LOKNETE DR. BALASAHEB VIKHE PATIL
(PADMA BHUSHAN AWARDEE)
PRAVARA RURAL EDUCATION SOCIETY'S
SIR VISVESVARAYA INSTITUTE OF TECHNOLOGY
PRAVARA TECHNICAL EDUCATION CAMPUS NASHIK
NASHIK

- SE Computer (22-23)

Microprocessor

(Course Code: 210254)

Unit IV- Protection

Subject In-charge -Dr. Prerana N.Khairnar

Need of Protection

- 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.

Overview of 80386DX Protection Mechanisms

- 80386DX uses system level protection & page level protection.
- protection has 5 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.

Segment Level protection

- All 5 aspects of protection apply to segment

(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.
- 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.

Segment Level protection

- 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 Checking used to detect whether any program is attempting to use segments in ways not intended by the programmer.
- Descriptors specifies the parameter as **Type of descriptor & intended usage of segment**
- Example:

If the segment is **read only segment (R bit=1)**,then its accessed is **limited to only reading purpose**.

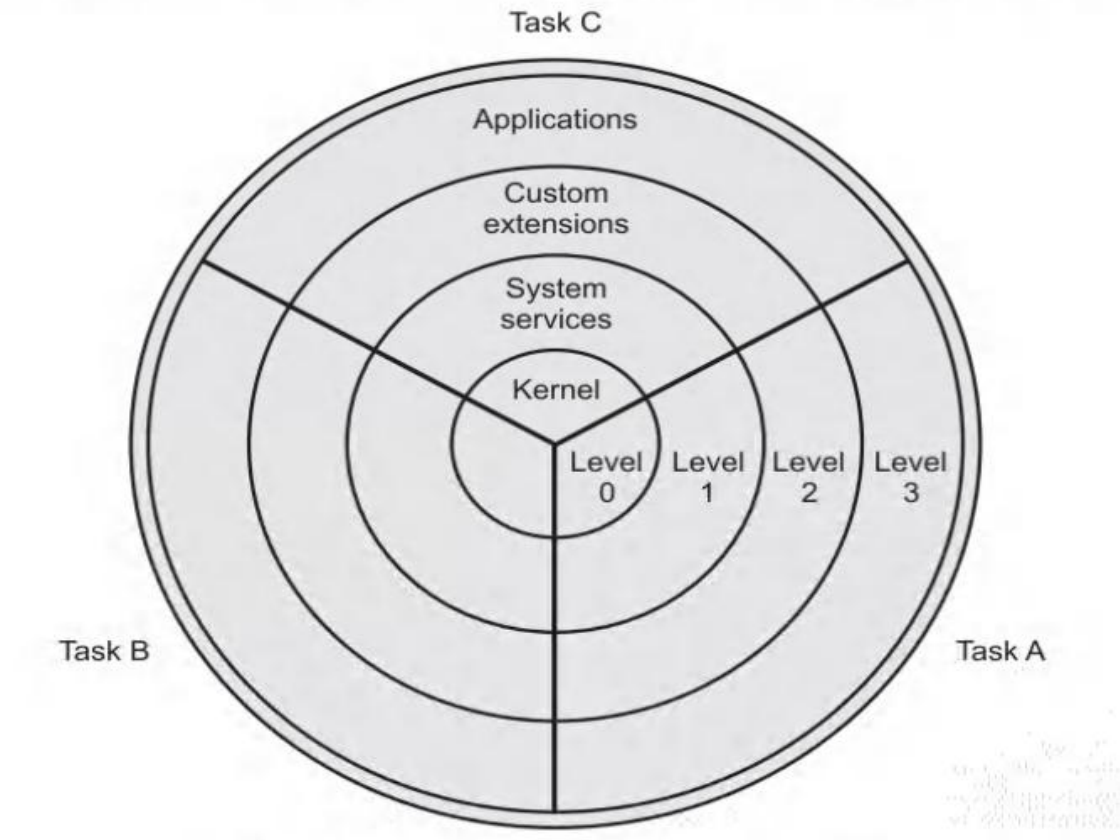
(Readable), C (Conforming), A (Accessed) and, E (Expanded-Down) bits from type field specify the usage of the segment and restrict segment for particular use only.

Limit Checking

- To prevent program from addressing outside the segments.
- **Limit < size of 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.
- Also check **ED(Expalnsion Direction)bit & B(Big)bit**
- 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)

Privilege Level Protection

- 80386 has 4 level of protection.
- The privilege levels (PL) are numbered 0 through 3.
- Level 0 is the most privileged or trusted level.



- Isolate & protect user program from each other in multi tasking OS

Concept of DPL,CPL,RPL,EPL

DPL	CPL	RPL	EPL
Descriptor Privilege Level	• Current Privilege Level -privilege of the currently executing code • Task privilege level	Requested Privilege Level	Effective Privilege Level
All 8 byte descriptors that define code, data, stack etc., segments have two bits reserved for DPL	Last two bits of CS register are considered as CPL.	These are the last two bits of DS, ES, SS, FS, GS registers	EPL = max { RPL,CPL }
DPL bits specify the minimum (or sometimes maximum) privilege required for using that segment contents.			

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.**
- **Normally, CPL = DPL** of the segment that the processor is currently executing.
- **CPL changes** as control is transferred to segments **with differing DPLs.**

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.

Requestor Privilege Level (RPL)

- RPL is the **two least significant bits of selector**
- RPL is used **to establish a less trusted privilege level than CPL** for the use of a segment and this level is called the task's **Effective Privilege Level** (EPL).
- 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

Protection

- **CPL** – Current Privilege Level is the privilege level of the currently executing program or task. It is stored in CS and SS segment registers
- **DPL** - Descriptor privilege level is the privilege level of a segment. It is stored in the DPL field of the segment descriptor
- **RPL** - Requested privilege level is an override privilege level use to specify the target segment.

Restricting Access to Data

- Assume that a task needs data from data segment.
- The privilege levels are checked at the time a selector for the target segment is loaded into the data segment register.
- Three privilege levels enter into privilege checking mechanism
  - CPL
  - RPL of the selector of target segment
  - DPL of the descriptor of the target segment
- A procedure can only access the data that is at the same or less privilege level (not numerically)
- Instructions may load a data-segment register only if

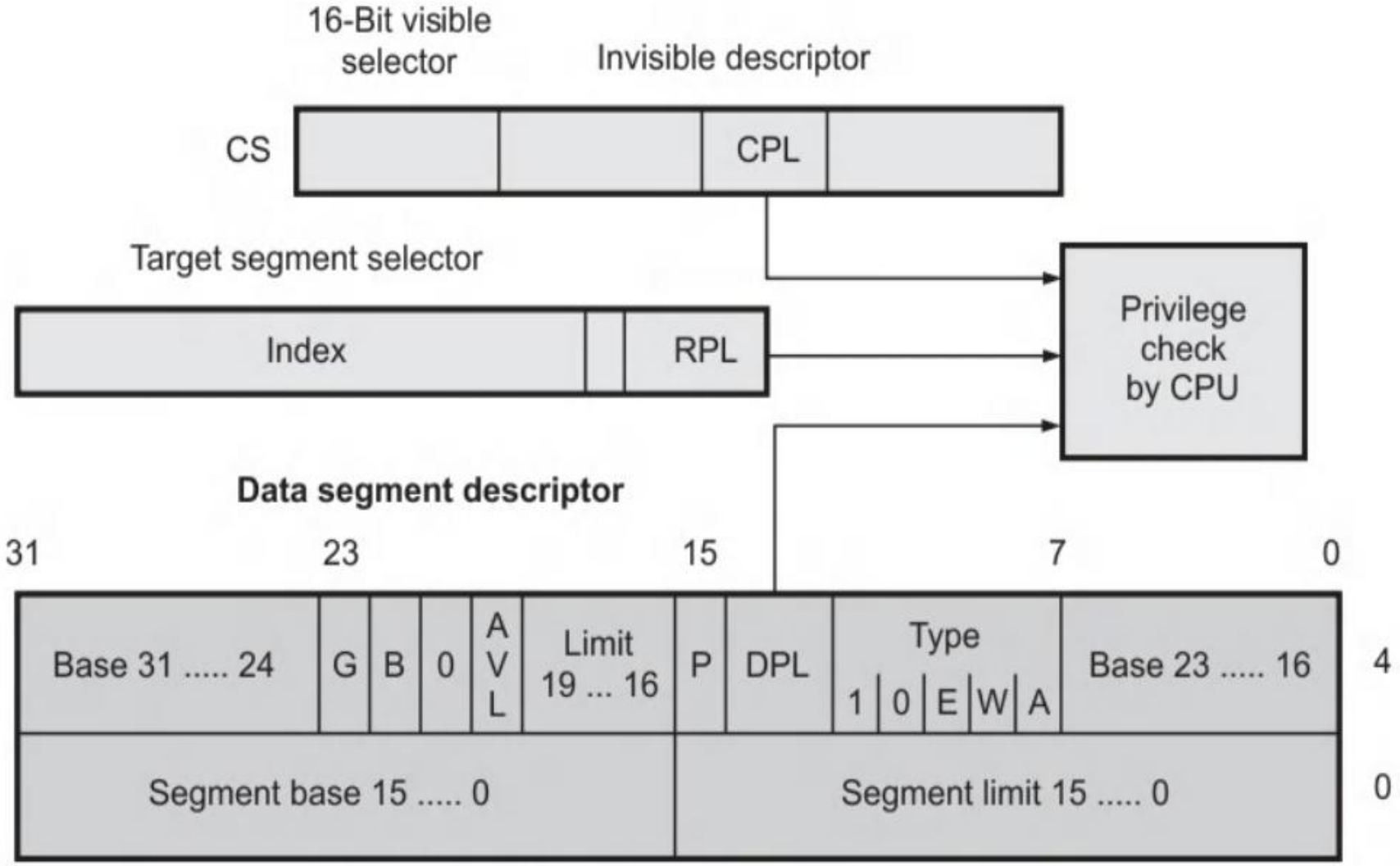
  DPL(data) ≥ max {CPL(proc),RPL} numerically

Data Access

DPL(data) ≥ max {CPL(proc),RPL}

No.	Privilege 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.Privilege check for data access



DPL(data) ≥ max {CPL(proc),RPL}

Accessing Data in code segments

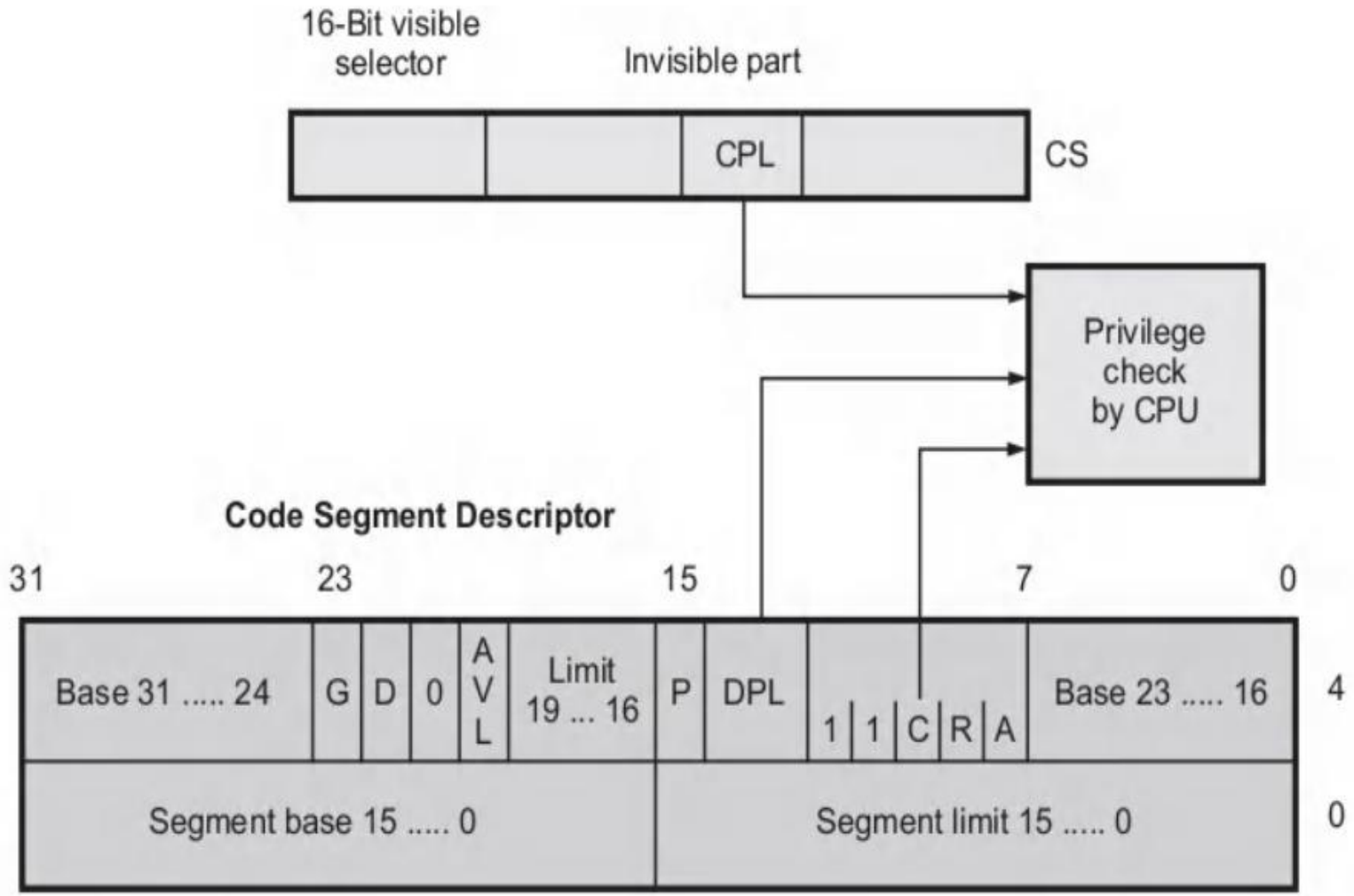
- 3 Ways of reading data from code segments

a) Load a data segment register with a selector of **non-conforming**, readable,executable segment.(Procedure can only acess data which is at **same or less privileged level)**

b) Load a data segment register with a selector of **conforming,** readable,executable segment.(Always **valid** because privilege level of segment whose **conforming bit is set**)

c) Use a CS override prefix to read a readable,executable segment whose selector is already loaded in CS register.(**valid** because **DPL of the code segment in CS is =CPL**)

Fig.Privilege check for control transfer



Restricting Control Transfer

- Transfer control instructions:JMP,CALL,RET,INT,IRET
- Transfer control instructions

1. NEAR -in same segment:JMP,CALL,RET do only limit checking
2. FAR-one segment to other segment:JMP,CALL,RET perform privilege checking

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

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.

Changing Privilege level

- Question:How to access segment with more privilege?
- (How user program accesss OS kernel,BIOS,utility procedures in segment which have more privilege)
- **Answer:**

2 solution

a) more privilege level segment must be conforming code segment(Without Call Gate Descriptor)

b) use call gate to access segment which have more privilege(complex)With Call Gate Descriptor

Conforming Code Segment

- An executable segment whose descriptor has the **conforming bit set**
- It permits **sharing** of procedures that may be called from various privilege levels but should **execute at the privilege level of the calling procedure.**
- Example: **math libraries**
- When control is transferred to a conforming segment, the CPL does not change

Nonconforming Code Segment

- Most code segments are not conforming
- For nonconforming segments, control can be transferred **without a gate only** to executable segments **at the same level of privilege**
- To transfer control to higher privilege levels(not numerical)-**for e.g. application want to use system service**- CALL instruction need to be used with **call-gate** descriptors
- **JMP** instruction **never** transfer control to a nonconforming segment whose **DPL ≠ CPL**.

Case 1: Accessing of conforming code segment

No.	Current Privilege Level (CPL)	DPL of conforming code segment	Access
1	3	2	Valid
2	2	0	Valid
3	1	1	Valid
4	1	2	Invalid
5	2	3	Invalid

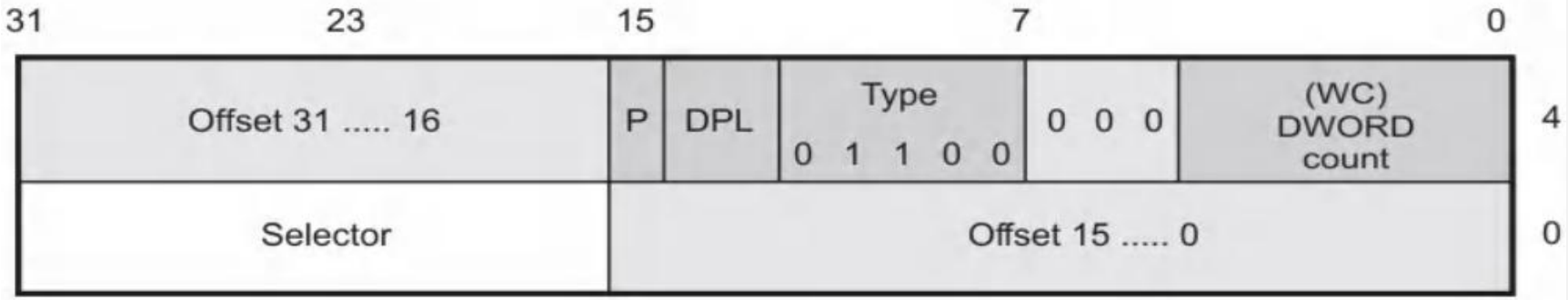
The DPL of the conforming descriptor must always be less than or equal to the current CPL.

You can never transfer control to a segment whose DPL is greater (less privileged) than the current segment.

conforming code segment must have higher or same privilege level than original segment to allow control to return back to the original segment.

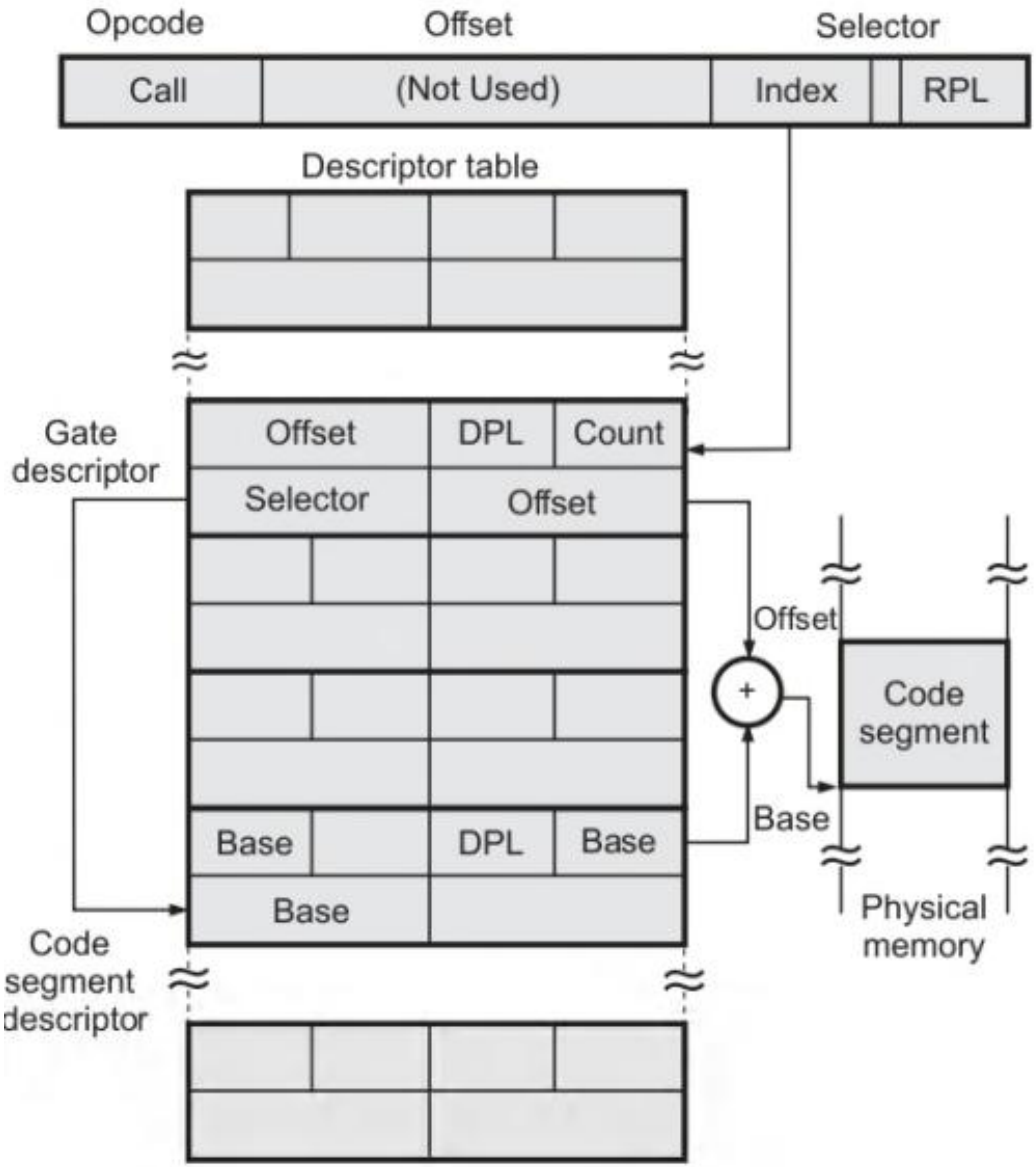
Case 2: Inter Privilege Level Transfer using Call Gates

- **Call Gates-do not define any memory space,**
- **no base addresss or limit fields**
- **Interface layer between code segments at different privilege level.**
- **Mechanism to call procedure located at higher level segments.**
- **JMPs are not allowed .Hence name call gate.**
- **Call gate descriptor is put in GDT or LDT.**



Format of 80386 call gate

Indirect transfer via call gate



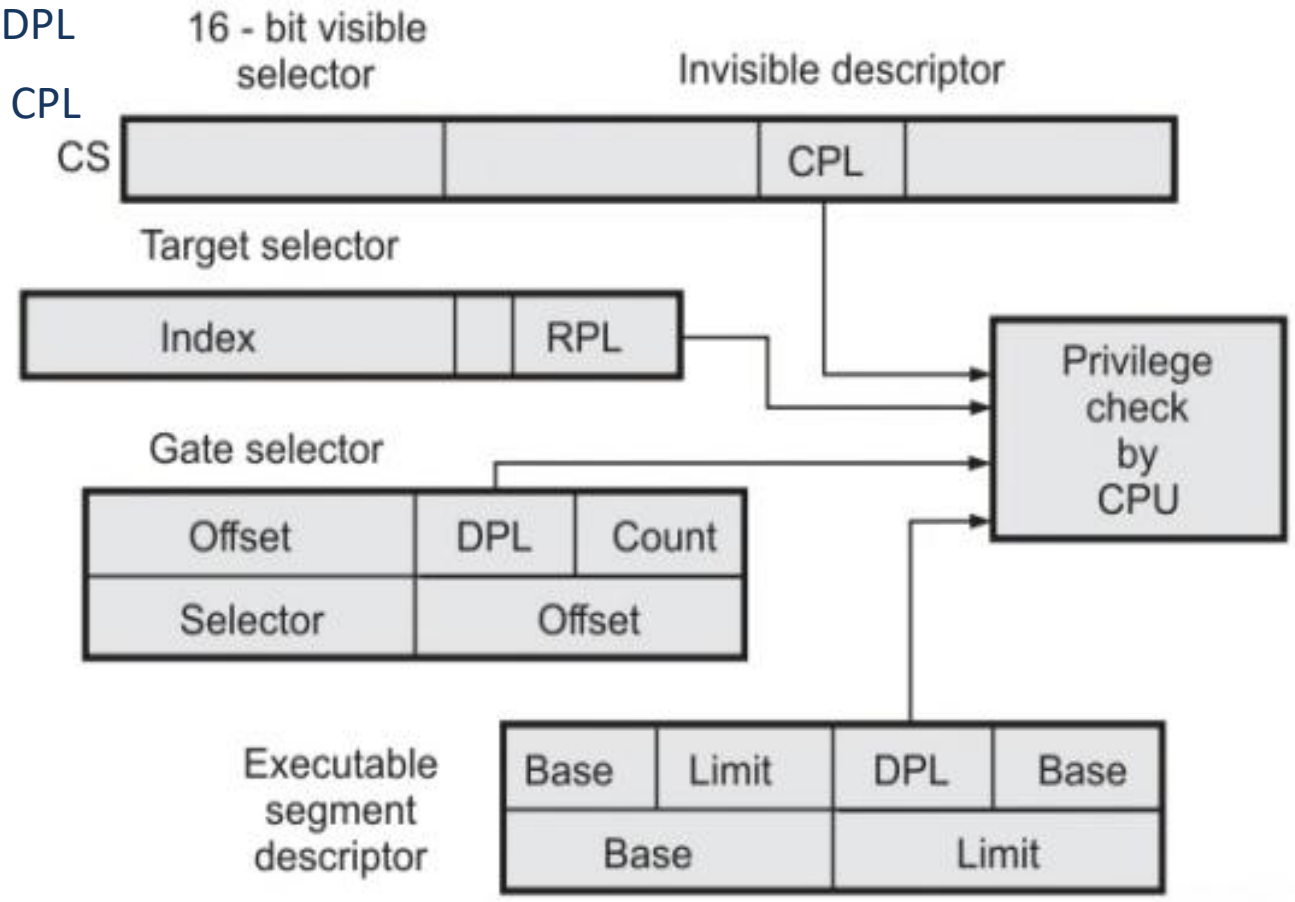
- The call gate descriptor contains two important things :
  1. Selector which points to the descriptor for the segment where the procedure is actually loaded.
  2. Offset of the called procedure in its segment.
- During this process the validity of control transfer is checked using four different privilege levels :
  1. The CPL (Current Privilege Level).
  2. The RPL (Requester'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.
- For valid control transfer, the transfer must satisfy the following privilege rules for CALL instruction

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

For a JMP instruction

MAX(CPL,RPL) ≤ gate DPL

target segment DPL = CPL

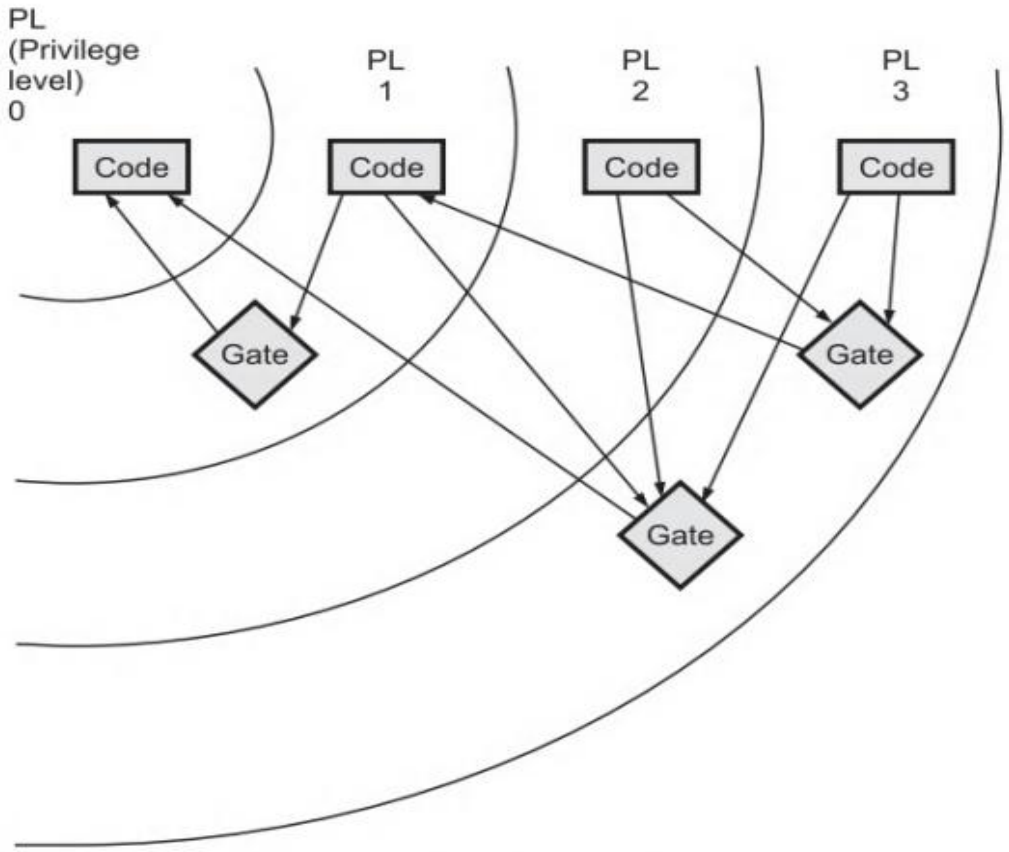


Privilege check via call gate

For a CALL instruction

MAX(CPL,RPL) ≤ gate DPL

target segment DPL ≤ CPL



Some valid accesses to higher privileged levels using call gates

For example, if you are running in a PL2 code segment (CPL=2), and you want to call a PL0 procedure (target DPL=0), you must use a gate to that procedure with a DPL of 2 or 3.

- DPL(Call Gate)>=Current privilege level
- DPL(Call Gate)>=RPL of gate selector
- DPL(Call Gate)>=target code segment DPL
- Target code segment DPL<=Current privilege level
- For a JMP instruction, the privilege rules are

  MAX(CPL,RPL) ≤ gate DPL

  target segment DPL = CPL(numerically)
- For a CALL instruction, the rules are

  MAX(CPL,RPL) ≤ gate DPL

  target segment DPL ≤ CPL(numerically)

Stack Switching

- The change in privilege level changes the address domain of the program. The 80386 also changes stacks in case of change in privilege level.
- When call gate causes a change in privilege, stack segment and pointer are saved, and a new stack is used that corresponds to the new, inner privilege level.
- When controls returned to outer level code, the use of the original stack is restored.
- If there is a valid call through gate, the 80386 uses a new stack. It takes segment selector and the pointer for this stack from the TSS. If user is calling procedure with privilege level 1 (PL1), the new stack selector and stack pointer are taken from SS1 and ESP1, respectively.
- The old stack selector and stack pointer are immediately pushed onto this new stack. Then 80386 finds the number of double word (32-bit) entries to be pushed from old stack to new stack from WC (Word Count) field from the call gate descriptor. This means that WC field decides number of passing parameters to the new stack. After this, old CS selector and EIP offset are pushed onto the new stack.
- Finally, CS is loaded from the selector field of the call gate descriptor, EIP is loaded from the offset field, and execution starts at the new address.

Page Level Protection

Page level protection involves two kinds of protections

1. Restriction of addressable domain
2. Type checking

The U/S and R/W fields of PDEs and PTEs are used to control access to pages.

Restriction of 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

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 Read/write access,

whereas at user level page access depends on R/$\overline{W}$ bi 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 readable and writeable.

When 80386DX is executing at user level, it cannot access page be supervisor level.

Combining segment & page level protection

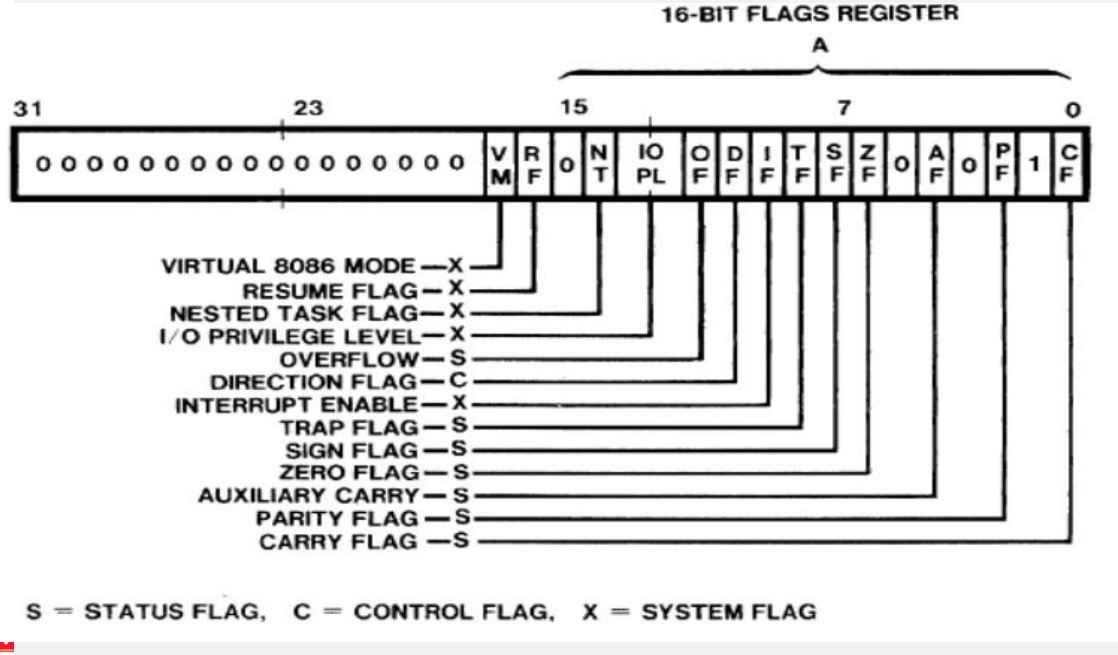
- When paging is enabled, the 80386 first evaluates segment protection, then evaluates page protection.
- If the processor detects a protection violation at either the segment or the page level, the requested operation cannot proceed; a protection exception occurs instead.
- It is possible to define 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).

I/O Protection

- 2 mechanism fpr protecting I/O ports in protected mode.

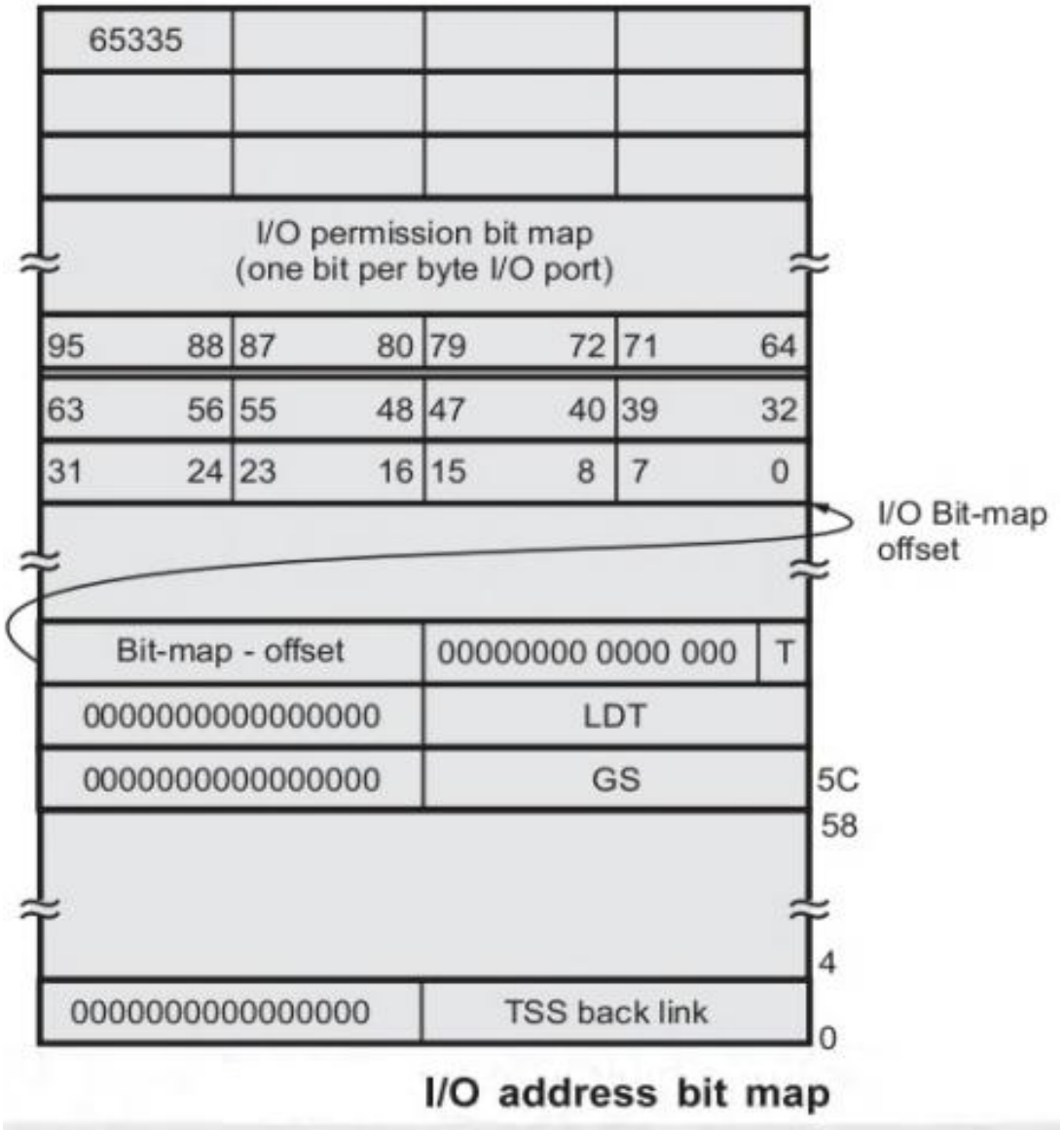
1) **IOPL field**(I/O Privilege Level) in EFLAG register-define right use of I/O instructios

2) **I/O permission bit map** of TSS Segment-define right use of ports in I/O address space



(I/O Privilege Level)	I/O permission bit map
For instructions(sensitive instruction) IN,INS,OUT,OUTS,CLI,STI **CPL<=IOPL**	Alow port to be accessed by specific task. I/O permission bit map is a bit vector. To access port, Bit in I/O bitmap=0
If CPL>=IOPL,General protection exception	When program attemps to access port, 80386 compare CPL with IOPL. CPL<=IOPL, and then make Bit in I/O bitmap=0
Each task have uniques flag register. Different task have different IOPL	Size & location of bit map is different in TSS. It is variable.
Only procedure exicuting at PL0 can change IOPL, otherwise IOPL unchanged.	16 bit ports use 2 bit each& 32 bit uses 4 bit each.

Fig-I/O address bit map



Privileged & I/O sensitive instruction(19 instructions)

Privileged instruction	IOPL sensitive instruction
Instruction affect system data structure	**IOPL in EFLAG defines right use of I/O instructions**
Instruction executed when CPL=0,Else general protection exception.	CPL<=IOPL

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

IOPL - sensitive instruct

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

Privileged instructions