Chapter - 9 Introduction to Microcontrollers **(9 - 1) to (9 - 22)**

9.1 Microcontroller and Embedded Processors ..... 9 - 1

9.2 Features of 8051 Microcontroller ..... 9 - 9

9.3 Block Diagram of 8051 Microcontroller ..... 9 - 10

9.4 Register Organization of 8051 Microcontroller ..... 9 - 12

9.5 Pin Diagram of 8051 ..... 9 - 14

9.6 Memory Organization ..... 9 - 17

9.7 External Memory Interfacing ..... 9 - 20

Solved SPPU Question Papers **(S - 1) to (S - 5)**

Unit III

4 Memory Management

4.1 : Address Translation Overview

Q.1 Draw and explain the protected mode addressing mechanism.

OR What is a logical address, linear address and physical address ?

☞ [SPPU : June-22, Marks 6]

Ans. :

In protected mode, the 80386 transforms logical addresses (i.e., addresses as viewed by programmers) into physical address (i.e., actual addresses in physical memory) in two steps :

- **Segment translation,** in which a logical address (consisting of a segment selector and segment offset) are converted to a linear address.
- **Page translation,** in which a linear address is converted to a physical address. This step is optional, at the discretion of systems-software designers.

In protected mode every segment selector contents of segment register has a linear base address associated with it, and it is stored in the segment descriptor. A selector is used to point a descriptor for the segment in a table of descriptors. The linear base address from the descriptor is then added to the 32-bit offset to generate the 32-bit linear address. This process is known as **segmentation** or **segment translation**. If paging unit is not enabled then the 32-bit linear address corresponds to the physical address. But if paging unit is enabled, paging mechanism translates the linear address space into the physical address space by paging translation.

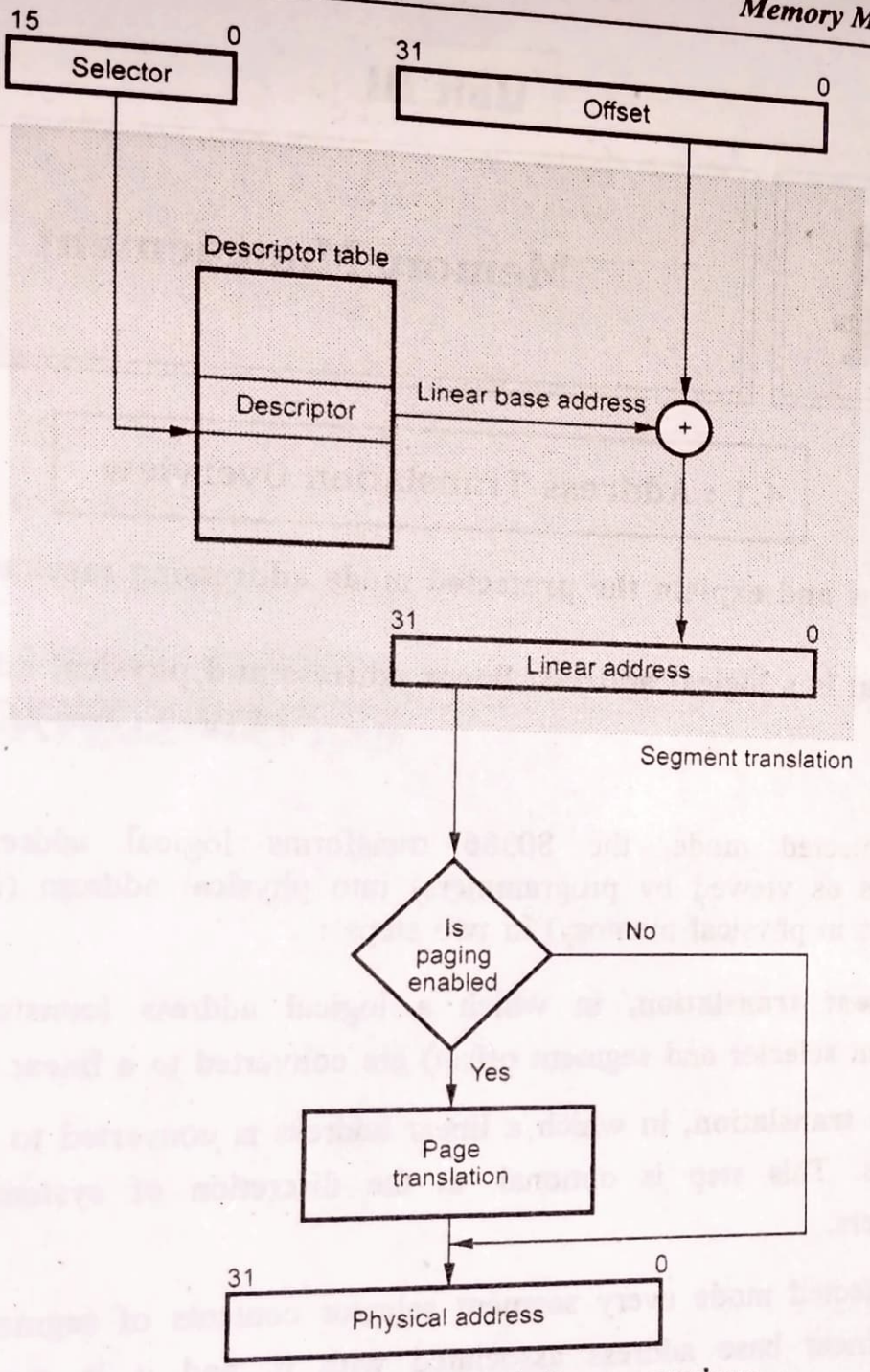


Fig. Q.1.1 Address translation overview

4.2 : Segment Translation

Q.2 Draw and explain how 80386 microprocessor translates logical address into linear address.

☞ [SPPU : Nov.-12, Dec.-18, May-19, Marks 10]

OR Explain memory segmentation of 80386 microprocessor.

☞ [SPPU : Nov.-12, Marks 8]

DECODE®

OR Explain the working of segment selector in protected mode operation of 80386 processor. ☞ [SPPU : Dec.-11, Marks 8]

OR How are descriptors used to access the operand ?

☞ [SPPU : May-06, June-22, Marks 4]

OR Explain segment address translation in detail.

☞ [SPPU : May-17, Dec.-22, Marks 6]

Ans. : Segment translation is a process of converting logical address into a linear address. Fig. Q.2.1 shows the segmentation mechanism. It shows how selector is used to access a descriptor in a descriptor table. The 13-bit index part of selector is multiplied by 8 and used as a pointer to the desired descriptor in a descriptor table. The index value is multiplied by 8 because each descriptor requires 8 bytes in the descriptor table. The descriptor in the descriptor table contains mainly base address, segment limit and access right byte. The 80386 adds the base address from the descriptor to the effective address or offset to generate a linear address.

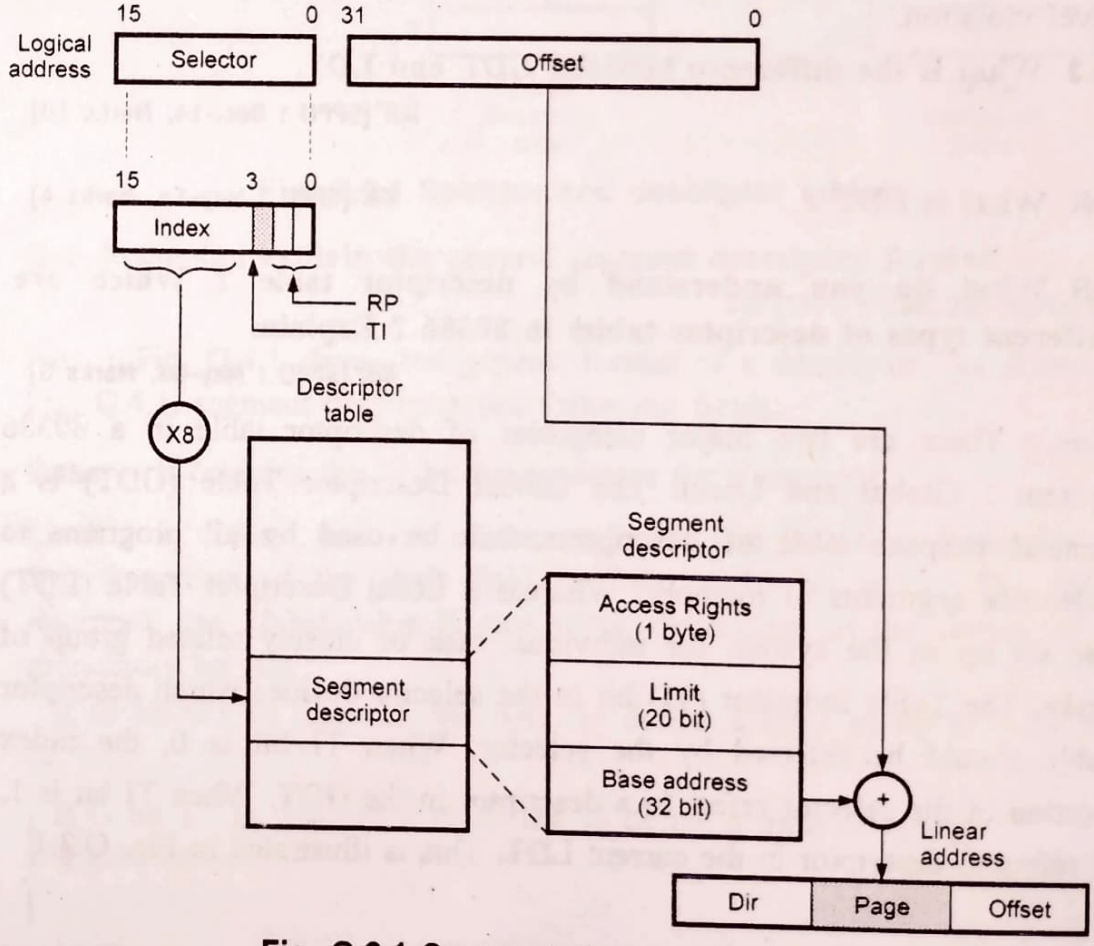


Fig. Q.2.1 Segmentation mechanism

The Table Indicator (TI) bit in the selector decides which dscriptor table should be referred by the selector : IT = 0 (GDT) and TI = 1 (LDT).

As shown in the Fig. Q.2.1, The selector component of each logical address contains 2 bits (RP) which represent the privilege level of the program section requesting access to a segment. Level 0 is the most privileged and level 3 is the least privileged. More privileged levels are numerically smaller than less privileged levels. The descriptor of each segment contains 2 bits which represent the privilege level of that segment. When an executing program attempts to access a segment, the memory management unit compares the privilege level in the selector with the privilege level in the descriptor. If the segment selector has the same or greater privilege level, then the memory management unit allows the segment to be accessed. If the selector privilege level is lower than the privilege level of the segment, the memory management unit denies the access and sends an interrupt signal to the CPU indicating a privilege level violation.

Q.3 What is the difference between GDT and LDT.

☞ [SPPU : Dec.-14, Marks 10]

OR What is LDT ? ☞ [SPPU : May-14, Marks 4]

OR What do you understand by descriptor table ? Which are different types of descriptor tables in 80386 ? Explain.

☞ [SPPU : May-08, Marks 8]

Ans. : There are two major categories of descriptor table in a 80386 system : Global and Local. The Global Descriptor Table (GDT) is a general purpose table of descriptors, can be used by all programs to reference segments of memory. Whereas a Local Descriptor Table (LDT) are set up in the system for individual task or closely related group of tasks. The Table Indicator (TI) bit in the selector decides which descriptor table should be referred by the selector. When TI bit is 0, the index portion of the selector refers to a descriptor in the GDT. When TI bit is 1, it refers to descriptor in the current LDT. This is illustrated in Fig. Q.3.1.

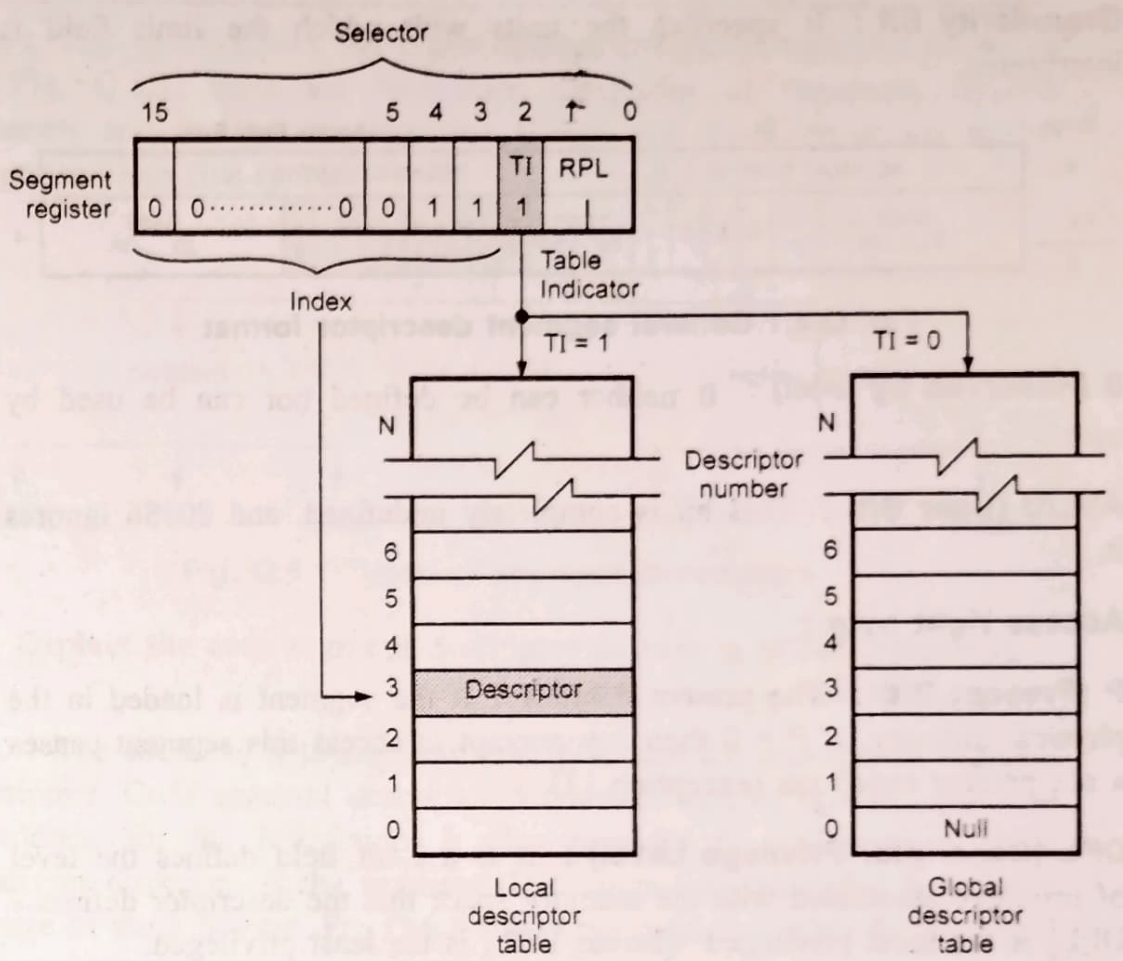


Fig. Q.3.1 Selector and descriptor tables

Q.4 Draw and explain the general segment descriptor format.

☞ **[SPPU : Dec.-22, Marks 6]**

Ans. : Fig. Q.4.1 shows the general format of a descriptor. As shown in Fig. Q.4.1, segment descriptor has following fields.

Base : It contains the 32-bit base address for a segment.

Limit : It defines the size of the segment. The 80386 concatenates the two fragments of the limit field to form a 20 bit value. The 80386 interprets this 20-bit value in two ways, depending on the setting of the granularity bit (G) :

If G bit 0 : In units of one byte, to define a limit of up to 1 M byte (2^{20})

If G bit 1 : In units of 4 kilobytes, to define a limit of up to 4 gigabytes.

The **Executable (E)** bit indicates whether segment is code or data segment. If E bit is 1, segment is code segment otherwise segment is data segment. The code segment may be executable, or executable and read. This is determined by Readable (R) bit. If R bit is 1, code segment is executable and readable otherwise it is only executable. If Conforming Bit (C) is 1, code segment can be executed and shared by programs at different privilege levels.

The Expansion Direction (ED) bit specifies expansion direction for the segment. If ED = 0, expansion direction is upward which is data segment and if ED = 1, expansion direction is downwards which is stack segment.

The Write (W) bit for data segment indicates whether the data segment is read only, or read and write. If W bit 0, data segment is read only; otherwise it is read/write segment. For stack segment W bit must be logic 1.

Q.7 Draw and explain the general format of system segment descriptor.

OR Explain the functions of LDT, GDT, TSS and Gate descriptors.

☞ [SPPU : June-22, Marks 3]

Ans. : Fig. Q.7.1 shows the general format of system segment descriptor. From Fig. Q.7.1 it can be seen that several descriptor fields (Base address, limit, Granularity bit G and Present bit P) are similar to the general segment descriptor. Type field of descriptor decides descripter amongst : LDT, TSS and Gate

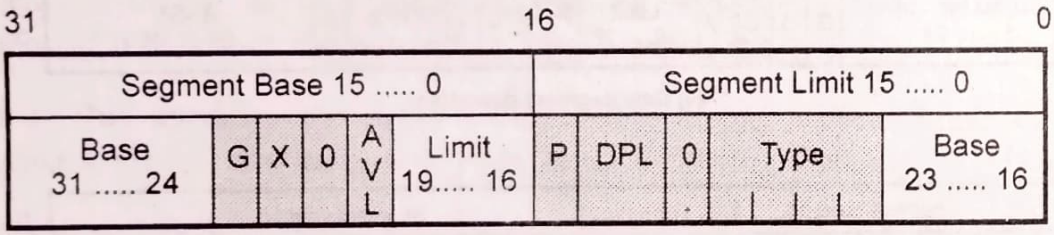


Fig. Q.7.1 System segment descriptor

a) LDT descriptors (S = 0, Type = 2) : The LDT descriptors are present only in the Global Descriptor Table (GDT). They contain the information about the local descriptor table

b) TSS descriptor (S = 0, Type = 1, 3, 9, B) : In a multitasking environment computer performs more than one task at a time, and it also switch between the task. A task can be a single program, or it can be a group of related programs. When it switches from task 1 to task 2, it

stores all the information necessary to restart the task1 later in time exactly as it was left. It involves saving the contents of all of the processor registers as well as any read/write memory variables and the address of next instruction to be executed. Such information is called **state of the task** or **context of the task.**

The 80386 uses a special segment called **Task State Segment** (TSS) to store the state/context of the task. This segment can be addressed with the help of Task State Segment (TSS) descriptor. The TSS descriptor contains information about the location, size and privilege level of a TSS.

Along with the context of the task, the TSS also contains the **linkage field** for the next task which allows the nesting of tasks. The TSS descriptor gives base address and limit for TSS. Its TYPE field is used to indicate whether task is currently BUSY (i.e. on a chain of active tasks) or the TSS is available. The TYPE field also indicates if the segment contains a 80286 or an 80386DX TSS.

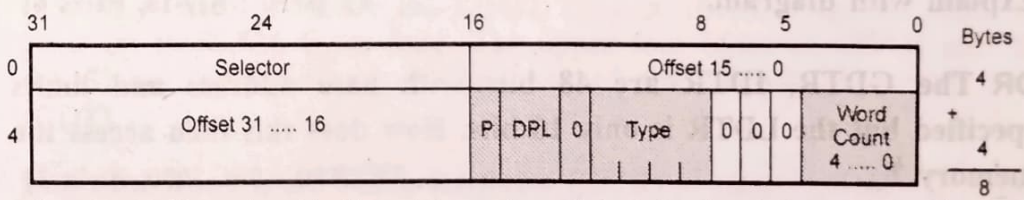
c) Gate descriptors (S = 0, TYPE = 4 – 7, C, F) : A gate is a special type of descriptor. It allows the 80386 to automatically perform protection checks. There are four types of gate descriptors :

- Call gate
- Task gate
- Interrupt gate
- Trap gate

Call gates are used to change privilege levels. Task gates are used to perform a task switch and interrupt and trap gates are used to specify interrupt service routines.

Q.8 Draw the format of gate descriptor.

Ans. : Fig. Q.8.1 shows the format of the four types of gate descriptors.



D word count : The number of double words to copy from caller's stack to the called procedure's stack. Only used with call gate.

Destination Selector :	(16 - bit)	Selector to the target code segment or Selector to the target task state segment for task gate
Destination Offset :	(32 - bit)	Entry point within the target code segment

Fig. Q.8.1 Gate descriptor formats

Q.9 Explain various types of descriptor tables.

OR What is IDT ? ☞ [SPPU : May-09, 11, Marks 2]

Ans. : There are three types of descriptor tables :

- **The Global Descriptor Table (GDT) :** It is a general purpose table of descriptors, can be used by all programs to reference segments of memory. The GDT can have any type of segment descriptor except for descriptors which are used for serving interrupts.
- **The Interrupt Descriptor Table (IDT) :** It holds the segment descriptors that define interrupt or exception handling routines. The IDT is a direct replacement for the interrupt vector table used in 8086 system.
- **A Local Descriptor Tables (LDT) :** They are set up in the system for individual task or closely related group of tasks.

Q.10 List and explain the different descriptor table register used in protected mode of 80386 processor. ☞ [SPPU : Dec.-11, Marks 10]

OR What is significance of global descriptor table registers ? Explain with diagram. ☞ [SPPU : May-13, Marks 4]

OR What is an IDTR ? What are its contents following reset of 80386 ? ☞ [SPPU : May-01, Marks 3]

OR Specify size and function of IDTR. ☞ [SPPU : May-10, Marks 2]

OR What is the significance of interrupt descriptor table registers ? Explain with diagram. ☞ [SPPU : May-13, Marks 4]

OR The GDTR, IDTR are 48 bits with base address and limits specified but the LDTR is only 16 bits. How does this then access the memory ? ☞ [SPPU : Dec.-2000, Marks 3]

OR Specify size and function of LDTR. ☞ [SPPU : May-10, Marks 2]

OR What is the significance of local descriptor table registers ? Explain with diagram. ☞ [SPPU : May-13, Marks 4]

OR Differentiate and explain GDTR, LDTR and IDTR.

☞ [SPPU : Dec.-22, Marks 6]

Ans. : Each of the tables has a register associated with it the GDTR, the LDTR and the IDTR.

Global Descriptor Table Register (GDTR) : Fig. Q.10.1 shows how the contents of the global descriptor table register.

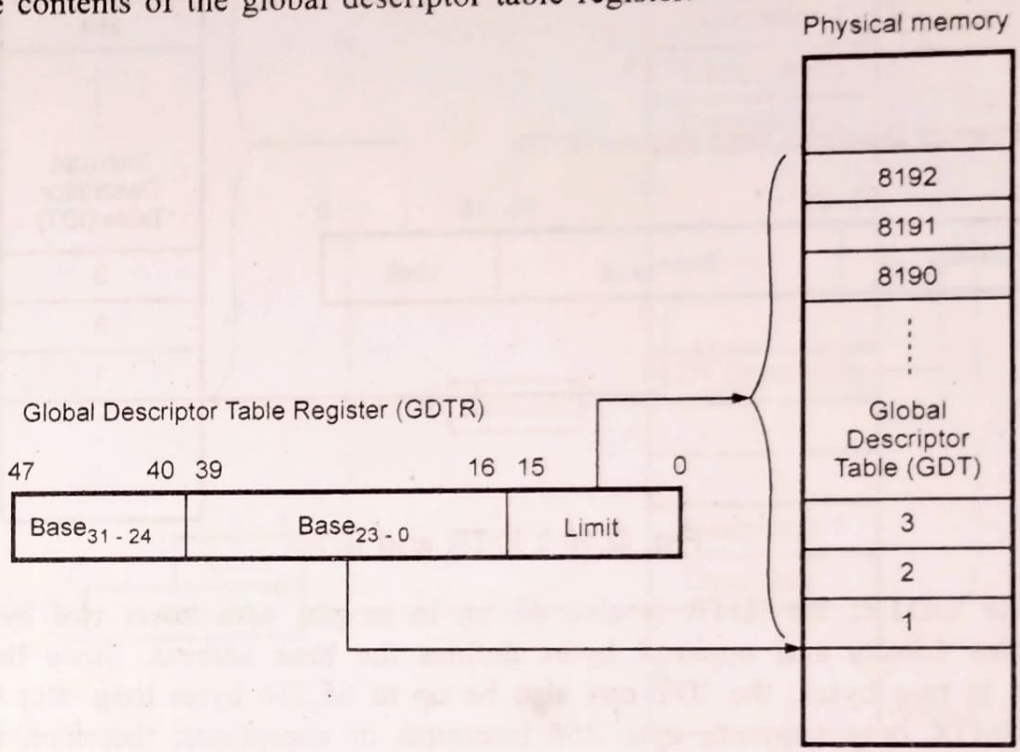


Fig. Q.10.1 GDTR and GDT

GDTR is a 48-bit register located inside the 80386DX. The lower two bytes of this register specifies the LIMIT, (in bytes) for the GDT. The value of limit is one less than the actual size of the table. For example, if LIMIT is 03FFH then the table is 1024 (1023 + 1) bytes in length ($03FFH = 1023_{10}$). Since the LIMIT field is 16 bit long, the GDT can grow up to 65,536 bytes long. The upper four bytes of GDTR specifies the 32-bit linear address of the base of the Global Descriptor Table (GDT).

Interrupt Descriptor Table Register (IDTR) : Like global descriptor table register, interrupt descriptor table register holds the 16-bit limit and 32-bit linear address of the base of the Interrupt Descriptor Table (IDT). Fig. Q.10.2 shows how the contents of the Interrupt Descriptor Table Register are used to define a Interrupt Descriptor Table (IDT) in the 80386DX physical memory address space.

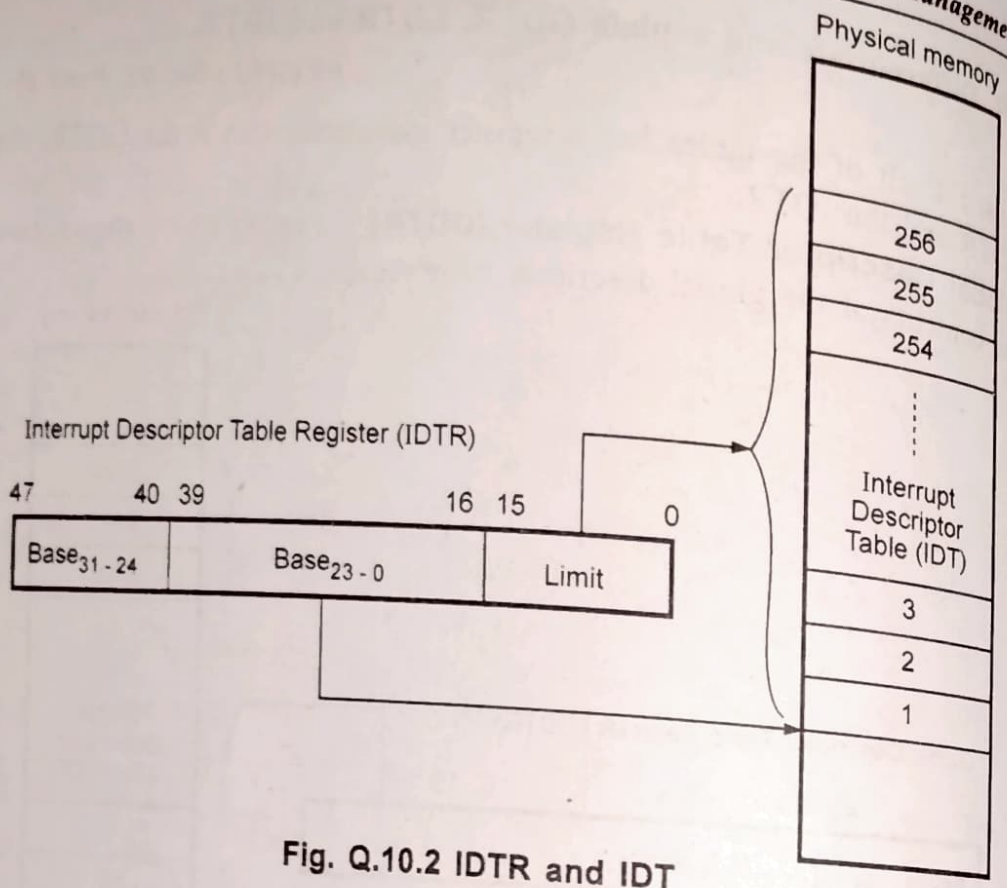


Fig. Q.10.2 IDTR and IDT

Like GDTR, the IDTR is also 48 bit in length, with lower two bytes defines Limits and upper 4 bytes defines the base address. Since limit field is two bytes, the IDT can also be up to 65,536 bytes long. But the 80386DX only supports upto 256 interrupts or exceptions; therefore, the size of the IDT should not be set to support more than 256 interrupts.

At reset, the contents of IDTR are : Base = 0
and Limit = 03FFH

Local Descriptor Table Register (LDTR) : Unlike GDTR and IDTR, the LDTR is a 16-bit register. It does not specify any limit or base address for the segment but it specifies the address of the LDT descriptor stored in the Global Descriptor Table (GDT).

Fig. Q.10.3 shows LDTR, GDT and LDT shows how contents of LDTR are used indirectly to define a Local Descriptor Table.

LDTR holds a selector that points to an LDT descriptor in the GDT. Whenever a selector is loaded into the LDTR, the corresponding descriptor is located in the global descriptor table. The contents of this descriptor defines the local descriptor table. The 32-bit base value defines starting point of the table in the 80386DX physical memory address space and 16-bit limit specifies the size of the table.

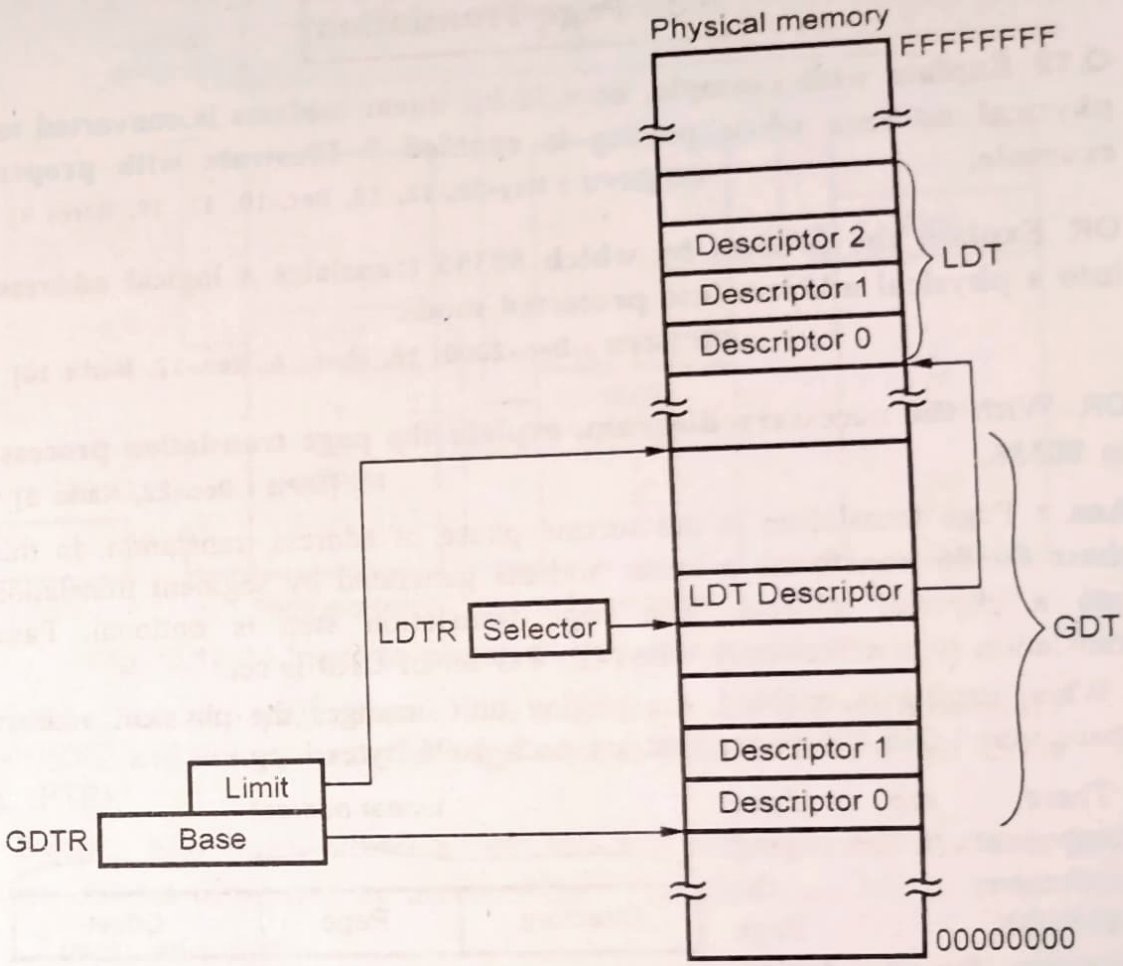


Fig. Q.10.3 Global and local descriptor tables

Q.11 What do you mean by segment descriptor cache ? Explain the use of it. ☞ **[SPPU : May-10, Marks 6]**

Ans. : The hidden part of segment register is referred to as segment descriptor cache register. Using these registers 80386DX stores information from descriptor, thereby avoiding the need to consult a descriptor table every time it accesses memory. Segment register (visible portion) contents are manipulated by programs whereas segment descriptor cache register (hidden portion) contents are manipulated by processor. Once the descriptors are cached, subsequent references to them are performed without any overhead for loading of the descriptor. This is the biggest advantage of segment descriptor cache registers.

4.3 : Page Translation

Q.12 Explain with example, how 32-bit linear address is converted to physical address when paging is enabled ? Illustrate with proper example. ☞ [SPPU : May-02, 12, 18, Dec.-10, 17, 19, Marks 8]

OR Explain the process by which 80386 translates a logical address into a physical address into protected mode.

☞ [SPPU : Dec.-2000, 18, Marks 4, Nov.-12, Marks 10]

OR With the necessary diagram, explain the page translation process in 80386. ☞ [SPPU : Dec.-22, Marks 6]

Ans. : Page translation is the second phase of address translation. In this phase 80386 transforms a linear address generated by segment translation into a physical address. The page translation step is optional. Page translation is in effect only when the PG bit of CR0 is set.

When paging is enabled, the paging unit arranges the physical address space into 1,048,496 pages that are each 4096 bytes long.

There are three components to the paging mechanism of the 80386DX : Page directory, the page tables, and the page. When paging is enabled 80386DX uses two levels of tables to translate the linear address (from the segment translation) into a physical address. Fig. Q.12.1 shows the format of linear address. Processor internally divides a linear address into three fields : Two fields of 10 bits each and one field of 12 bits. The most significant 10 bits (DIR field) of the linear address are used as an index into a page directory. The next most significant 10 bits (PAGE field) of the linear address are used as an index into the page table determined by the page directory. The least significant 12 bits (OFFSET) select one of 4096 bytes of memory from the page frame determined by the page table. The physical address of the current page directory is stored in the control register (CR3) which is also referred to as **Page Directory Base Register (PDBR)**. Fig. Q.12.2 shows how the 80386DX converts the DIR, PAGE and OFFSET fields of a linear address into the physical address by consulting two levels of page tables.

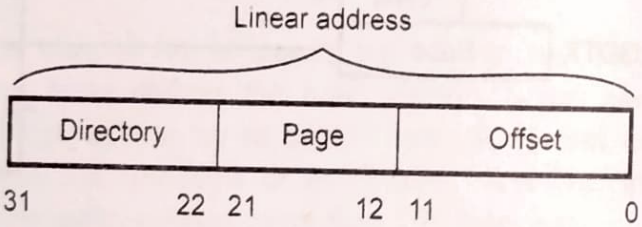


Fig. Q.12.1 Linear address format

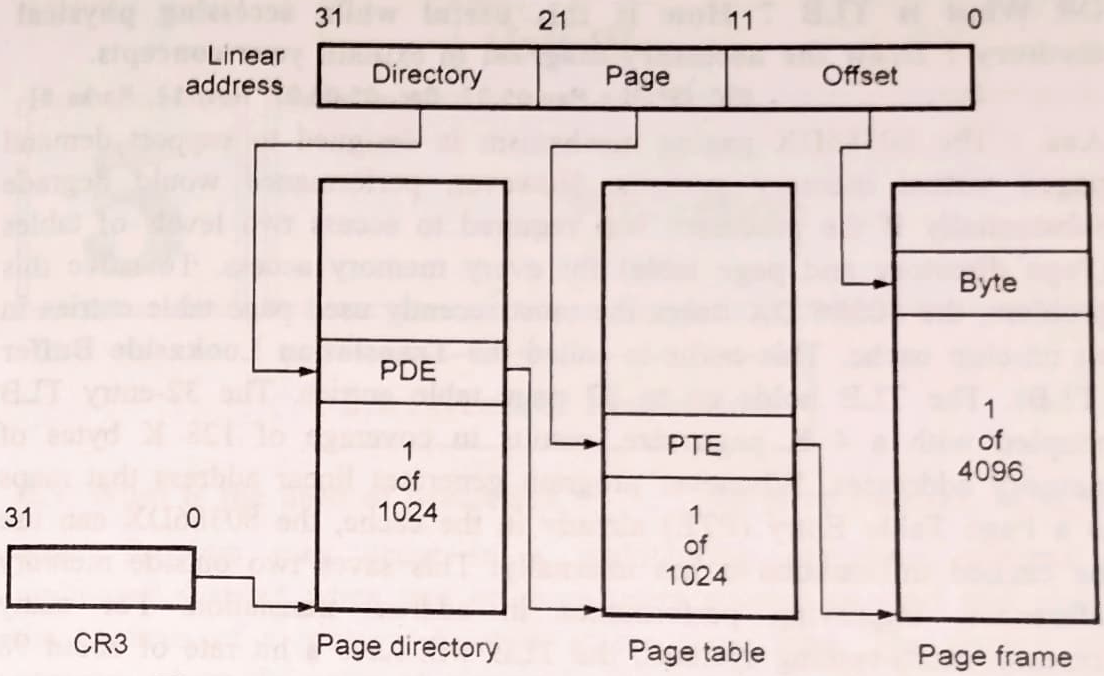


Fig. Q.12.2 Linear to physical address translation

The descriptor in a page directory is referred to as a **Page Directory Entry** (PDE) and descriptor in the page table is referred to as **Page Table Entry** (PTE).

Q.13 Explain how linear address 0080400 AH will be translated into physical address using paging mechanism. ☞ [SPPU : Dec.-19, Marks 4]

Ans. : Linear address

31 Directory 22	21 Page 12	11 Offset 0
0000 0000 10	00 0000 0100	0000 0000 1010

- Index to page directory = 10. This entry in the page directory gives the base address of the page table.
- Index to page table = 100. This entry in the page table gives the base address of the page frame.
- Index to page frame = 1010. Selects 1010th (10th byte) of memory from the page frame.
- The physical address generated will be different from the linear address.

Q.14 What is the purpose of TLB and descriptor cache ? How do they reduce system overheads ?

OR Explain the role of TLB in the process.

☞ [SPPU : May-04, Marks 2]

OR What is TLB ? How is this useful while accessing physical memory ? Draw the necessary diagram to explain your concepts.

[SPPU : May-05,07, Dec.-05,06,07, Nov.-12, Marks 8]

Ans. : The 80386DX paging mechanism is designed to support demand paged virtual memory systems. However, performance would degrade substantially if the processor was required to access two levels of tables (Page directory and page table) for every memory access. To solve this problem, the 80386 DX stores the most recently used page table entries in an on-chip cache. This cache is called the **Translation Lookaside Buffer (TLB)**. The TLB holds up to 32 page table entries. The 32-entry TLB coupled with a 4 K page size, results in coverage of 128 K bytes of memory addresses. Whenever program generates linear address that maps to a Page Table Entry (PTE) already in the cache, the 80386DX can use the cached information it has internally. This saves two outside memory references, improving performance in address translation. For many common multi-tasking systems, the TLB will have a hit rate of about 98 %. This means that the processor will only have to access the two-level page structure on 2 % of all memory accesses. Fig. Q.13.1 illustrates how the TLB supports the 80386DX paging mechanism.

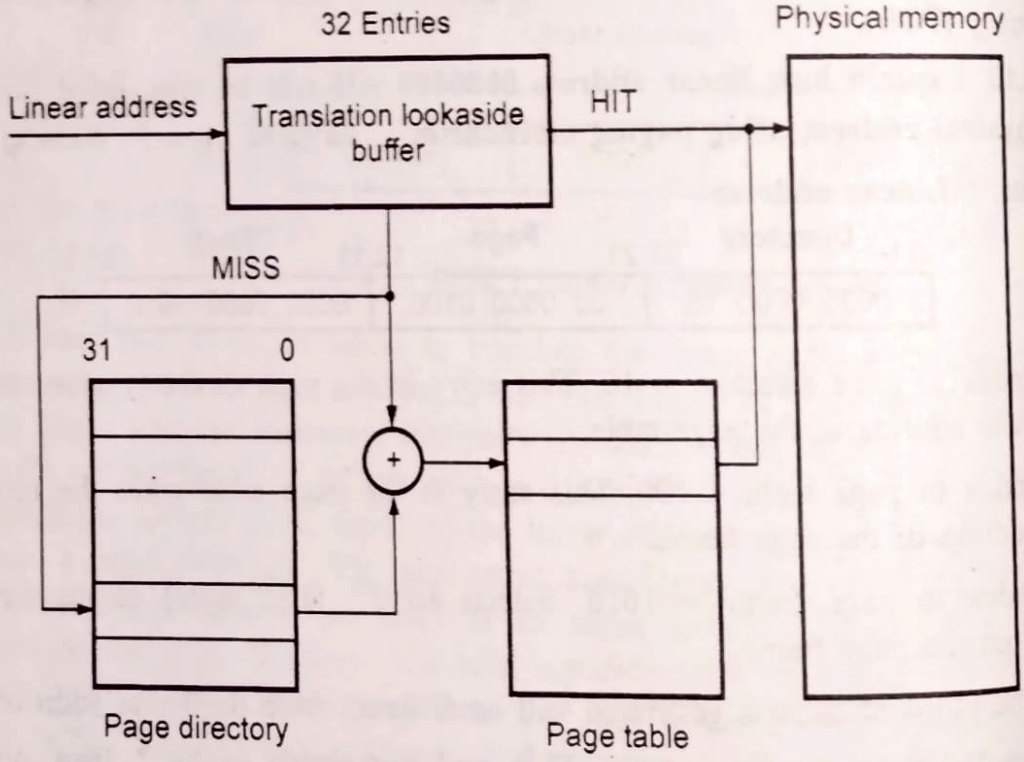


Fig. Q.14.1 Translation lookaside buffer

END...

Unit IV

5 Protection

5.1 : Need of Protection

Q.1 What is the need of protection ? ☞ [SPPU : Dec.-22, Marks 5]

Ans. : Problem may occur in a multitasking operating systems or multi-user systems when two or more users attempt to read and change the contents of a memory location at the same time. The section of a program where the value of a variable is being read and changed (critical section) must be protected from access by other tasks until the operation is complete.

Another region that requires protection is the operating system code. The incorrect address in a user program may cause program to write over the critical sections of the operating system corrupting the operating system code and data areas. The system then 'locks-up' and the only way to get control again is to reboot the system. In a multitasking system this is intolerable, so several methods are used to protect the operating system.

5.2 : Overview of 80386DX Protection Mechanisms

Q.2 State the protection mechanism supported by 80386.

OR List the five aspects of protection in the 80386.

☞ [SPPU : May-18, Dec.-22, Marks 2]

Ans. : The 80386 uses segment level protection and page level protection mechanisms to protect critical sections.

Protection in the 80386 has five aspects:

1. Type checking
2. Limit checking
3. Restriction of addressable domain
4. Restriction of procedure entry points
5. Restriction of instruction set

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.

5.3 : Segment Level Protection

Q.3 What is segment level protection ?

OR Define the function of type checking and limiting checking in protection. ☞ [SPPU : June-22, Marks 6]

Ans. : Segment level protection involves :

1. Type checking 2. Limit checking
3. Restriction of addressable domain
4. Restriction of procedure entry points
5. Restriction of instruction set

The segment is the unit of protection, and segment descriptors store protection parameters. Protection checks are performed automatically by the CPU when the selector of a segment descriptor is loaded into a segment register and with every segment access. Segment registers hold the protection parameters of the currently addressable segments.

When an attempt is made to access a segment first of all, the 80386 checks to see if the descriptor table indexed by the selector contains a valid descriptor for that selector. If the selector attempts to access a location outside the limit of the descriptor table or the location indexed by the selector in the descriptor table does not contain a valid descriptor, then an exception is produced.

The 80386 also checks to see if the segment descriptor is of the right type to be loaded into the specified segment register cache. The descriptor for a read-only data segment, for example cannot be loaded into the SS register, because a stack must be able to be written to. A selector for a code segment which has been marked "execute only" cannot be loaded into the DS register to allow reading the contents of the segment.

If all above protection conditions are met, the limit, base, and access rights bytes of the segment descriptor are copied into the hidden part of the segment register. The 80386 then checks the P (Present) bit of the

access byte to see if the segment for that descriptor is present, a type 11 exception is generated.

After a segment selector and descriptor are loaded into a segment register, further checks are made each time a location in the actual segment is accessed. These checks are type checking and limit checking.

Type Checking : Type field of the descriptor specifies. 1. Type of the descriptor and 2. Intended usage of the segment.

Type checking is used to detect whether any program is attempting to use segments in ways not intended by the programmer.

Limit Checking : The 80386DX uses limit field of a segment descriptor to prevent programs from addressing outside the segments.

Privilege Level Protection : The 80386 has four levels of protection which are optimized to support the needs of a multi-tasking operating system to isolate and protect user programs from each other and the operating system.

Q.4 How many privilege levels do 80386 supports ?

☞ [SPPU : May-05, Marks 3]

OR What do you mean by privilege levels ? What is the need ?

☞ [SPPU : May-05, 19, Dec.-07, Marks 4]

OR Explain the four levels of protection mechanism implemented by 80386. ☞ [SPPU : May-13, Dec.-15, 22, June-22, Marks 12]

OR What is DPL, RPL and CPL ?

☞ [SPPU : May-10,12, Dec.-17, 18, Marks 5]

OR What are the privilege checks made if accessed area is code, data or stack ? ☞ [SPPU : Dec.-2000, Marks 4]

OR Write privilege checks performed by 80386 while accessing code or data with protection mechanism. ☞ [SPPU : May-10,12, Marks 5]

OR Explain CPL, RPL and DPL concepts while accessing the code or data with protection mechanism of 80386. Also draw the data segment descriptor format and code segment descriptor format to demonstrate mechanism of privilege level checks.

☞ [SPPU : May-05, June-22, Dec.-22, Marks 5]

Ans. : The four level of protections are four privilege levels, numbered from 0 to 3. The value zero represents highest privilege level and value 3 represents lowest privilege level. Fig. Q.4.1 shows how a 80386DX protected mode system can be set up with four privilege levels. It shows that operating system kernel is assigned with the highest privilege level, which is privilege level 0 (PL0). The system services such as BIOS procedures are assigned with PL1, whereas custom device drivers are assigned with PL2 and finally application programs are assigned with PL3.

The 80386DX assigns these levels to different objects such as descriptors and selectors. The assigned privilege levels are stored in the respective fields as given below.

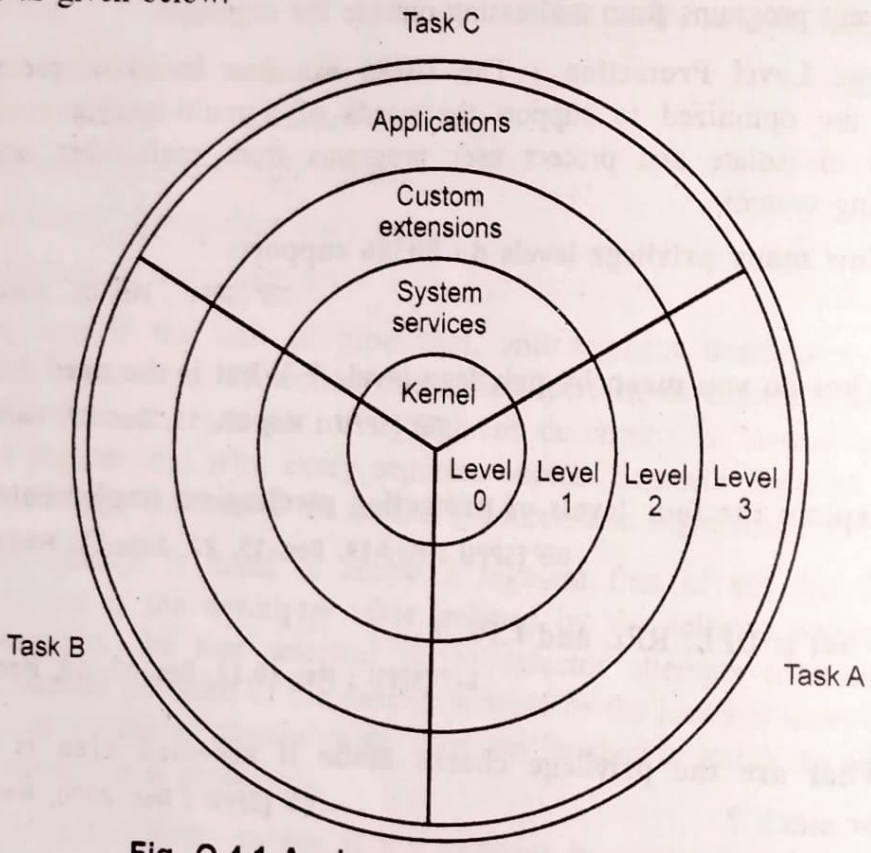


Fig. Q.4.1 Assignment of privilege levels

- Descriptors contain field called the **Descriptor Privilege Level (DPL)** This is the least privileged level at which task may access that descriptor.
- Selectors contain field called the **Requester's Privilege Level (RPL)**. The RPL is intended to represent the privilege level of the procedure that originates a selector.

- The 80386DX stores the descriptors in the internal cache (hidden portion of segment registers) for currently executing segments. Privilege levels for such descriptors are referred to as **Current Privilege Level (CPL)**.

Restricting Access to Data

Program can 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.

Accessing Data in Code Segments

It is possible to read data from code segment. There are three ways of reading data from code segments.

1. Load a data segment register with a selector of a non conforming, 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.

Restricting Control Transfers

The 80386DX can transfer program control with the help of JMP, CALL, RET, INT and IRET instructions. The "near" forms of JMP, CALL and RET transfer control within the current segment so these are subjected to only limit checking. But in case of far JMP, CALL and RET transfers, control is transferred to other segment. In such cases 80386DX performs privilege checking.

To successfully transfer the control to other segment, both the RPL and the CPL must be a number less than or equal to the DPL of the segment. In other words, the privilege level of the requesting selector and current privilege level must both be greater than or equal to the privilege level of the desired segment.

$$\text{Max (CPL, RPL)} \leq \text{DPL}$$

Q.5 What do you mean by conforming code ? Explain. Is there any alternative for this mechanism ? Explain. ☞ **[SPPU : Dec.-06, Marks 10]**

OR Explain conforming code segment and non-conforming code segment. ☞ **[SPPU : Dec.-08, May-10, Marks 6]**

OR Explain the working of conforming code segment.

☞ [SPPU : May-13, Marks 6]

OR What is call gate ? ☞ [SPPU : May-02, Marks 5]

OR What are different methods to call a function with higher privilege level ? Explain with the help of suitable example.

☞ [SPPU : Dec.-11, 22 May-05,06,08,12, June-22, Marks 10]

OR What is call gate descriptor ? Give its significance in detail.

☞ [SPPU : Dec.-08, Marks 10]

OR What is call gate ? Explain its role in changing PL.

☞ [SPPU : May-09, Dec.-18, Marks 10]

OR CALL gate acts as an interface layer to a code with different privilege levels. Justify the statement with the help of CALL gate descriptor.

☞ [SPPU : May-10, 19, Marks 8]

OR Explain CALL gate mechanism in detail.

☞ [SPPU : Nov.-12, Marks 6]

Ans. : There are two ways to access a procedure located in a segment which has a higher privilege level.

1. The first option has a restriction that the segment which has a higher privilege level must be a **conforming code segment**.
2. The second option is more complex, but allows to access the segment which has a higher privilege level using special structure known as **call gate**.

Conforming Code Segment

A code segment is considered conforming if bit 2 of the access rights byte of its descriptor is set. (C = 1 for conforming and C = 0 for non-conforming). Conforming code segments differ from the **non-conforming code** segment in that they have no inherent privilege level of their own; they confirm to that level of the code that CALLs them or JMPs to them. For example, if a program in a PL3 segment transfers control to a conforming code segment, then the conforming code runs with CPL equal to 3. If the same segment is invoked by PL0 code, it runs with a CPL of 0.

Even though conforming code segments do not have any particular privilege level associated with them, there is still one restriction regarding when a conforming segment can be used. The DPL of the conforming descriptor must always be less than or equal to the current CPL. You can never transfer control to a segment whose DPL is greater (less privileged) than the current segment. This is done because at the time of transferring control back to the original segment from conforming code segment there is change in privilege level. Here, conforming code segment must have higher or same privilege level than original segment to allow control to return back to the original segment.

Call Gates

A call gate is simply a special type of descriptor as shown in Fig. Q.5.1. Unlike code, data, or stack descriptors or the system type LDT descriptor, call gate descriptors **do not define any memory space**. They have **no base address** or **limit fields**. Actually, they are not descriptors at all, but it is convenient to place them in descriptor tables. It acts as an interface layer between code segments at different privilege levels. The **call gate** is the only mechanism that allows to call a procedure located in any segment (conforming or non-conforming) which has a higher privilege level. JMPs are not allowed. Hence the name **call gate**.

Call Gates

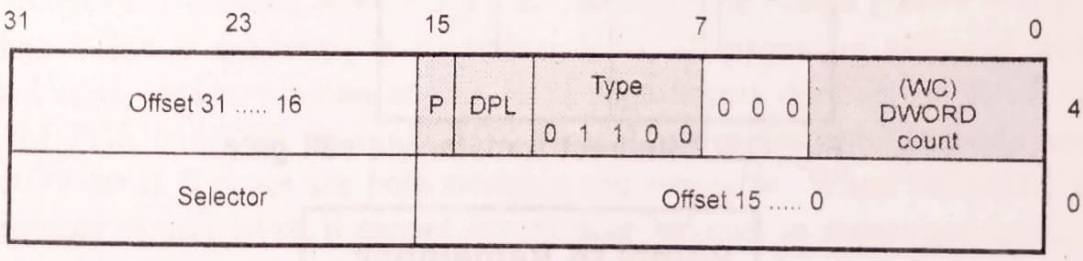


Fig. Q.5.1 Format of 80386 call gate

The call gate descriptor contains two important things :

1. Selector which points to the descriptor for the segment where the procedure is actually loaded.
2. Offset of the called procedure in its segment.

If the call is valid, the selector from the call gate (points to the descriptor for the segment where the procedure is actually loaded) is placed in the visible portion of CS register and the corresponding segment descriptor is loaded into the hidden portion of CS register. The 80386DX then uses the base address from the segment descriptor and the offset from the call gate descriptor to calculate the physical address of the called procedure as shown in Fig. Q.5.2.

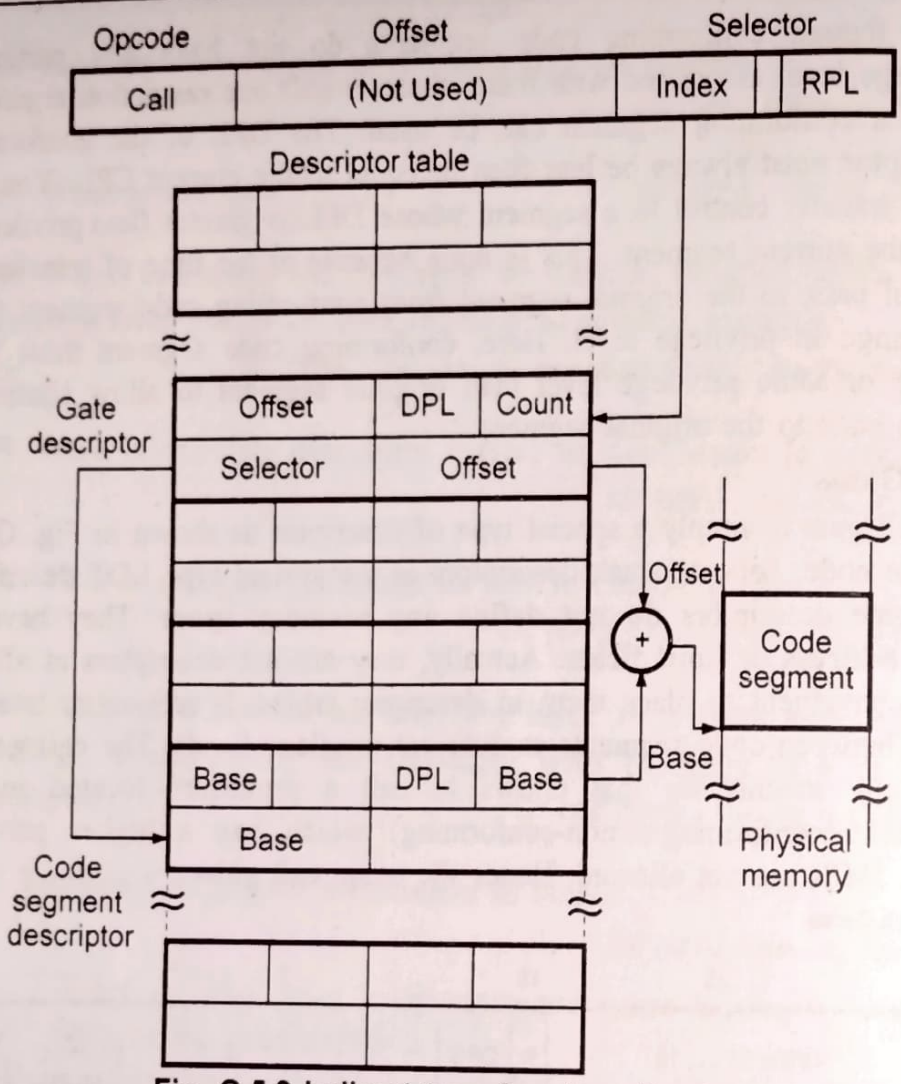


Fig. Q.5.2 Indirect transfer via call gate

Points to Remember

Privilege requirements to use a call gate :

- Call gate DPL must be numerically greater than or equal to the current privilege level.
- Call gate DPL must be numerically greater than or equal to the RPL of the gate selector.
- Call gate DPL must be numerically greater than or equal to the target code segment DPL.
- Target code segment DPL must be numerically less than or equal to the current privilege level.

5.4 : Page Level Protection

Q.6 What is page level protection ?

OR List aspects of protection related to pages.

☞ [SPPU : May-19, Dec.-19, June-22, Marks 6]

Ans. : Page level protection involves two kinds of protections

1. Restriction of addressable domain 2. Type checking

Here, The U/$\overline{S}$ and R/$\overline{W}$ fields of PDEs and PTEs are used to control access to pages.

Restricting Addressable Domain

The U/$\overline{S}$ bit is 0 for the operating system and other system software and related data. It is a supervisor level. When the 80386DX is executing at supervisor level, all pages are addressable. If U/$\overline{S}$ bit is 1, 80386DX is executing at user level. In this case, only pages that belongs to the user level are addressable.

Type Checking

At the level of page addressing two types of accessing are defined.

1. Read only access (R/$\overline{W}$ = 1) 2. Read/$\overline{Write}$ access (R/$\overline{W}$ = 0)

When 80386 is executing at supervisor level, all pages are assigned with Read/write access, whereas at user level page access depends on R/$\overline{W}$ bit in the PDE and PTE fields. If R/$\overline{W}$ bit is 1 pages are only readable and if R/$\overline{W}$ bit is 0 pages are both readable and writeable. When 80386DX is executing at user level, it cannot access page belongs to supervisor level.

5.5 : Combining Segment and Page Level Protection

Q.7 Explain how segment and page level protections function in combination. ☞ [SPPU : Dec.-22, Marks 5]

Ans. : When paging is enabled, the 80386 first evaluates segment protection, then evaluates page protection. If the processor detects a protection violation at either the segment or the page level, the requested operation cannot proceed; a protection exception occurs instead.

It is possible to define a large data segment which has some subunits that are read-only and other subunits that are read-write. In this case, the page directory (or page table) entries for the read-only subunits would have the

U/S and R/W bits set to X0, indicating no write rights for all the pages described by that directory entry (or for individual pages).

5.6 : I/O Addressing

Q.8 State the different ways supported by 80386 for I/O addressing.

☞ [SPPU : Dec.-19, Marks 2]

Ans. : The 80386 allows input/output to be performed in either of two ways :

- By means of a separate I/O address space (using specific I/O instructions)
- By means of memory-mapped I/O (using general-purpose operand manipulation instructions).

Q.9 Write a note on I/O address space in 80386.

Ans. : The 80386 provides a separate I/O address space, distinct from physical memory, that can be used to address the input/output ports that are used for external 16 devices. The I/O address space consists of 2^{16} (64K) individually addressable 8-bit ports; any two consecutive 8-bit ports can be treated as a 16-bit port; and four consecutive 8-bit ports can be treated as a 32-bit port. Thus, the I/O address space can accommodate up to 64K 8-bit ports, or up to 16K 32-bit ports.

The 80386 can transfer 32, 16, or 8 bits at a time to a device located in the I/O space.

The instructions IN and OUT move data between a register and a port in the I/O address space. The instructions INS and OUTS move strings of data between the memory address space and ports in the I/O address space.

Q.10 Write a note on memory mapped I/O in 80386.

Ans. : I/O devices also may be placed in the 80386 memory address space. Memory-mapped I/O provides additional programming flexibility. Any instruction that references memory may be used to access an I/O port located in the memory space. For example, the MOV instruction can transfer data between any register and a port; and the AND, OR, and TEST instructions may be used to manipulate bits in the internal registers of a device. Memory-mapped I/O allows to use additional addressing

modes for selecting the desired I/O device (e.g., direct address, indirect address, base register, index register, scaling).

Memory-mapped I/O, like any other memory reference, is subject to access protection and control when executing in protected mode.

5.7 : I/O Instructions

Q.11 Write a note on I/O instructions.

Ans. : The I/O instructions of the 80386 provide access to the processor's I/O ports for the transfer of data to and from peripheral devices. These instructions have as one operand the address of a port in the I/O address space. There are two classes of I/O instruction :

1. Those that transfer a single item (byte, word, or doubleword) located in a register.
2. Those that transfer strings of items (strings of bytes, words, or doublewords) located in memory. These are known as **"string I/O instructions"** or **"block I/O instructions"**.

Register I/O Instructions

The I/O instructions IN and OUT are provided to move data between I/O ports and the EAX (32-bit I/O), the AX (16-bit I/O), or AL (8-bit I/O) general registers. IN and OUT instructions address I/O ports either directly, with the address of one of up to 256 port addresses coded in the instruction, or indirectly via the DX register to one of up to 64K port addresses.

Block I/O Instructions

The block (or string) I/O instructions INS and OUTS move blocks of data between I/O ports and memory space. Block I/O instructions use the DX register to specify the address of a port in the I/O address space.

5.8 : Protection and I/O

Q.12 State the mechanisms supprted by 80386 for protecting I/O ports.

Ans. : The 80386 supports two mechanisms for protecting I/O ports in protected mode :

1. The IOPL field in the EFLAG register defines the right to use I/O related instructions (I/O privilege level).
2. The I/O permission bit map of a 80386 TSS segment defines the right to use ports in the I/O address space.

Q.13 Explain the significance of IOPL in 80386 processor.

☞ [SPPU : May-13, Dec.-19, Marks 2]

Ans. : In I/O privilege level mechanism, for execution of IN, INS, OUT, OUTS, CLI and STI instructions, the CPL of a procedure or task must be the same or a lower number than the number represented by the IOPL bits. (CPL ≤ IOPL). Any attempt by a less privileged procedure to use these instructions result in a general protection exception. The instructions IN, INS, OUT, OUTS, CLI and STI are called **sensitive instructions** because they are sensitive to IOPL.

Each task has its own unique copy of the flag register. Thus each task can have a different IOPL. The task can change IOPL. But only procedure executing at privilege level 0 can change IOPL; otherwise IOPL remains unaltered. It does not result in an exception.

Q.14 Explain the function of I/O permission bit map.

☞ [SPPU : May-18, Marks 3]

OR What I/O permission bit map ?

Ans. : The second mechanism for protecting I/O ports from unauthorized access is an optional I/O permission bit map which allows ports to be associated only with specific task. The Fig. Q.7.1 shows the I/O address bit map.

The I/O permission bit map is a bit vector. The size of the map and its location in the TSS (Task State Segment) are variable. The 80386 locates the I/O permission map. By means of the I/O map base field which is in the fixed portion of the TSS. Each bit in the map corresponds to an I/O port byte address. Thus 16-bit ports use 2-bits each and 32-bit ports use 4-bit each. To access I/O port the corresponding bit in the I/O bitmap must be 0.

When program attempts to access a port, the 80386 first compares the CPL of the task with the IOPL. If the access passes the IOPL test and an I/O bit map is compulsory, the 80386 checks the map bit corresponding to the addressed port. If corresponding bit is 0, access is granted.

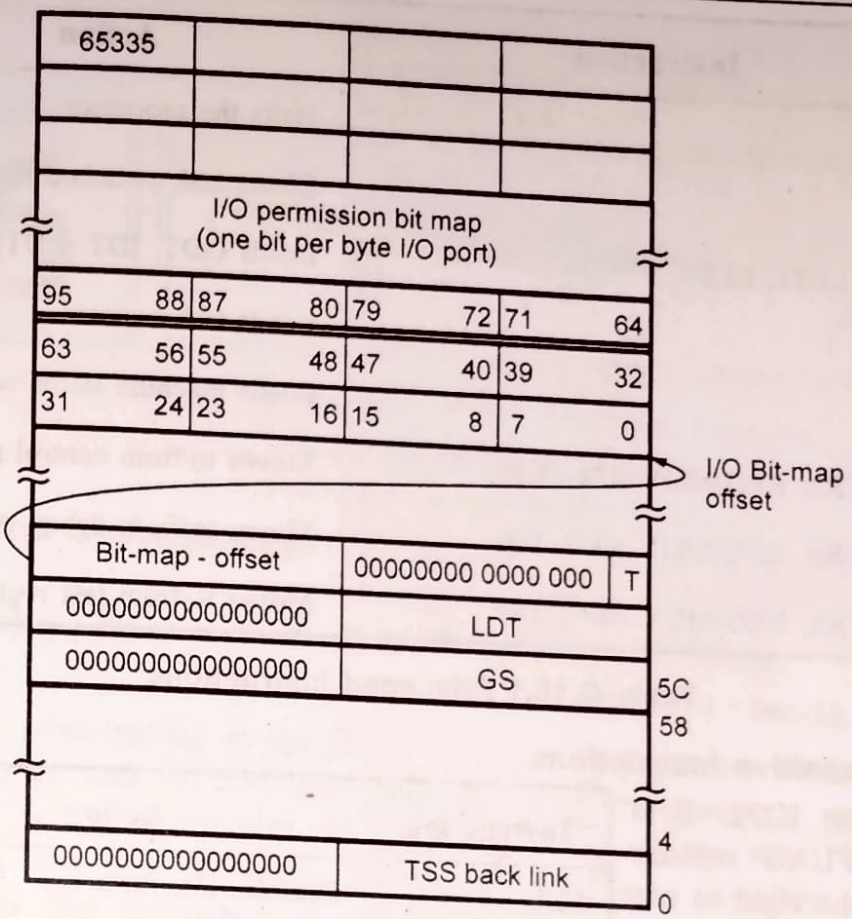


Fig. Q.14.1 I/O address bit map

Q.15 What is privileged instruction ? Explain its significance with examples.

[SPPU : May-05,13, Dec.-08,10,17, June-22, Marks 6]

Ans. : There are 19 privileged instructions supported by 80386. Privileged instructions are those that affect the segmentation and protection mechanism, alter the interrupt flag, or perform peripheral I/O. These instructions are divided in two groups.

1. Privileged Instructions (Group I)
2. IOPL - Sensitive Instructions (Group II)

Privileged Instructions

The instructions that affect the system data structures are come under first group. The instructions under this group must be executed when CPL is 0; otherwise 80386 generates general protection exception. Table Q.15.1 shows the instructions from group I (Privileged Instructions).

Instruction	Action
HLT	Halts the processor
CLTS	Clears task-switched flag
LGDT, LIDT, LLDT	Loads GDT, IDT, LDT registers
LTR	Loads task register
LMSW	Loads machine status word
MOV CRn, REG/MOV REG, CRn	Moves to/from control registers
MOV DRn, REG/MOV REG, DRn	Moves to/from debug registers
MOV TRn, REG/MOV REG, TRn	Moves to/from test registers

Table Q.15.1 Privileged instructions

IOPL Sensitive Instructions

Here, the IOPL field in the FLAG register defines the right to use I/O related instructions. Hence the instructions from this group are called **sensitive instructions**. Table Q.15.2 shows the IOPL sensitive instructions.

Instruction	Action
CLI	Disables interrupts
STI	Enables interrupts
IN, INS	Inputs data from I/O port
OUT, OUTS	Outputs data to I/O port

Table Q.15.2 IOPL - sensitive instructions

In order to execute these instructions, the CPL of a procedure or task must be the same or a lower number than the number represented by the IOPL bits (CPL ≤ IOPL).

END...

Unit V

6 Multitasking

6.1 : Introduction

Q.1 What is multitasking ?

OR Write a short note on multitasking feature of 80386.

☞ [SPPU : Dec.-19, Marks 4]

Ans. : Multitasking is the ability of a computer to run more than one program or task at the same time. On a single processor multitasking system, multiple processes do not actually run at the same time since there is only one processor. Instead, the processor switches among the processes that are active at any given time. Because of this action, it appears to the user as though the processor is executing all of the tasks at once.

Q.2 List the registers and data structures that are used in multitasking.

☞ [SPPU : May-17, Marks 2]

Ans. : • The 80386 has special registers, **special data structures** to support efficient and protected multitasking system. These are :

- Task State Segment (TSS)
- Task State Segment Descriptor
- Task Register
- Task Gate Descriptor.

6.2 : Task State Segment

Q.3 What is a TSS ? What is its size and contents ? Explain clearly indicating checks and its usage by 80386.

☞ [SPPU : Dec.-2000,08,10,17,22, May-01,08,13, Marks 7]

OR What is back-link field of TSS ? What is its offset in TSS ?

☞ [SPPU : May-02, 08, Marks 4]

OR Explain the significance of the TSS.

☞ [SPPU : Dec.-02, May-12, Marks 5]

OR What is TSS ? What are the contents of it ? Discuss its use in multitasking. ☞ [SPPU : Nov.-12, May-18, Dec.-18, Marks 8]

Ans. : Fig. Q.3.1 shows the format of a TSS. It is a special type of segment, used to manage the task. The 80386 uses TSS like a scratch-pad. It stores everything it needs to know about a task in TSS. This means that task environment (context) is stored in the TSS. TSS is not accessible to the general user program or program even at privilege level 0. The fields within TSS are accessible to only 80386. The fields of a TSS are divided into two sets : Dynamic set and Static set.

1. Dynamic Set : The 80386 updates dynamic set when it switches from one task to another task. This set includes :

- The general registers (EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI)
- The segment registers (CS, SS, DS, ES, FS, GS)
- The flag registers (EFLAGS)
- The instruction pointer (EIP)
- Back link.

The first four fields (general registers, segment registers/selectors, flags and instruction pointer) save the state of the microprocessor, 80386. Saving EIP guarantees that the task can be restarted at the point at which it was stopped and saving EFLAGs allows 80386 to execute conditional instructions properly, when the task is restarted.

The back link is used by the 80386 to keep track of a previous task. By executing a return instruction at the end of the new task, the back link selector for the previous TSS is automatically loaded into task register. This activates the previous task and restores the prior program environment.

2. Static Set : The 80386 only reads fields from this set. This set includes :

- The selector for the task's LDT
- The register (PDBR) that contains the base address of the task's page directory

31		0	
Bit Map Offset	000000000000000	T	64
0000000000000000	LDT		60
0000000000000000	GS		5C
0000000000000000	FS		58
0000000000000000	DS		54
0000000000000000	SS		50
0000000000000000	CS		4C
0000000000000000	ES		48
EDI			44
ESI			40
EBP			3C
ESP			38
EBX			34
EDX			30
ECX			2C
EAX			28
EFLAGS			24
EIP			20
CR3			1C
0000000000000000	SS2		18
EIP2			14
0000000000000000	SS1		10
EIP1			0C
0000000000000000	SS0		8
EIP0			4
0000000000000000	Back link		0

Fig. Q.3.1 Task state segment

- Pointers to the stacks for privilege levels 0-2
- The T-bit (debug trap bit) which causes the 80386 to raise a debug exception when a task switch occurs.
- The I/O map offset.

Note : TSS static set saves the selector for the task's LDT. This means that TSS descriptors must appear only in the GDT.

Task switching may change the privilege level changing the addressable domain of the program. As rule says the privilege level of the stack segment must exactly match the privilege level of the code segment at all times, the 80386 has to change stack when there is change in privilege level. Due to this previous stack segment and pointer are abandoned, and a new stack is used that corresponds to the new privilege level. When control is returned to previous level, the previous stack is restored. To store stack pointer and stack selector of the previous task fields ESP0, ESP1, ESP2, SS0, SS1, SS2 hold the stack segment pointers and stack selectors for privilege levels 0, 1 and 2.

The I/O map base holds the 16-bit offset of the beginning of the I/O permission bit map. It is implemented on a task-by-task basis and affects the hardware privilege checking only for I/O instructions.

6.3 : TSS Decriptor

Q.4 What is TSS descriptor ? ☞ [SPPU : May-10, Dec.-13, Marks 3]

OR. Explain the siginificance and format of the TSS descriptor.

☞ [SPPU : Dec.-02, May-12, Marks 5]

OR. Draw neat diagram to explain TSS descriptor.

☞ [SPPU : May-13, June-22, Dec.-22, Marks 6]

Ans. : The task state segment is defined by descriptor called **TSS descriptor**. Fig. Q.4.1 shows the task state segment descriptor. It contains fields like other segments. The B-bit in the type field indicates whether the task is busy. Tasks are not re-entrant, The B-bit allows 80386 to detect an attempt to switch to a task that is already busy. The BASE. LIMIT, and DPL fields and the G-bit and the P-bit have functions similar

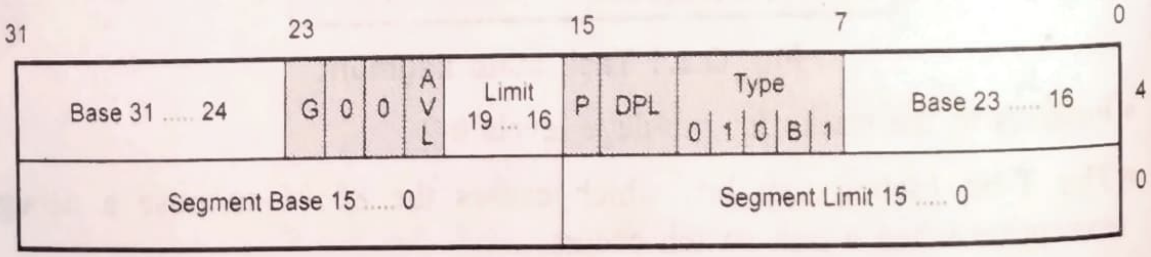


Fig. Q.4.1 Task state segment descriptor

to other descriptors. The limit field, however must have a value equal to or greater than 103 (104-1), because 80386 requires minimum 104 bytes of storage in order to perform a context save. A larger limit is permissible and it is required if an I/O permission map is present. The maximum limit for TSS is 4 Gbyte.

To access TSS descriptor, the procedure must have privilege level less than or equal to (numerically) privilege level specified by DPL field of the TSS descriptor. Usually this access is restricted for only trusted softwares, whose privilege level is zero. This can be done by setting DPL fields of TSS descriptor to zero. Thus only trusted softwares has the right to perform task switching.

6.4 : Task Register

Q.5 Explain task register. ☞ [SPPU : Dec.-08, Marks 2]

OR Specify size and function of TR. ☞ [SPPU : May-10, Marks 2]

OR Explain the role of task register in multitasking and the instructions used to modify and read TR.

☞ [SPPU : May-18, June-22, Dec.-22, Marks 6]

Ans. : The Task Register (TR) is a 16-bit register. It specifies the currently executing task by pointing to the TSS.

It has both visible portion and invisible portion. The selector in the visible portion is used to specify a TSS descriptor in the GDT and invisible portion is used to cache the base and limit values from the TSS descriptor. Holding the base and limit in the invisible portion of the Task Register makes execution of the task more efficient, because the processor does not need to repeatedly fetch these values from memory when it references the TSS of the current task. The 80386 gives two instructions to read and modify the visible protion of the task register : LTR (Load Task Register) and STR (Store Task Register).

6.5 : Task Gate Descriptor

Q.6 What is task ? ☞ [SPPU : Dec.-03, Marks 2, May-07, 13, Marks 3]

OR What is the use to task gate ? ☞ [SPPU : May-14, Marks 3]

Ans. : Task gate is a special system gate. It has its own descriptor. A task gate descriptor does not define a memory segment but instead acts as an interface point between user code and a task state segment. It provides an indirect and protected reference to a TSS. Fig. Q.6.1 shows the format of a task gate descriptor. A task gate descriptor defines a selector to a TSS descriptor which uniquely identifies a task. Like the selector to a call gate, the selector to a task gate can be used in place of a selector to a code segment in FAR JMP and FAR CALL instructions.

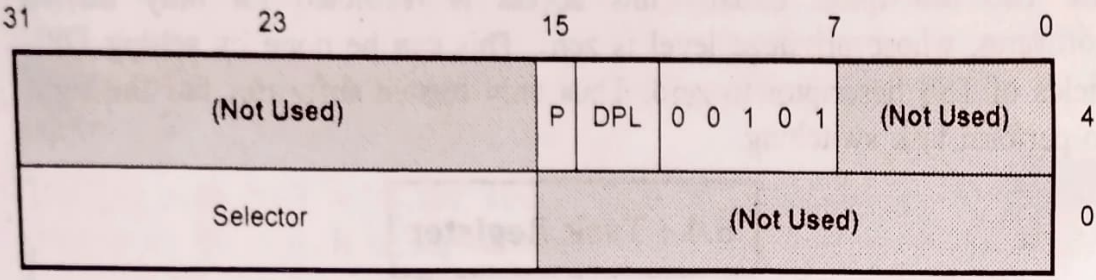


Fig. Q.6.1 Task gate descriptor

The DPL field of a task gate controls the right to use the descriptor to cause a task switch. Procedure selects a task gate descriptor only when the maximum of selector's RPL and the CPL of the procedure is numerically less than or equal to the DPL of the descriptor.

$$\text{MAX (CPL, RPL)} \leq \text{task gate DPL}$$

Now if DPL privilege level is 0. Then privilege constraint prevents untrusted procedures (procedures having privilege level from 1 to 3) from causing task switch. But through task gates we can switch from lower privilege to higher privilege because when a task gate is used, the DPL of the target TSS descriptor is not used for privilege checking. Thus a procedure that has access to a task gate has the power to cause a task switch.

6.6 : Task Swithcing

Q.7 Explain the cases during which 80386 does task switching.

Ans. : The 80386 does task switching in any of four cases :

1. A long jump or call instruction contains a selector which refers to a TSS descriptor. This is the simplest method and can be easily implemented by the operating system Kernel at the end of a time slice.
2. The selector in a long jump or call instruction refers to a task gate. In this case the selector for the destination TSS is in the task gate. This

3. The interrupt selector refers to a task gate in the interrupt descriptor table. The task gate contains the selector for the new TSS. If the access passes all the privilege level tests, the selector and descriptor for the interrupt task will be loaded into the task register. The Nested Task (NT) bit in the EFLAGs register will be set.
4. An IRET instruction is executed with the NT bit in the EFLAGs register set. The IRET instruction uses the back link selector in the TSS to return execution to the interrupted task.

Q.8 Explain the process of task switching in processor 80386.

☞ [SPPU : May-02,03,07,12 Dec.-02,03,07, Marks 8]

OR Write short note on task switching in 80386

☞ [SPPU : May-09, Marks 6]

Ans. : Fig. Q.8.1 shows task switch operation.

Steps involved in Task Switching (Without Task Gate) :

1. **Privilege Check :** If the DPL of the TSS descriptor is numerically greater than or equal to the maximum of CPL and the RPL of the selector then only the current task is allowed to switch to the designated task.
2. **Limit and Present Bit Checking :** The TSS descriptor for the designated task is checked for its limits and presence.
3. **Saving the State of the Current Task :** The 80386DX 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, DS, SS, FS, GS, the flag register and EIP). The EIP field of the TSS points to the instruction after the one that caused the task switch. The selector for the current task is saved as a back link selector in the new task.
4. **Loading of Task Register :** The visible portion of the task register is loaded with the selector of the designated task's TSS descriptor. This sets the TS (Task switch) bit in the Machine Status Word (MSW). The B bit in the new task's descriptor is marked busy. Then the corresponding task state descriptor is read from the GDT and loaded into the task register cache (hidden portion of task register).

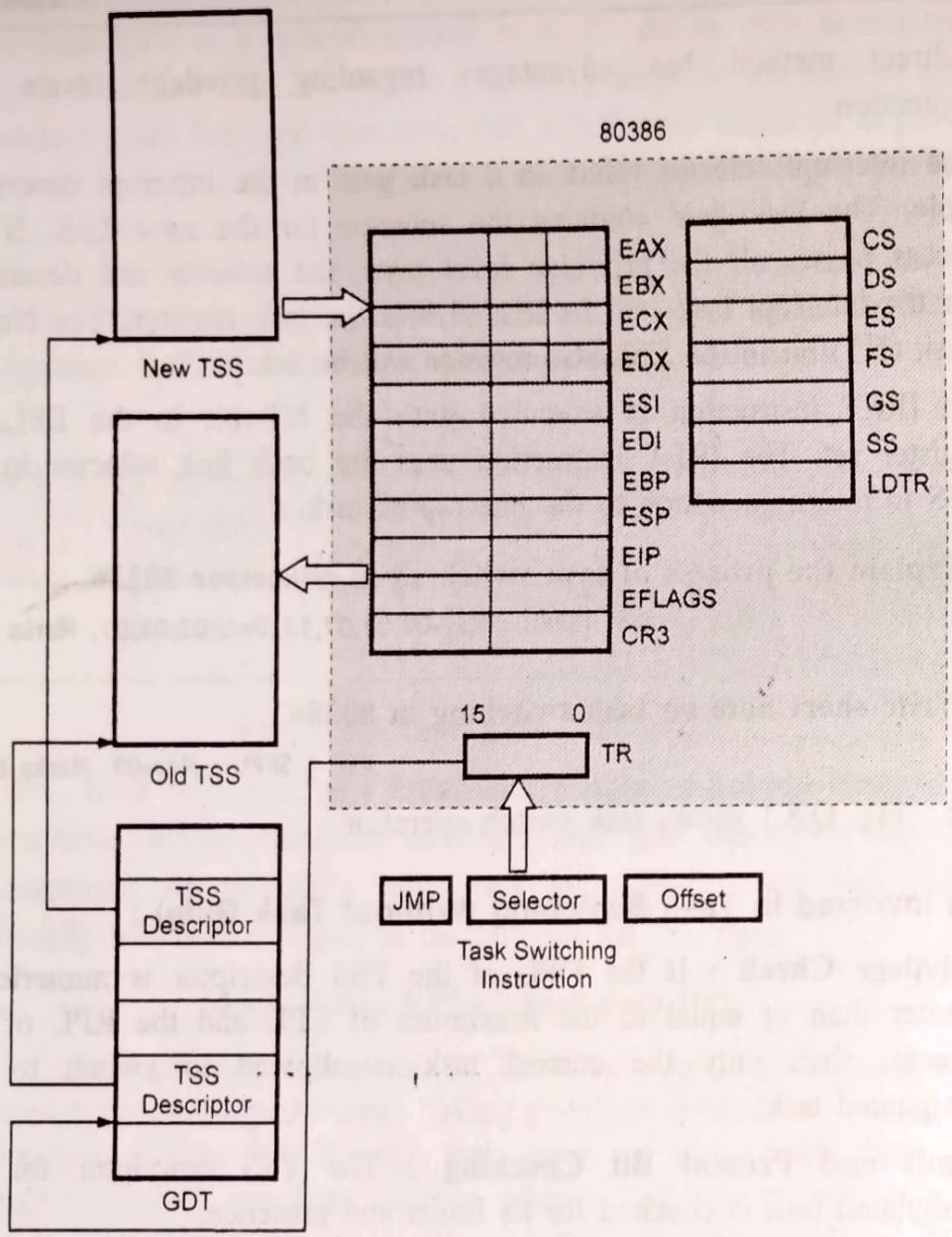


Fig. Q.8.1 Task switch operation

5. **Resuming Execution :** Finally, 80386 starts execution of designated task, with the instruction pointed by the new contents of the code segment selector (CS) and instruction pointer (EIP).

Steps involved in Task Switching (Using Task Gate) :

Privilege Check : The DPL of the task gate is compared with the CPL and RPL of the gate selector. If the DPL of the task gate is numerically greater than or equal to the maximum of CPL and the RPL of the gate selector, the current task is allowed to switch to the designated/new task. The remaining steps are similar excepts that for loading selector for TSS descriptor into TR task gate is referred instead of CALL or JMP instruction.

In case of exceptions, interrupts and IRETs regardless of the DPL of the new task gate or TSS descriptor, the current task is allowed to switch to the new task.

6.7 : Task Linking

Q.9 Explain : NT (Nested Task) bit.

☞ [SPPU : Dec.-02, May-08, 09, Marks 6]

OR What is 'Nested task' ? How are they handled in 80386 ?

☞ [SPPU : Dec.-08, Marks 2]

OR Write a note on task linking. ☞ [SPPU : Dec.-19, Marks 4]

Ans. : Nested tasks are analogous to nested subroutines. If task switch was caused by a FAR CALL instruction or by an exception, fault or trap, the new task is considered to be nested within the old task that invoked it. In any of these cases, when the task executes an IRET instruction, the 80386 automatically task-switches back to the task that invoked it. To do so, there is a mechanism of linking the tasks, which is equivalent of a call/return stack. The task linking mechanism consists of Back Link and NT (Nested Task) flag. The Back Link is used to keep a track of a previous task. By executing a IRET instruction at the end of the new task, the back link selector for the previous TSS is automatically loaded into task register. This activates the previous task and restores the prior program environment. The 80386 sets the NT (Nested Task) flag in the EFLAGS register, when one system task invokes another task. The 80386 uses NT as a flag so that it can tell whether the Back Link field in the current TSS is valid.

6.8 : Task Address Space

Q.10 What is the necessity of distinct address space for different tasks ?

Ans. : The ability for tasks to have distinct address spaces is an important aspect of 80386 protection. A module in one task cannot interfere with a module in another task if the modules do not have access to the same address spaces. The flexible memory management features of the 80386 allow systems designers to assign areas of shared address space to those modules of different tasks that are designed to cooperate with each other.

Q.11 Explain the two ways of arranging the linear-to-physical mappings of tasks.

Ans. : The choices for arranging the linear-to-physical mappings of tasks fall into two general classes :

1. One linear-to-physical mapping shared among all tasks

When paging is not enabled, this is the only possibility. Without page tables, all linear addresses map to the same physical addresses. When paging is enabled, this style of linear-to-physical mapping results from using one page directory for all tasks.

2. Several partially overlapping linear-to-physical mappings

This style is implemented by using a different page directory for each task. Because the PDBR (page directory base register) is loaded from the TSS with each task switch, each task may have a different page directory.

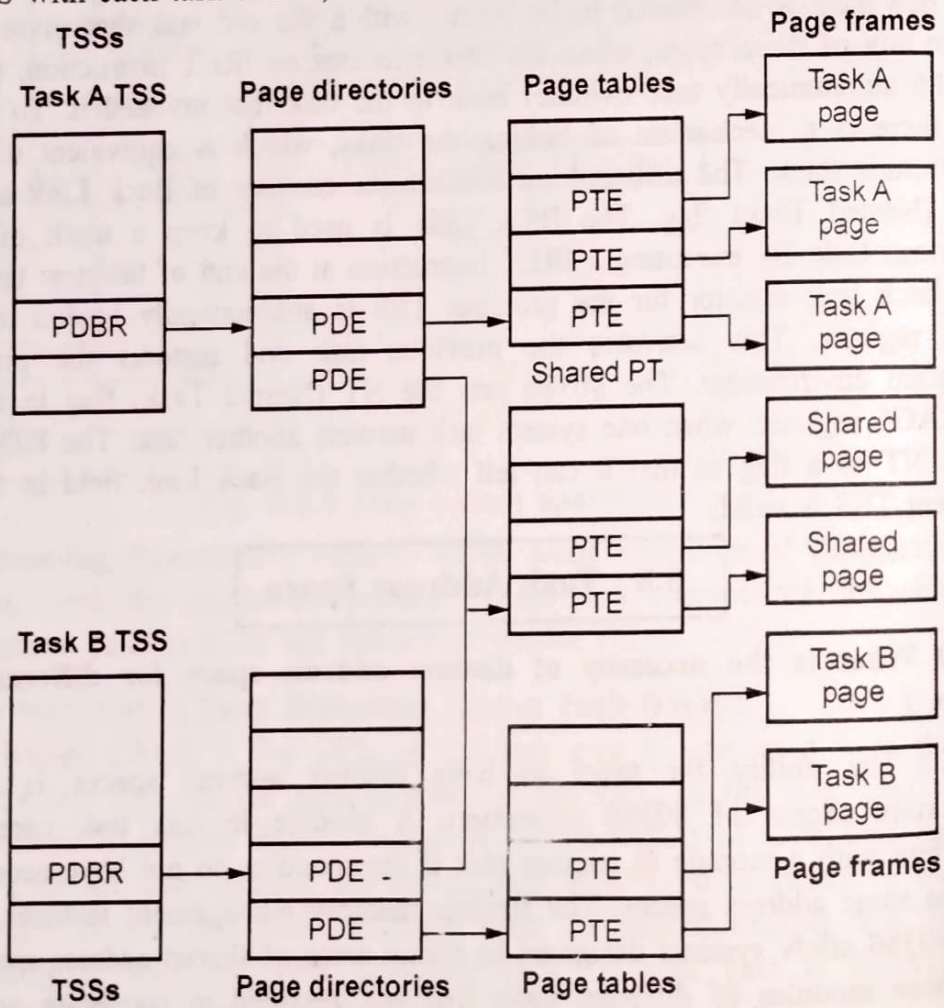


Fig. Q.11.1 Partially-overlapping linear spaces

In theory, the linear address spaces of different tasks may map to completely distinct physical addresses. If the entries of different page directories point to different page tables and the page tables point to different pages of physical memory, then the tasks do not share any physical addresses.

In practice, some portion of the linear address spaces of all tasks must map to the same physical addresses. The task state segments must lie in a common space so that the mapping of TSS addresses does not change while the processor is reading and updating the TSSs during a task switch. The linear space mapped by the GDT should also be mapped to a common physical space; otherwise, the purpose of the GDT is defeated. Fig. Q.11.1 shows how the linear spaces of two tasks can overlap in the physical space by sharing page tables.

Q.12 Write a short note on Task Logical Address Space.

OR Explain the three ways to creat common logical to physical address-space mappings.

Ans. : By itself, a common linear-to-physical space mapping does not enable sharing of data among tasks. To share data, tasks must also have a common logical-to-linear space mapping; i.e., they must also have access to descriptors that point into a shared linear address space. There are three ways to create common logical-to-physical address-space mappings :

1. **Via the GDT :** All tasks have access to the descriptors in the GDT. If those descriptors point into a linear-address space that is mapped to a common physical-address space for all tasks, then the tasks can share data and instructions.
2. **By sharing LDTs :** Two or more tasks can use the same LDT if the LDT selectors in their TSSs select the same LDT segment. Those LDT-resident descriptors that point into a linear space that is mapped to a common physical space permit the tasks to share physical memory. This method of sharing is more selective than sharing by the GDT; the sharing can be limited to specific tasks. Other tasks in the system may have different LDTs that do not give them access to the shared areas.
3. **By descriptor aliases in LDTs :** It is possible for certain descriptors of different LDTs to point to the same linear address space. If that linear address space is mapped to the same physical space by the page mapping of the tasks involved, these descriptors permit the tasks to

share the common space. Such descriptors are commonly called **aliases.** This method of sharing is even more selective than the prior two; other descriptors in the LDTs may point to distinct linear addresses or to linear addresses that are not shared.

END...

Unit V

7 Virtual 8086 Mode

7.1 : Introduction

Q.1 Explain the advantages of virtual 86 mode.

☞ [SPPU : May-05, Dec.-05, Marks 3]

OR What is virtual 8086 mode ?

☞ [SPPU : May-04,05,06,07,08,09,12,16, Dec.-05,06, Marks 6]

OR List any three differences between Virtual 86 mode and 8086.

☞ [SPPU : Dec.-18, Marks 3]

OR Write short note on virtual 8086 mode.

☞ [SPPU : May-18, Dec.-19, Marks 3]

OR Explain features of "Virtual 8086 mode".

☞ [SPPU : May-19, June-22, Marks 6]

Ans. : In multitasking system, it is necessary to switch back and forth between real and protected mode. Because in multitasking system, there is a mixture of tasks, some use segment-offset addressing (Real mode addressing) and some use descriptors (Protected mode addressing). The 8086 virtual mode solves this problem.

Features of Virtual 8086 Mode

1. A 80386 operating in protected mode can easily switch to virtual 8086 mode to execute a time slice of an 8086 program and then easily switch back to protected mode to execute a time slice of protected mode task.
2. The 80386 allows execution of one or more 8086, 8088, 80186 or 80188 programs in an 80386 protected mode environment, as different tasks in the virtual 8086 mode.

3. In 8086 virtual mode, the 80386 treats the segment registers exactly the same way as it does in real mode. Therefore, the **address range of a virtual 8086 mode task is 1 Mbyte.**
4. The segment and offset registers together give the linear address instead of physical address.
5. The physical address is generated from the linear address with the help of page translation. Thus the physical address may be anywhere in the 4 gigabyte memory addressable by the 80386.
6. In 8086 virtual mode, the 80386 provides mechanism to selectively trap and manage input/output and interrupt activity. Using software it is possible to determine the Input/Output Privilege Level (IOPL) that selectively controls input/output transfer and it is also possible to use the input/output port permission map to selectively control access to input/output ports.

7.2 : Executing 8086 Code

Q.2 List the instructions in virtual 8086 mode.

Ans. : In virtual mode, 80386 can execute normal 8086 instructions as well as new instructions introduced by 80186/80188, 80286 and 80386 as listed below.

1. New instructions introduced by 80186/80188 and 80286

- PUSH immediate data
- PUSH ALL and POP ALL (PUSH A and POP A)
- Multiply immediate data
- Shift and rotate by immediate count
- String I/O
- ENTER and LEAVE
- BOUND

2. New instructions introduced by 80386

- LSS, LFS, LGS instructions
- Long displacement conditional jumps

- Single bit instruction
- Bit scan
- Byte set on condition
- Double shift instruction
- Move with sign/zero extension
- Generalized multiply

7.3 : Memory Management in Virtual Mode

Q.3 Describe how physical address is obtained in virtual 8086 mode.

OR Explain linear address formation in virtual mode of 80306.

☞ **[SPPU : Dec.-17,18, 22, June-22, Marks 6]**

Ans. : In virtual 8086 mode, the segment register contents are used to generate linear address with the help of offset. The linear address is generated by adding the contents of the appropriate segment register which are shifted left by 4 bit to an effective address/offset. Fig. Q.3.1 shows virtual 8086 mode address generation.

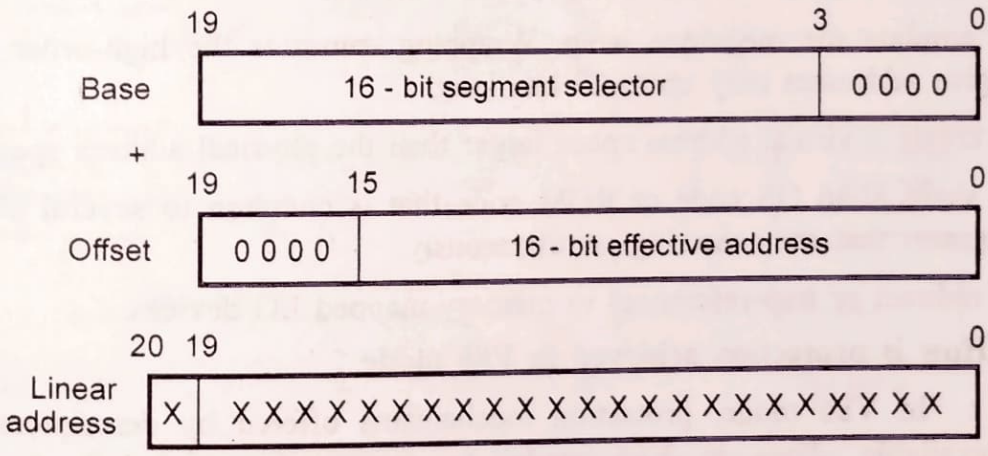


Fig. Q.3.1 Virtual 8086 mode address generation

If there is a carry generated after addition of shifted segment register contents and effective address, unlike 8086, resulting 21 bit address is a linear address.

Q.4 Explain the two way for implementing the 8086 operating system.

Ans. :

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 re-implement 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.

Q.5 State how paging is useful for V86 tasks.

Ans. : Paging is not necessary for a single V86 task; however, paging is useful or necessary for any of the following reasons :

- To create multiple V86 tasks. Each task must map the lower megabyte of linear addresses to different physical locations.
- To emulate the megabyte wrap. Wrapping truncates the high-order bit to give addresses only up to 20 bits long.
- To create a virtual address space larger than the physical address space.
- To share 8086 OS code or ROM code that is common to several 8086 programs that are executing simultaneously.
- To redirect or trap references to memory-mapped I/O devices.

Q.6 How is protection achieved in V86 mode ?

Ans. : In V86 mode, protection mechanisms offered by descriptors is not available. Thus in V86 mode, to protect the systems software designers may follow either of these approaches :

- Reserve the first megabyte (plus 64 kilobytes) of each task's linear address space for the 8086 program. An 8086 task cannot generate addresses outside this range.

- Use the U/S bit of page-table entries to protect the virtual-machine monitor and other systems software in each virtual 8086 task's space. When the processor is in V86 mode, CPL is 3. Therefore, an 8086 program has only user privileges. If the pages of the virtual-machine monitor have supervisor privilege, they cannot be accessed by the 8086 program.

7.4 : Entering and Leaving Virtual 8086 Mode

Q.7 Write a note on entering and leaving virtual 8086 mode.

☞ [SPPU : Dec.-17, 22, May-18, 19, Marks 6]

OR How to switch from real mode to VM-86 mode of 80386.

☞ [SSPU : May-08,09,16, Marks 2]

OR How to switch from protected mode to virtual 86 mode.

☞ [SPPU : May-12, Marks 4]

Ans. : The 80386 enters or leaves 8086 virtual mode due to any of the three reasons as shown in Fig. Q.7.1.

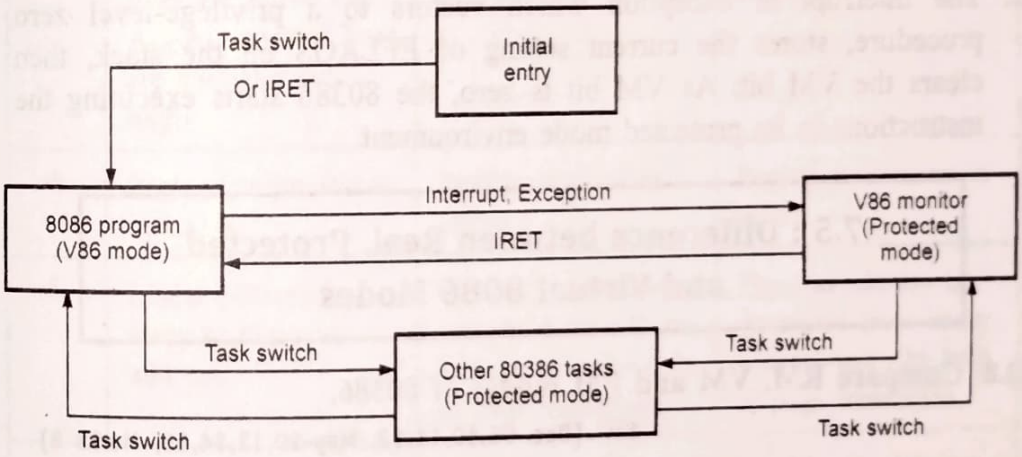


Fig. Q.7.1 Entering and leaving an 8086 program

1. An interrupt that vectors to a task gate.
2. An action of the schedule of the 80386 operating system.
3. An IRET when the NT (Nested Task) flag is set.

Entering 8086 Virtual Mode : The 80386 can enter 8086 virtual mode by either of two means :

1. A task switch to an 80386 task loads the image of EFLAGs from the new TSS. If the VM bit in EFLAGS register is set, the 80386 enters virtual 8086 mode to execute the new task. If the VM bit is not set, the 80386 executes the new task as a normal protected mode task.
2. An IRET from a procedure that loads the EFLAGs image changes the VM bit if the Current Privilege Level (CPL) at the time of IRET is zero. If changed status of the VM bit is 1 then 80386 enters in 8086 virtual mode.

Leaving 8086 Virtual Mode : The 80386 leaves the 8086 virtual mode when an interrupt or exception occurs.

1. A task switching from a 8086 virtual task to any other task caused by interrupt or exception loads EFLAGS from the TSS of the new task. If the new TSS is an 80386 TSS and the VM bit is zero, or if the TSS is an 80286 TSS, the 80386 clears the VM bit of EFLAGS. It then loads the segment registers as defined by the new TSS and begins executing the instructions of the new task according to 80386 protected mode description.
2. The interrupt or exception which vectors to a privilege-level zero procedure, stores the current setting of EFLAGS on the stack, then clears the VM bit. As VM bit is zero, the 80386 starts executing the instructions in its protected mode environment

7.5 : Difference between Real, Protected and Virtual 8086 Modes

Q.8 Compare RM, VM and PM modes of 80386.

☞ [Dec.-05,10,11,12, May-10,13,14,15, Marks 8]

OR Difference between real mode and virtual 8086 mode.

☞ [SPPU : June-22, Dec.-22, Marks 6]

Ans. :

Sr. No.	Real mode	Protected mode	Virtual 8086 mode
1.	In this mode 80386 maintains the compatibility of the object code with 8086. It supports same architecture as 8086, but it can access the 32-bit register set of 80386DX.	This mode provides a sophisticated memory management and the hardware-assisted protection mechanism.	This mode allows the execution of real mode programs that are incapable of running directly in protected mode.
2.	Memory size is limited to 1 Mbyte.	It increases the linear address space to 4 GB and allows the running of virtual memory programs of 64 terabytes.	Memory address range of virtual 8086 mode task is limited by 1 Mbyte.
3.	Only A_2-A_{19} address lines are active and A_{20}-A_{31} address lines are normally held high.	All address lines are active	All address lines are active.
4.	Paging mechanism is in-active	Paging mechanism is active.	Paging mechanism is active.
5.	Linear address is same as physical address.	Physical address is generated from linear address with the help of page translation.	Physical address is generated from linear address with the help of page translation.
6.	Protection mechanism is not available	Protection mechanism is available	Provides mechanism to selectively trap and manage I/O and interrupt activity.
7.	Multitasking is not supported	Multitasking is supported	Multitasking is supported.

8.	When 80386 is reset or powered up it is initialized in real mode. It is possible to enter into real mode from protected mode by resetting the PE bit of the CR0 register.	80386 enters into protected mode by setting PE bit in the CR0 register to logic 1.	By setting VM bit in EFLAGS register to logic 1, the 80386 enters in virtual 8086 mode.

END...

Unit VI

8 Interrupts and Exceptions

8.1 : Introduction

Q.1 Explain the sources of interrupt and exception.

☞ [SPPU : Dec.-19, Marks 3]

Ans. : Sources for External Interrupts

- Maskable interrupts (INTR)
- Nonmaskable interrupt (NMI)

Sources for Exceptions • **Processor detected** : These are further classified as faults, traps and aborts.

- **Programmed** : The instructions INTO, INT 3, INT n and BOUND can trigger exceptions. These instructions are often called "software interrupts", but the processor handles them as exceptions.

Q.2 What do you mean by faults, taps and aborts ?

☞ [SPPU : May-17, 19, June-22, Dec.-22, Marks 6]

Ans. : Exceptions are classified as faults, traps or aborts depending on the way they are reported and whether restart of the instruction that caused the exception is supported.

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

8.2 : Indentifying Interrupts

Q.3 List the assignment of interrupt and exception identifiers.

Ans. : • The processor associates an identifying number with each different type of interrupt or exception.

• Table Q.3.1 shows the assignment of interrupt and exception identifiers.

Interrupt number	Cause of exception	Description
0	DIV, IDIV	Divide error
1	All	Debug exceptions
3	INT	Breakpoint
4	INTO	Overflow
5	BOUND	Bounds check
6	Any undefined opcode or LOCK used with wrong instruction	Invalid opcode
7	ESC or WAIT	Coprocessor not available
8	INT vector is not within IDTR limit	Interrupt table limit too small
9-11		Reserved
12	Memory operand crosses offset 0 or 0FFFFH	Stack fault
13	Memory operand crosses offset 0FFFFH or attempt to execute past offset 0FFFFH or instruction longer than 15 bytes	Pseudo-protection exception

14,15		Reserved
16	ESC or WAIT	Coprocessor error
17-31	Reserved	
32-255	Available for external	Interrupts via INTR pin

Table Q.3.1 Interrupts and exception ID assignment

8.3 : Enabling and Disabling Interrupts

Q.4 Write a note on enabling an disabling interrupts.

☞ [SPPU : Dec.-19, June-22, Marks 6]

Ans. : Certain conditions and flag settings cause the processor to inhibit certain interrupts and exceptions at instruction boundaries. These conditions and settings are :

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

IF Masks INTR : The IF (interrupt-enable flag) controls the acceptance of external interrupts signalled via the INTR pin. When IF = 0, INTR interrupts are inhibited; when IF = 1, INTR interrupts are enabled.

RF Masks Debug Faults : The RF bit in EFLAGS controls the recognition of debug faults.

MOV or POP to SS Masks Some Interrupts and Exceptions : Software that needs to change stack segments often uses a pair of instructions; for example :

MOV SS, AX MOV ESP, StackTop

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.

8.4 : Priority Among Simultaneous Interrupts and Exceptions

Q.5 State the priorities of different interrupts of 80386.

Ans. :

Priority	Interrupt /Exception
HIGHEST	Faults except debug faults
	Trap instructions INTO, INT n, INT 3
	Debug traps for this instruction
	Debug faults for next instruction
	NMI interrupt
LOWEST	INTR interrupt

Table Q.5.1 Priority among simultaneous interrupts and exceptions

8.5 : Interrupt Descriptor Table

Q.6 Write a note on interrupt descriptor table.

☞ [SPPU : May-17, Dec.-18, 22, June-22, Marks 6]

Ans. : In protection mode, each interrupt or exception is associated with a descriptor which gives the information about interrupt service routine. These descriptors are stored in a special descriptor table called the Interrupt Descriptor Table or IDT. This table can be located anywhere in memory. Like the GDT and LDTs, the IDT is an array of 8 byte descriptors. The base address and limit for the interrupt descriptors table are loaded into the Interrupt Descriptor Table Register (IDTR) as shown in Fig. Q.6.1.

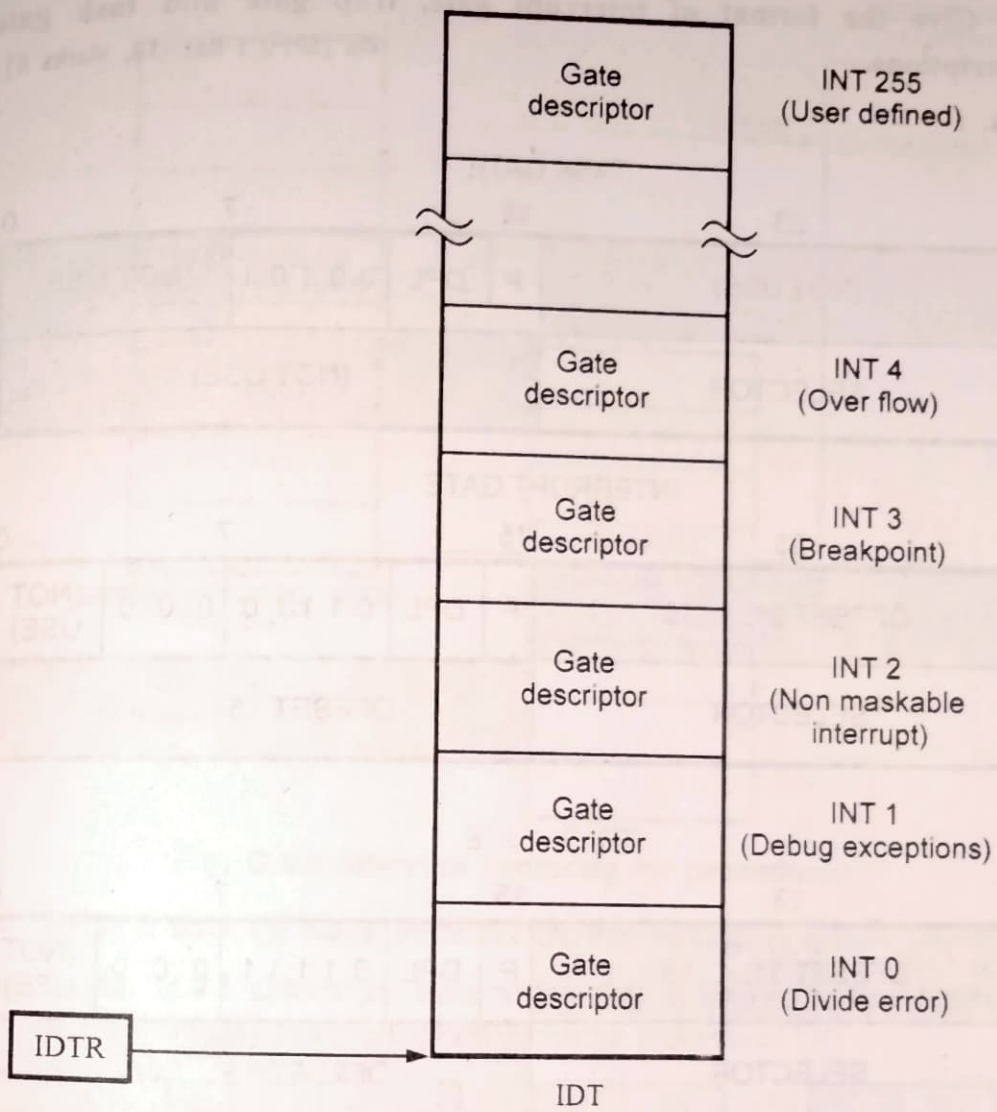


Fig. Q.6.1 Interrupt descriptor table and IDTR

LIDT (Load IDT register)/ SIDT (Store IDT register) instructions are used to load/store the IDT register with the linear base address and limit values, respectively.

Because there are only 256 identifiers, the IDT need not contain more than 256 descriptors. There are three types of descriptors can be used in the IDT.

- Trap gate descriptor
- Interrupt gate descriptor
- Task gate descriptor

Q.7 Give the format of interrupt gate, trap gate and task gate descriptions. [SPPU : Dec.-18, Marks 6]

Ans. :

TASK GATE

31 – 23	15		7 – 0		
(NOT USE)	P	DPL	0 0 1 0 1	(NOT USE)	4
SELECTOR	(NOT USE)				0

INTERRUPT GATE

31 – 23	15		7 – 0			
OFFSET 31.....16	P	DPL	0 1 1 1 0	0 0 0	(NOT USE)	4
SELECTOR	OFFSET 15 0					0

TRAP GATE

31 – 23	15		7 – 0			
OFFSET 31.....16	P	DPL	0 1 1 1 1	0 0 0	(NOT USE)	4
SELECTOR	OFFSET 15 0					0

Fig. Q.7.1 IDT gate descriptors

8.6 : Interrupt Tasks and Interrupt Procedures

Q.8 Write a note on interrupt procedure. [SPPU : Dec.-18, Marks 6]

Ans. : 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. Q.8.1. 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.

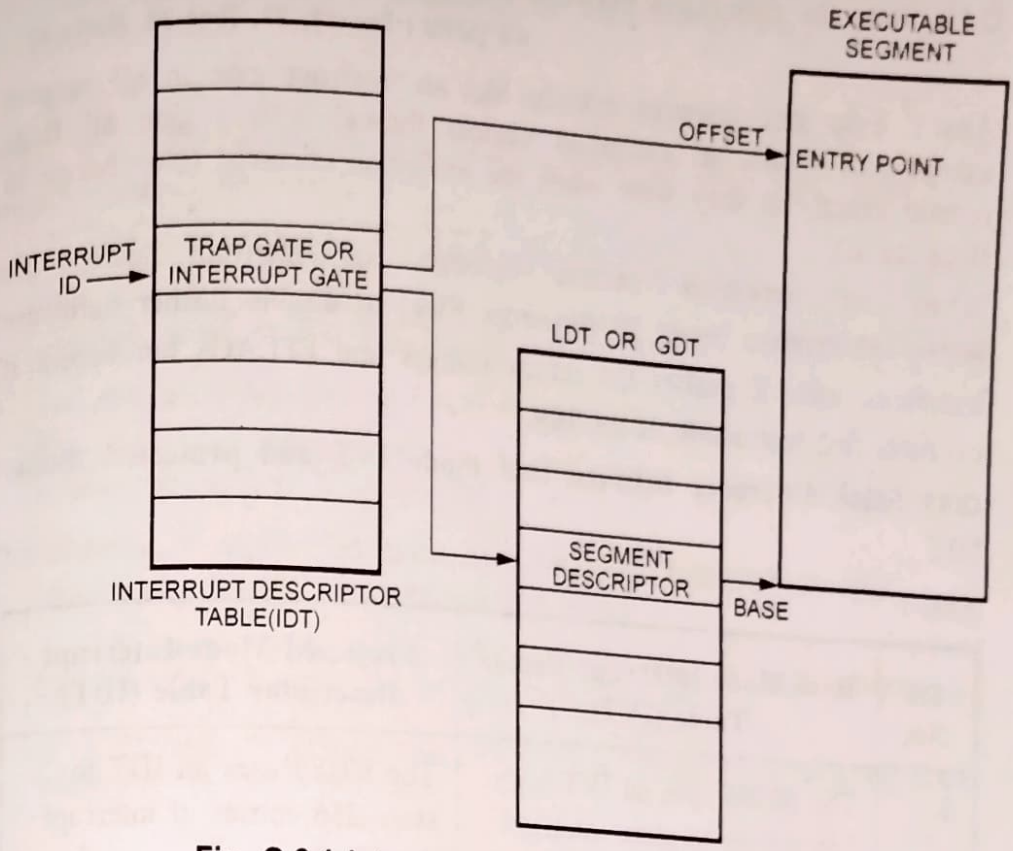


Fig. Q.8.1 Interrupt vectoring for procedures

Q.9 Write a note on stack of interrupt procedure.

Ans. : Fig. Q.9.1 shows the information that is stacked before control is transferred to interrupt or exception handling procedure.

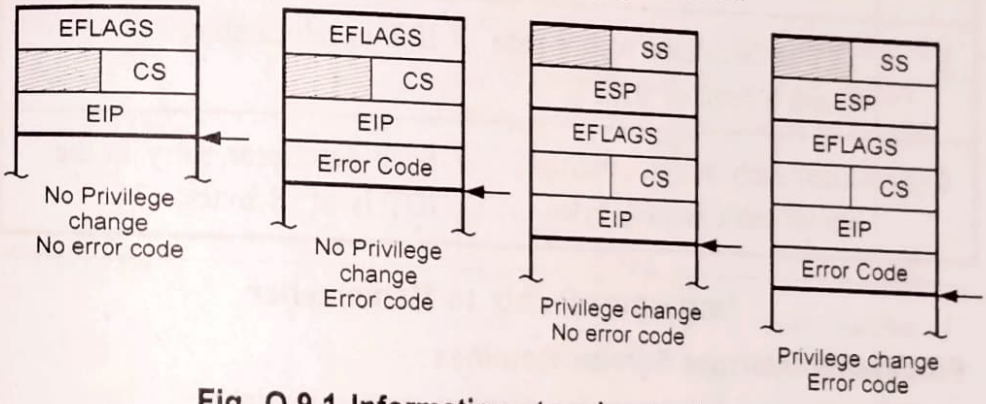


Fig. Q.9.1 Information stored onto the stack

As shown in Fig. Q.9.1 if an exception cause a special error code it is saved on the stack. If exception requires a privilege change then old stack segment and pointer are also saved on the stack.

Q.10 State the difference between trap gates in interrupt gate.

☞ [SPPU : Dec.-18, 22, June-22, Marks 6]

Ans. : Trap gate operates exactly like an interrupt gate in all respect except one. When an exception vectors through a trap gate all flags remain exactly as they were when the exception occurred (No change in flags status).

When an exception vectors through an interrupt gate, the 80386/486/Pentium resets IF (Interrupt Flag) to disable further hardware interrupts, after it pushes the return address and EFLAGs but before it executes first instruction of the ISR.

Q.11 State difference between real mode IVT and protected mode IDT

Ans. :

Sr. No.	Real Mode Interrupt Vector Table (IVT)	Protected Mode Interrupt Descriptor Table (IDT)
1.	The 80386 uses an IVT with 256 entries to store interrupt vectors.	The 80386 uses an IDT to store 256 entries of interrupt descriptors.
2.	In IVT, vectors specify the segment and offset of the interrupt handler.	Descriptor specify the selector, offset, type of segment and its privilege level.
3.	IVT is initialized with a base 0 and a limit of 03FFH.	IDT is relocatable.
4.	Each entry in the interrupt vector table is of 4 bytes.	Each descriptor entry in the IDT is of 8 bytes.

Important Points to Rememeber

Processing Interrupt Service Routines

- IDT stores the descriptors for interrupt service routines.
- Only trap gate, interrupt gate and task gate descriptors are allowed.
- Operate like programmed procedures/subroutines.
- Before transferring saves all register that are used.

- ISRs are invoked by interrupts or exceptions instead of CALL instructions.
- ISRs terminate with IRET instead of RET instructions.

Q.12 State the advantages and disadvantages of using task gate over trap and interrupt states.

Ans. : • The entire context of the interrupted task is saved automatically.

- The exception handler does not need to be concerned with contaminating the interrupted code.
- The exception handler can run at any privilege level.
- The exception handler can use it own private code and data space because it can have its own LDT.

Drawbacks of using task gate over trap and interrupt gates :

- More time is required to perform task switch.
- A task gate cannot specify where in the task to begin execution. Dormant tasks always resume where they left off.
- It is difficult to retrieve any information about the interrupted code when it is in a different task.

8.7 : Error Code

Q.13 Draw and explain the error code format of 80386.

Ans. : The format of error code is as shown in Fig. Q.13.1.

EX (External, bit 0) — When this bit is set, it indicates that the exception was initially caused by an external (hardware) interrupt or that it occurred when the 80386 was already handling another exception.

I (IDT, bit 1) — This bit is like a second TI bit. When set, it indicates that the index portion of the error code refers to a descriptor in the IDT.

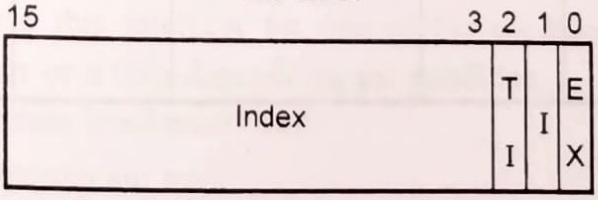


Fig. Q.13.1 Error code format

TI (Table Indicator, bit 2)	This is just like the flag of the same name in a segment selector. When set, the index portion of the error code refers to a descriptor in the current LDT at the time the exception occurred. When clear, the index selects a GDT descriptor. This bit is undefined when I is set.
INDEX (Descriptor Index, bits 3 through 15)	This field uniquely identifies a descriptor in the GDT, LDT or IDT, depending on the status of the I and TI bits.

8.8 : Exception Conditions

Point to Remember :

Description	Interrupt number	Return address points to faulting instruction	Type	Function that can generate the exception
Divide error	0	YES	FAULT	DIV, IDIV
Debug exceptions	1	YES	TRAP	Any Instruction
NMI	2	NO	NMI	INT 2 or NMI
Break point	3	NO	TRAP	One-byte INT 3
Overflow	4	NO	TRAP	INTO
Bounds check	5	YES	FAULT	BOUND
Invalid opcode	6	YES	FAULT	Any illegal instruction
Coprocessor not available	7	YES	FAULT	ESC, WAIT
Double fault	8	YES	ABORT	Any instruction that can generate an exception

Coprocessor segment overrun	9	NO	ABORT	Any operand of an ESC instruction that wraps around the end of a segment.
Invalid TSS	10	YES	FAULT	JMP, CALL, IRET, any interrupt
Segment not present	11	YES	FAULT	Any segment-register modifier
Stack exception	12	YES	FAULT	Any memory reference through SS
General protection	13	YES	FAULT/ ABORT	Any memory reference or code fetch
Page fault	14	YES	FAULT	Any memory reference or code fetch
Reserved by Intel	15	—	—	—
Coprocessor error	16	YES	FAULT	ESC, WAIT
Two-byte SW Interrupt	0-255	NO	TRAP	INT n

Q.14 Explain any four exception conditions that occur in 80386.

Ans. :

Interrupt 1 : Debug Exceptions

The 80386 triggers this interrupt for one of the conditions; whether the exception is a fault or a trap depends on the condition :

- Instruction address breakpoint fault.
- Data address breakpoint trap.

- General detect fault.
- Single-step trap.
- Task-switch breakpoint trap.
- The 80386 Pentium does not push an error code for this exception. An exception handler can examine the debug registers to determine which condition caused the exception.

Interrupt 2 : Non Maskable Interrupt

- As the name suggests, this interrupt can not be disabled by any software instruction. This interrupt is activated by low to high transition on 80386 NMI input pin. In response, 80386 triggers a type 2 interrupt.

Interrupt 3 : Breakpoint

- The type 3 interrupt is used to implement BREAK POINT function in the system. The type 3 interrupt is produced by execution of the INT 3 instruction. Break point function is often used as a debugging aid in cases where single stepping provides more detail than wanted. When you insert a breakpoint, the system executes the instructions upto the breakpoint, and then goes to the breakpoint procedure. In the break point procedure you can write a program to display register contents, memory contents and other information that is required to debug your program. You can insert as many breakpoints as you want in your program.

Interrupt 4 : Overflow Interrupt

- The type 4 interrupt is used to check overflow condition after any signed arithmetic operation in the system.
- If the overflow flag is set, the 80386 will trigger a type 4 interrupt after executing the INTO instruction.

END...

Unit VI

9 Introduction to Microcontrollers

9.1 : Microcontroller and Embedded Processors

Q.1 What is mean by microcontroller ?

Ans. : A device which contains the microprocessor with integrated peripherals like memory, serial ports, parallel ports, timer/counter, interrupt controller, data acquisition interfaces like ADC, DAC is called microcontroller.

Q.2 Draw and explain the simplified block diagram of microcontrolller.

☞ [SPPU : June-22, Marks 5]

Ans. : • The key feature of microprocessor based computer system is that it is possible to design a system with a great flexibility. It is possible to configure a system as large system or small system by adding suitable peripherals.

- On the other hand, the microcontroller incorporates all the features that found in microprocessor. However, it has also added features to make a complete microcomputer system on its own. The microcontroller has built-in ROM, RAM, parallel I/O, serial I/O, counters and a clock circuit. Fig. Q.2.1 shows the simplified block diagram of a microcontroller. (Refer Fig. Q.2.1 on next page.)
- As shown in the Fig. Q.2.1, the microcontroller has on-chip (built-in) peripheral devices. These on-chip peripherals make it possible to have single-chip microcomputer system. There are few more advantages of built-in peripherals :
  - Built-in peripherals have smaller access times hence speed is more.
  - Hardware reduces due to single chip microcomputer system.

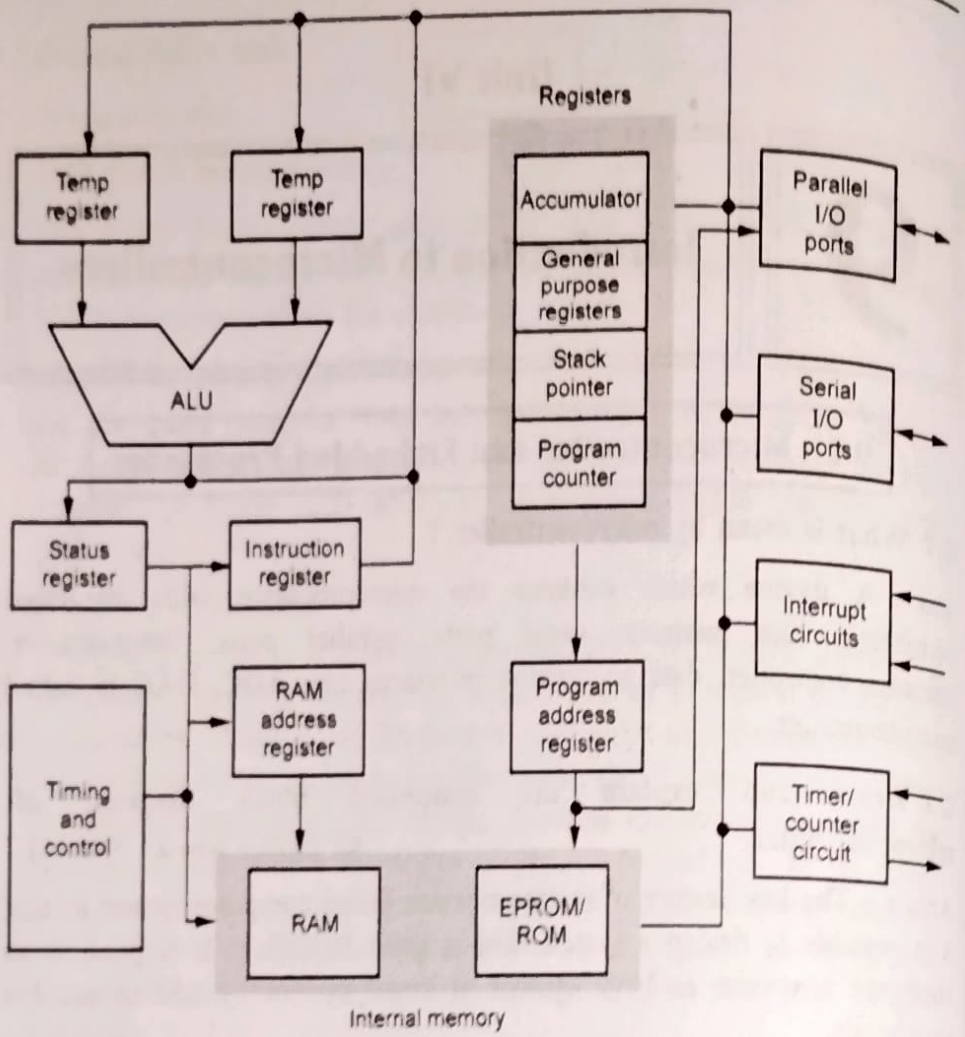


Fig. Q.2.1 Block diagram of microcontroller

- Less hardware, reduces PCB size and increases reliability of the system.

Q.3 Differentiate between microprocessor and microcontroller.

[SPPU : Dec.-22, Marks 5]

Ans. :

Sr. No.	Microprocessor	Microcontroller
1.	Microprocessor contains ALU, control unit (clock and timing circuit), different register and interrupt circuit.	Microcontroller contains microprocessor, memory (ROM and RAM), I/O interfacing circuit and peripheral devices such as A/D converter, serial I/O, timer etc.

2.	It has many instructions to move data between memory and CPU.	It has one or two instructions to move data between memory and CPU.
3.	It has one or two bit handling instructions.	It has many bit handling instructions.
4.	Access times for memory and I/O devices are more.	Less access times for built-in memory and I/O devices.
5.	Microprocessor based system requires more hardware.	Microcontroller based system requires less hardware reducing PCB size and increasing the reliability.
6.	Microprocessor based system is more flexible in design point of view.	Less flexible in design point of view.
7.	It has single memory map for data and code.	It has separate memory map for data and code.
8.	Less number of pins are multifunctioned.	More number pins are multifunctioned.

Q.4 What are the different addressing modes supported by 8051 ?

☞ [SPPU : May-14, 18, Marks 6]

OR List down the various addressing modes used in instruction set of 8051. Give one example of each.

☞ [SPPU : May-15, Dec.-15, Marks 6]

OR Explain with examples the different addressing modes used in 8051.

Ans. : Immediate Addressing : In this addressing mode source operand is a constant and it is incorporated in the instruction. Sign "#" indicates it is a immediate addressing mode.

Example : Add the constant 52 decimal in accumulator.

```
MOV A, #52
```

Register Addressing

- In the register addressing mode, the data to be operated is within the registers.

Example :

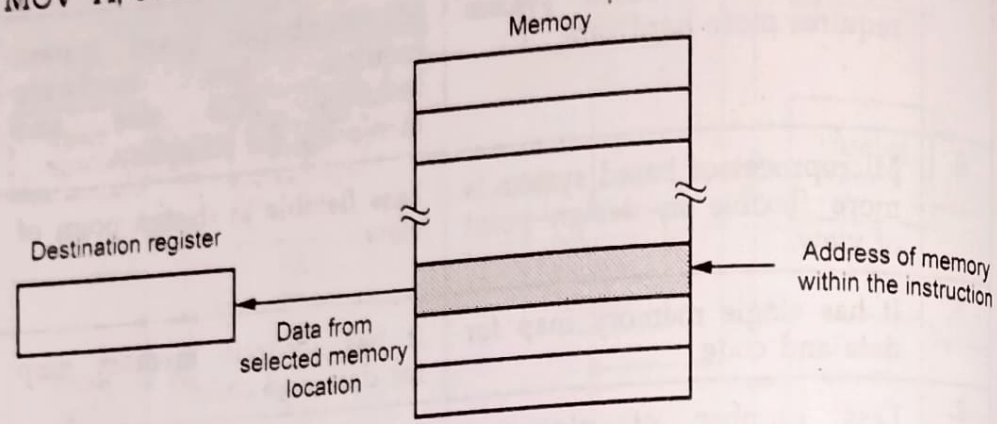
MOV A, R3 ; Copies content of R3 to A

Direct Addressing

- Direct addressing can access any on-chip variable or hardware register, i.e. on-chip RAM and special function register.

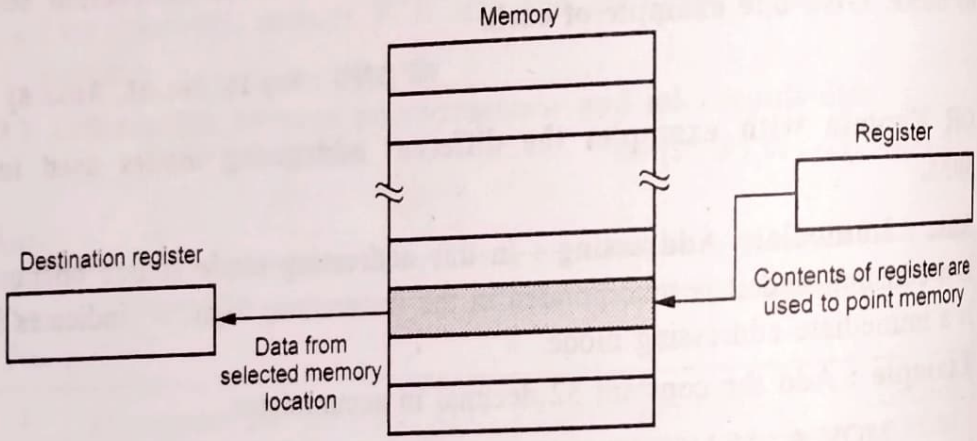
Example :

MOV A, 50H ; Load byte from address 50H into A



Indirect or Register Indirect Addressing

- In this addressing mode R0 and R1 of each register bank can be used as an index or pointer register. R0 and R1 point to the contents in the RAM. The instruction with indirect addressing uses the '@' sign.



Example :

MOV A, @R0 ; Load the contents pointed by R0 in A

Relative Addressing

- The JMP (JUMP) and CALL instructions in 8051 use the relative addressing.
- We know that, a JMP and CALL instructions replace the contents of the program counter with a new program address.
- In relative addressing, a new program address is not directed specified. It is specified as a difference, in bytes, of the new address from the address in the program counter called **offset**.
- The relative offset is a signed 8-bit value gives an address range of + 127 to – 128 locations.

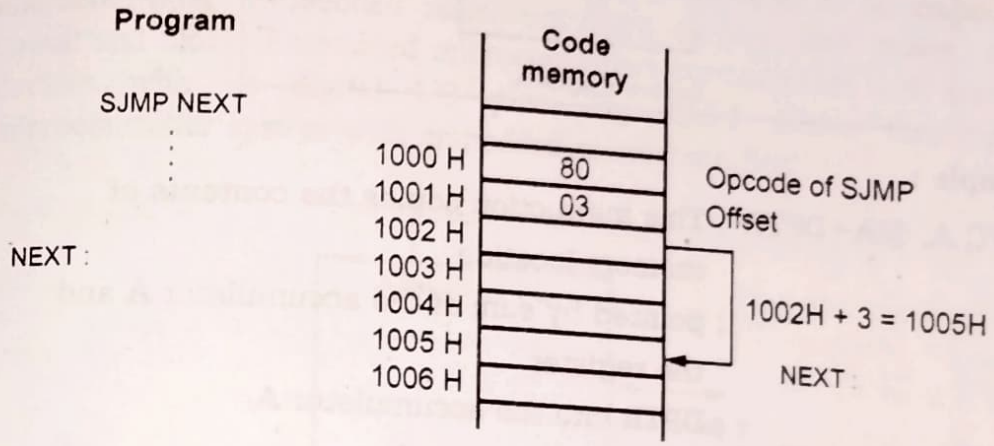


Fig. Q.4.1 Relative addressing mode

Absolute Addressing

- This addressing mode is similar to that of relative addressing except the offset length. Here, the offset is 11-bit instead of 8-bit in case of relative addressing.

Example :

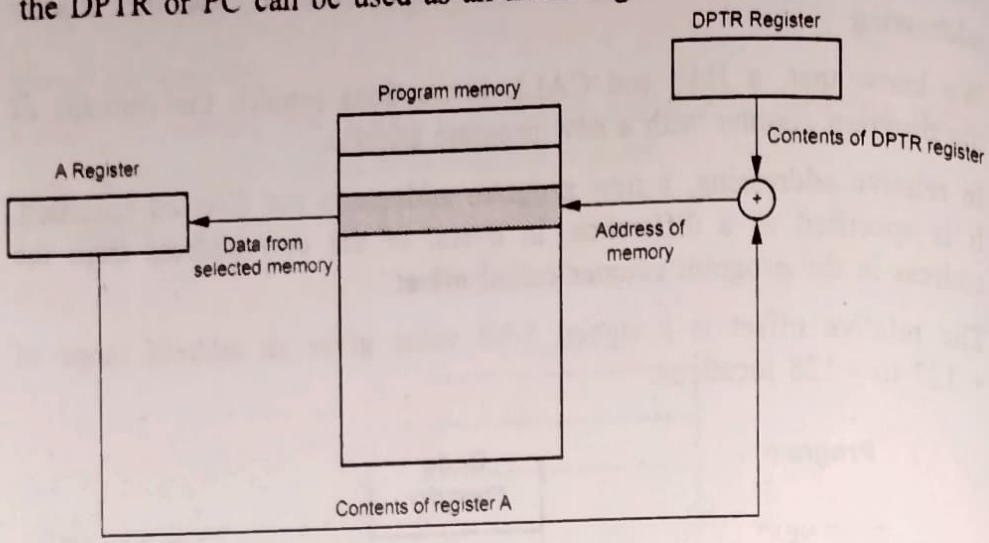
```
AJMP NEXT      ; Executes the instruction at label NEXT after
               ; execution of the current instruction
```

Long Addressing

- In the long addressing mode, the new address for JMP or CALL instruction (LJMP or LCALL) is specified as the entire new 16-bit address.

Indexed Addressing

- Only program memory can be accessed in the index addressing. Either the DPTR or PC can be used as an index register.



Example :

```
MOVC A, @A+DPTR ; This instruction copies the contents of
                  memory location
                ; pointed by sum of the accumulator A and
                  the register
                ; DPTR into the accumulator A.
```

Inherent Addressing

- Inherent in the instruction, these refer to a specific register such as accumulator or DPTR.

Example :

```
SWAP A ; Swap nibbles within the Accumulator
```

Bit Inherent Addressing

- In this addressing, the address of the flag which contains the operand, is implied in the opcode of the instruction.

Example :

```
CLR C   ; Clears the carry flag to 0
```

Bit Direct Addressing

- The RAM address space from 20H to 2FH and some Special Function Registers (SFRs) are bit addressable.

Example : SETB 06H; Sets the bit 6 of 20H RAM space

Q.5 Write a note on types of microcontroller.

Ans. : Like microprocessors, the microcontrollers have family of microcontrollers. Different microcontrollers require different support chips and resources to develope particular microcontroller system. To choose and appropriate device to meet system requirements we must understand differences, different options and features of various microcontrollers.

Embedded Microcontrollers

- When a complete hardware required to run a particular application is provided on the microcontroller chip, it is referred to as an embedded microcontroller. Embedded microcontrollers only require power, reset circuit and clock. Embedded microcontrollers communicate with external devices with its digital I/O pins. Fig. Q.5.1 shows the typical microcontroller system with embedded microcontroller.

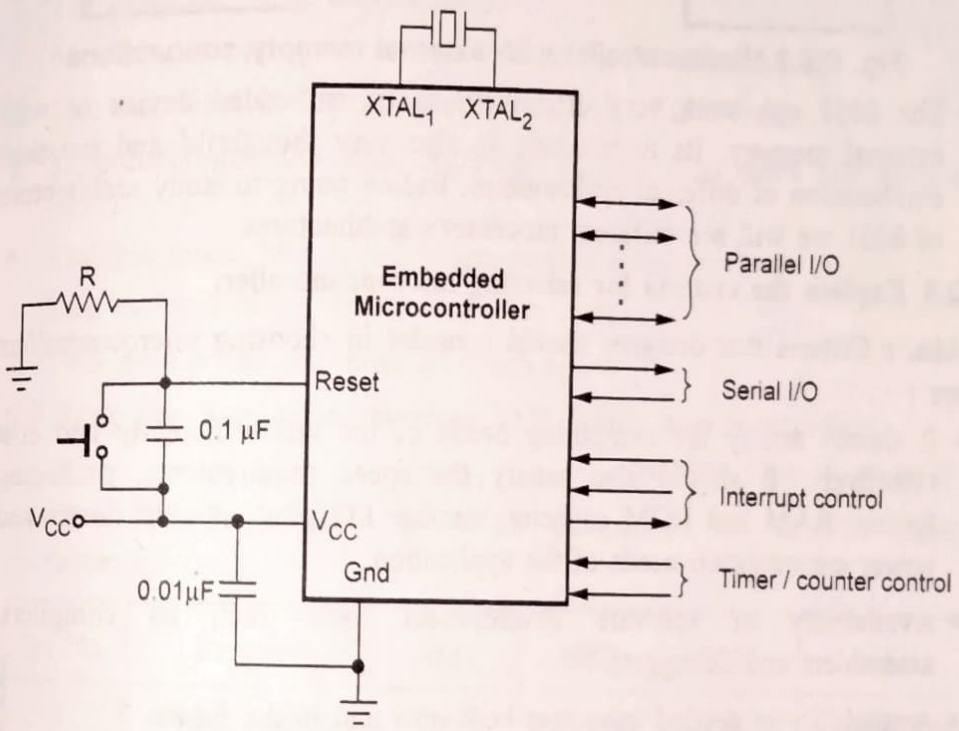


Fig. Q.5.1 Typical microcontroller system with embedded microcontroller

External Memory Microcontrollers

- Sometimes, for large systems, the built-in program memory and data memory are insufficient. To overcome this problem some microcontrollers allow the connection of external memory. For the connection of external memory some parallel port pins are used as

address and data lines. Thus connecting external memory to the microcontroller reduces its parallel input/output capabilities. Fig. Q.5.2 shows the microcontroller with external memory connections. As shown in the Fig. Q.5.2, many times address and data lines are multiplexed and separated by external latch and ALE signal from the microcontroller.

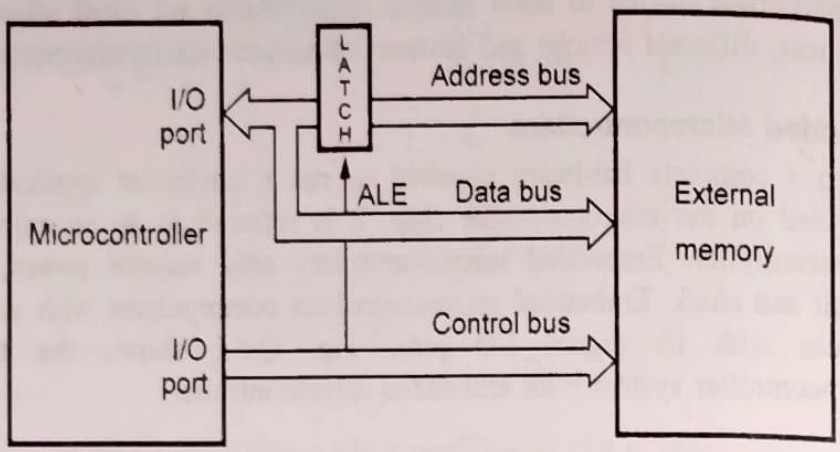


Fig. Q.5.2 Microcontroller with external memory connections

- The 8051 can work very effectively as an embedded device or with external memory. Its architecture is also very thoughtful and practical combination of different philosophies. Before going to study architecture of 8051 we will see different processor's architectures.

Q.6 Explain the criteria for selecting microncontroller.

Ans. : Criteria that designer should consider in choosing microcontrollers are :

- It should satisfy the computing needs of the task efficiently and cost effectively. It should also satisfy the speed requirements, packaging format, RAM and ROM capacity, number I/O pins, on-chip timers and power consumption needs of the application.
- Availability of software development tools such as compilers, assemblers and debuggers.
- Availability in needed quantities both now and in the future.
- Its ability to upgrade to higher-performance or lower power consumption versions.

Q.7 State the applications of microcontroller.

☞ [SPPU : Dec.-22, Marks 5]

Ans. : Microcontrollers are preferred in embedded products. Some applications of microcontroller are :

- Home appliances : Washing machine, refrigerators, microwave ovens etc.
- Calculators, keyboards, printers, modems, mobile phones etc.
- Industrial controllers, data acquisition systems communication systems etc.
- Automobile engines, flight control systems, traffic light control systems etc. Millitary applications.

9.2 : Features of 8051 Microcontroller

Q.8 List the salient features of 8051 microcontroller.

Ans. : • The features of the 8051 family are as follows :

1. 4096 bytes on - chip program memory.
2. 128 bytes on - chip data memory.
3. 64 kilobytes each program and external RAM addressability.
4. One microsecond instruction cycle with 12 MHz crystal.
5. 32 bidirectional I/O lines organized as four 8-bit ports (16 lines on 8031).
6. Multiple mode, high-speed programmable serial port.
7. Two multiple mode, 16-bit timers/counters.
8. Two-level prioritized interrupt structure.

Q.9 Give the comparison between 8051 family microcontrollers.

☞ [SPPU : May-18, Marks 6]

Ans. : • The Table Q.9.1 gives the comparison of MCS-51 family microcontrollers.

Sr. No.	Feature	8031	8051	8052	8751
1.	Program memory (in bytes)	None	4 K ROM	8 K ROM	4 K EPROM
2.	Data memory (in Bytes)	128 RAM	128 RAM	256 RAM	128 RAM
3.	Timers / Counters (16-bit)	2	2	3	2

4.	I/O pins	32	32	32	32
5.	Serial Port	1	1	1	1
6.	Interrupt Sources (Reset not included)	5	5	6	5

Table Q.9.1 Comparison of MCS-51 family microcontrollers

9.3 : Block Diagram of 8051 Microcontroller

Q.10 Draw the architecture of 8051 microcontroller.

[SPPU : Dec.-18 Marks 3]

OR With the help of neat diagram explain the internal block diagram of 8051.

Ans. : • Fig. Q.10.1 shows the internal block diagram of 8051. It consists of a CPU, two kinds of memory sections (data memory - RAM and program memory - ROM), input/output ports and control logic needed for a timer/counter, serial port and interrupt functions. These elements communicate through an eight bit data bus which runs throughout the chip referred as internal data bus. This bus is buffered to the outside world through an I/O port when memory or I/O expansion is desired.

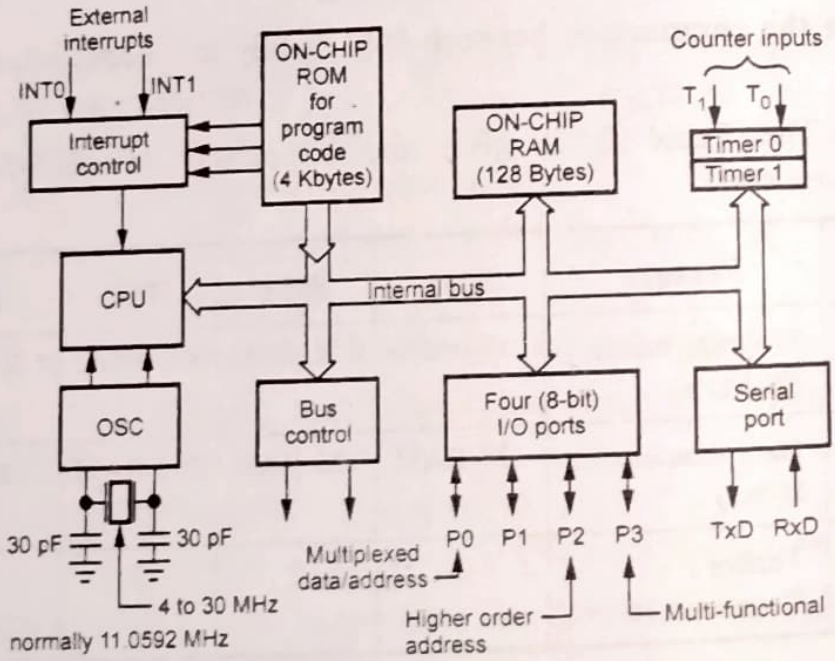


Fig. Q.10.1 Block diagram of 8051 microcontroller

- **Central Processor Unit (CPU) :** It monitors and controls all operations that are performed by microcontroller. The CPU of 8051 consists of eight-bit arithmetic and logic unit with associated registers like A, B, PSW, SP, the sixteen bit program counter and "Data pointer" (DPTR) registers. Along with these registers it has a set of special function registers and control unit.

- **ROM :** A code of 4K memory is incorporated as on-chip ROM in 8051. The 8051 ROM is a non-volatile memory meaning that its contents cannot be altered.

- **RAM :** The 8051 microcontroller is composed of 128 bytes of internal RAM. This is a volatile memory since its contents will be lost if power is switched off. These 128 bytes of internal RAM are divided into 32 working registers which in turn constitute
  4 register banks (Bank 0-Bank 3) with each bank consisting of 8 registers (R0 - R7). There are 128 addressable bits in the internal RAM.

- **I/O Ports :** The 8051 microcontroller has four 8-bit input/output ports : P0, P1, P2 and P3. All Ports can used as general purpose ports. In the presence of external memory, Port 0 functions as a multiplexed address and data bus and Port 2 functions as a higher order byte address bus. All port pins of port 3 are multifunctional. Therefore, each pin of port 3 can be programmed to use as I/O or as one of the alternate function.

- **Interrupt Control :** It supports both an internal (software) and external (hardware) interrupts. In 8051, 5 sources of interrupts are provided.

- **Timers :** 8051 supports two multiple mode, 16-bit timers/counters. In timer mode, they can be used to generate a delay of particular and in counter mode they can be used to count external pulses.

- **Serial Port :** Provides a method of establishing serial communication by transmitting and receiving data bits. It uses TxD and RxD pins to transmit and receive bits, respectively.

- **Oscillator :** It is used for providing the clock to 8051 and decide baud rate for serial communication.

9.4 : Register Organization of 8051 Microcontroller

Q.11 Draw the register organization of 8051.

Ans. : • Fig. Q.11.1 shows the register organization of 8051. It shows all CPU registers along with the registers used for timers, interrupts and serial communication.

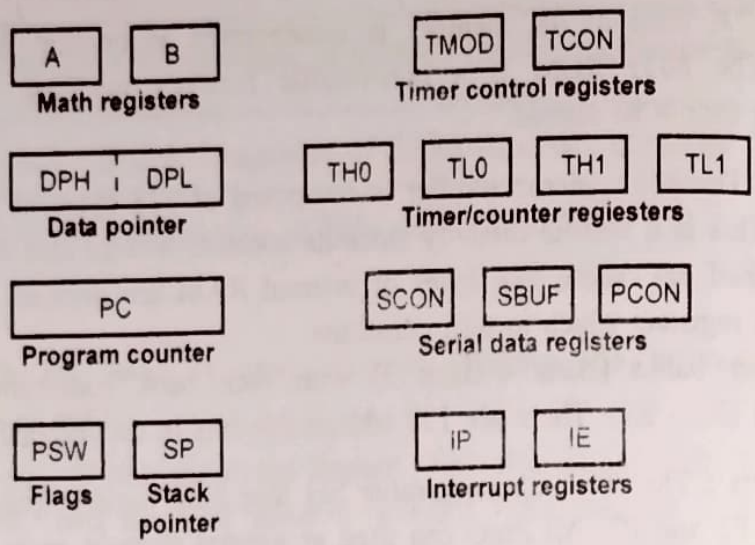


Fig. Q.11.1 Register organization of 8051

Q.12 Explain the function of register A of 8051.

Ans. : • It is an 8-bit register called **accumulator**. It holds a source operand and receives the result of the arithmetic instructions (addition, subtraction, multiplication and division).

- Several functions apply exclusively to the accumulator : Rotate, parity computation, testing for zero and so on.

Q.13 Explain the function of register B of 8051.

Ans. : It is used for the hardware multiply/divide operation.

Q.14 What is DPTR ? State its function.

Ans. : • The data pointer (DPTR) consists of a high byte (DPH) and a low byte (DPL). Its function is to hold a 16-bit address. It may be manipulated as a 16-bit data register or as two independent 8-bit registers. It serves as a base register in indirect jumps, lookup table instructions and external data transfer.

Q.15 Explain the function of program counter.

Ans. : • The 8051 has a 16-bit program counter. It is used to hold the address of memory location from which the next instruction is to be fetched.

Q.16 Draw PSW and explain various flags in 8051 microcontroller.

☞ [SPPU : May-15, 17, Marks 7]

OR Draw the program status word of 8051 ?

☞ [SPPU : Dec.-11, May-14, Marks 3]

Ans. : • The Fig. Q.16.1 shows the bit pattern of Program Status Word (PSW) of 8051. PSW is also known as **flag register**.

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
CY	AC	FO	RS1	RS0	OV	-	P

Fig. Q.16.1

- **CY-Carry Flag :** This flag is set if there is an overflow out of bit 7. The carry flag also serves as a borrow flag for subtraction. In both the examples shown below, the carry flag is set.
- **AC-Auxiliary Carry Flag :** This flag is set if there is an overflow out of bit 3 i.e., carry from lower nibble to higher nibble (D_3 bit to D_4 bit).
- **FO** - Available for user for general purpose.
- **RS1 - RS0 (Register Bank Select) :** They select the working register bank as follows :

RS1	RS0	Bank Selection	
0	0	00H - 07H	Bank 0
0	1	08H - 0FH	Bank 1
1	0	10H - 17H	Bank 2
1	1	18H - 1FH	Bank 3

- **OV-Over Flow Flag :** This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit.

- **P-Parity Flag :** Parity is defined by the number of ones present in the accumulator. P = 0, if number of ones are even and P = 1, if number of ones are odd.

Q.17 Show how to select bank 2 with a single instruction after power-up. ☞[SPPU : May-17, Marks 2]

Ans. : Instruction to select Bank 2 is MOV PSW, # 10.

Q.18 Write a note on special function registers of 8051.

Ans. : • The group of registers, implemented to perform special functions and are located immediately above the 128 bytes of RAM are called **special function registers.** All access to the four I/O ports, the CPU registers, interrupt control registers, the timer/counter, UART and power control are performed through registers between 80H and FFH.

- Special Function Registers (SFRs) are a sort of used for running and monitoring the operation of the microcontroller. Each of these registers as well as each bit they include, has its name, address in the scope of RAM and precisely defined purpose such as timer control, interrupt control, serial communication control etc.
- Examples of special function register are : P0, P1, P2, P3, PSW, A, B, IP, IE, SCON, PCON, TCON, TMOD etc.

Q.19 What is stack ? How it operates in 8051 ?

☞[SPPU : Dec.-17, Marks 2]

Ans. : • The stack refers to an area of internal RAM that is used to store and retrieve data quickly. The stack pointer register is used by the 8051 to hold an internal RAM address that is called **top of stack.** The stack pointer register is 8-bit wide. It is increased **before** data is stored during PUSH and CALL instructions and decremented **after** data is restored during POP and RET instructions.

9.5 : Pin Diagram of 8051

Q.20 Draw the pin-diagram of 8051 and explain the function of each pin.

OR Explain the function of pins $\overline{PSEN}$, $\overline{EA}$ and ALE.

☞[SPPU : Dec.-17, Marks 6]

OR Explain the functions of following pins of 8051
i) $\overline{EA}$ ii) $\overline{PSEN}$ iii) RST [SPPU : May-14, Marks 3]

Ans. : Fig. Q.20.1 shows the pin diagram of 8051.

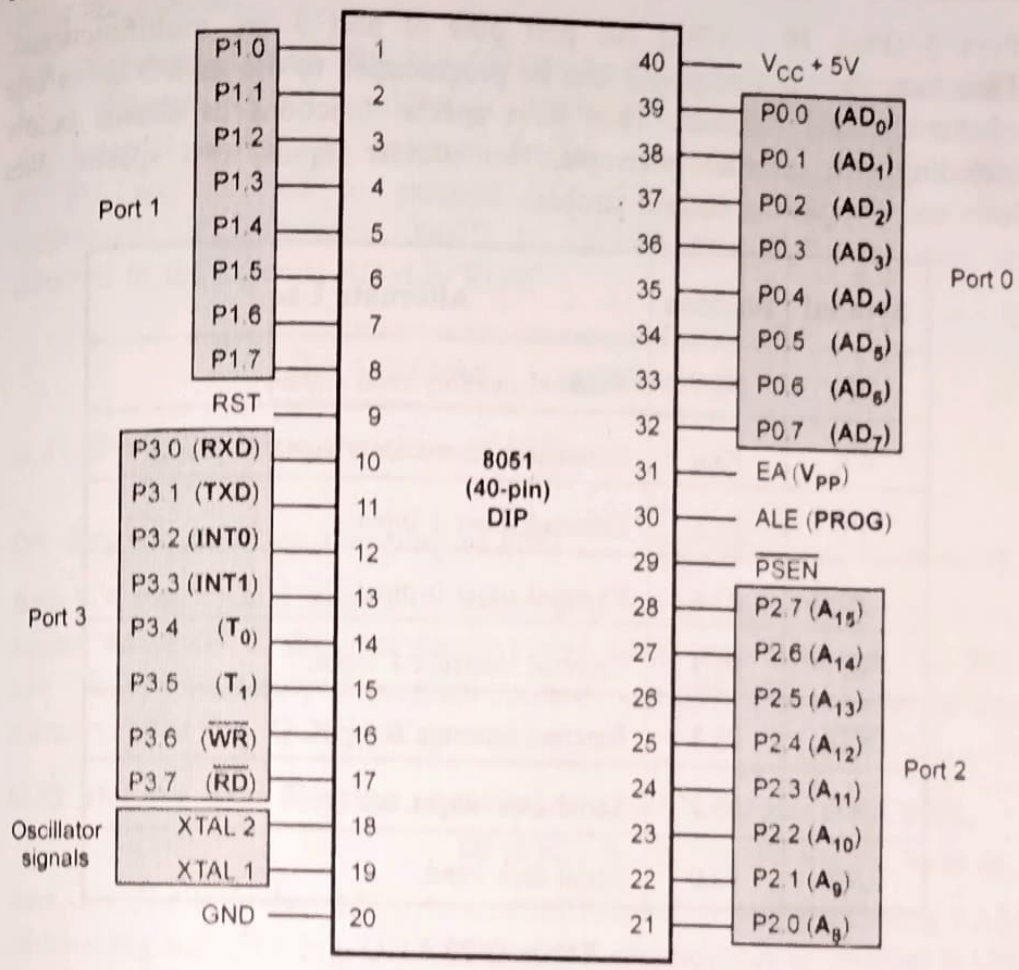


Fig. Q.20.1 Pin-out of 8051

- The 8051 has 32 I/O pins configured as four eight-bit parallel ports (P0, P1, P2 and P3). All four ports are bidirectional i.e. each pin will be configured as input or output (or both). All port-pins are multiplexed except the pins of port 1. Each port consists of a latch, an output driver and an input buffer.

Port 0 (Pins 32 - 39) : Port 0 pins can be used as I/O pins. The output drives and input buffers of port 0 are used to access external memory. Port 0 outputs the low order byte of the external memory address, time multiplexed with the data being written or read. Thus, port 0 can be used as a multiplexed address/data bus.

Port 1 (Pins 1 - 8) : Port 1 pins can be used only as I/O pins.

Port 2 (Pins 21 - 28) : The output drives of port 2 are used to access external memory. Port 2 outputs the high order byte of the external memory address when the address is 16 bits wide. Otherwise, port 2 is used as an I/O port.

Port 3 (Pins 10 - 17) : All port pins of port 3 are multifunctional. Therefore, each pin of port 3 can be programmed to use as I/O or as one of the alternate function. They have special functions as shown below including two external interrupts, two counter inputs, two special data lines and two timing control strobes.

Symbol	Position	Alternate Use
$\overline{RD}$	P3.7	External memory read signal.
$\overline{WR}$	P3.6	External memory write signal.
T1	P3.5	External timer 1 input.
T0	P3.4	External timer 0 input.
$\overline{INT1}$	P3.3	External interrupt 1 input.
$\overline{INT0}$	P3.2	External interrupt 0 input.
TXD	P3.1	Serial data output.
RXD	P3.0	Serial data input.

Table Q.20.1

Power-supply Pins V_{CC} (Pin 40) and V_{SS} (Pin 20) : 8051 operates on d.c. power supply of +5 V with respect to ground. The +5 V is to be connected to pin V_{CC} and ground to pin V_{SS}.

Oscillator Pins XTAL2 (Pin 18) and XTAL1 (Pin 19) : For generating an internal clock signal, the external oscillator is connected at these two pins.

ALE (Address Latch Enable, Pin 30) : AD_0 to AD_7 lines are multiplexed. To demultiplex these lines and for obtaining lower half of an address, an external latch and ALE signal of 8051 is used.

RST (Reset, Pin 9) : This pin is used to reset 8051.

$\overline{PSEN}$ (Program Store Enable, Pin 29) : It is the active low output control signal used to activate the enable signal of the external ROM/EPROM. It is activated every six oscillator periods while reading the external memory. Thus, this signal acts as the read strobe to external program memory.

$\overline{EA}$ (External Access, Pin 31) : When the $\overline{EA}$ pin is high (connected to V_{CC}), program fetches to addresses 0000H through 0FFFH are directed to the internal ROM and program fetches to addresses 1000H through FFFFH are directed to external ROM/EPROM. When $\overline{EA}$ is low (grounded), all addresses (0000H to FFFFH) fetched by program are directed to the external ROM/EPROM.

9.6 : Memory Organization

Q.21 Draw the basic structure of 8051.

OR Explain memory mapping of 8051. ☞ [SPPU : Dec.-18, Marks 3]

Ans. : • Fig. Q.21.1 shows the basic memory structure for 8051. It can access up to 64 K program memory and 64 K data memory. The 8051 has 4 K bytes of internal program memory and 256 bytes of internal data memory. (See Fig. Q.21.1 on next page)

Q.22 Draw a neat diagram of the internal RAM structure of 8051.

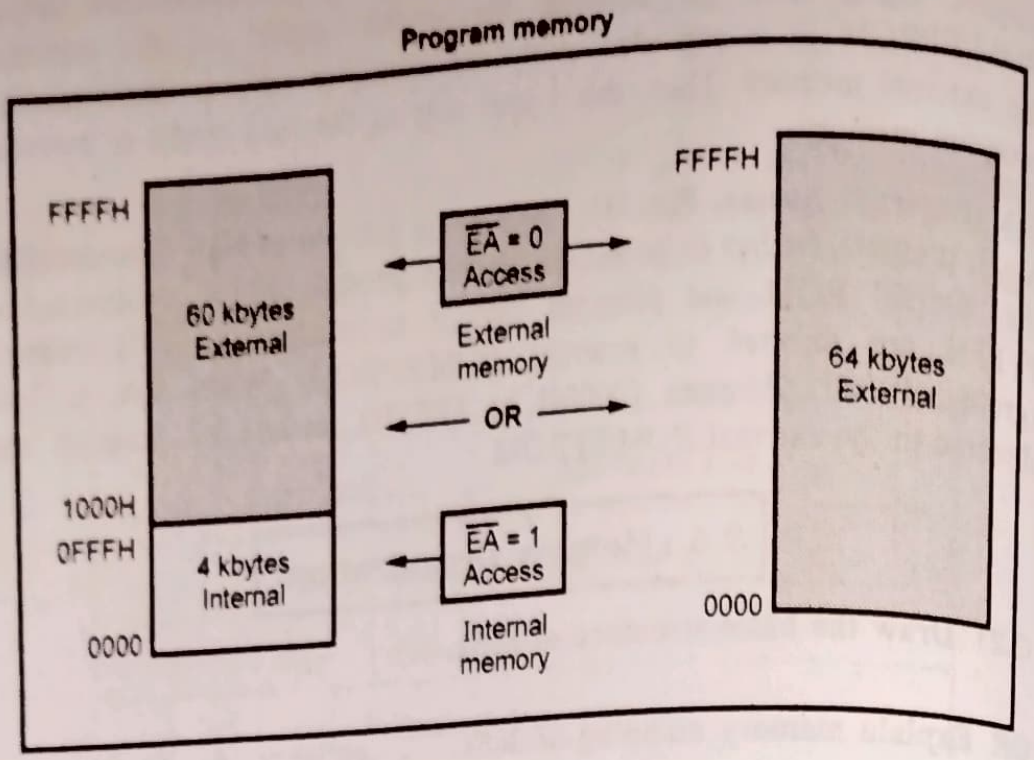
☞ [SPPU : Dec.-14, 15, 17, May-16, Marks 4]

Ans. : • The 8051 has 128-byte internal RAM. It is accessed using RAM address register. The Fig. Q.22.1 shows the organization of internal RAM. It is organized into three distinct areas : (See Fig. Q.22.1 on page 9 - 19)

- Register bank
- Bit addressable
- General purpose.

8051 Register Banks (Working Registers)

- The first 32-bytes from address 00H to 1FH of internal RAM constitute 32 working registers. They are organized into four banks of eight registers each. The four register banks are numbered 0 to 3 and are consists of eight registers named R_0 to R_7.
- Each register can be addressed by name or by its RAM address.



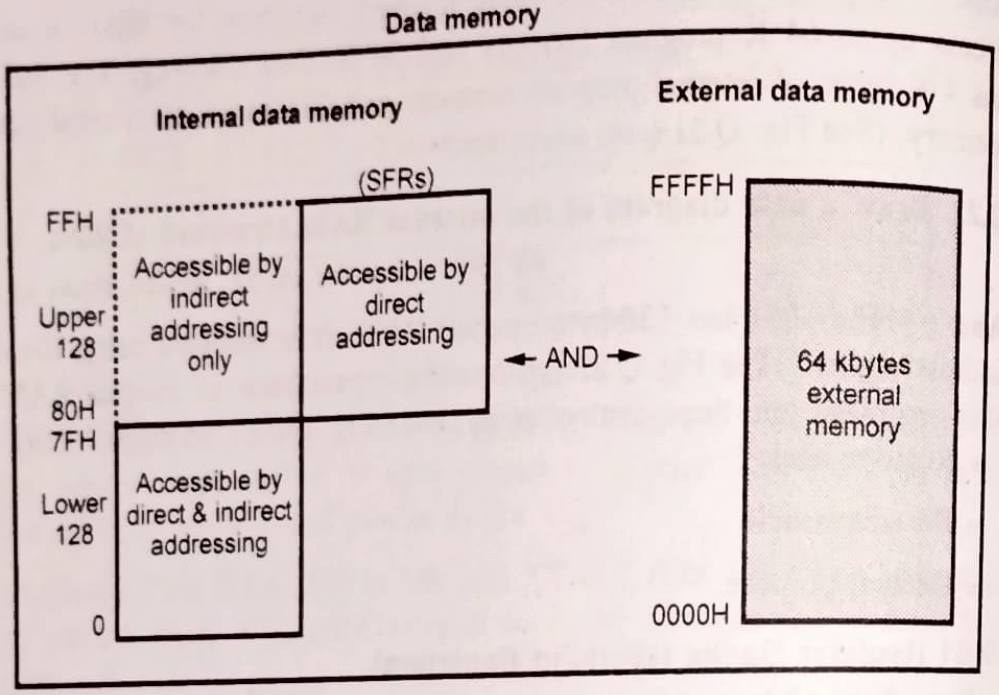


Fig. Q.21.1

- Only one register bank is in use at a time. Bits RS0 and RS1 in the PSW determine which bank of registers is currently in use.
- On reset, the bank 0 is selected.

Bit / Byte Addressable

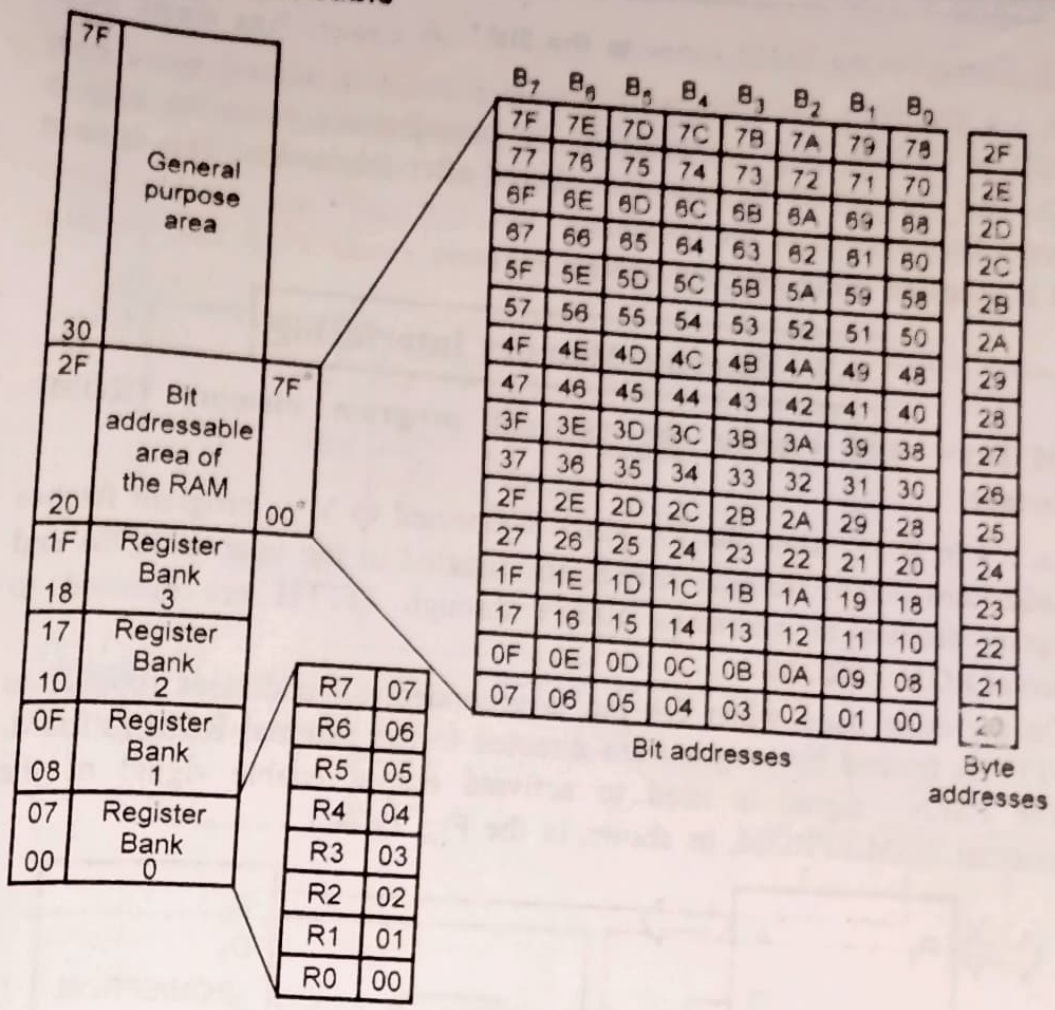


Fig. Q.22.1 Organization of internal RAM of 8051

- The 8051 provides 16 bytes of a bit-addressable area. It occupies RAM byte addresses from 20H to 2FH, forming a total of 128 (16 × 8) addressable bits.
- An addressable bit may be specified by its bit address of 00H to 7FH, or 8 bits may form any byte address from 20H to 2FH.
- For example, bit address 4EH refers bit 6 of the byte address 29H.

General Purpose RAM

The RAM area above bit addressable area from 30H to 7FH is called general purpose RAM. It is addressable as byte.

Q.23 Comment on ROM space in the 8051.

Ans. : • The 8051 has 4 kbyte of internal ROM with address space from 0000H to 0FFFH. It is programmed by manufacturer when the chip is built. This part cannot be erased or altered after fabrication. This is used to store final version of the program.

• It is accessed using program address register.

9.7 : External Memory Interfacing

Q.24 Draw and explain the external program memory (ROM) interface.

Ans. : • In 8051, when the $\overline{EA}$ pin is connected to V_{CC}, program fetches to addresses 0000H through 0FFFH are directed to the internal ROM and program fetches to addresses 1000H through FFFFH are directed to external ROM/EPROM.

• On the other hand when $\overline{EA}$ pin is grounded, all addresses (0000H to FFFFH) fetched by program are directed to the external ROM/EPROM. The $\overline{PSEN}$ signal is used to activate output enable signal of the external ROM/EPROM, as shown in the Fig. Q.24.1.

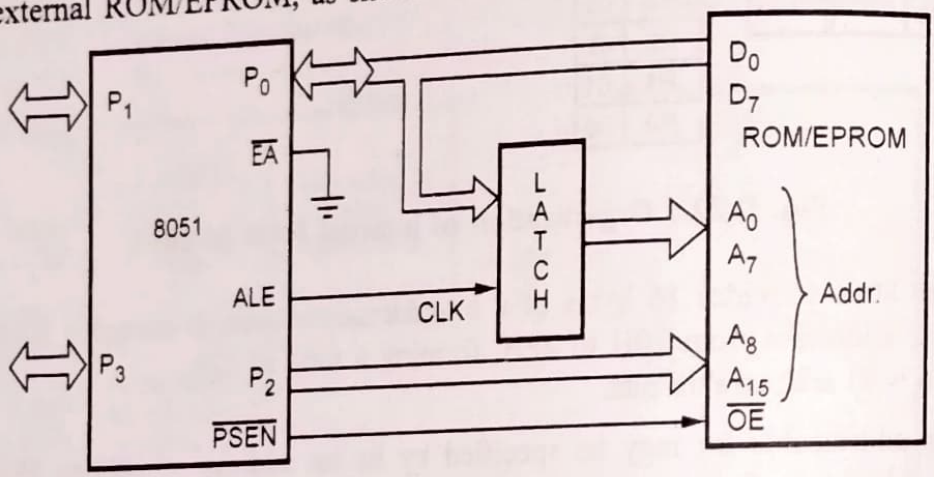


Fig. Q.24.1 Accessing external program memory

• As shown in the Fig. Q.24.1, the port 0 is used as a multiplexed address/bus. It gives lower order 8-bit address in the initial T-cycle and later it is used as a data bus. The 8-bit address is latched using external latch and ALE signal generated by 8051.

• The port 2 provides the higher order 8-bit address.

Q.25 Draw and explain the external data memory (RAM) interface.

Ans. : • Fig. Q.25.1 shows the circuit diagram for connecting external data memory. The multiplexed address/data bus provided by port 0 is demultiplexed by external latch and ALE signal. Port 2 gives the higher order address bus. The $\overline{RD}$ and $\overline{WR}$ signals from 8051 selects the memory read and memory write operation, respectively.

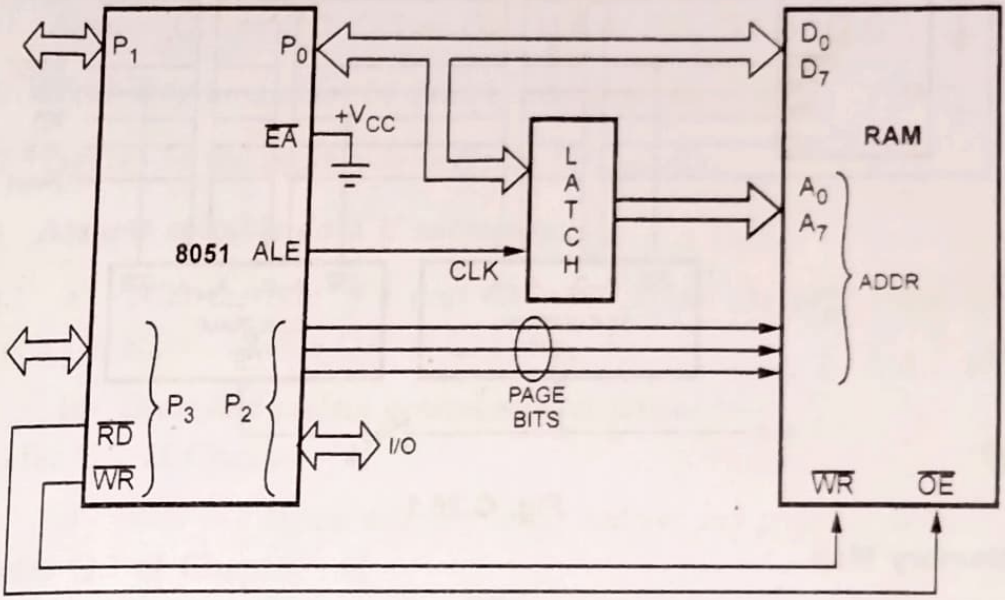


Fig. Q.25.1 Accessing external data memory

Q.26 Discuss interfacing of external 32 K EPROM and 32 K RAM with the microcontroller. Draw diagram and explain.

Ans. : To address 32 K EPROM and RAM needs 15 address lines (A_0 - A_{14}). The remaining address line A_{15} is used for selection of EPROM or RAM. When A_{15} is low EPROM is selected and when A_{15} is high RAM is selected.

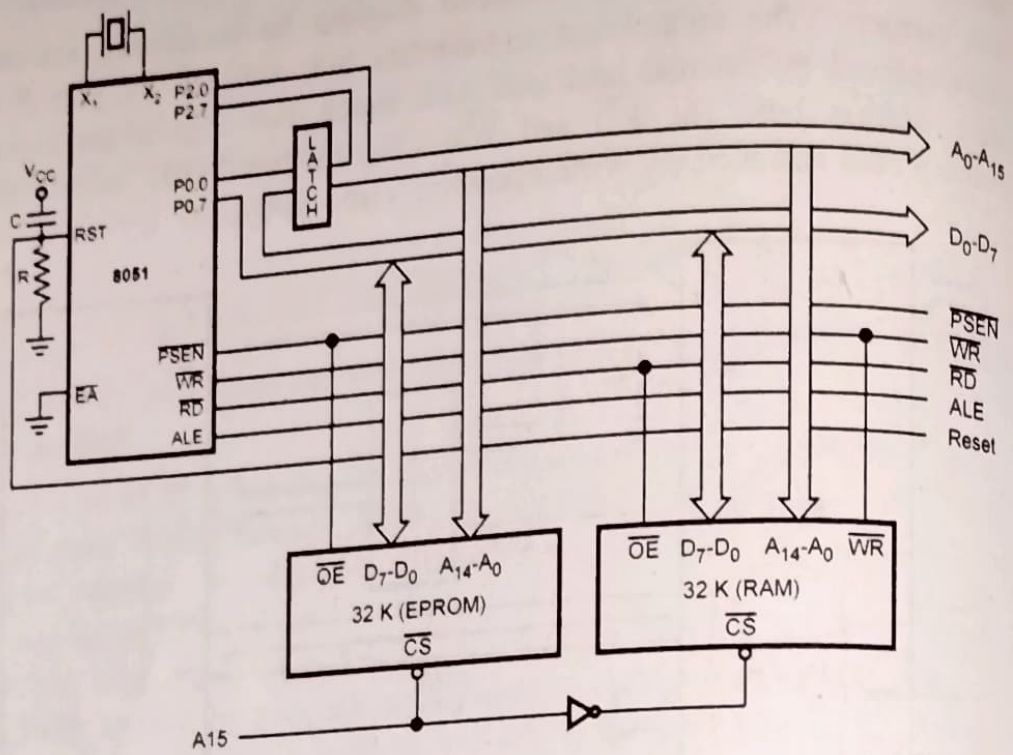


Fig. Q.26.1

Memory Map

Memory	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Address
RAM	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000H (Start)
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFFH (End)
EPROM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H (Start)
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFFH (End)

END...

JUNE - 2022 [5869] - 279 | Solved Paper

Course 2019

Time : $2\frac{1}{2}$ Hours] **[Maximum Marks : 70**

Instruction to the candidates :

1) *Answer Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.*

2) *Neat diagrams must be drawn whenever necessary.*

3) *Figures to the right side indicate full marks.*

4) *Assume suitable data if necessary.*

Q.1 **a)** *With the help of a neat diagram, explain the page translation process in 80386.* **[Refer Q.6 of Chapter - 5]** [6]

b) *Draw and explain general selector format.*
[Refer Q.2 of Chapter - 4] [6]

c) *What is a logical address, linear address and physical address ?*
[Refer Q.1 of Chapter - 4] [6]

OR

Q.2 **a)** *Explain the use of following instructions in detail :*
i) SGDT ii) LIDT iii) SLDT
[Refer pages 2 - 36, 2 - 23 and 2 - 37] [6]

b) *Explain the segment translation process with a neat diagram of 80386.* **[Refer Q.2 of Chapter - 4]** [6]

c) *Enlist various types of system and non-system descriptors in the 80386. Explain their use in brief.*
[Refer Q.5 of Chapters - 4, 5 and 6] [6]

Q.3 **a)** *Write a short note on CPL, DPL and RPL.*
[Refer Q.4 of Chapter - 5] [6]

b) *Explore the role of various fields in page level protection.*
[Refer Q.6 of Chapter - 5] [6]

c) *List and explain various privilege instructions.*
[Refer Q.15 of Chapter - 5] [5]

OR

Q.4 a) *What is call gate ? Explain how it is used in calling functions with higher privilege levels.* **[Refer Q.5 of Chapter - 5]** [6]

b) *Define the functions of type checking and limit checking in protection.* **[Refer Q.3 of Chapter - 5]** [6]

c) *Explain different levels of protection? State the rules of protection check.* **[Refer Q.4 of Chapter - 5]** [5]

Q.5 a) *Explore the role of task register in multitasking and the instructions used to modify and read task register.*
[Refer Q.5 of Chapter - 6] [6]

b) *Draw and Explain the task state segment of 80386.*
[Refer Q.3 of Chapter - 6] [6]

c) *Difference between real mode and virtual 8086 mode.*
[Refer Q.8 of Chapter - 7] [6]

OR

Q.6 a) *Explain the TSS descriptor of 80386 with a neat diagram.*
[Refer Q.4 of Chapter - 6] [6]

b) *Explore memory management in the virtual 8086 mode.*
[Refer Q.3 of Chapter - 7] [6]

c) *List and explain various features of virtual 8086 mode.*
[Refer Q.1 of Chapter - 7] [6]

Q.7 a) *Explain the process of enabling and disabling interrupts in 80386.* **[Refer Q.4 of Chapter - 8]** [6]

b) *Differentiate and explain the interrupt gate and trap gate descriptor.* **[Refer Q.10 of Chapter - 8]** [6]

c) *Differentiate between microprocessor and microcontroller.*
[Refer Q.3 of Chapter - 9] [5]

OR

Q.8 a) *With the help of the necessary diagram, explain the structure of IDT in 80386.* **[Refer Q.6 of Chapter - 8]** [6]

b) *Explain different types of exceptions in 80386 with suitable examples.* **[Refer Q.2 of Chapter - 8]** [6]

c) *Draw and explain the architecture of a typical microcontroller.* **[Refer Q.2 of Chapter - 9]** [5]

DECEMBER - 2022 [5925] - 263 | Solved Paper

Course 2019

Time : $2\frac{1}{2}$ Hours] **[Maximum Marks : 70**

Instructions to the candidates :

1) *Solve Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.*

2) *Neat diagrams should be drawn wherever necessary.*

3) *Use of non-programmable calculator is allowed.*

4) *Assume suitable data if necessary.*

Q.1 a) *Explain the segment translation process with a neat diagram of 80386.* **[Refer Q.2 of Chapter - 4]** [6]

b) *Differentiate and explain GDTR, LDTR and IDTR.* **[Refer Q.10 of Chapter - 4]** [6]

c) *Demonstrate general selector format in brief.* **[Refer Q.2 of Chapter - 4]** [6]

OR

Q.2 a) *Demonstrate general descriptor format available in various descriptor tables.* **[Refer Q.4 of Chapter - 4]** [6]

b) *With the necessary diagram, explain the page translation process in 80386.* **[Refer Q.12 of Chapter - 4]** [6]

c) *Explain the use of following instructions in detail : i) LGDT ii) SIDT iii) LLDT* **[Refer pages 2 - 23, 2 - 36, and 2 - 24]** [6]

Q.3 a) *What is call gate ? Explain how it is used in calling functions with higher privilege levels.* **[Refer Q.5 of Chapter - 5]** [6]

b) *Explore five aspects of protection applied in segmentation.*
[Refer Q.2 of Chapter - 5] [6]

c) *Explore the need for a protection mechanism in 80386.*
[Refer Q.1 of Chapter - 5] [5]

OR

Q.4 a) *Explain the following terminologies. i) CPL ii) DPL iii) RPL*
[Refer Q.4 of Chapter - 5] [6]

b) *Explain different levels of protection. Describe the rules of protection check.* **[Refer Q.4 of Chapter - 5]** [6]

c) *Elaborate on the concept of combining segment protection and page level protection in 80386.* **[Refer Q.7 of Chapter - 5]** [5]

Q.5 a) *Explore memory management in the Virtual 8086 Mode.*
[Refer Q.3 of Chapter - 7] [6]

b) *Explain the TSS descriptor of 80386 with a neat diagram.*
[Refer Q.4 of Chapter - 6] [6]

c) *Explore the role of task register in multitasking and the instructions used to modify and read task register.*
[Refer Q.5 of Chapter - 6] [6]

OR

Q.6 a) *Draw and explain the task state segment of 80386.*
[Refer Q.4 of Chapter - 6] [6]

b) *With the necessary diagram, explain entering and leaving the virtual mode of 80386.* **[Refer Q.7 of Chapter - 7]** [6]

c) *Difference between real mode and virtual 8086 Mode.*
[Refer Q.8 of Chapter - 7] [6]

Q.7 a) *Explain the following exception conditions with an example : Faults, Traps and Aborts.* **[Refer Q.2 of Chapter - 8]** [6]

b) *With the help of the necessary diagram, explain the structure of IDT in 80386.* **[Refer Q.6 of Chapter - 8]** [6]

c) *List and elaborate on different applications of microcontrollers.*
[Refer Q.7 of Chapter - 9] [5]

OR

Q.8 a) *Differentiate and explain the interrupt gate and trap gate descriptor.* **[Refer Q.10 of Chapter - 8]** [6]

b) *How interrupts are handled in protection mode. Explain with the help of a neat diagram.* **[Refer Q.6 of Chapter - 8]** [6]

c) *Differentiate between microprocessor and microcontroller.* **[Refer Q.3 of Chapter - 9]** [5]

END...