UNIT 3
Protection and Multitasking

Why Protection?

- The purpose of the protection features of the 80386 is **to help detect and identify bugs (Unauthorized accesses).**

- 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.**

Overview of 80386DX Protection Mechanisms

1. Type checking
2. Limit checking
3. Restriction of addressable domain
4. Restriction of procedure entry points
5. Restriction of instruction set
   - The concept of "**Privilege**" is central to several aspects of protection (numbers 3, 4, and 5 in the above list).

Segment-Level Protection

- All five aspects of protection apply to segment translation:

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

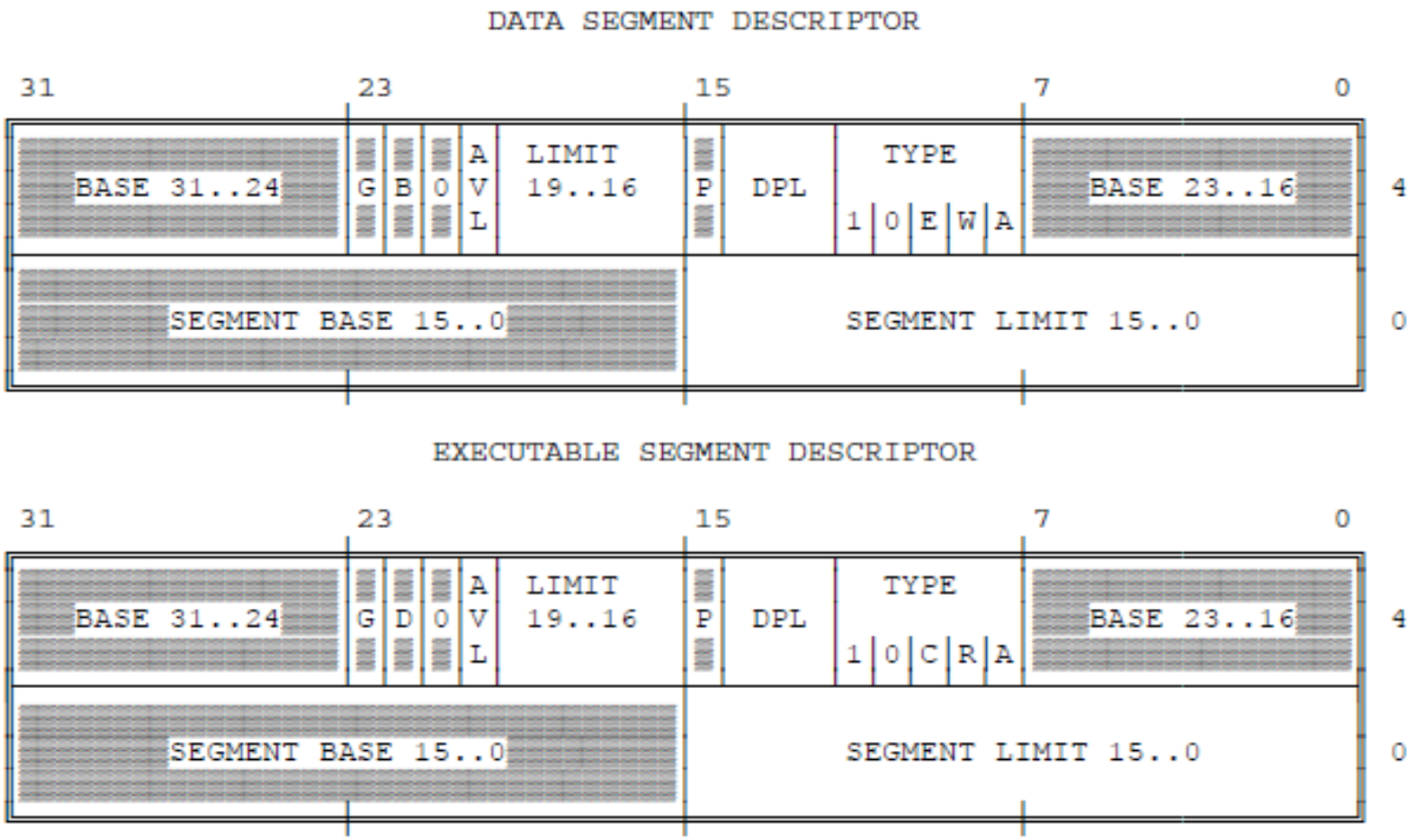
Descriptors Store Protection Parameters

- The **protection parameters are placed in the descriptor by systems software** at the time a descriptor is created.

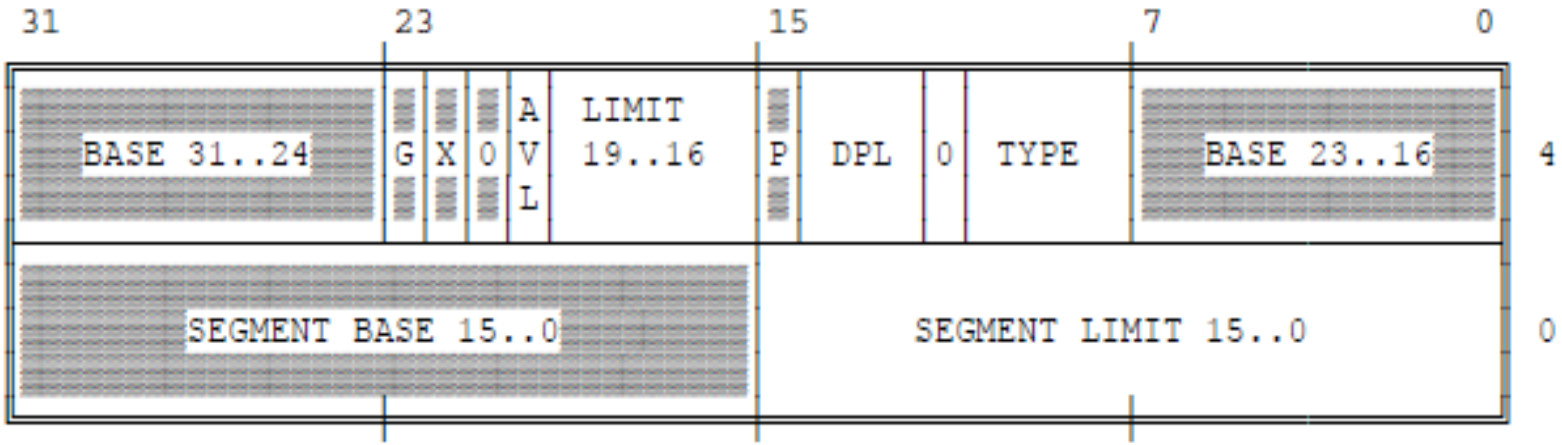
- When a program loads a selector from a segment register, **the processor loads not only the address of the segment but also protection information**.

- **Each segment register has bits in the invisible portion for storing base, limit, type, and privilege level;** therefore, subsequent protection checks on the **same segment do not consume additional clock cycles.**

- Clear regions in Figure highlights the protection-related fields of segment descriptors.



SYSTEM SEGMENT DESCRIPTOR



A - ACCESSED
AVL - AVAILABLE FOR PROGRAMMERS USE
B - BIG
C - CONFORMING
D - DEFAULT
DPL - DESCRIPTOR PRIVILEGE LEVEL
E - EXPAND-DOWN
G - GRANULARITY
P - SEGMENT PRESENT
R - READABLE
W - WRITABLE

1. Type Checking

- The TYPE field of a descriptor has two functions:

1. **It distinguishes among different descriptor formats.** (system/non-system).

2. **It specifies the intended usage of a segment.**

   eg. If the segment is read only segment then its accessed is limited to only reading purpose.

Table 6-1. System and Gate Descriptor Types

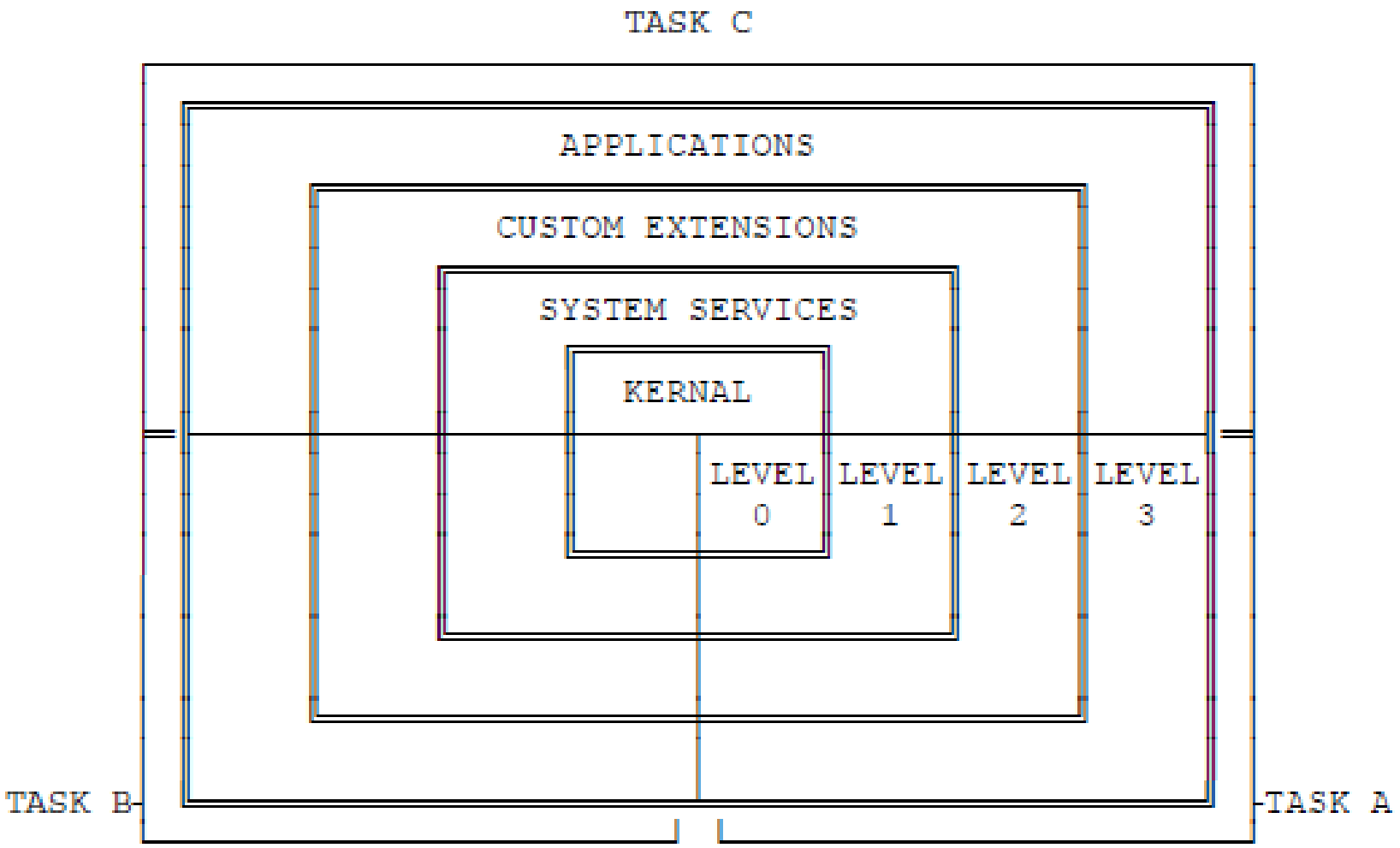
Code	Type of Segment or Gate
0	-reserved
1	Available 286 TSS
2	LDT
3	Busy 286 TSS
4	Call Gate
5	Task Gate
6	286 Interrupt Gate
7	286 Trap Gate
8	-reserved
9	Available 386 TSS
A	-reserved
B	Busy 386 TSS
C	386 Call Gate
D	-reserved
E	386 Interrupt Gate
F	386 Trap Gate

2. Limit Checking

- **The limit field of a segment descriptor is used by the processor to prevent programs from addressing outside the segment.**

- The processor's interpretation of the limit depends on the setting of the **G (granularity) bit.**

3. Privilege Levels



Protection Check

1. It checks, If the descriptor table index by the selector contain a **valid descriptor for that selector.**

2. It also checks to see **if the segment descriptor is of the right type to be loaded into specified segment register cache.**

3. It checks the **P** bit of the access byte.

4. Further checks are made each time a location in the **actual segment is accessed.**

- **The rules regarding the stack segment are slightly different than those involving data segments.**

- Instructions that load selectors into SS, **must refer to data segment descriptors for read write permission**.

- And also, the DPL and RPL must equal the CPL.

Terminology

- **Privilege Level (PL):**
  - One of the four hierarchical privilege levels.
  - Level 0 is the most privileged level.
  - Level 3 is the least privileged level.
- **Requestor Privilege Level (RPL):**
  - The privilege level of the **original supplier of the selector.**
  - RPL is determined by the least two significant bits of a selector.

- **Current Privilege Level (CPL):**
  - The privilege level at which a task is currently executing, which equals the **privilege level of the code segment being executed.**
  - CPL can also be determined by examining the lowest 2 bits of the CS register, except for conforming code segments.

- **Descriptor Privilege Level (DPL):**
  - The DPL of the descriptor of the target segment.

- **Effective Privilege Level (EPL):**
  - The effective privilege level is the **least privileged of the RPL and DPL.** EPL is the numerical maximum of RPL and DPL.

- **I/O Privilege Level (IOPL):**

  - Defines the **least privileged level at which I/O instructions can be unconditionally performed.**
  - I/O instructions can be unconditionally performed when CPL <= IOPL.
  - IOPL-sensitive instructions- **CLI and STI.**
  - IF bit can be changed by loading a new value into the EFLAGS register.

Rules of Privilege

Data stored in a segment with privilege level **P** can be accessed only by code executing at a privilege level at least as privileged as **P**.

A code segment/procedure with privilege level **P** can only be called by a task executing at the same or a lesser privilege level than **P**.

Table 4-3. Descriptor Types Used for Control Transfer

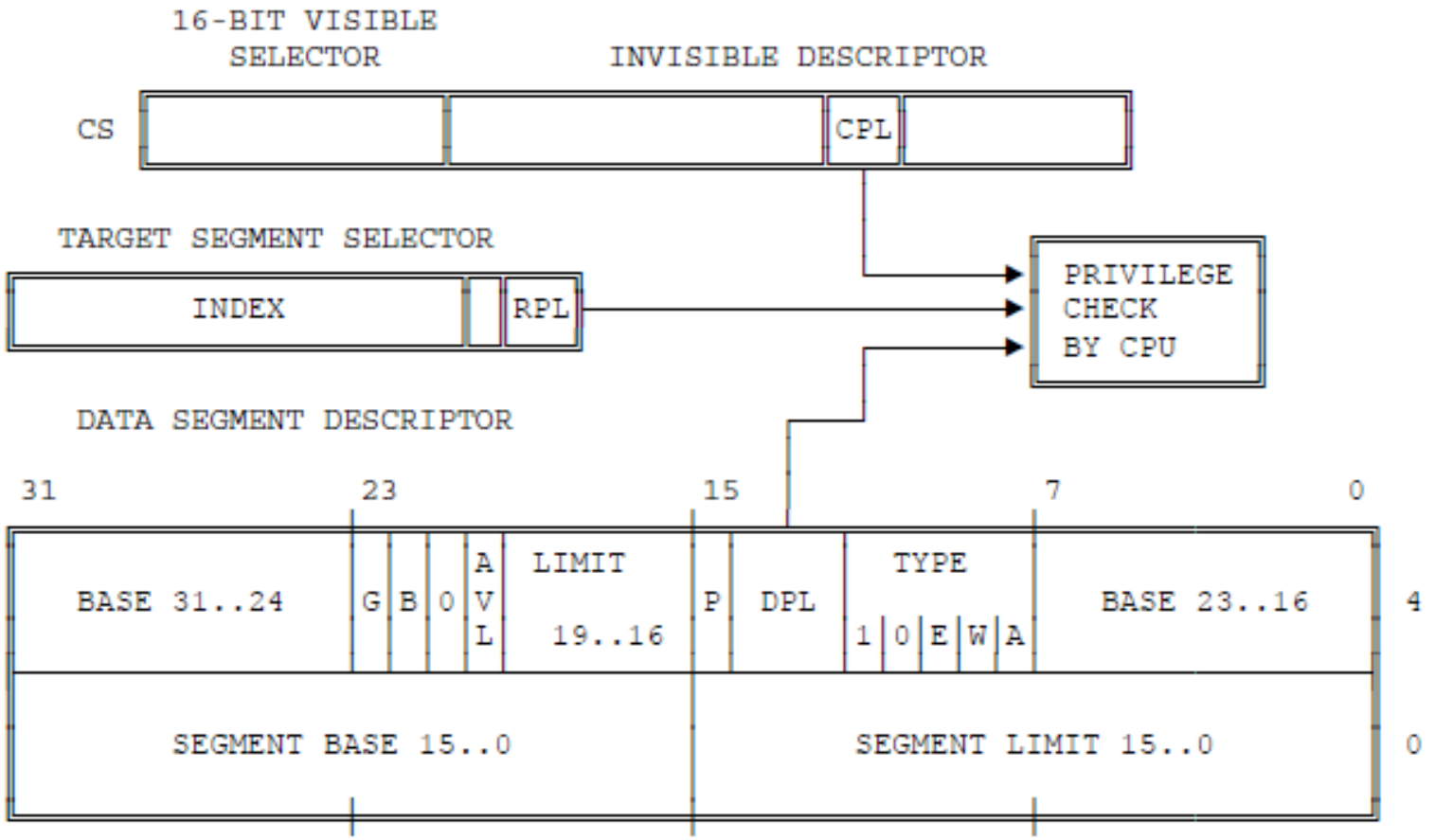
Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag register) = 0
**NT (Nested Task bit of flag register) = 1

Privilege Check for Data Access

- 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

- The addressable domain of a task varies as CPL changes.
- When CPL is zero, data segments at all privilege levels are accessible;
- when CPL is one, only data segments at privilege levels one through three are accessible;
- when CPL is three, only data segments at privilege level three are accessible.



CPL - CURRENT PRIVILEGE LEVEL
RPL - REQUESTOR'S PRIVILEGE LEVEL
DPL - DESCRIPTOR PRIVILEGE LEVEL

- A procedure can only access the data that is at the same or less privilege level (not numerically).

Control Transfer

- The far JMP and CALL can be done in 2 ways:

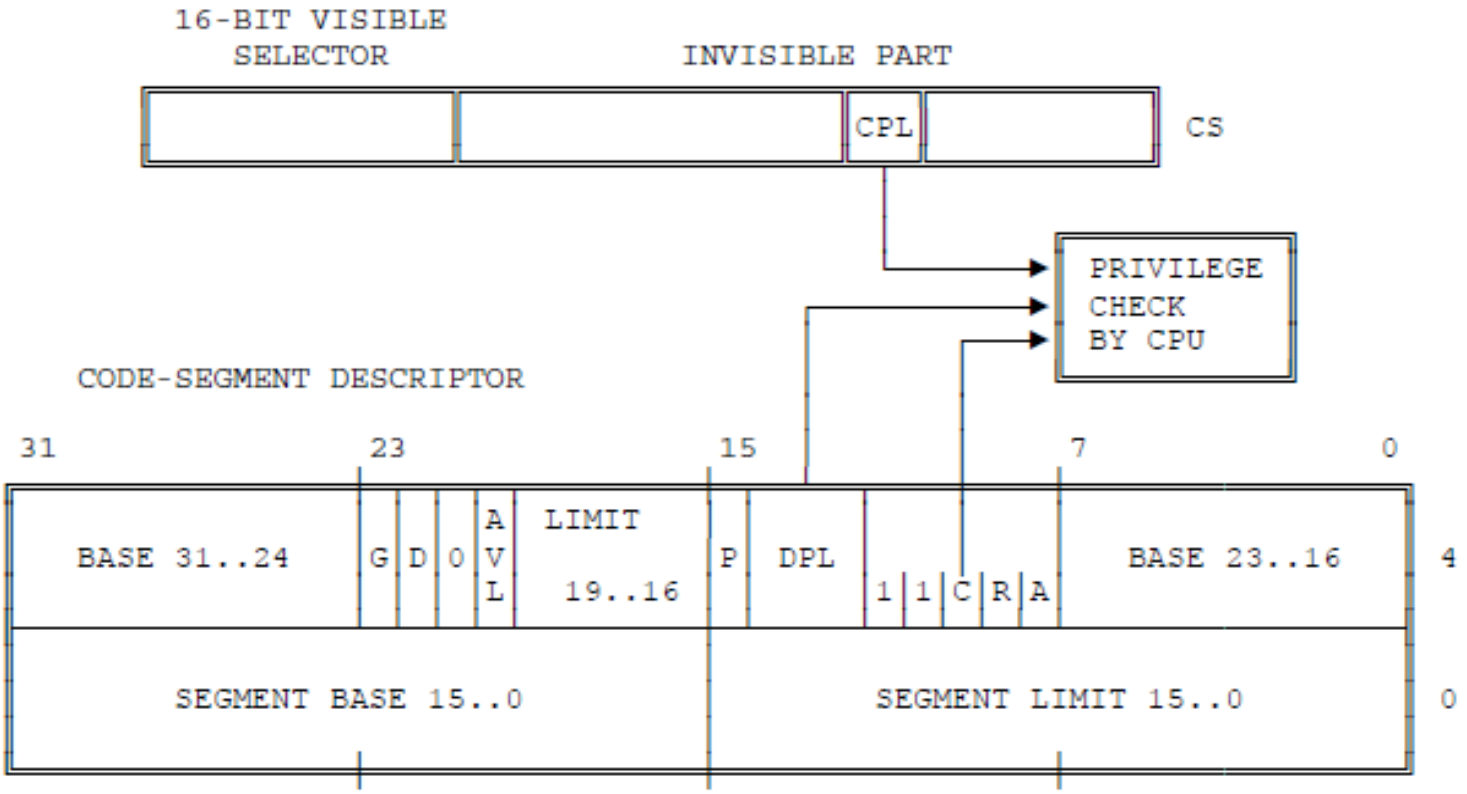
1. Without Call Gate Descriptor

2. With Call Gate Descriptor

Without Call Gate

- The processor permits a JMP or CALL directly to another segment only if
  1. DPL of the target segment = CPL of the calling segment
  2. Confirming bit of the target code is set and DPL of the target segment ≤ CPL
- **Confirming Segment:** These segments may be called from various privilege levels but execute at the privilege level of the calling procedure. (e.g. math library, system calls).

Privilege Check for Control Transfer without Gate



CPL - CURRENT PRIVILEGE LEVEL
DPL - DESCRIPTOR PRIVILEGE LEVEL
C - CONFORMING BIT

With Call Gate

- The FAR pointer of the control transfer instruction uses the selector part of the pointer and selects a gate.

- The selector and offset fields of a gate form a pointer to the entry of a procedure.

Privilege level transitions can only occur via gates.

- **JMPs** can be made to a non-conforming code segment with the **same privilege or to a conforming code segment with greater or equal privilege.**

- **CALLs** can be made to a **non-conforming code segment with the same privilege or via a gate to a more privileged level.**

In case of Interrupts:

- Interrupts handled within the task obey the same privilege rules as CALLs.

- **Conforming Code** segments are accessible by privilege levels which are the same or less privileged than the conforming-code segment's DPL.

- The code segment selected in the gate must be the same or more privileged than the task's CPL.

- **Task switches can be performed by a CALL, JMP or INT** which references either a task gate or task state segment who's DPL is less privileged or the same privilege as the old task's CPL.

Call Gates

- One of the major uses of gates is to **provide a secure method of privilege transfers within a task.**

- Gates can be accessed by a task if the

  EPL <= gate descriptor's DPL.

- Call Gates are **accessed via a CALL instruction.**

Inter-level call gate

1. Load CS:EIP from gate check for validity.
2. SS is pushed zero-extended to 32 bits.
3. ESP is pushed.
4. Copy Word Count 32-bit parameters from the old stack to the new stack.
5. Push Return address on stack.

Accessing Data in Code Segments

- The following methods of accessing data in code segments are possible:

1. Load a data-segment register with a selector of a nonconforming, readable, executable segment.

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

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

Restricting Control Transfers

- With the 80386, 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.

- The operands of the "**Far**" forms of JMP and CALL refer to other segments; therefore, the processor performs privilege checking. There are two ways a JMP or CALL can refer to another segment:

1. The operand selects the descriptor of another executable segment.

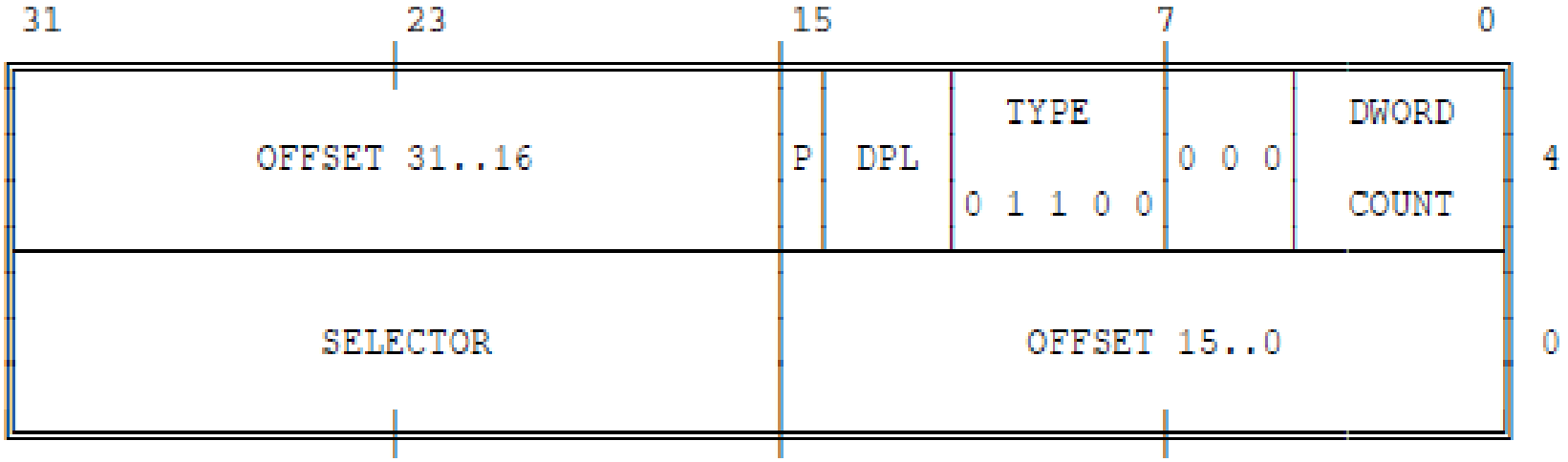
2. The operand selects a call gate descriptor.

Gate Descriptors Guard Procedure Entry Points

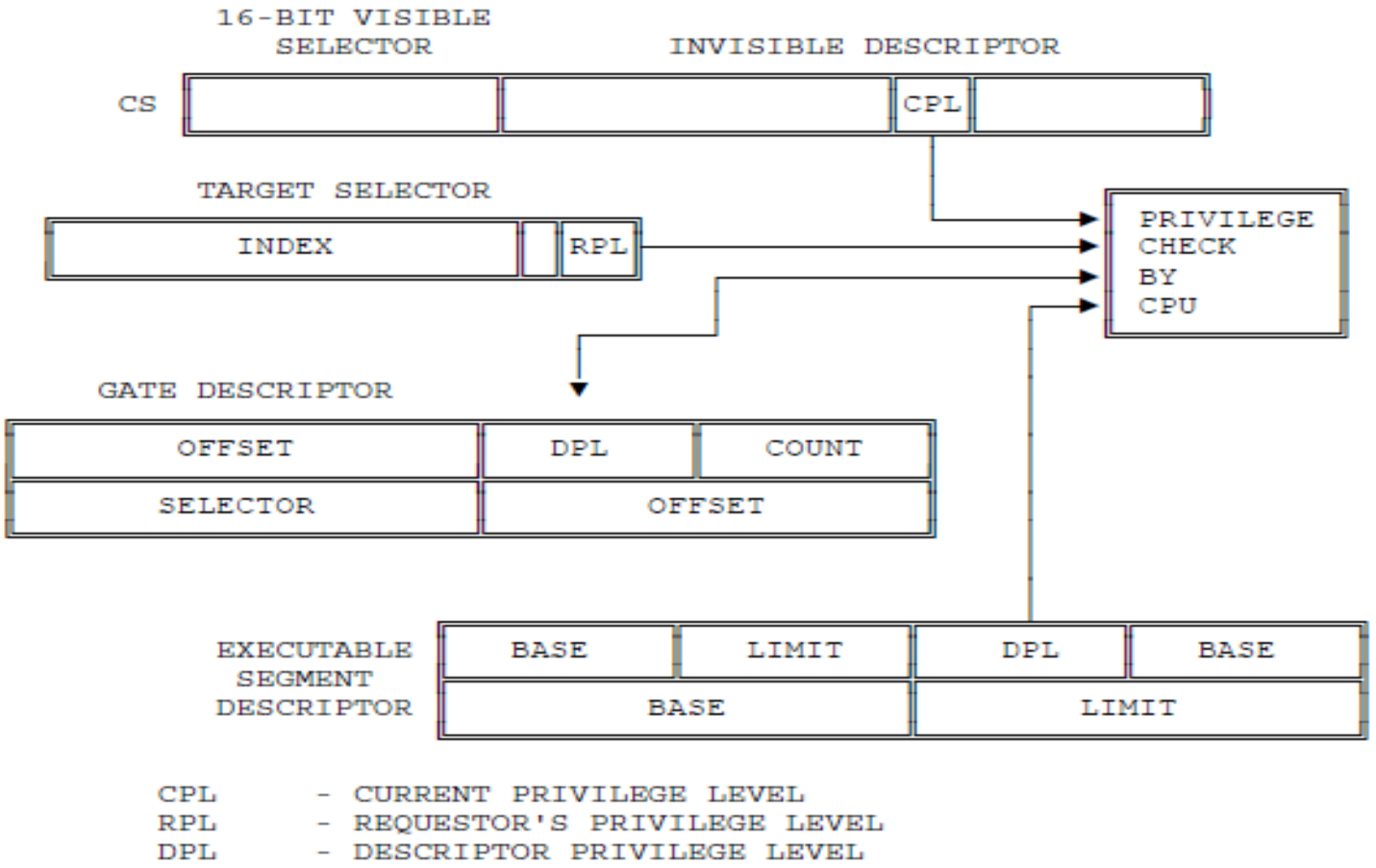
- To provide protection for control transfers among executable segments at different privilege levels, the 80386 uses gate descriptors. There are four kinds of gate descriptors:

  - Call gates
  - Trap gates
  - Interrupt gates
  - Task gates

Format of 80386 Call Gate



Privilege Check via Call Gate



- Gates can be used for control transfers to numerically smaller privilege levels or to the same privilege level (though they are not necessary for transfers to the same level).

- Only CALL instructions can use gates to transfer to smaller privilege levels.

- A gate may be used by a JMP instruction only to transfer to an executable segment with the same privilege level or to a conforming segment.

Page-Level Protection

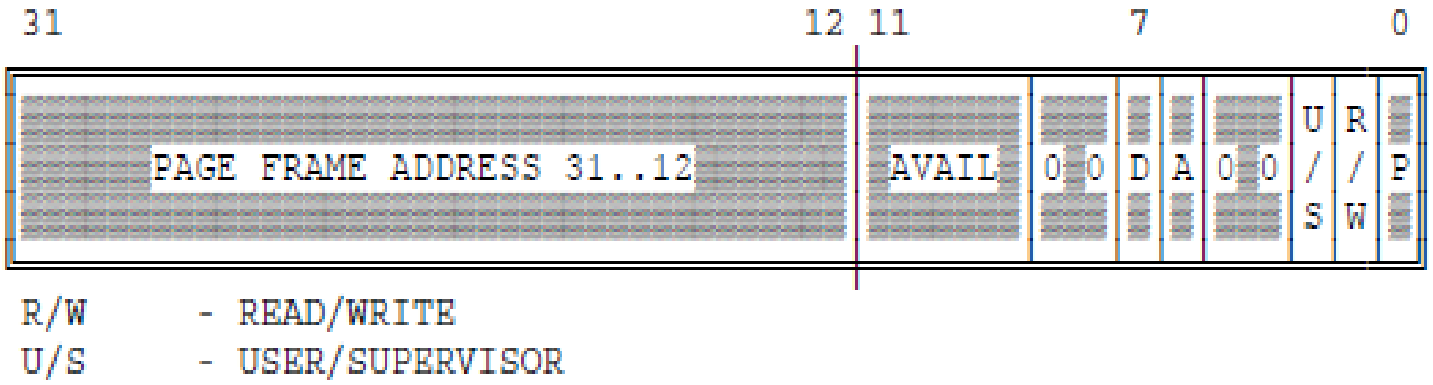
- Two kinds of protection are related to pages:

1. Restriction of addressable domain.

2. Type checking.

Page-Table Entries Hold Protection Parameters

- Figure highlights the fields of PDEs and PTEs that control access to pages.



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.

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 Page and Segment 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.

Combining Directory and Page Protection

Page Directory Entry		Page Table Entry		Combined Protection	
U/S	R/W	U/S	R/W	U/S	R/W
S-0	R-0	S-0	R-0	S	x
S-0	R-0	S-0	W-1	S	x
S-0	R-0	U-1	R-0	S	x
S-0	R-0	U-1	W-1	S	x
S-0	W-1	S-0	R-0	S	x
S-0	W-1	S-0	W-1	S	x
S-0	W-1	U-1	R-0	S	x
S-0	W-1	U-1	W-1	S	x
U-1	R-0	S-0	R-0	S	x
U-1	R-0	S-0	W-1	S	x
U-1	R-0	U-1	R-0	U	R
U-1	R-0	U-1	W-1	U	R
U-1	W-1	S-0	R-0	S	x
U-1	W-1	S-0	W-1	S	x
U-1	W-1	U-1	R-0	U	R
U-1	W-1	U-1	W-1	U	W

NOTE
S — Supervisor
R — Read only
U — User
W — Read and Write
x indicates that when the combined U/S attribute is S, the R/W attribute is not checked.

Multitasking

- To provide efficient, protected multitasking, the 80386 employs several special data structures.

  - **Task State Segment**
  - **Task State Segment Descriptor**
  - **Task Register**
  - **Task Gate Descriptor**

- With these structures the 80386 can rapidly switch execution from one task to another, saving the context of the original task so that the task can be restarted later.

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

1. Interrupts and exceptions can cause task switches. (To call Interrupt and return from interrupt)
2. With each switch to another task, the 80386 can also switch to another LDT and to another Page Directory.

Task State Segment

- All the information which is needed to manage a task, is stored in a special type of segment, a **Task State Segment (TSS).**

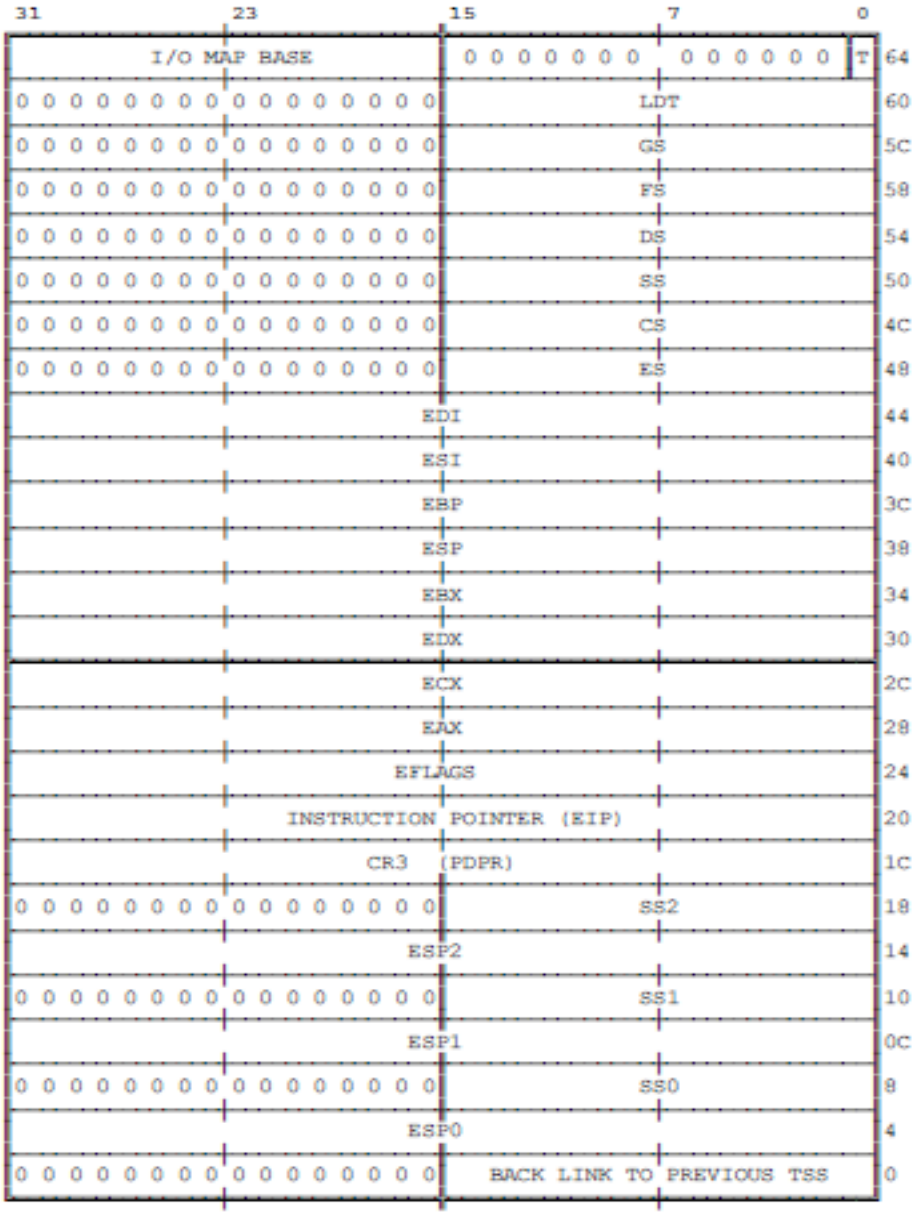
- The fields of a TSS belong to two classes:
  1. Dynamic Set
  2. Static Set

1. **A Dynamic Set**:- The processor updates this set with each switch from the task. This set includes the fields that store:

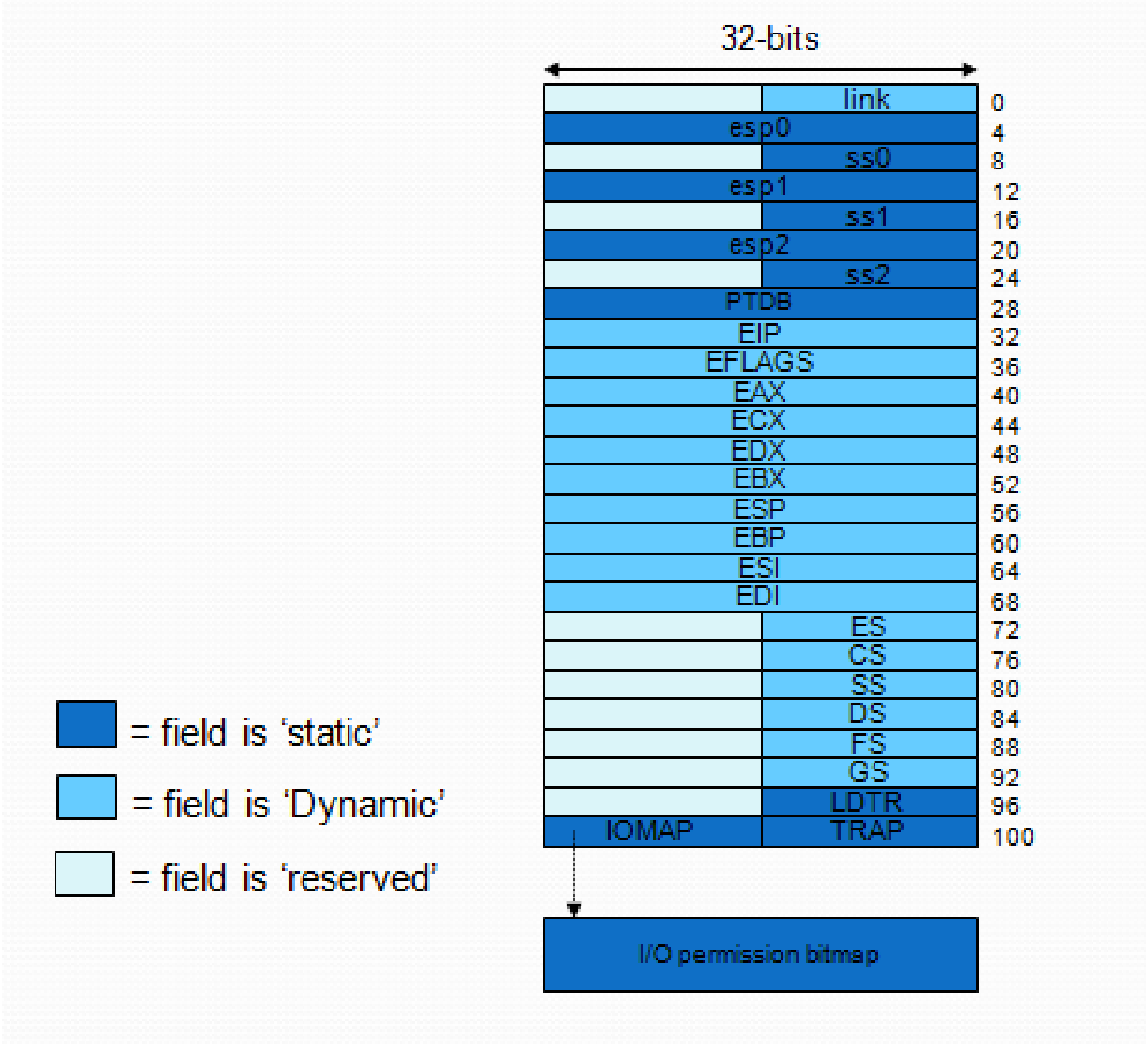
   - **The general registers** (EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI).
   - **The segment registers** (ES, CS, SS, DS, FS, GS).
   - **The flags register** (EFLAGS).
   - **The instruction pointer** (EIP).
   - **The selector of the TSS of the previously executing task** (updated only when a return is expected).

2. **A Static Set:-** The processor reads this set but does not change. This set includes the fields that store:

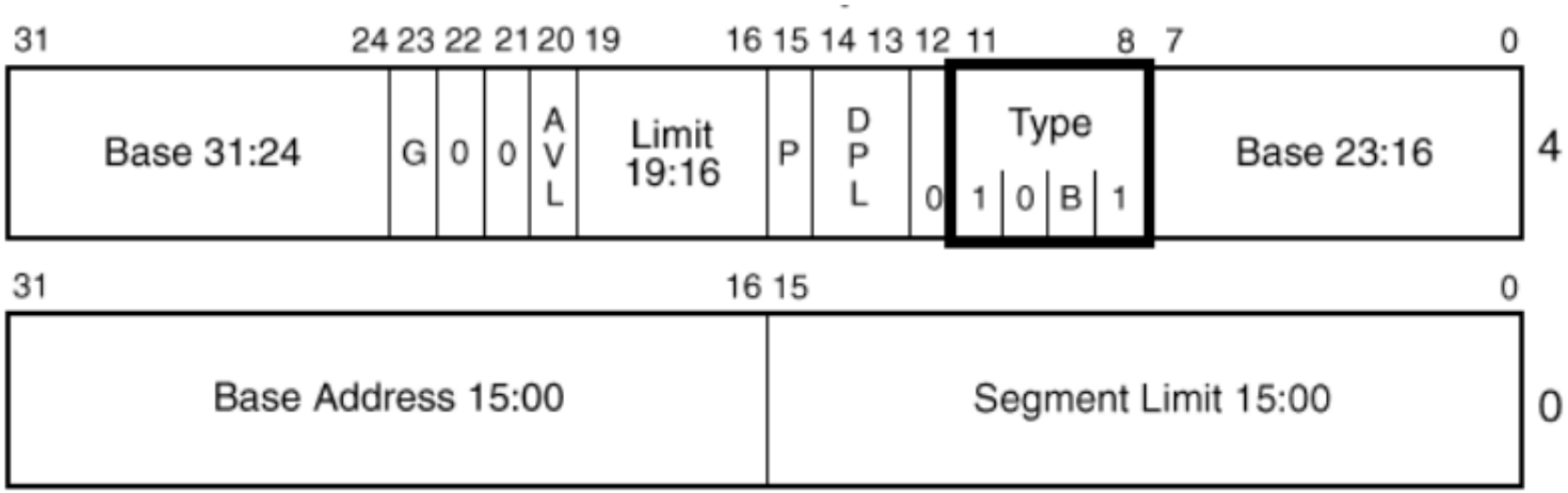
   - The **selector of the task's LDT.**
   - The register **(PDBR)** that contains the base address of the task‘s page directory (read only when paging is enabled).
   - Pointers to the stacks for privilege levels 0-2.
   - The **T-bit (debug trap bit)** which causes the processor to raise a debug exception when a task switch occurs.
   - The I/O map base.


31
23
15
7
0
I/O MAP BASE
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
T
64
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LDT
60
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GS
5C
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FS
58
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DS
54
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SS
50
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CS
4C
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ES
48
EDI
44
ESI
40
EBP
3C
ESP
38
EBX
34
EDX
30
ECX
2C
EAX
28
EFLAGS
24
INSTRUCTION POINTER (EIP)
20
CR3 (PDPR)
1C
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SS2
18
ESP2
14
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SS1
10
ESP1
0C
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SS0
8
ESP0
4
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BACK LINK TO PREVIOUS TSS
0

TSS


32-bits
link
0
esp0
4
ss0
8
esp1
12
ss1
16
esp2
20
ss2
24
PTDB
28
EIP
32
EFLAGS
36
EAX
40
ECX
44
EDX
48
EBX
52
ESP
56
EBP
60
ESI
64
EDI
68
ES
72
CS
76
SS
80
DS
84
FS
88
GS
92
LDTR
96
IOMAP
TRAP
100
I/O permission bitmap
= field is 'static'
= field is 'Dynamic'
= field is 'reserved'

TSS Descriptor



AVL	Available for use by system software
B	Busy flag
BASE	Segment Base Address
DPL	Descriptor Privilege Level
G	Granularity
LIMIT	Segment Limit
P	Segment Present
TYPE	Segment Type

Task Register

- The **Task Register (TR)** identifies the currently executing task by pointing to the TSS.

- The task register has both a **"visible"** portion (i.e., can be read and changed by instructions) and an **"invisible"** portion (maintained by the processor to correspond to the visible portion; cannot be read by any instruction).

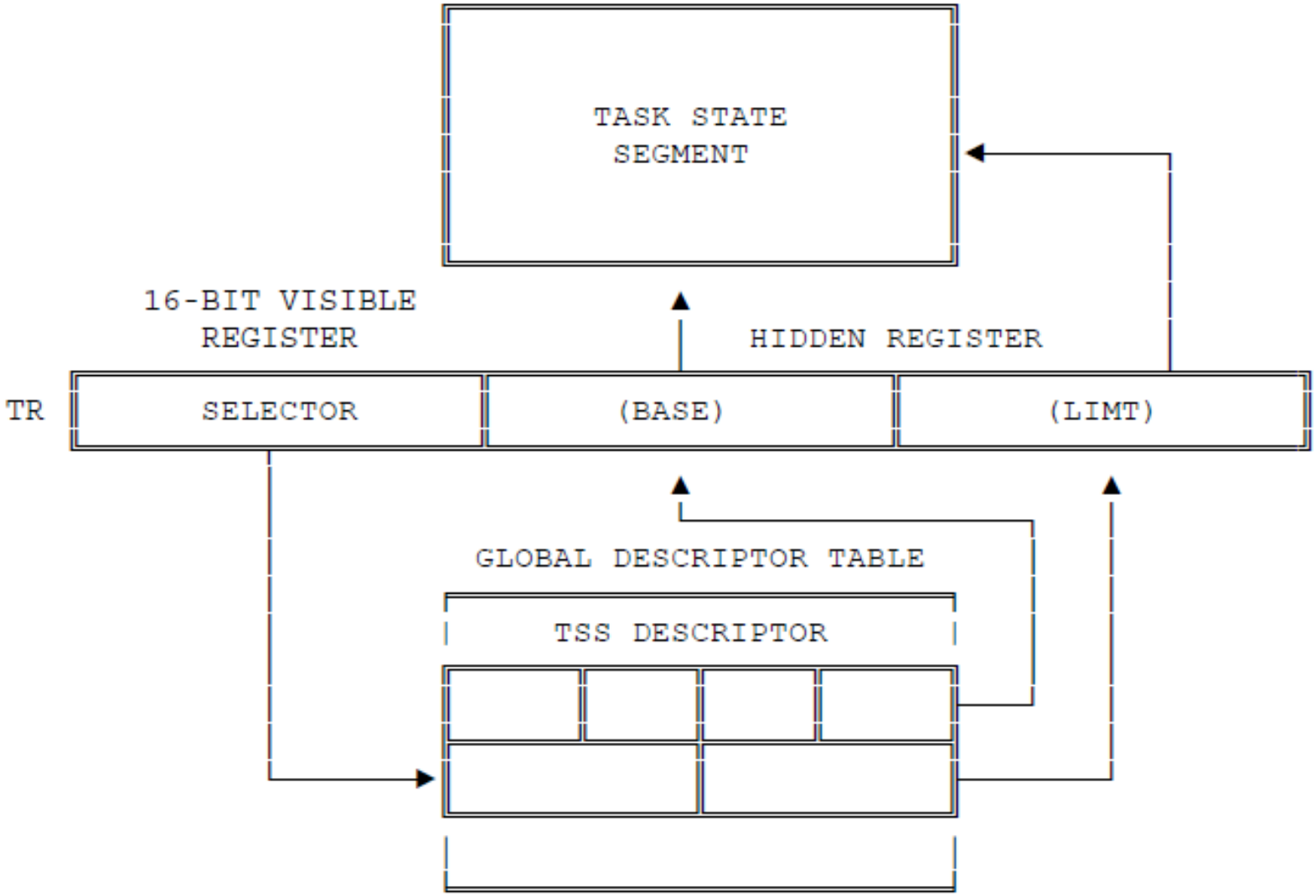
- The **selector in the visible portion selects a TSS descriptor in the GDT.**

- The processor **uses the invisible portion to cache the base and limit values from the TSS descriptor.**

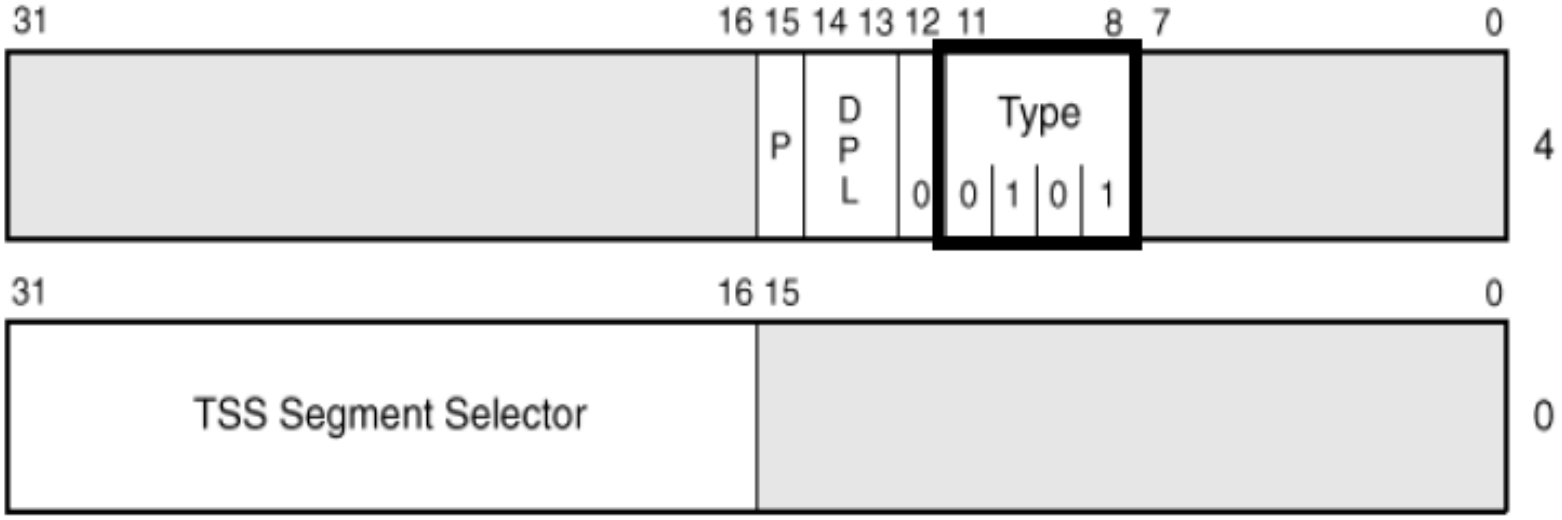
- **Holding the base and limit in a register makes execution of the task more efficient,** because the processor does not need to repeatedly fetch these values from memory when it references the TSS of the current task.

- The instructions **LTR** and **STR** are used to modify and read the visible portion of the task register.

- Both instructions take one operand, a 16-bit selector located in memory or in a general register.


TASK STATE
SEGMENT
16-BIT VISIBLE
REGISTER
HIDDEN REGISTER
TR
SELECTOR
(BASE)
(LIMT)
GLOBAL DESCRIPTOR TABLE
TSS DESCRIPTOR

Task Gate Descriptor



DPL Descriptor Privilege Level
P Segment Present
TYPE Segment Type

Reserved

- A **Task Gate Descriptor** provides **an indirect, protected reference to a TSS.**

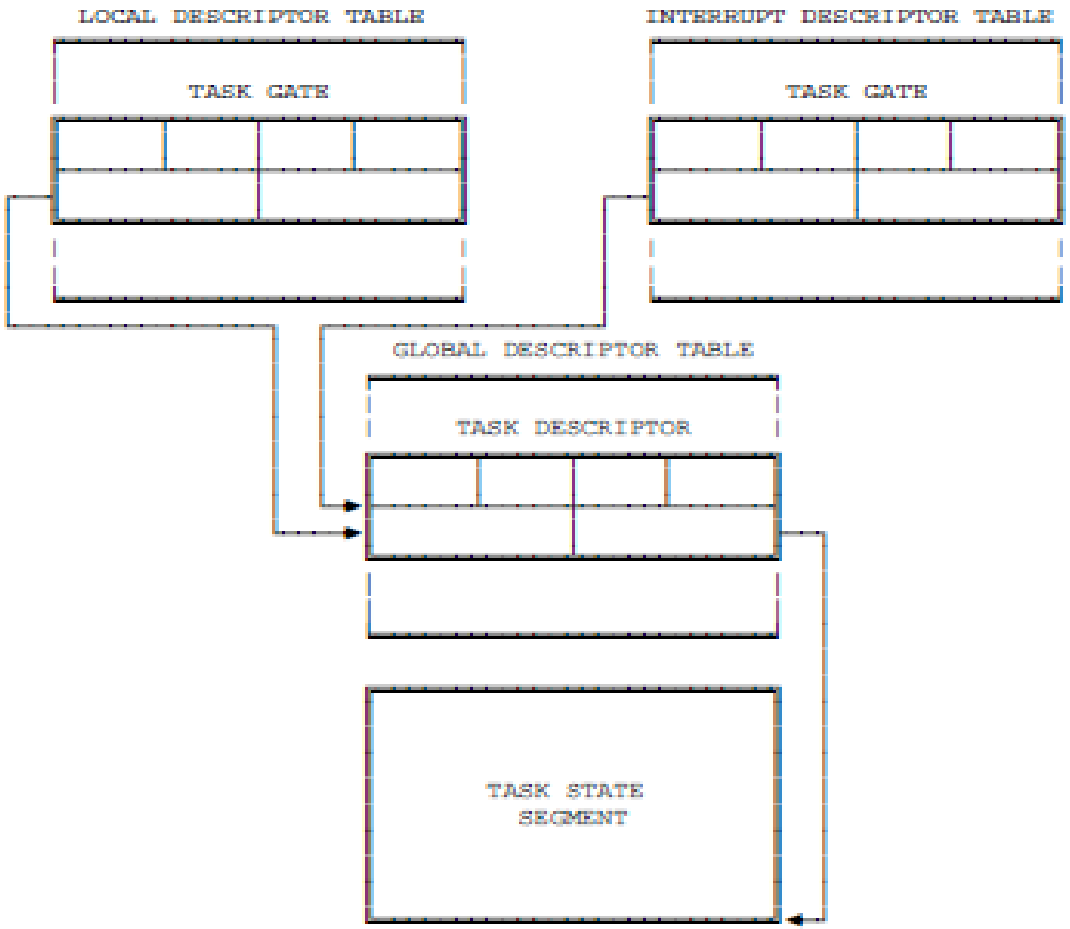
- The SELECTOR field of a task gate must refer to a TSS descriptor.

- The DPL field of a task gate controls the right to use the descriptor to cause a task switch.

- The 80386 has task gates in addition to TSS descriptors to satisfy three needs:

1. **One task can execute different multiple task switches**, i.e. one TSS may require multiple Task gate Descriptors.
2. **The need to provide selective access to tasks.** Using PL of a task, Task Gate Descriptors choose high priority task to be executed.
3. **The need for an interrupt or exception to cause a task switch.** Task gates may also reside in the IDT, making it possible for interrupts and exceptions to cause task switching.

- **Task gate in LDT and a task gate in the IDT can identify the same task.**



Task Switching

- The 80386 switches execution to another task in any of four cases:

1. The current task executes a JMP or CALL that refers to a TSS descriptor.

2. The current task executes a JMP or CALL that refers to a Task Gate.

3. An interrupt or exception vectors to a Task Gate in the IDT.

4. The current task executes an IRET when the NT flag is set.

- **A task switching operation involves these steps:**

1. **Checking that the current task is allowed to switch to the designated task.** Data-access privilege rules apply in the case of JMP or CALL instructions.

2. **Checking that the TSS descriptor of the new task is marked present and has a valid limit.**

3. **Saving the state of the current task.** The processor finds the base address of the current TSS cached in the task register. It copies the registers into the current TSS (EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI, ES, CS, SS, DS, FS, GS, and the flag register). The EIP field of the TSS points to the instruction after the one that caused the task switch.

4. **Loading the Task Register with the selector of the incoming task‘s TSS descriptor**, marking the incoming task's TSS descriptor as busy, and setting the TS (task switched) bit of the MSW.

5. **Loading the incoming task's state from its TSS and resuming execution.** The registers loaded are the LDT register; the flag register; the general registers EIP, EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI; the segment registers ES, CS, SS, DS, FS, and GS; and PDBR.

Tests conducted during a Task Switch

Test	Test Description	Exception	Error Code Selects
1	Incoming TSS descriptor is present	NP	Incoming TSS
2	Incoming TSS descriptor is marked not-busy	GP	Incoming TSS
	marked not-busy		
3	Limit of incoming TSS is greater than or equal to 103	TS	Incoming TSS
	-- All register and selector values are loaded --		
4	LDT selector of incoming task is valid	TS	Incoming TSS
5	LDT of incoming task is present	TS	Incoming TSS
6	CS selector is valid	TS	Code segment
7	Code segment is present	NP	Code segment
8	Code segment DPL matches CS RPL	TS	Code segment
9	Stack segment is valid	GP	Stack segment
10	Stack segment is present	SF	Stack segment
11	Stack segment DPL = CPL	SF	Stack segment
12	Stack-selector RPL = CPL	GP	Stack segment
13	DS, ES, FS, GS selectors are valid	GP	Segment
14	DS, ES, FS, GS segments are readable	GP	Segment
15	DS, ES, FS, GS segments are present	NP	Segment
16	DS, ES, FS, GS segment DPL >= CPL (unless these are conforming segments)	GP	Segment

NP = Segment-not-present exception

GP = General protection fault

TS = Invalid TSS

SF = Stack fault

Validity tests of a selector checks whether selector referring to the proper
Table or not (eg., the LDT selector refers to the GDT).

Task Linking

- The **back-link field of the TSS** and the **NT (nested task) bit of the flag** work together allow the 80386 to automatically return to a task that CALLed another task or was interrupted by another task.

- When a CALL instruction, an interrupt instruction, an external interrupt, or an exception causes a switch to a new task, the **80386 automatically fills the back-link of the new TSS with the selector of the outgoing task's TSS** and, at the same time, sets the NT bit in the new task's flag register.

Effect of Task Switch on BUSY, NT, and Back-Link

Affected Field	Effect of JMP Instruction	Effect of CALL Instruction	Effect of IRET Instruction
Busy bit of incoming task	Set, must be 0 before	Set, must be 0 before	Unchanged, must be set
Busy bit of outgoing task	Cleared	Unchanged (already set)	Cleared
NT bit of incoming task	Cleared	Set	Unchanged
NT bit of outgoing task	Unchanged	Unchanged	Cleared
Back-link of incoming task	Unchanged	Set to outgoing TSS selector	Unchanged
Back-link of outgoing task	Unchanged	Unchanged	Unchanged

Task Address Space

- In 80386, data can be stored in form of segments or pages.
- The LDT selector and PDBR can be used to select the appropriate segment or page to access the data.
- By appropriate choice of the segment and page mappings for each task, **tasks may share address spaces.**

1. Task Linear-to-Physical Space Mapping

- 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.

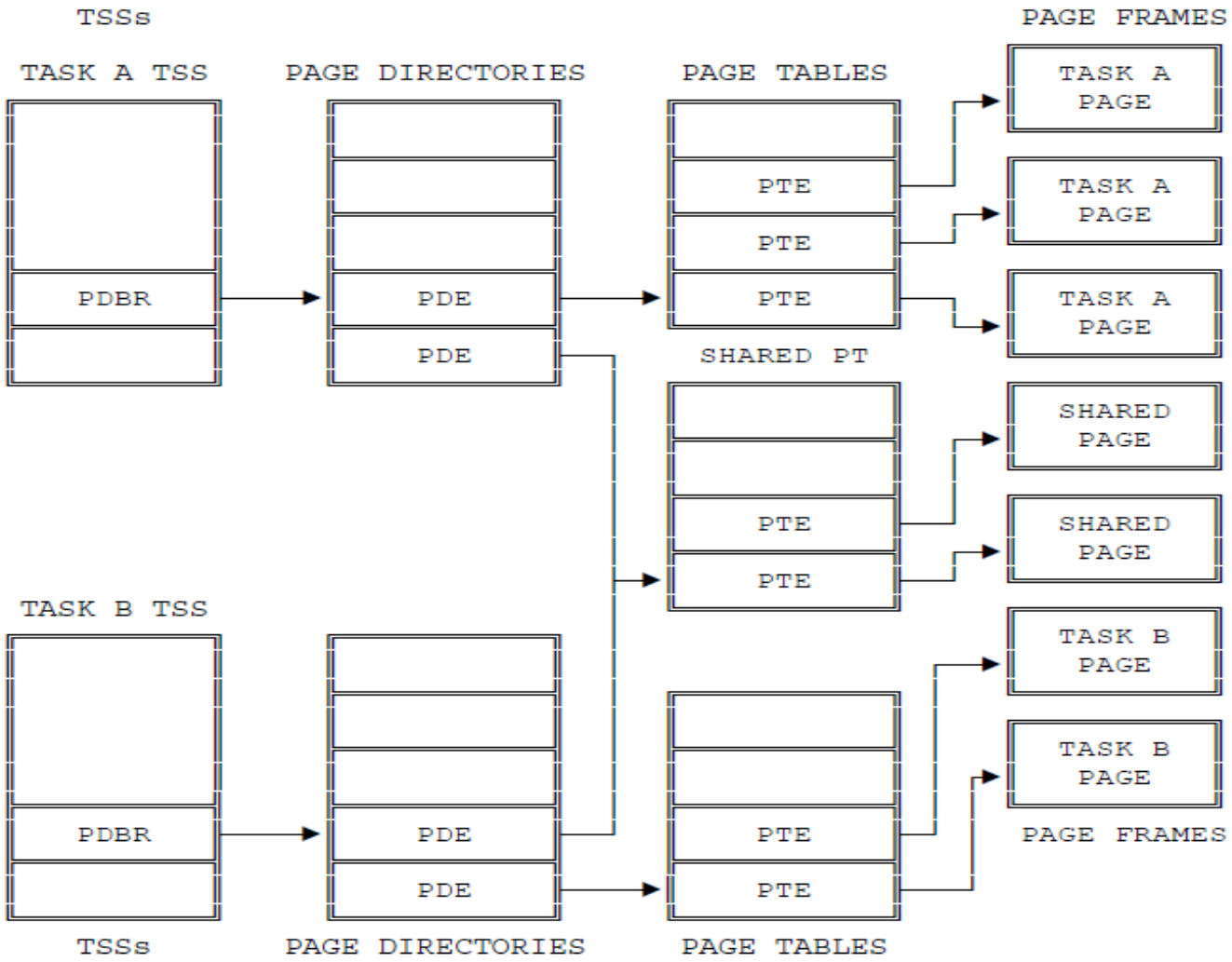
- When paging is enabled, one page directory is shared among all executing tasks.

2. Several partially overlapping “linear-to-physical mappings”.

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

- **In theory**, the linear address spaces of different tasks may map to completely distinct physical addresses.
- As 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 are mapped to the same physical addresses (see figure on next slide).

Partially-Overlapping Linear Spaces



2. Task Logical Address Space

- Actually, **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.

3. **By descriptor aliases in LDTs.** It is possible for certain descriptors of different LDTs to point to the same linear address space, if they share common aliases between descriptors.

Task-Nesting

- Tasks can be nested…