data (Data bus typically 8, 16 bits wide).
5.	Microprocessor are very fast (> 1 GHz).	Microcontroller are relatively slow (typically 10-20 MHz) since most I/O devices being controlled are relatively slow.
6.	Microprocessors are expensive.	Microcontrollers are cheap.

Q. 2 What are the advantages of microcontrollers.

Ans. : **(4 Marks)**

The advantages of microcontrollers like 8051 over microprocessors are :

1. Microcontrollers have on-chip memory i.e. ROM as well as RAM. ROM is used to store programs and RAM to store data.
2. Microcontrollers have on-chip I/O controllers. The I/O devices can be directly connected to these I/O pins. Devices like switches, displays, LEDs etc. can be connected on I/O pins of microcontroller.
3. Microcontrollers have on-chip timer/counter. Hence to keep track of time or to count a particular event, no external chip is required.
4. Microcontrollers also have on-chip interrupt controller, and hence do not require any external interrupt controller. This makes the system compact as all the required operations are implemented on a single chip.

Q. 3 What are the features of microcontroller. **(6 Marks)**

Ans. :

The 8051 is the most widely used microcontroller. Let us discuss the features of 8051 :

1. 8051 operates at a maximum frequency of 12 MHz.
2. It has power saving modes.
3. It has 32 programmable I/O lines. Each line is separately programmable, hence we can connect a separate device on each line and hence have 32 devices connected to the processor.
4. It has an on-chip data memory of 128 Byte. This can also be referred to as Data RAM.
5. It has 4 KB on-chip program memory that can also be referred to as 4 KB EPROM.
6. A maximum of 64 K External Program Memory can be interfaced, if the internal EPROM falls short.
7. A maximum of 64K External Data Memory can be interfaced, if the internal data memory falls short.
8. It has two 16-bit Timer / Counters. The timer/counter can be used to either keep a track of count of times an even occurs or also be used for time delays.
9. It has 5 Interrupt sources and all are vectored interrupts.
10. It has a full duplex (simultaneous reception and transmission possible) programmable Serial Port for serial communication. In case of serial communication, one bit is transmitted at a time. This type of communication is useful for long distance communication.

Q. 4 Compare between 8050, 8051, 8052 microcontroller. **(6 Marks)**

Ans. :

Sr. No.	Feature	8048	8049	8050	8031	8051	8052
1.	Internal Data RAM	8048 has 64 bytes of internal Data RAM.	8049 has 128 bytes of internal Data RAM	8050 has 256 bytes of internal Data RAM	8031 has 128 bytes of internal Data RAM	8051 has 128 bytes of internal Data RAM	8052 has 256 bytes of internal Data RAM
2.	Internal Program ROM	8048 has 1 KB of internal ROM.	8049 has 2 KB of internal ROM.	8050 has 4 KB of internal ROM.	No internal ROM.	8051 has 4 KB of internal ROM.	8052 has 8 KB of Internal ROM.
3.	Timer / Counter	It has one 8-bit timer counter.	It has one 8-bit timer counter.	It has one 8-bit timer counter.	8031 has two 16-bit timer / counter.	8051 has two 16-bit timer / counter.	8052 has three 16-bit timer / counter.
4.	Interrupt sources	It has two interrupt sources.	It has two interrupt sources.	It has two interrupt sources.	It has 5 interrupt sources.	It has 5 interrupt sources.	It has 6 interrupt sources.

Sr. No.	Feature	8048	8049	8050	8031	8051	8052
5.	Speed	It operates at 11 MHz.	It operates at 11 MHz.	It operates at 11 MHz.	It operates at 12 MHz.	It operates at 12 MHz.	It operates at 12 MHz.
6.	I/O pin count	It has 27 I/O pins.	It has 27 I/O pins.	It has 27 I/O pins.	It has 32 I/O pins.	It has 32 I/O pins.	It has 32 I/O pins.
7.	External RAM	It can have a maximum of 256 bytes external RAM.	It can have a maximum of 256 bytes external RAM.	It can have a maximum of 256 bytes external RAM.	It can have a maximum of 64 KB external RAM.	It can have a maximum of 64 KB external RAM.	It can have a maximum of 64 KB external RAM.
8.	External ROM	It can have a maximum of 4 KB external ROM.	It can have a maximum of 4 KB external ROM.	It can have a maximum of 4 KB external ROM.	It can have a maximum of 64 KB external ROM.	It can have a maximum of 64 KB external ROM.	It can have a maximum of 64 KB external ROM.

Q. 5 What are the applications of microcontroller. **(4 Marks)**

Ans. :

1. Many home appliances like Microwave oven, washing machine, Television (TV), Air Conditioner (AC) etc. have microcontrollers in them to control their operation.
2. Most of the electronic gadgets like mobile phones, cameras, tablets, health band etc. have microcontrollers.
3. Electronic weighing scale, digital thermometer and many such instruments have microcontrollers embedded in them.
4. It has a wide application in automation systems like home automation and home security system.
5. Microcontrollers also have their application in Robotics, automated car etc.
6. Medical instruments like ECG, lung function test, X-Ray machine etc. also use microcontrollers.

❑❑❑