Introduction
Advanced 32-bit microprocessor optimized for multitasking operating systems

- Designed for applications needing very high performance.
- The 32-bit registers and data paths support 32-bit addresses and data types.
- Can address up to 4 gigabytes of physical memory and 64 terabytes (2^46 bytes) of virtual memory.

- The on-chip memory-management facilities include
  - Address translation registers
  - Advanced multitasking hardware
  - Protection mechanism
  - Paged virtual memory.

- Special debugging registers provide data and code breakpoints even in ROM-based software.
Block Diagram
Pin Discription
These signals are separated in *Six* major groups:

1. Bus / Memory Interface Unit
2. Code Prefetch Unit
3. Instruction Decode Unit
4. Execution Unit
5. Segmentation Unit
6. Paging Unit
Memory/IO Interface

1. Data Bus:
   - The data bus consists of 32 pins (D31 – D0). These lines are used to transfer 8, 16, 24, or 32-bit data at one time.

2. Address Bus:
   - The 80386DX generates 32-bit address.
   - The higher 30-bits of address are sent on the A31-A2.
   - The lower 2-bits, select one of four bytes of the 32-bit data bus.
   - These two bits are internally decoded and sent on the four byte enable pins (BE3- BE0)
**Bus Status Signals:**

- The bus status signals *decide the type of bus cycle* to be performed. These signals are:

1. Address status
2. Write/Read
3. Memory/IO
4. Data/Control
5. LOCK
Register Set
General Purpose Register
The 80386 contains **32-bit general purpose register** called EAX, EBX, ECX, EDX, ESP, EBP, ESI, and EDI.

- The *lower 16-bits of each* of the general purpose register can be accessed individually.

- These 16-bit registers are accessed as AX, BX, CX, DX, SP, BP, SI, and DI respectively.

- The AX, BX, CX and DX registers can be further divided into two separate bytes: Higher byte and lower byte.
Segment Registers
Fig. 2.4.3 Segment registers

- Bit 15
- Bit 0

- CS: Code segment
- DS: Data segment
- SS: Stack segment
- ES: Extra segment
- FS: Extra segment
- GS: Extra segment
The 80386 has a 1 MB address space in real mode.

But all of this memory cannot be active at one time.

It supports six simultaneously accessible memory blocks called segments.

A segment represents an independently accessible block of memory consisting of 64 K consecutive byte-wide storage locations.

These segments are addressed by 16-bit registers: CS, DS, ES, SS, FS and GS.
1. **CS (Code Segment)**: holds the base address of the currently active code segment.

2. **DS (Data Segment)**: is used to hold the address of currently active data segment.

3. **ES (Extra Segment) FS, & GS**: are used as general data segment registers.
   - These registers hold the base addresses of three different memory segments.
   - These segments are referred as Extra Segments.

4. **SS (Stack Segment)**: The base address of the currently active stack segment is contained in the SS register.
Index, Pointers & Base Registers
<table>
<thead>
<tr>
<th>Segment register</th>
<th>Offset registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS (Code Segments)</td>
<td>(E) IP (Instruction Pointer)</td>
</tr>
<tr>
<td>SS (Stack Segments)</td>
<td>(E) SP (Stack Pointer)</td>
</tr>
<tr>
<td></td>
<td>(E) BP (Base Pointer)</td>
</tr>
<tr>
<td>DS (Data Segments)</td>
<td>(E) BX (Base Register)</td>
</tr>
<tr>
<td></td>
<td>(E) SI Source Index Register</td>
</tr>
<tr>
<td></td>
<td>(E) DI Destination Index Register</td>
</tr>
<tr>
<td>ES, FS and GS (Extra Segments)</td>
<td>(E) BX (Base Register)</td>
</tr>
<tr>
<td></td>
<td>(E) SI Source Index Register</td>
</tr>
</tbody>
</table>
Flag Registers
1. **Status Flags**
   i. CF (Carry Flag)
   ii. PF (Parity Flag)
   iii. AF (Auxiliary Carry Flag)
   iv. ZF (Zero Flag)
   v. SF (Sign Flag)
   vi. OF (Overflow Flag)

2. **Control Flags**
   i. DF (Direction flag)

3. **System Flags**
   i. VM (Virtual Memory) flag
   ii. R (Resume) flag
   iii. NT (Nested flag)
   iv. IOPL (I/O Privilege level)
   v. IF (Interrupt Flag)
   vi. TF (Trap Flag)
1. **VM (Virtual Memory) flag:**
   - *Indicates operating mode of 80386.*
   - When VM flag is set, 80386 *switches from protected mode to virtual 8086 mode.*

2. **R (Resume) flag:**
   - This flag, when set *allows selective masking* of some *exceptions* at the time of debugging.
Flag Register

1. NT (Nested flag):
   - This flag is set when one *system task invokes another* task.
   - (i.e. nested task).

2. IOPL (I/O Privilege level):
   - The *two bits* in the IOPL are used by the processor and the operating system to
     determine your application's access to I/O facilities.
5. **IF (Interrupt Flag):**
   - When interrupt flag is set, the 80386 recognizes and handles external hardware interrupts on its INTR pin.
   - If the interrupt flag is cleared, 80386 ignores any inputs on this pin.
   - The *IF flag is set and cleared with the STI and CLI instructions*, respectively.

6. **TF (Trap Flag):**
   - Trap flag allows user to *single-step through programs*. An 80386 detects that this flag is set, it executes one instruction and then automatically generates an internal exception 1.
   - After servicing the exception, the processor executes next instruction and repeats the process.
   - This *single stepping continues until program code resets this flag* for debugging programs single step facility is used.
System Address Registers
System Addresses Register

There are four system address registers:

1. TR (Task Register)
2. IDTR (Interrupt Descriptor Table Register)
3. GDTR (Global Descriptor Table Register)
4. LDTR (Local Descriptor Table Register).

These registers hold the addresses for the four special descriptor table segments.
1. TR (Task Register)
   • Points to the *Task state segment*

2. IDTR (Interrupt Descriptor Table Register)
   • Points to the *Interrupt Descriptor table (IDT)*

3. GDTR (Global Descriptor Table Register)
   • Points to the *Global Descriptor Table (GDT)*

4. LDTR (Local Descriptor Table Register)
   • Points to the *Local Descriptor Table (LDT)*
Control Register
There are four control registers:

- CR0, CR1, CR2 and CR3.
- These registers define the machine state that affects all the tasks in the systems.
Control Register

**CR0 : Holds the MSW (Machine Status Word).**

- It contains six status bits:
  1. PE (Protection Enable)
  2. MP (Math Present)
  3. EM (Emulate Coprocessor)
  4. TS (Task Switched)
  5. ET (Extension Type)
  6. PG (Paging).

![Diagram of CR0 register](image-url)
1. **PE (Protection Enable):**
   - This bit is *similar to the VM* bit in EFLAGS in that it controls the 80386's mode of operation.
   - When PE is set, it is in *Protection mode* otherwise it operates in *Real Mode*.

2. **MP (Math Present):**
   - When this bit is set, the 80386 assumes that real floating point hardware (80287 or 80387) is present in the system.
   - When this bit is clear, the 80386 assumes that no such coprocessor exists, and will not attempt to use real floating point hardware.
3. **EM (Emulate coprocessor):**
   - When this bit is set, the **80386 will generate an exception 11 (device not available)** whenever it attempts to execute a floating point instruction.
   - Programmer can use this exception handler to emulate floating point hardware in software.

4. **TS (Task Switched):**
   - The **80386 sets the bit automatically every time it performs a task switch.**
   - *It will never clear this bit on its own. But programmer can clear this bit using CLTS instruction.*
5. ET (Extension Type):
   - When power is applied, *80386 detects whether numeric Processor connected is 80287 or 80387 & sets ET to logic 1, if numeric processor is 80387.*
   - This is necessary because the 80387 uses a slightly different protocol than 80287.

6. PG (Paging):
   - *This bit enables or disables paging mechanism in Memory Management Unit (MMU).*
   - If bit is set, paging is enabled.
Control Register

1. Control Register 1 (CR1)
   - This is \textit{reserved by Intel.}

2. Control Register 2 (CR2)
   - CR2 is read-only register.
   - The \textit{80386, itself writes the last 32-bit linear address} of page fault routine in this register.
   - When page fault occurs, the 80386 generates exception 14 (page fault)
   - This address \textit{is important for writing page fault routine.}
   - The page \textit{fault routine helps programmer to find cause of the fault.}
Control Register

3. Control Register 3 (CR3)

- Control register 3 holds the physical address of the root of the two level paging tables used when paging is enabled. It is also called Page Directory Base Register.
Physical Address Space
Physical Address Space

- The physical memory of an 80386 system is organized as a sequence of 8-bit bytes.

- Each byte is assigned a unique address that ranges from zero to a maximum of $2^{32} - 1$ (4 gigabytes).

- 80386 programs, however, are independent of the physical address space.
This means that programs can be written without knowledge of how much physical memory is available and without knowledge of exactly where in physical memory the instructions and data are located.

- The model of memory organization seen by applications programmers is determined by systems-software designers.

- The architecture of the 80386 gives designers the freedom to choose a model for each task.
The model of memory organization can range between the following extremes:

1. Flat address space
2. Segmented address space

1. Flat address space:
   - In a "flat" model of memory organization, the applications programmer sees a single array of up to $2^{32}$ bytes (4 gigabytes).
   - The processor maps the 4 gigabyte flat space onto the physical address space by the address translation mechanisms.
2. **Segmented address space:**

- A segmented model consisting of a collection of up to 16,383 linear address spaces of up to 4 gigabytes each.
- In a segmented model of memory organization, the address space as viewed by an applications program (called the logical address space) is a much larger space of up to $2^{46}$ byte (64 terabytes).
- The processor maps the 64 terabyte logical address space onto the physical address space (up to 4 gigabytes) by the address translation mechanisms.
- Both models can provide memory protection. Different tasks may employ different models of memory organization.
Data Types
Data Types

- Data types supported by 80386 DX
Data Types

- Byte
- Word
- Dword
- Near pointer
- Far pointer
- Unpacked BCD
- Packed BCD
- Long integer
- Short real
- Long real
- Extended real
- Packed BCD
- Byte string
- Word string
- Dword string
- Bit string
Data Types

Figure 1-1. Example Data Structure

Figure 1-1. Example Data Structure

GREATEST ADDRESS

DATA STRUCTURE

31  23  15  7  0 <---BIT OFFSET

28
24
20
16
12
8
4

UNDEFINED

BYTE 3  BYTE 2  BYTE 1  BYTE 0

SMALLEST ADDRESS

BYTE OFFSET---+
Operating Modes
80286 and above operate in either the real or protected mode.

Real mode operation allows addressing of only the first 1M byte of memory space—even in Pentium 4 or Core2 microprocessor.

- the first 1M byte of memory is called the real memory, conventional memory, or DOS memory system.
Segments and Offsets

- All real mode memory addresses must consist of a segment address plus an offset address.
  - **segment address** defines the beginning address of any 64K-byte memory segment
  - **offset address** selects any location within the 64K byte memory segment
- Figure 2–3 shows how the **segment plus offset** addressing scheme selects a memory location.
FIGURE 2–3  THE REAL MODE MEMORY-ADDRESSING SCHEME, USING A SEGMENT ADDRESS PLUS AN OFFSET.

- this shows a memory segment beginning at 10000H, ending at location FFFFFH
  - 64K bytes in length

- also shows how an offset address, called a displacement, of F000H selects location 1F000H in the memory
Protected Mode Memory Addressing

- Allows access to data and programs located within & above the first 1M byte of memory.
- **Protected mode** is where Windows operates.
- In place of a segment address, the segment register contains a **selector** that selects a descriptor from a descriptor table.
- The **descriptor** describes the memory segment's location, length, and access rights.
The **descriptor** is located in the segment register & describes the location, length, and access rights of the segment of memory.

- it selects one of 8192 descriptors from one of two tables of descriptors

- In protected mode, this segment number can address any memory location in the system for the code segment.

- Indirectly, the register still selects a memory segment, but not directly as in real mode.
- **Global descriptors** contain segment definitions that apply to all programs.
- **Local descriptors** are usually unique to an application.
  - a global descriptor might be called a **system descriptor**, and local descriptor an **application descriptor**
- Figure shows the format of a descriptor for the 80286 through the Core2.
  - each descriptor is 8 bytes in length
  - global and local descriptor tables are a maximum of 64K bytes in length
The 80286 through Core2 64-bit descriptors.
Selectors and Descriptors

- The **base address** of the descriptor indicates the starting location of the memory segment.
  - the paragraph boundary limitation is removed in protected mode
  - segments may begin at any address

- The G, or **granularity bit** allows a segment length of 4K to 4G bytes in steps of 4K bytes.
  - 32-bit offset address allows segment lengths of 4G bytes
  - 16-bit offset address allows segment lengths of 64K bytes.
Protected Mode Addressing Mechanism
The 80386 DX has three distinct address spaces:
- Logical,
- Linear and
- Physical

A logical address (also known as virtual address) consists of a *selector and an offset*.

A selector is the contents of a segment register.
Protected Mode Addressing Mechanism

- **A selector**: is used to point a *descriptor* for the segment in a *table of descriptors*.

- **In real mode**:
  - the segmentation unit *shift’s the selector left four bits* & adds the result to the offset to form the linear address.

- **In protected mode**:
  - Every *segment selector* has a *linear base address* associated with it, and it is stored in the *segment descriptor*.
  - *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*.
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.
Segment Descriptor
<table>
<thead>
<tr>
<th>Bytes</th>
<th>31</th>
<th>23</th>
<th>15</th>
<th>Access Right Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BASE 31 ..... 24</th>
<th>G</th>
<th>X</th>
<th>0</th>
<th>AVL</th>
<th>LIMIT 19 ..... 16</th>
<th>P</th>
<th>DPL</th>
<th>S</th>
<th>TYPE</th>
<th>A</th>
<th>BASE 23 ..... 16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**BASE** Base address of the segment

**LIMIT** The length of the segment

**P** Present bit: 1 = Present 0 = Not present

**DPL** Descriptor privilege level 0 - 3

**S** Segment descriptor: 0 = System descriptor 1 = Code or Data segment descriptor

**TYPE** Type of segment

**A** Accessed bit

**G** Granularity bit: 1 = Segment length is page granular 0 = Segment length is byte granular

**0** Bit must be zero (0) for compatibility with future processors

**AVL** Available field for user or OS
In **protected mode**, Memory Management Unit (MMU) uses the **segment selector** to access a **descriptor** for the desired segment in a table of descriptors in memory.

**Segment descriptor**:  
- is a special structure which **describes the segment**.  
- Exactly **one segment descriptor** must be defined for each segment of the memory.
Descriptors are eight type quantities which contain attributes about a given region of linear address space (i.e. a segment).

These attributes include

- the **32-bit base linear address** of the segment,
- the **20-bit length and granularity** of the segment,
- the **protection level**,
- **read, write or execute privileges**,
- the **default size of the operands** (15-bit or 32Bit), and
- the **type of segment**.
Segment Descriptor

- **Base:**
  - It contains the *32-bit base address for a segment.*
  - Thus defines the location of the segment within the 4 gigabyte linear address space.
  - The 80386 concatenates the three fragments of the base address to form a single 32-bit address.

- **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 *1 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.
Granularity Bit:
- It specifies the *units with which the limit field is interpreted.*
- When *bit is 0*, the limit is interpreted in units of *one byte*;
- otherwise limit is interpreted in units of *4 Kbytes*.

0 (Reserved by Intel):
- It neither can be defined nor can be used by user.
- This bit must be zero for compatibility with future processors.

AVL/U (User Bit):
- This bit is *completely undefined*, and 80386 *ignores it*.
- This is *available field/bit for user* or operating system.
Segment Descriptor

**Access rights byte :**

- **P (Present Bit) :**
  - If \( P = 1 \) if the segment is loaded in the physical memory,
  - If \( P = 0 \) then any attempt to access this segment causes a not present exception (exception 11).

- **DPL (Descriptor Privilege Level) :**
  - It is a 2-bit field defines the level of privilege associated with the memory space that the descriptor defines – DPL\(_0\) is the most privileged whereas DPL\(_3\) is the least privilege

- **Type :**
  - This specifies the specific descriptors among various kinds of descriptors.
Segment Descriptor

- **S (System Bit):**
  - The segment S bit in the segment descriptor determines if a given segment is a **system segment** or a **code** or a **data segment**.
  - If the S bit is 1 then the segment is either a **code or data segment**, if it is 0 then the segment is **system segment**.

- **A (Accessed Bit):**
  - The 80386 **automatically** sets this bit when a selector for the descriptor is loaded into a **segment register**.
  - This means that 80386 sets accessed bit **whenever a memory reference is made by accessing the segment**.
Types Segment Descriptor
Types Segment Descriptor

Segment Descriptors

System
- LDT
- TSS
- Gate

Non-system
- Code
- Data
Types Segment Descriptor: **Non-System Segment Descriptors**

**Non-System Segment Descriptors**

**Data Segment Descriptor**
Types Segment Descriptor: *Non-System Segment Descriptors*

**Code Segment Descriptor**

![Code Segment Descriptor Diagram]
**Systeme Segment Descriptor**

<table>
<thead>
<tr>
<th>Type</th>
<th>Defines</th>
<th>Type</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved by Intel</td>
<td>8</td>
<td>Reserved by Intel</td>
</tr>
<tr>
<td>1</td>
<td>Available 80286 TSS</td>
<td>9</td>
<td>Available Intel 80386DX TSS</td>
</tr>
<tr>
<td>2</td>
<td>LDT</td>
<td>A</td>
<td>Undefined (Intel reserved)</td>
</tr>
<tr>
<td>3</td>
<td>Busy 80286 TSS</td>
<td>B</td>
<td>Busy Intel 80386DX TSS</td>
</tr>
<tr>
<td>4</td>
<td>80286 call gate</td>
<td>C</td>
<td>Intel 80386DX call gate</td>
</tr>
<tr>
<td>5</td>
<td>Task gate (for 80286 or Intel 80386DX task)</td>
<td>D</td>
<td>Undefined (Intel reserved)</td>
</tr>
<tr>
<td>6</td>
<td>80286 Interrupt gate</td>
<td>E</td>
<td>Intel 80386DX interrupt gate</td>
</tr>
<tr>
<td>7</td>
<td>80286 Trap gate</td>
<td>F</td>
<td>Intel 80386DX trap gate</td>
</tr>
</tbody>
</table>
Types Segment Descriptor: **System Segment Descriptors**

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 tables. The local descriptor tables contain the segment descriptors which are unique to a particular task. The DPL (Descriptor Privilege field) of this descriptor is ignored because it can be accessed with only privilege level 0.
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 task 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 \((S = 0, \text{TYPE} = 4 - 7, \text{C, F})\): 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.
Types Segment Descriptor: **System Segment Descriptors**

**Gate Descriptor**

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
Descriptor Table
Segment descriptors are grouped and placed one after the other in contiguous memory locations.

This group arrangement is known as a descriptor table.

The maximum limit for the length of descriptor table is 64 kbytes and each descriptor takes 8 bytes to store the information of a particular segment.

So descriptor table can have as many as 8192 descriptors.

The upper 13 bits of a selector are used as an index into the descriptor table.
There are three types of descriptor tables:

- **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.

- **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 8085 system.

- **Local Descriptor Tables (LDT):**
  - They are set up in the system for individual task or closely related group of tasks.
Descriptor Table

Task 1
Local address space
LDT

Task 1
Virtual address space

Task 2
Virtual address space

Task 3
Virtual address space

Task 3
Local address space
LDT

Task 2
Local address space
LDT

Global address space
GDT
Descriptor Table: GDTR
Descriptor Table : GDTR

- 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 1 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 =102316).
- Since the LIMIT field is 16 bit long, the GDT can grow up to 65,536 bytes long.
- The upper four bytgs of GDTR specifies the 32-bit linear address of the base of the Global Descriptor Table (GDT).
Descriptor Table: IDTR
Descriptor Table : IDTR

- Like global descriptor table register, Interrupt descriptor table register (IDTR) holds the 16-bit limit and 32-bit linear address of the base of the Interrupt Descriptor Table (IDT).
- Interrupt Descriptor Table Register are used to define a Interrupt Descriptor Table (IDT) in the 80386DX physical memory address space.
- 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 803B6DX only supports upto 256 interrupts or exceptions;
- therefore, the size of the IDT should not be set to support more than 256 interrupts.
Descriptor Table: 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).

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.
The GDT can contain many LDT descriptors.

To put particular LDT in service, it is necessary to load the LDTR with corresponding selector.

For loading the values in GDTR, IDTR and LDTR registers, 80336DX provides LGDT, LLDT, and LIDT instructions.

It also provides SGDT, SLDT and SIDT instructions.

These (48 bits) instructions copy the contents of the descriptor table registers into the six bytes of memory pointed by the destination operand.

These tables are manipulated by the operating system.

Thus, the instructions used for loading the descriptor tables are privileged instructions.
Segmentation
Segmentation

- Segmentation is a process of converting logical address into a linear address.
- *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.
Segmentation

- The selector component of each logical address contains 2 bits 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.
Segmentation

- 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.
There are two major categories of descriptor table in a 80386 system:

- **Global Descriptor Table (GDT)** is a general purpose table of descriptors, can be used by all programs to reference segments of memory.

- **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.