# UNIT-IV Protection

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

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

## Need of Protection:

The purpose of the protection features of the 80386 is to help detect and identify bugs.

The 80386 supports sophisticated applications that may consist of hundreds or thousands of program modules.

In such applications, the question is how bugs can be found and eliminated as quickly as possible and how their damage can be tightly confined.

To help debug applications faster and make them more robust in production, the 80386 contains mechanisms to verify memory accesses and instruction execution for conformance to protection criteria.

These mechanisms may be used or ignored, according to system design objectives.

# Overview of 80386 Protection Mechanisms :

The 80386 uses Segment level protection and Page level protection mechanisms to protect critical section.

**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**

The protection hardware of the 80386 is an integral part of the memory management hardware.

**Protection applies both to segment translation and to page translation Available in protected mode only.**

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.

# Segment-Level Protection

**All five aspects of protection apply to segment translation:**
Type checking
Limit checking
Restriction of addressable domain
Restriction of procedure entry points
Restriction of instruction set

The segment is the unit of protection, and segment descriptors store protection parameters.

The CPU performs protection checks automatically 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.

**1 Type Checking**

**The TYPE field of a descriptor has two functions:**
1.Type field of the descriptor specifies
2.Type of descriptor (system/non-system)
Intended usage of the segment.
Ex. If the segment is read only segment then its accessed is limited to only reading purpose.

**2 Limit Checking:**

To prevent program from addressing outside the segments.

- It interprets limit field depending on the setting of the G (granularity) bit, which specifies whether limit value counts 1 byte or 4 Kb.

- The 80386DX causes a general protection exception when program attempts to
  - Access memory byte at an address > limit
  - Access memory word at an address>= limit
  - Access memory dword at address>= (limit-2)

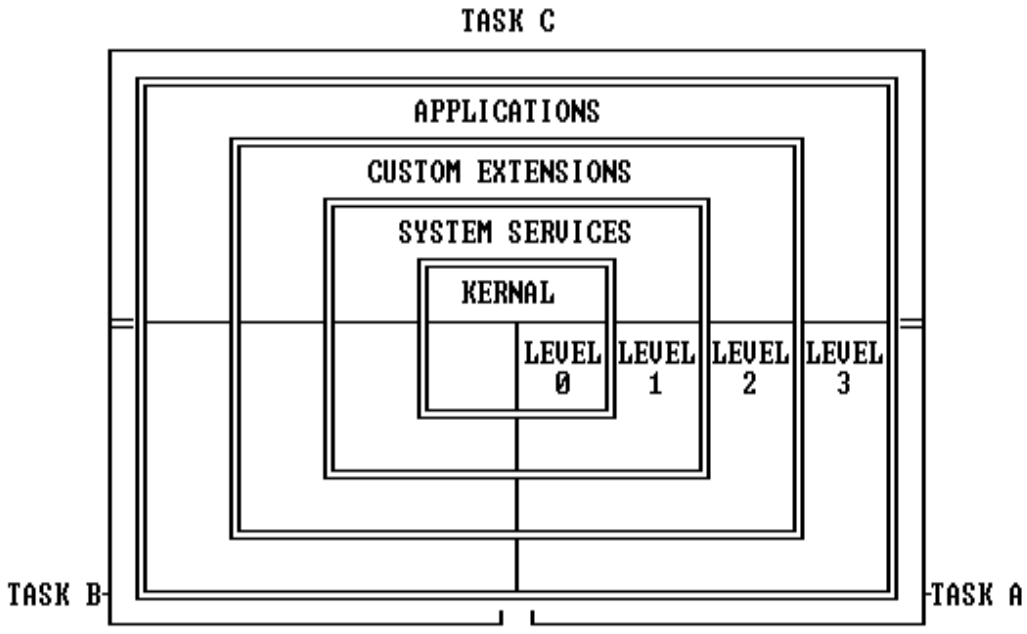
## Protection Levels: Privilege Level Protection

80386 DX has four levels of protection, which isolate and protect user programs from each other and the operating system.

It offers an additional type of protection on a page basis, when paging is enabled (using U/S and R/W fields)

The four-level hierarchical privilege system is illustrated as follow

Figure 6-2. Levels of Privilege




00: Highest Privilege
01
10
11: Lowest Privilege

**Level 0 is the most privileged or trusted level.**
**Level 3 is the least privileged level.**

**Privilege Level**

There are 4 different types of privilege level entering into the privilege level checks:

**Current Privilege Level (CPL)**

**Descriptor Privilege Level (DPL)**

**Requestor Privilege Level (RPL)**

**Effective Privilege Level (EPL)**

1) **Current Privilege Level (CPL) :**

- Also called Task Privilege Level
- It specifies privilege level of currently executing task
- A task's CPL can only be changed by control transfers through gate -descriptors to a code segment with a different privilege level.
- E.g. an application program running at PL = 3 may call an OS routine at PL = 1 (via a gate) which would cause the task's CPL to be set to 1 until the OS routine is finished.
- CPL is stored in the selector of currently executing CS register.

- It represents the privilege level(PL) of the currently executing task.
- It is also PL in the descriptor of the code segment.
- It is also designated as Task Privilege Level(TPL)

2) **Descriptor Privilege Level (DPL) :**

- It is the PL of the object, which is being attempted to be accessed by the current task.
- It is PL of target segment and is contained in the descriptor of the segment.
- It is the least privileged level at which a task may access that descriptor -and the segment associated with that descriptor.
- It is contained in the access right byte of the descriptor of the segment.

3) **Requestor Privilege Level (RPL) :**

- Selectors contain field called the RPL.
- It is the lowest two bits of any selector.
- It can be used to weaken the CPL if desired.

RPL is the two least significant bits of selector

**4) Effective Privilege Level (EPL) :**

- When access to a new memory segment is desired, an Effective Privilege Level (EPL) is computed.
- The Effective Privilege Level(EPL) is EPL = max (CPL,RPL) (here numbers)
- EPL is defined as EPL = max { RPL,CPL } (numerically)
  Thus, the task becomes less privileged
- E.g. If RPL = 2 and CPL = 1, EPL = 2 task becomes less privileged

## Restricting Access to Data :

To address operands in memory, an 80386 program must load the selector of a data segment into a data-segment register (DS, ES, FS, GS, SS).

The processor automatically evaluates access to a data segment by comparing privilege levels.

The evaluation is performed at the time a selector for the descriptor of the target segment is loaded into the data-segment register.

**Three different privilege levels enter into this type of privilege check:**

1. The CPL (current privilege level).
2. The RPL (requestor's privilege level) of the selector used to specify the target segment.
3. The DPL of the descriptor of the target segment.

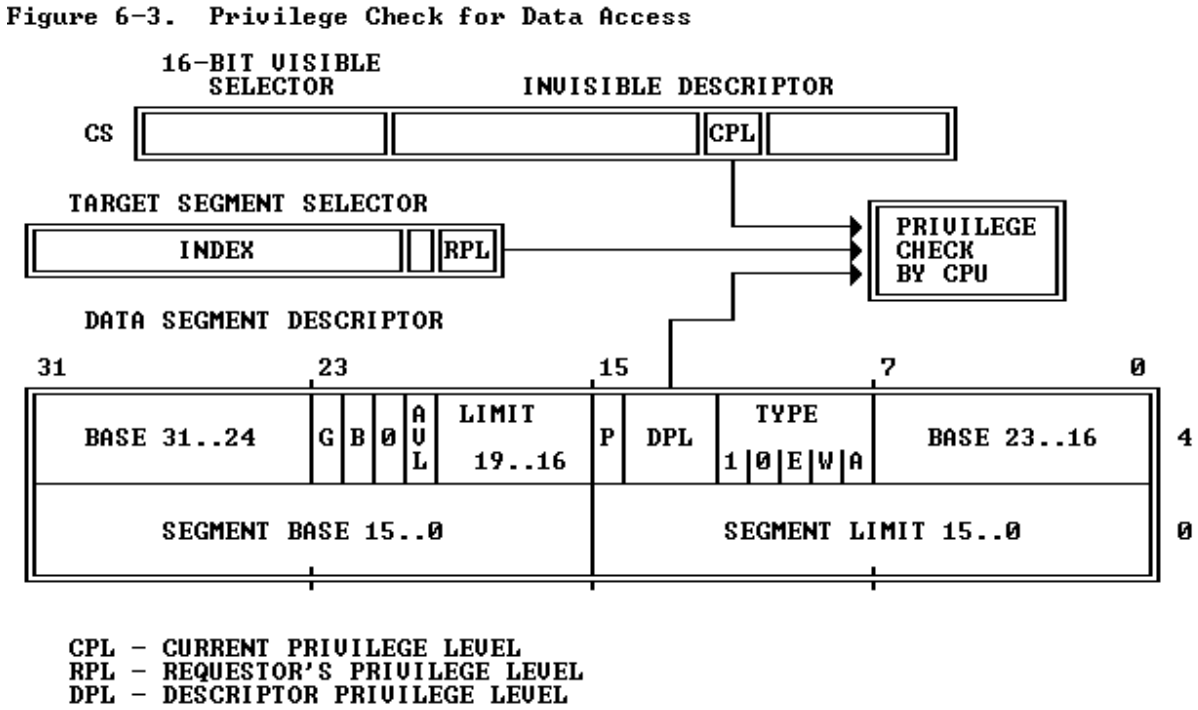
| No. | Previlege Levels | | | Access |
|---|---|---|---|---|
| | DPL | CPL | RPL | |
| 1 | 2 | 0 | 1 | Valid |

| | | | | |
|---|---|---|---|---|
| 2 | 3 | 1 | 2 | Valid |
| 3 | 1 | 1 | 0 | Valid |
| 4 | 1 | 2 | 0 | Invalid |
| 5 | 2 | 2 | 3 | Invalid |

Fig: Data Accesses

Instructions may load a data-segment register (and subsequently use the target segment) 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.

   In Case 1, procedure can only access data that is at the same or less privileged level.

2. Load a data-segment register with a selector of a conforming, readable, executable segment.

   In Case 2, is always valid because the privilege level of the segment whose conforming bit is set.

3. Use a CS override prefix to read a readable, executable segment whose selector is already loaded in the CS register.

   In Case 3, is also always valid because the DPL of the code segment in CS is by definition ,equal to CPL.

## Restricting Control Transfers :

The 80386DX control transfers are accomplished by the instructions JMP, CALL, RET, INT, and IRET, as well as by the exception and interrupt mechanisms .

The "near" forms of JMP, CALL, and RET transfer within the current code segment, and therefore are subject only to limit checking.

The processor ensures that the destination of the JMP, CALL, or RET instruction does not exceed the limit of the current executable segment.

This limit is cached in the CS register; therefore, protection checks for near transfers require no extra clock cycles.

- **To Sucessfully transfer the control to other segment, both the RPL and CPL must be a number less than or equal to the DPL of Segment**

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

## Changing Privilege Levels :

Figure 6-4. Privilege Check for Control Transfer without Gate

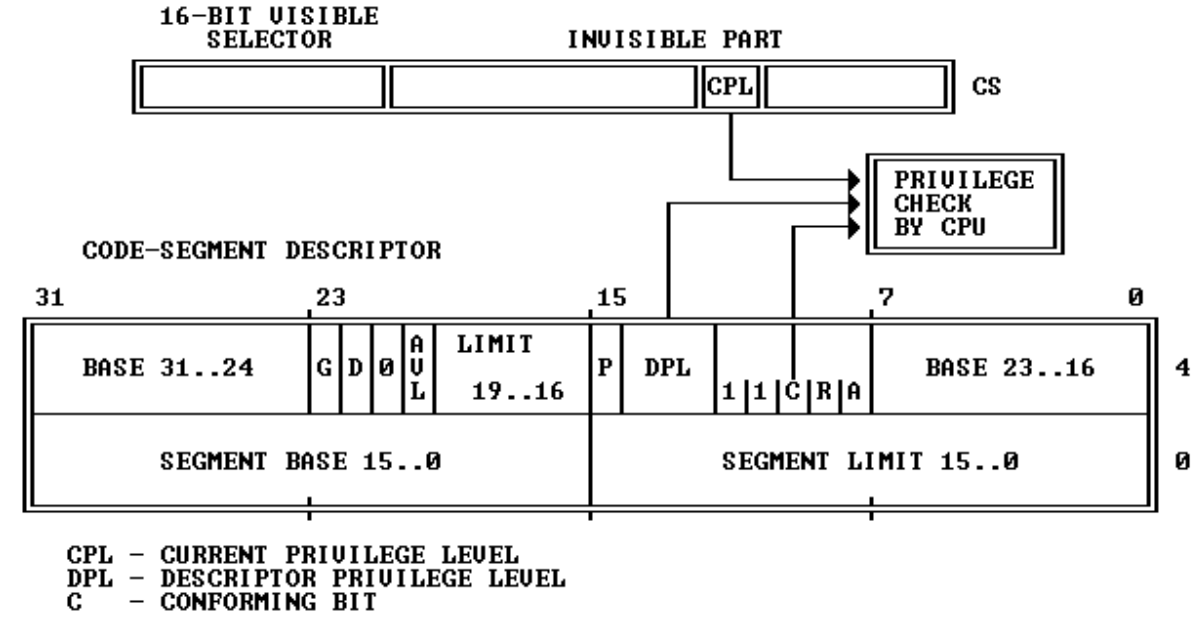


Figure 6-5. Format of 80386 Call Gate

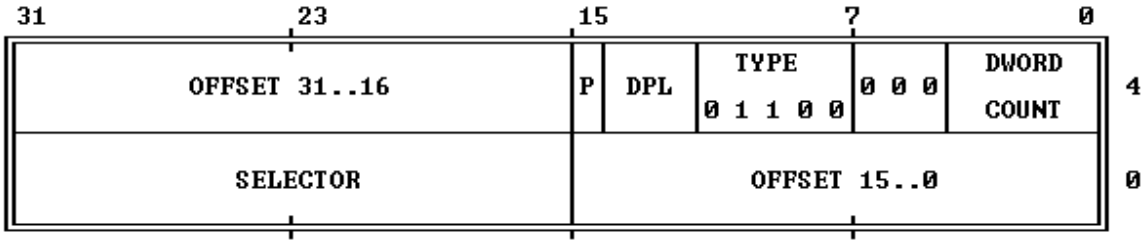
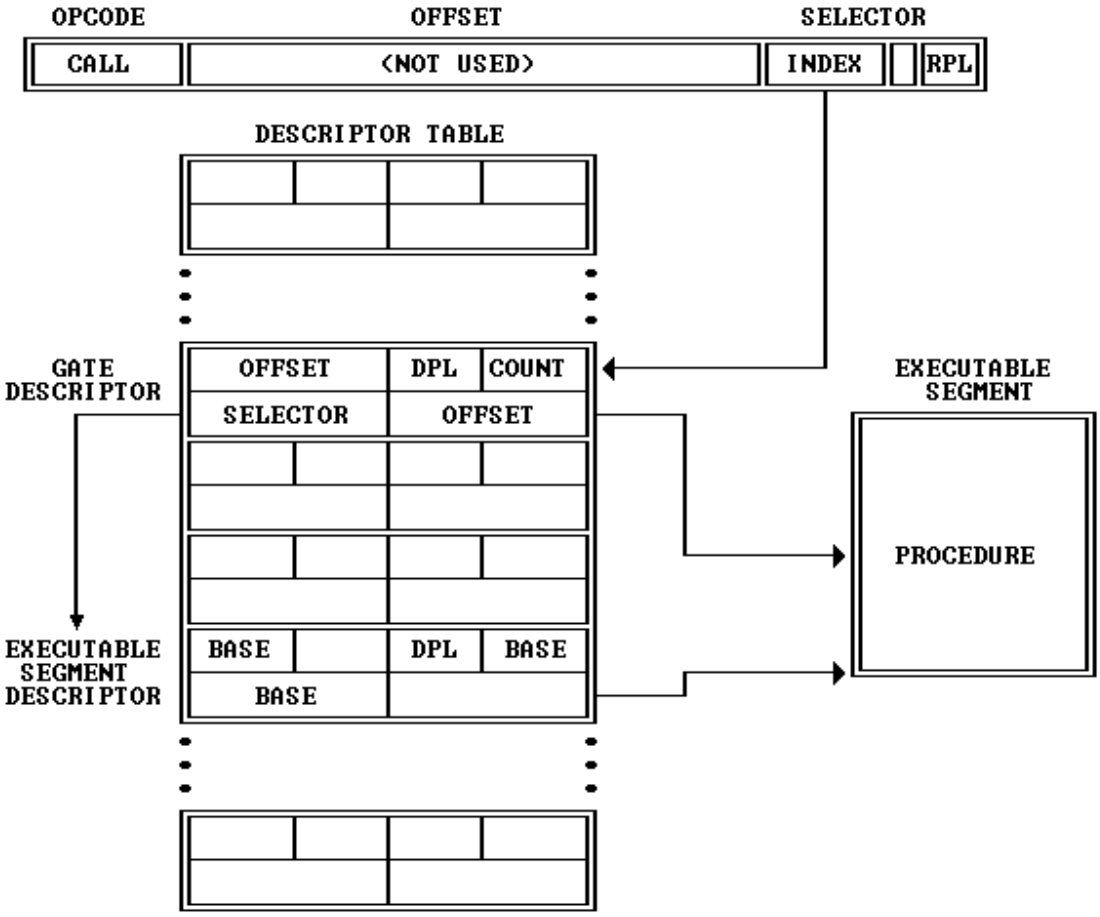


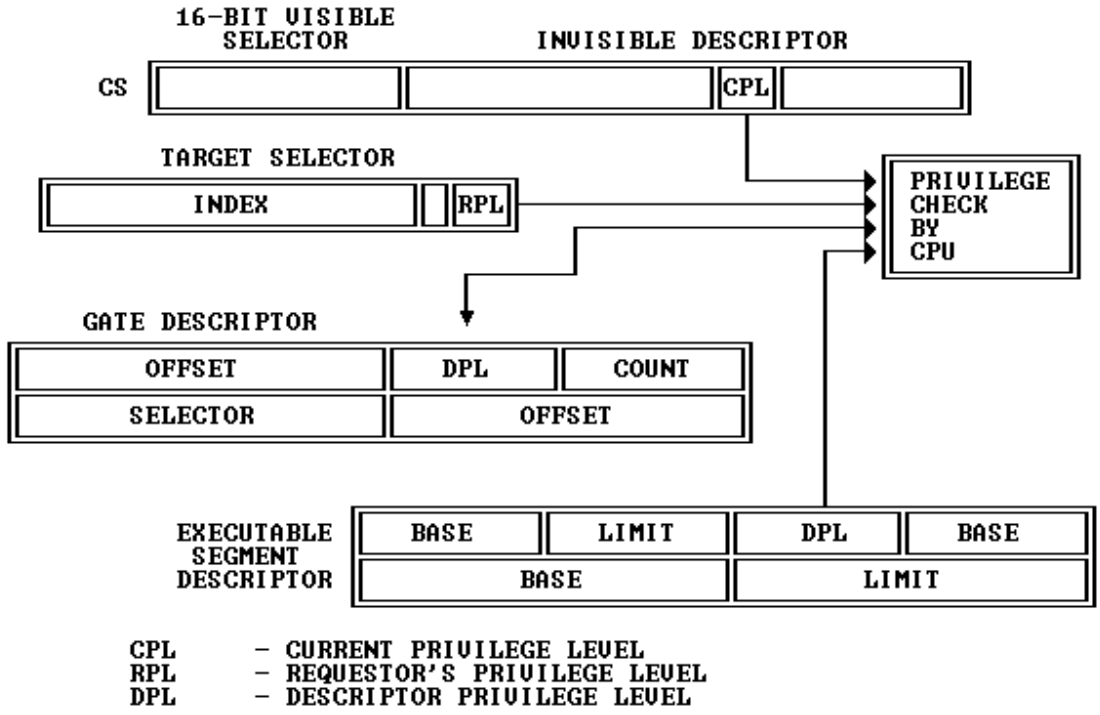

Figure 6-6. Indirect Transfer via Call Gate



Four different privilege levels are used to check the validity of a control transfer via a call gate:

1. The CPL (current privilege level).
2. The RPL (requestor's privilege level) of the selector used to specify the call gate.
3. The DPL of the gate descriptor.
4. The DPL of the descriptor of the target executable segment.

Figure 6-7. Privilege Check via Call Gate




- For Valid control transfer, the transfer must satisfy the following privilege rules for CALL instructions.

**Target DPL ≤ Max (RPL,CPL) ≤ Call Gate DPL**

## Stack Switching

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

To maintain system integrity, each privilege level has a separate stack. These stacks assure sufficient stack space to process calls from less privileged levels.

Without them, a trusted procedure would not work correctly if the calling procedure did not provide sufficient space on the caller's stack.

The processor locates these stacks via the task state segment.

Each task has a separate TSS, thereby permitting tasks to have separate stacks.

Systems software is responsible for creating TSSs and placing correct stack pointers in them.

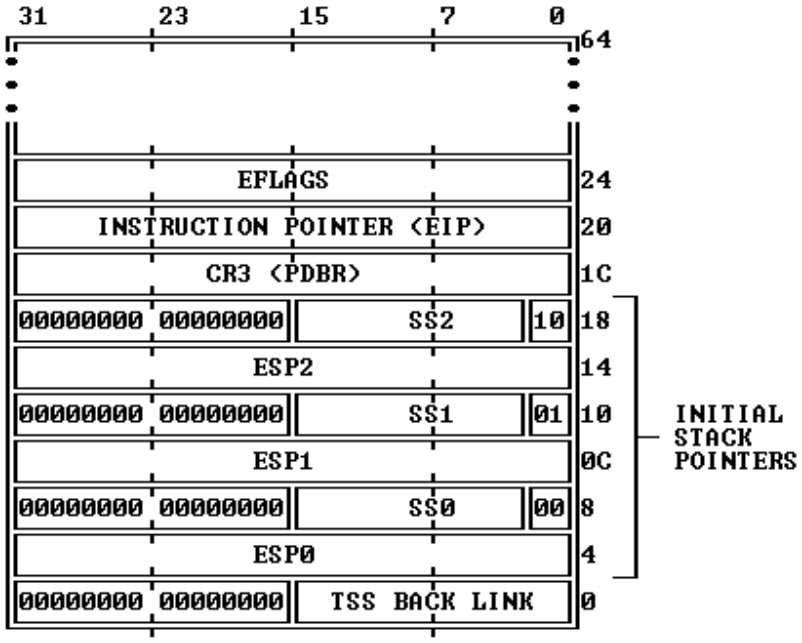
The initial stack pointers in the TSS are strictly read-only values.

The processor never changes them during the course of execution.

When a call gate is used to change privilege levels, a new stack is selected by loading a pointer value from the Task State Segment (TSS).

The processor uses the DPL of the target code segment (the new CPL) to index the initial stack pointer for PL 0, PL 1, or PL 2.

Figure 6-8. Initial Stack Pointers of TSS




# Page-Level Protection :

**Q. Explore the role of various fields in Page Level Protection.**

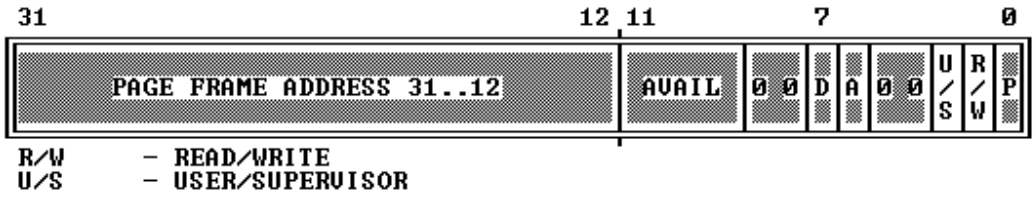
Two kinds of protection are related to pages:

1. Restriction of addressable domain.

2. Type checking.

**Page-Table Entries Hold Protection Parameters**

Figure 6-10. Protection Fields of Page Table Entries




## 1 Restricting Addressable Domain :

The concept of privilege for pages is implemented by assigning each page to one of two levels:

1. Supervisor level (U/S=0) -- for the operating system and other systems software and related data.
2. User level (U/S=1) -- for applications procedures and data.

The current level (U or S) is related to CPL. If CPL is 0, 1, or 2, the processor is executing at supervisor level. If CPL is 3, the processor is executing at user level.
When the processor is executing at supervisor level, all pages are addressable, but, when the processor is executing at user level, only pages that belong to the user level are addressable.

## 2 Type Checking

At the level of page addressing, two types are defined:

1. Read-only access (R/W=0)
2. Read/write access (R/W=1)

When the processor is executing at supervisor level, all pages are both readable and writable.

When the processor is executing at user level, only pages that belong to user level and are marked for read/write access are writable;
pages that belong to supervisor level are neither readable nor writable from user level.

### Combining Protection of Both Levels of Page Tables

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.

For example, 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).

This technique might be used, for example, in a UNIX-like system to define a large data segment, part of which is read only (for shared data or ROMmed constants).

This enables UNIX-like systems to define a "flat" data space as one large segment, use "flat" pointers to address within this "flat" space, yet be able to protect shared data, shared files mapped into the virtual space, and supervisor areas.

# I/O Protection :

Two mechanisms provide protection for I/O functions:

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

These mechanisms operate only in protected mode, including virtual 8086 mode; they do not operate in real mode.
In real mode, there is no protection of the I/O space; any procedure can execute I/O instructions, and any I/O port can be addressed by the I/O instructions.

1. **I/O Privilege Level :**

Instructions that deal with I/O need to be restricted but also need to be executed by procedures executing at privilege levels other than zero.

For this reason, the processor uses two bits of the flags register to store the I/O privilege level (IOPL).

The IOPL defines the privilege level needed to execute I/O-related instructions.

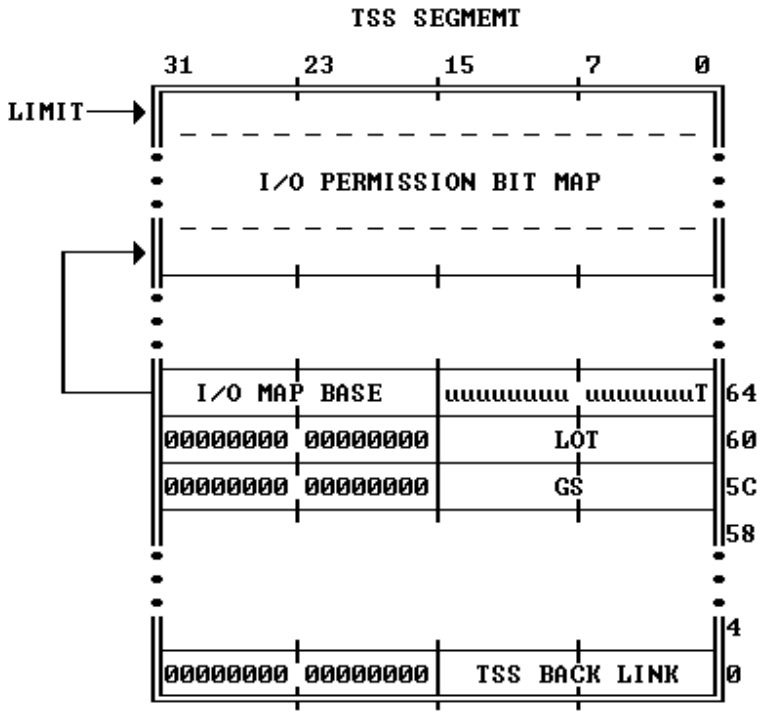
The following instructions can be executed only if CPL <= IOPL:

- IN -- Input
- INS -- Input String
- OUT -- Output
- OUTS -- Output String
- CLI -- Clear Interrupt-Enable Flag
- STI -- Set Interrupt-Enable

These instructions are called "sensitive" instructions, because they are sensitive to IOPL.

## 2. I/O Permission Bit Map :

Figure 8-2. I/O Address Bit Map




The I/O instructions that directly refer to addresses in the processor's I/O space are IN, INS, OUT, OUTS.

The 80386 has the ability to selectively trap references to specific I/O addresses.

The structure that enables selective trapping is the I/O Permission Bit Map in the TSS segment.

The I/O permission map is a bit vector.

The size of the map and its location in the TSS segment are variable.

The processor locates the I/O permission map by means of the I/O map base field in the fixed portion of the TSS.

The I/O map base field is 16 bits wide and contains the offset of the beginning of the I/O permission map.

The upper limit of the I/O permission map is the same as the limit of the TSS segment.

In protected mode, when it encounters an I/O instruction (IN, INS, OUT, or OUTS), the processor first checks whether CPL <= IOPL.

If this condition is true, the I/O operation may proceed. If not true, the processor checks the I/O permission map. (In virtual 8086 mode, the processor consults the map without regard for IOPL

Each bit in the map corresponds to an I/O port byte address;

for example, the bit for port 41 is found at I/O map base + 5, bit offset 1.

The processor tests all the bits that correspond to the I/O addresses spanned by an I/O operation;

for example, a doubleword operation tests 4 bits corresponding to four adjacent byte addresses.

If any tested bit is set, the processor signals a general protection exception.

If all the tested bits are zero, the I/O operation may proceed.

# Privilege and I/O Sensitive Instructions :

## Q.List and explain various Privilege Instructions.

Privileged instructions are those that affect the segmentation and protection mechanism, after the interrupt flag or perform peripheral I/O.

These instructions are divided in two groups.
1) Privileged instructions
2) IOPL- Sensitive Instructions

## 1) Privileged instructions :

| Instructions | Actions |
|---|---|
| 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 : Privileged Instructions**

# 1)HLT -- Halt

### Operation

Enter Halt state;

### Description

HALT stops instruction execution and places the 80386 in a HALT state. An enabled interrupt, NMI, or a reset will resume execution.

**Flags Affected : None**

# 2) CLTS -- Clear Task-Switched Flag

**Operation : TS Flag in CR0 := 0;**

## Description

CLTS clears the task-switched (TS) flag in register CR0. This flag is set by the 80386 every time a task switch occurs.

The TS flag is used to manage processor extensions as follows:

- Every execution of an ESC instruction is trapped if the TS flag is set.
- Execution of a WAIT instruction is trapped if the MP flag and the TS flag are both set.

CLTS appears in operating system software, not in application programs. It is a privileged instruction that can only be executed at privilege level 0.

**Flags Affected : TS := 0 (TS is in CR0, not the flag register)**

# 3)LGDT/LIDT/LLDT – Load GDT/IDT/LDT Descriptor Table Register :

## Description

The LGDT and LIDT instructions load a linear base address and limit value from a six-byte data operand in memory into the GDTR or IDTR, respectively.

If a 16-bit operand is used with LGDT or LIDT, the register is loaded with a 16-bit limit and a 24-bit base, and the high-order eight bits of the six-byte data operand are not used.

If a 32-bit operand is used, a 16-bit limit and a 32-bit base is loaded; the high-order eight bits of the six-byte operand are used as high-order base address bits.

LLDT loads the Local Descriptor Table register (LDTR). The word operand (memory or register) to LLDT should contain a selector to the Global Descriptor Table (GDT).

The GDT entry should be a Local Descriptor Table. If so, then the LDTR is loaded from the entry. The descriptor registers DS, ES, SS, FS, GS, and CS are not affected. The LDT field in the task state segment does not change.

# 4)LTR -- Load Task Register :

## Description

LTR loads the task register from the source register or memory location specified by the operand. The loaded task state segment is marked busy. A task switch does not occur.

LTR is used only in operating system software; it is not used in application programs.

**Flags Affected : None**

# 5) LMSW -- Load Machine Status Word

## Description

LMSW loads the machine status word (part of CR0) from the source operand. This instruction can be used to switch to Protected Mode; if so, it must be followed by an intrasegment jump to flush the instruction queue.

LMSW will not switch back to Real Address Mode.

LMSW is used only in operating system software. It is not used in application programs.

**Flags Affected : None**

# 6)MOV CRn , REG/MOV REG, CRn--Move to/from Control Registers

## Description

Moves the contents of a control register (CR0, CR2, CR3, or CR4) to a general-purpose register or vice versa.
The operand size for these instructions is always 32 bits, regardless of the operand-size attribute.

# 7) MOV DRn , REG/MOV REG, DRn--Move to/from Debug Registers

## Description

Moves the contents of a debug register (DR0, DR1, DR2, DR3, DR4, DR5, DR6, or DR7) to a general-purpose register or vice versa.
The operand size for these instructions is always 32 bits in non-64-bit modes, regardless of the operand-size attribute.
The instructions must be executed at privilege level 0 or in real-address mode.

# 8) MOV TRn , REG/MOV REG, TRn--Move to/from Test Registers

## Description

This form of mov stores or loads the Test Register TR6 or TR7 to or from a general purpose register.

These instructions are always used with 32-bit operands.

Example

```
movl %tr7, %ebp
movl %ebp, %tr7
```

## 2) IOPL- Sensitive Instructions :

The IOPL field in the FLAG register defines the right to use I/O releted instructions. Hence the instructions from this group are called **Sensitive instructions.**

| Instructions | Actions |
|---|---|
| CLI Clear Interrupt) | Disables interrupts |
| STI (Set Interrupt-Enable) | Enables interrupts |
| IN,INS(Input/ Input String) | Inputs data from I/O port |
| OUT,OUTS (Output/ Output String) | Outputs data from I/O port |

**Table : IOPL- Sensitive Instructions**