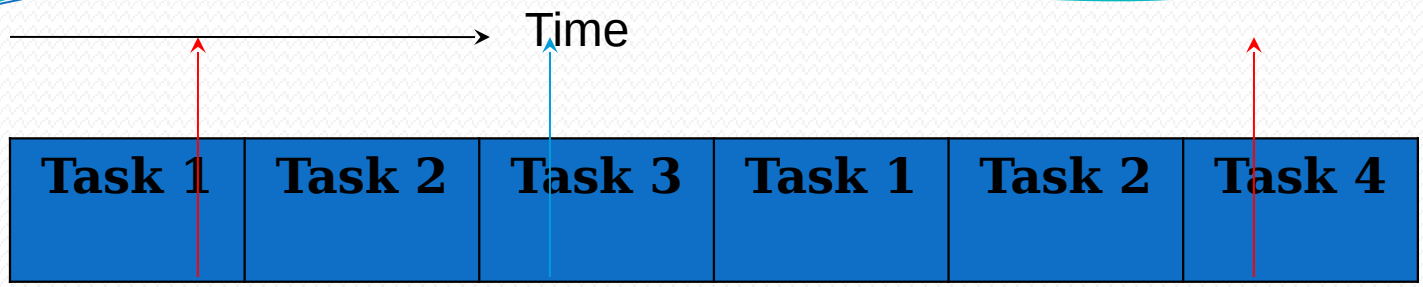
Unit V

MULTITASKING and Virtual Mode



Task switch

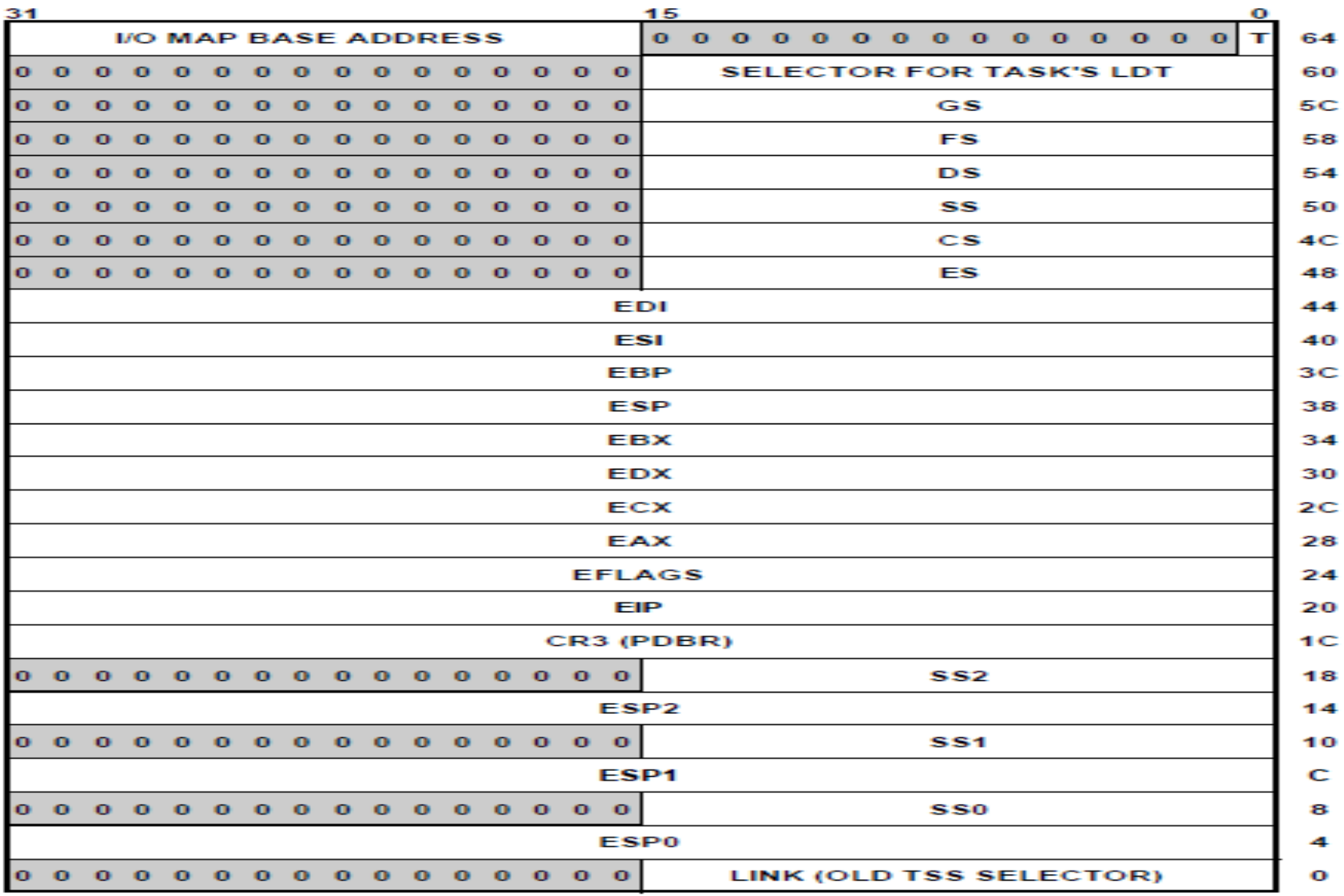
Task 3 completes

Task 4 begins

Fig: Running Multiple Tasks Simultaneously

- The x386 processor provides hardware support for multitasking.
- A task is a program which is running, or waiting to run while another program is running.
- A task is invoked by an interrupt, exception, jump, or call.
- When one of these forms of transferring execution is used with a destination specified by an entry in one of the descriptor tables, this descriptor can be a type which causes a new task to begin execution after saving the state of the current task.
- There are two types of task-related descriptors which can occur in a descriptor table: task state segment descriptors and task gates.
- When execution is passed to either kind of descriptor, a task switch occurs.

- A task switch is like a procedure call, but it saves more processor state information.
- A task switch transfers execution to a completely new environment, the environment of a task.
- This requires saving the contents of nearly all the processor registers, including the EFLAGS register and the segment registers.
- Unlike procedures, tasks are not re-entrant.
- A task switch does not push anything on the stack.
- The processor state information is saved in a data structure in memory, called a task state segment.



ADDRESSES ARE SHOWN IN HEXADECIMAL.
NOTE: BITS MARKED AS 0 ARE RESERVED. DO NOT USE.

APM62

Figure 13-1. 32-Bit Task State Segment

TASK STATE SEGMENT

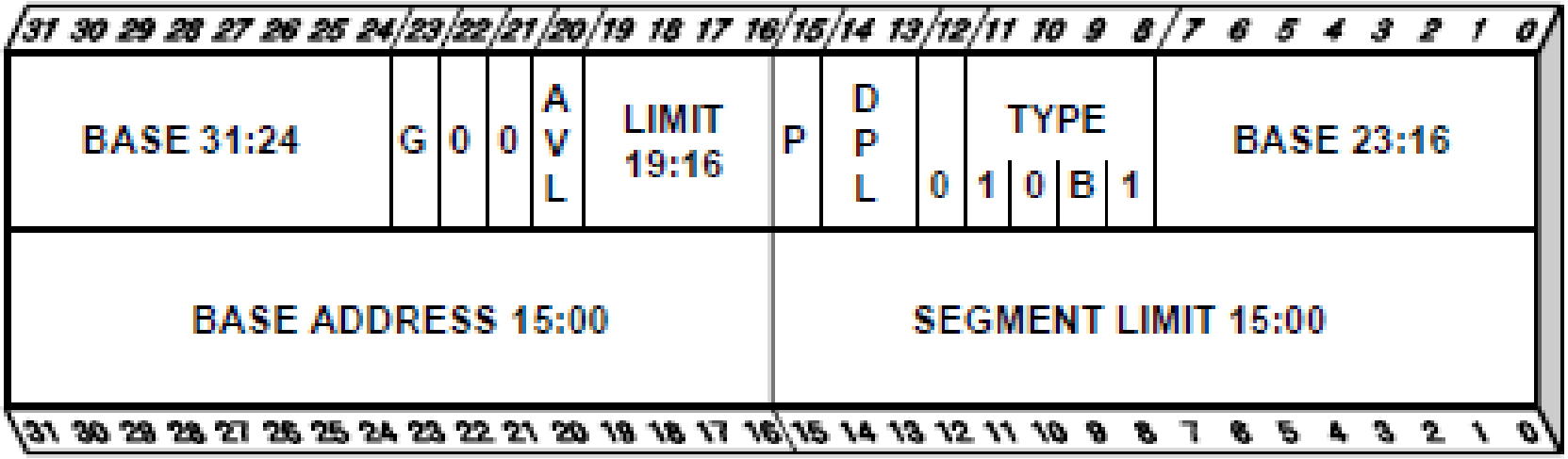
- The processor state information needed to restore a task is saved in a type of segment, called a task state segment or TSS.
- Figure :format of a TSS for tasks designed for 32- bit CPUs.
- The fields of a TSS are divided into two main categories:

1. Dynamic fields the processor updates with each task switch.
2. Static fields the processor reads, but does not change.

TSS DESCRIPTOR

- The task state segment, like all other segments, is defined by a descriptor.
- The format of a TSS descriptor.
- The Base, Limit, and DPL fields and the Granularity bit and Present bit have functions similar to their use in data-segment descriptors.

TSS DESCRIPTOR



AVL	AVAILABLE FOR USE BY SYSTEM SOFTWARE
B	BUSY BIT
BASE	SEGMENT BASE ADDRESS
DPL	DESCRIPTOR PRIVILEGE LEVEL
G	GRANULARITY
LIMIT	SEGMENT LIMIT
P	SEGMENT PRESENT
TYPE	SEGMENT TYPE

APM61

Figure 13-2. TSS Descriptor

- The Busy bit in the Type field indicates whether the task is busy.
- A busy task is currently running or waiting to run.
- A Type field with a value of 9 indicates an inactive task; a value of 11 (decimal) indicates a busy task.
- Tasks are not recursive.
- The processor uses the Busy bit to detect an attempt to call a task whose execution has been interrupted.

multitasking

- Task State Segment
- TSS Descriptor
- Task Register
- Task Gate Descriptor
- Task Switching
- Task Linking
- Task Address Space.

Multitasking……

- To provide efficient, protected multitasking, the 80386 employs several special data structures.
- It does not use special instructions to control multitasking
- It interprets ordinary control-transfer instructions differently when they refer to the special data structures.
- The registers and data structures that support multitasking are:

1. Task state segment
2. Task state segment descriptor
3. Task register
4. Task gate descriptor

Task MGMT Feature #1

- 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.

Task MGMT Feature #2

- Interrupts and exceptions can cause task switches (if needed in the system design).
- The processor not only switches automatically to the task that handles the interrupt or exception, but it automatically switches back to the interrupted task when the interrupt or exception has been serviced.
- Interrupt tasks may interrupt lower-priority interrupt tasks to any depth.

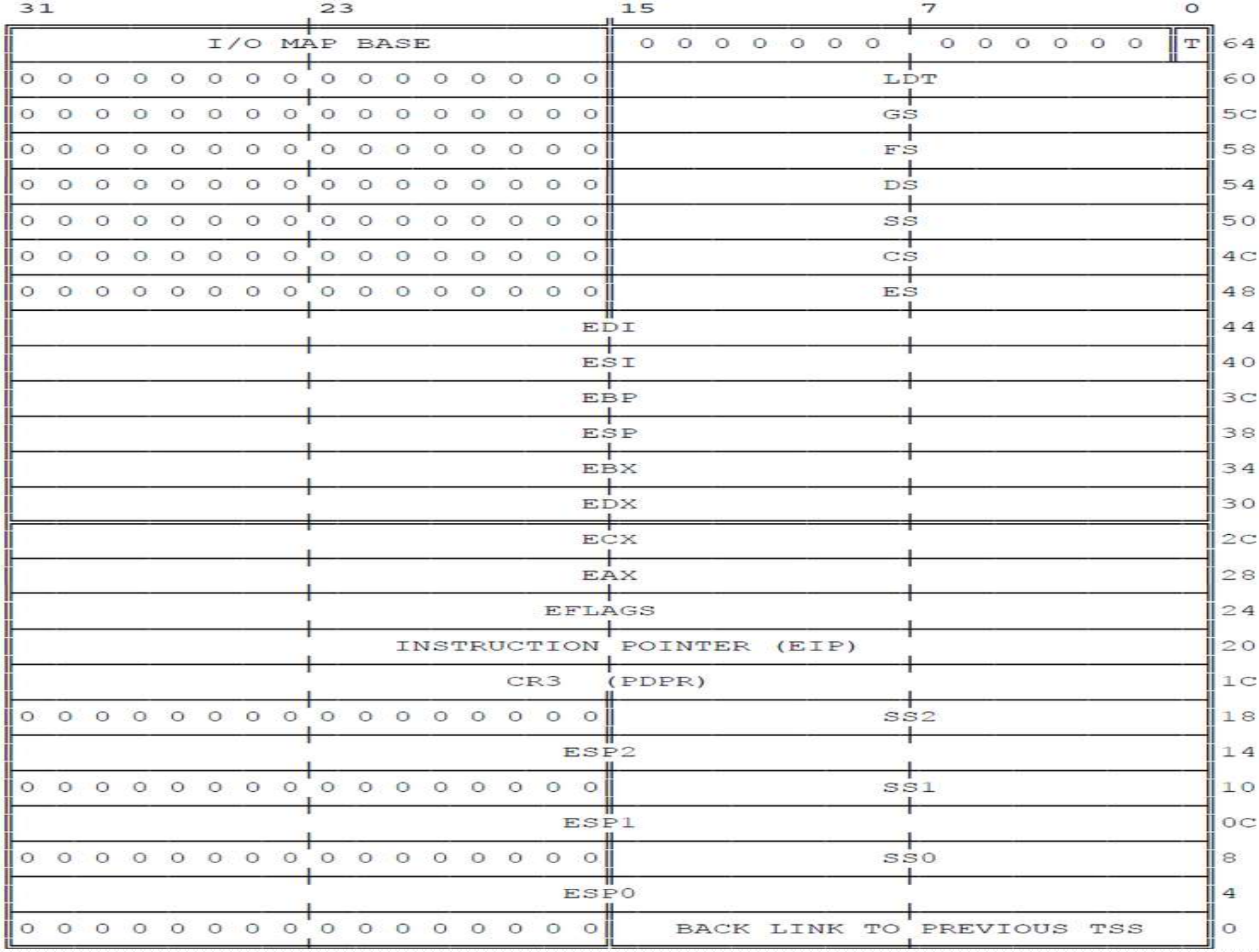
Task MGMT Feature #3

- With each switch to another task, the 80386 can also switch to another LDT and to another page directory.
- Thus each task can have a different logical-to-linear mapping and a different linear-to-physical mapping.
- This is yet another protection feature, because tasks can be isolated and prevented from interfering with one another.

6. Task State Segment

- All the information the processor needs in order to manage a task is stored in a special type of segment, a task state segment (TSS).
- Figure
- Format of a TSS for executing 80386 tasks

Diagram :
80386 32-Bit Task State Segment


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 Fields

- The fields of a TSS belong to two classes:

1. A dynamic set that the processor updates with each switch from the task.
2. A static set that the processor reads but does not change.

Dynamic Set

- This set includes the fields that store:

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

Static Set

- This set includes the fields that store:

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

Bit map

- The base address for the I/O permission bit map and interrupt redirection bitmap.
- If present, these maps are stored in the TSS at higher addresses.
- The base address points to the beginning of the I/O map and the end of the 32-byte interrupt map.

Task State Segment (TSS)

Two classes of TSS format

Static set

- **It is that where processor reads 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
  - The I/O map base

Dynamic set

- **That where processor updates 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).

Format of 80386 TSS (32 bit)

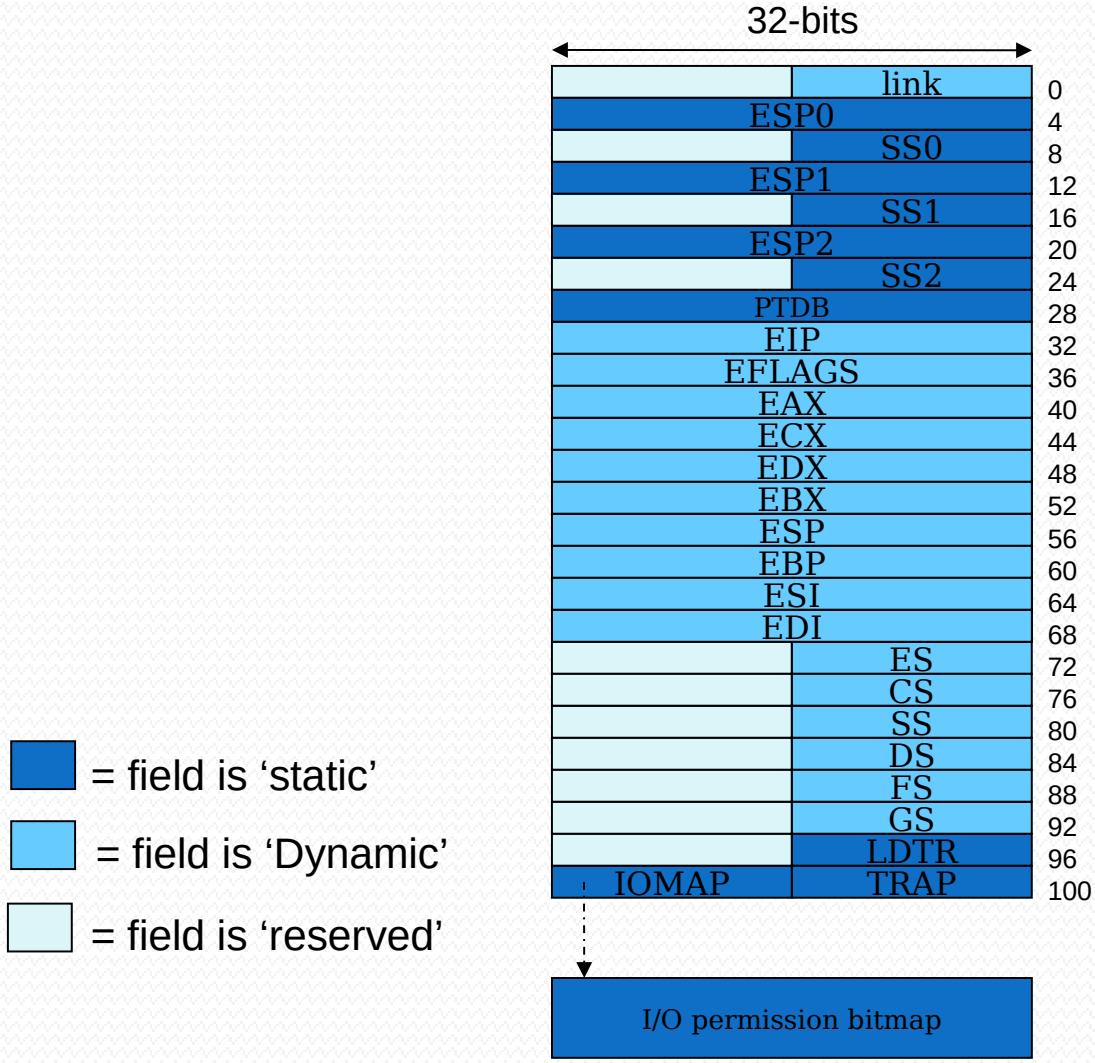


Fig. 1 :Task state

- Task state segments may reside anywhere in the linear space.
- The only case that requires caution is :
- when the TSS spans a page boundary and the higher-addressed page is not present.
- In this case, the processor raises an exception if it encounters the not-present page while reading the TSS during a task switch.

- Such an exception can be avoided by either of two strategies:

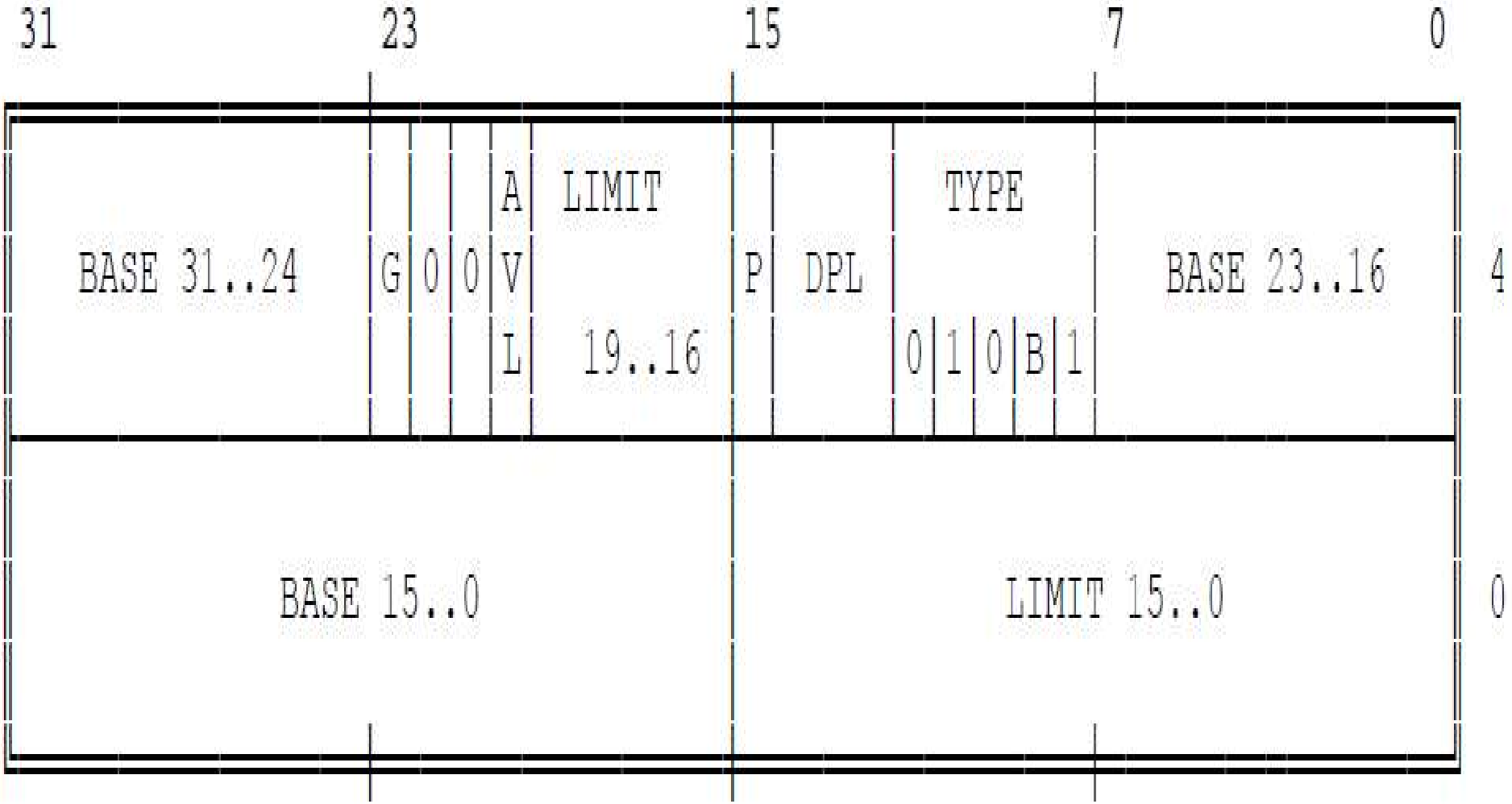
1. By allocating the TSS so that it does not cross a page boundary.

2. By ensuring that both pages are either both present or both not-present at the time of a task switch. If both pages are not-present, then the page-fault handler must make both pages present before restarting the instruction that caused the task switch.

7. TSS Descriptor

- The task state segment, like all other segments, is defined by a descriptor.
- Figure (Next) :the format of a TSS descriptor
- The B-bit in the type field indicates whether the task is busy.
- A type code of 9 indicates a non-busy task; a type code of 11 indicates a busy task.
- Tasks are not reentrant.
- The B-bit allows the processor to detect an attempt to switch to a task that is already busy.

Figure 7-2. TSS Descriptor for 32-bit TSS



Fields of TSS Descriptor

- The BASE, LIMIT, and DPL fields and the G-bit and P-bit have functions similar to their counterparts in data-segment descriptors.
- The LIMIT field, must have a value equal to or greater than 103.
- An attempt to switch to a task whose TSS descriptor has a limit less that 103 causes an exception.
- A larger limit is permissible, and a larger limit is required if an I/O permission map is present.
- A larger limit may also be convenient for systems software if additional data is stored in the same segment as the TSS.

- A procedure that has access to a TSS descriptor can cause a task switch.

- In most systems the DPL fields of TSS descriptors should be set to zero, so that only trusted software has the right to perform task switching.

- Having access to a TSS-descriptor does not give a procedure the right to read or modify a TSS.
- Reading and modification can be accomplished only with another descriptor that redefines the TSS as a data segment.
- An attempt to load a TSS descriptor into any of the segment registers (CS, SS, DS, ES, FS, GS) causes an exception.

- TSS descriptors may reside only in the GDT.

- An attempt to identify a TSS with a selector that has TI=1 (indicating the current LDT) results in an exception.

8.TR

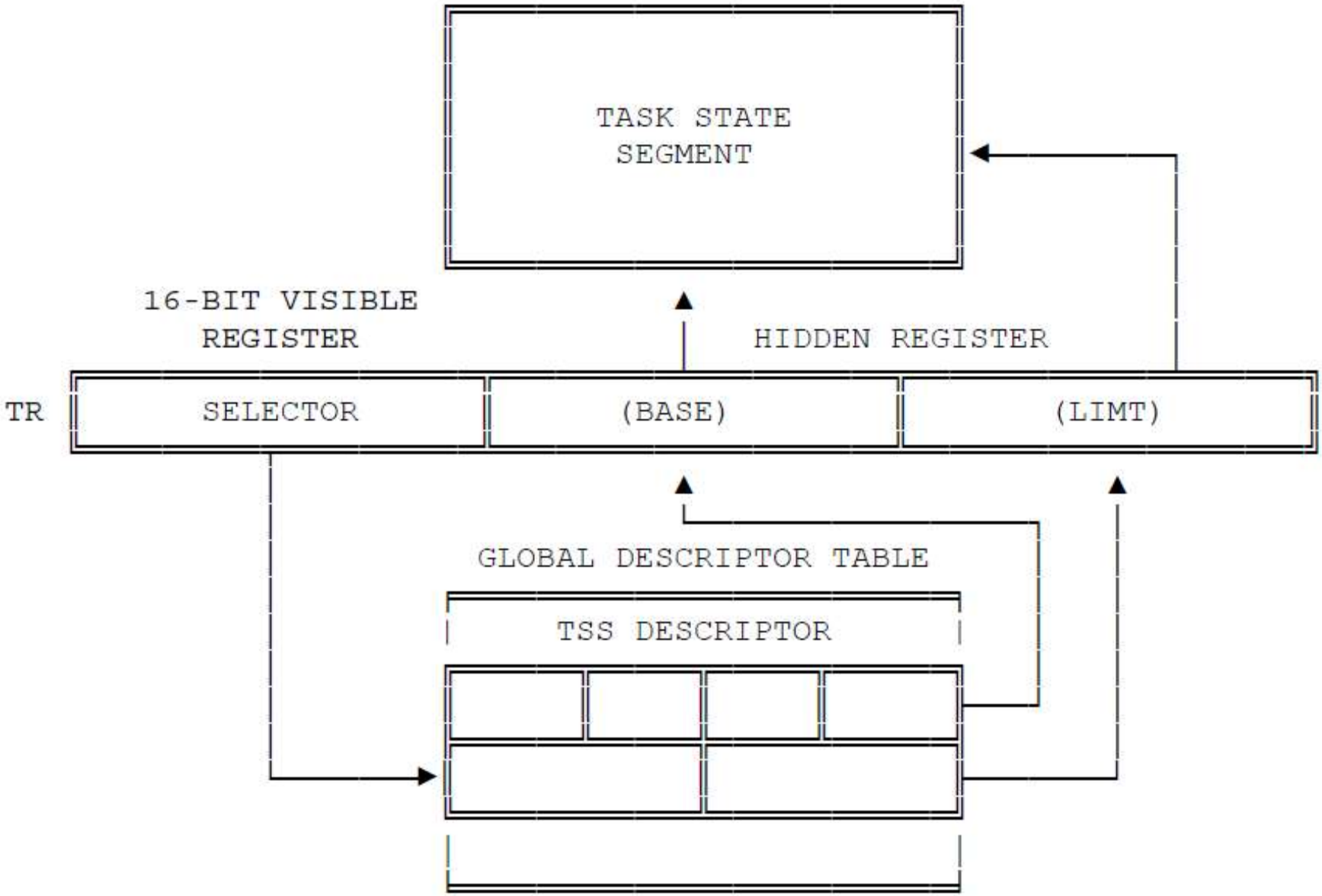
- TSS descriptors may only be loaded into GDT.
- When multiple TSS descriptors exist in GDT, the TSS currently in use is accessed through the use of the Task Register.
- TR is used as an index pointer into the GDT to locate a TSS descriptor.

TR.....

- The task register (TR) identifies the currently executing task by pointing to the TSS.

- Figure (Next):

the path by which the processor accesses the current TSS



Task Register

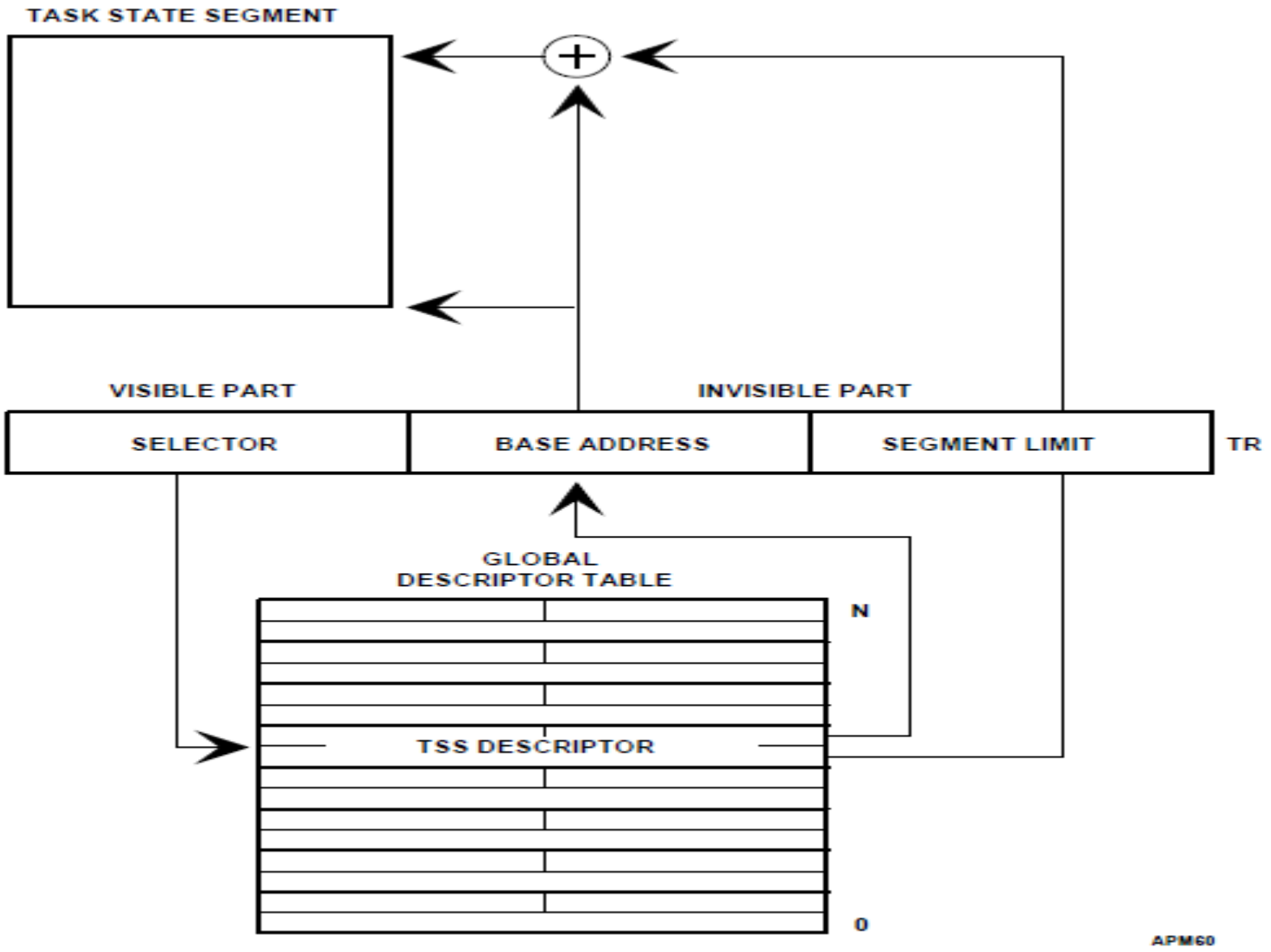


Figure 13-3. Task Register

TASK REGISTER

- The task register (TR) is used to find the current TSS.
- The task register has :
  - a visible part (i.e., a part which can be read and changed by software),
  - an invisible part (i.e., a part maintained by the processor and inaccessible to software).

- The selector in the visible portion indexes to a TSS descriptor in the GDT.
- The processor uses the invisible portion of the TR register to retain the base and limit values from the TSS descriptor.
- Keeping these values in a register makes execution of the task more efficient, because the processor does not need to fetch these values from memory to reference the TSS of the current task.

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

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

LTR : Load task register

- Loads the visible portion of the task register with the selector operand, which must select a TSS descriptor in the GDT.
- LTR also loads the invisible portion with information from the TSS descriptor selected by the operand.
- LTR is a privileged instruction; it may be executed only when CPL is zero.
- generally use during system initialization to give an initial value to the task register
- Then, contents of TR are changed by task switch operations.

STR : Store task register

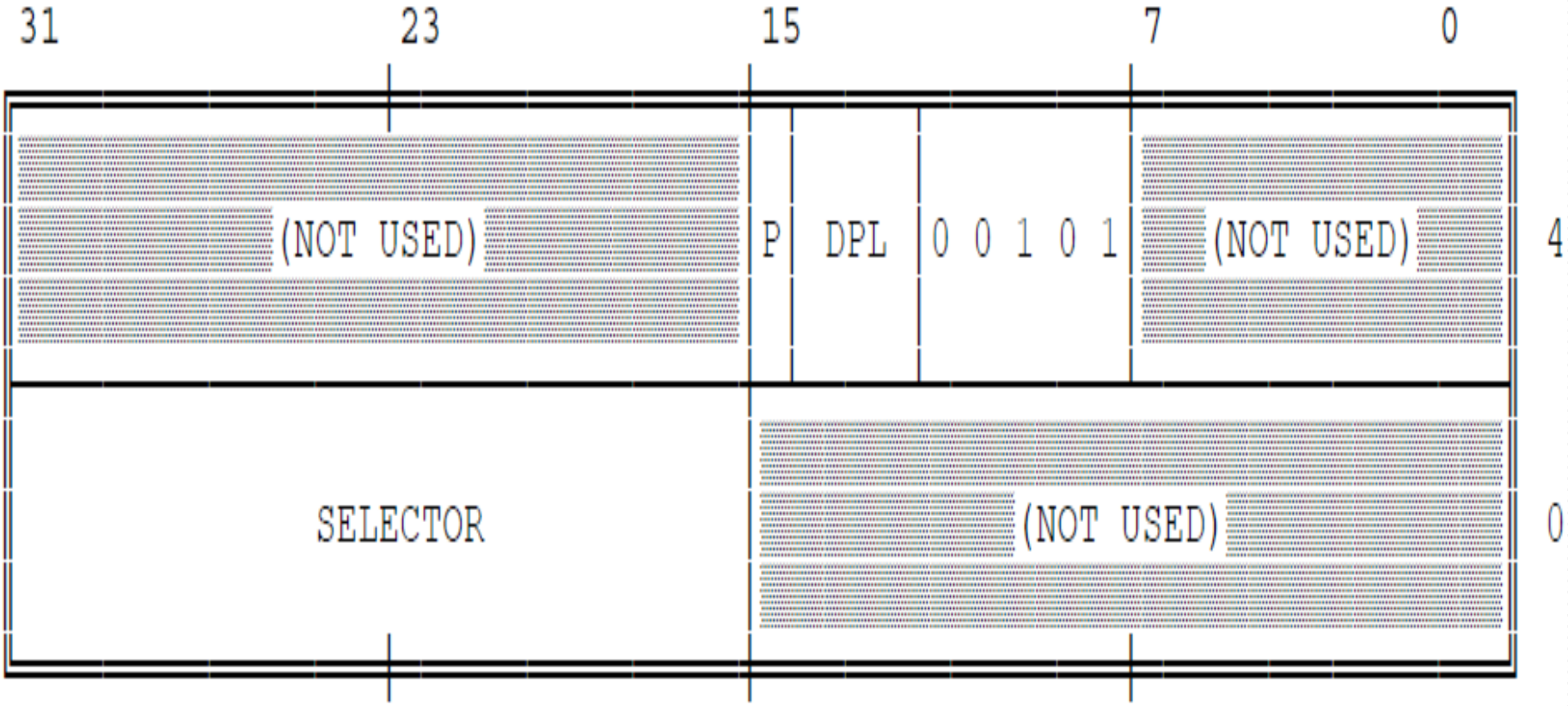
- stores the visible portion of the task register in a general register or memory word.

- STR is not privileged.

9.Task Gate Descriptor

- A task gate descriptor provides an indirect, protected reference to a TSS.
- Figure (Next) :the format of a task gate
- The SELECTOR field of a task gate must refer to a TSS descriptor.
- The value of the RPL in this selector is not used by the processor.

Figure 7-4. Task Gate Descriptor



- The DPL field of a task gate controls the right to use the descriptor to cause a task switch.
- A procedure may not select a task gate descriptor unless the maximum of the selector's RPL and the CPL of the procedure is numerically less than or equal to the DPL of the descriptor.
- This constraint prevents untrusted procedures from causing a task switch.
- Note : when a task gate is used, the DPL of the target TSS descriptor is not used for privilege checking.

- A procedure that has access to a task gate has the power to cause a task switch, just as a procedure that has access to a TSS descriptor.

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

1. The need for a task to have a single busy bit.
2. The need to provide selective access to tasks.
3. The need for an interrupt or exception to cause a task switch.

Need #1

To have a single busy bit for a task:

- Because the busy-bit is stored in the TSS descriptor, each task should have only one such descriptor.

- There may, however, be several task gates that select the single TSS descriptor.

Need #2

To provide selective access to tasks:

- Task gates fulfill this need, because they can reside in LDTs and can have a DPL that is different from the TSS descriptor's DPL.
- A procedure that does not have sufficient privilege to use the TSS descriptor in the GDT (which usually has a DPL of 0) can still switch to another task if it has access to a task gate for that task in its LDT.
- With task gates, systems software can limit the right to cause task switches to specific tasks.

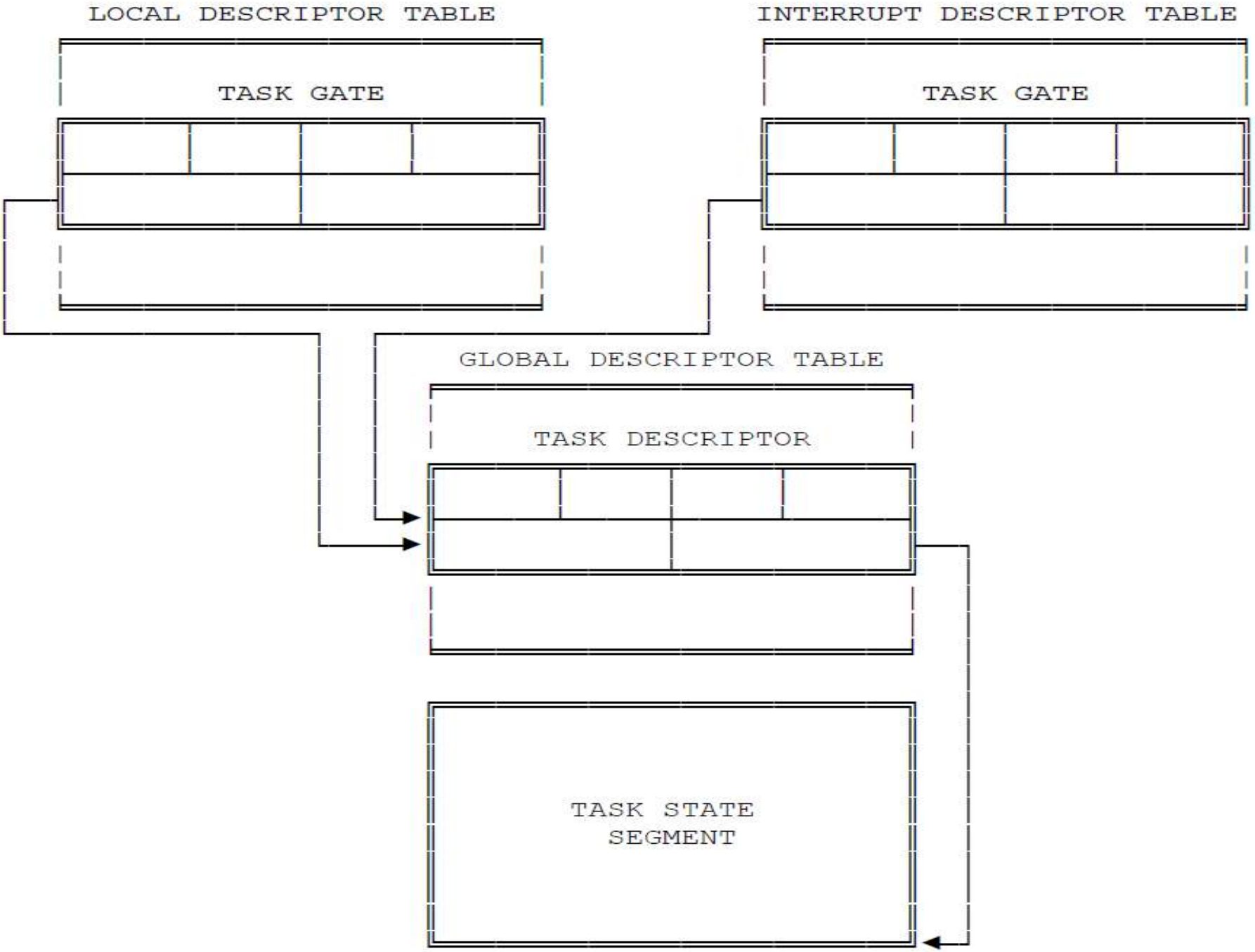
Need #3

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.
- When interrupt or exception vectors to an IDT entry that contains a task gate, the 80386 switches to the indicated task.
- Thus, all tasks in the system can benefit from the protection afforded by isolation from interrupt tasks.

Figure :
How both a task gate in an LDT and a task gate in the IDT can identify the same task

Figure 7-5. Task Gate Indirectly Identifies Task



10. 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.

- JMP, CALL, IRET, interrupts, and exceptions are all ordinary mechanisms of the 80386 that can be used in circumstances that do not require a task switch.

- Either the type of descriptor referenced or the NT (nested task) bit in the flag word distinguishes between the standard mechanism and the variant that causes a task switch.

- To cause a task switch, a JMP or CALL instruction can refer either to a TSS descriptor or to a task gate.
- The effect is the same in either case: the 80386 switches to the indicated task.
- An exception or interrupt causes a task switch when it vectors to a task gate in the IDT.
- If it vectors to an interrupt or trap gate in the IDT, a task switch does not occur.

- Whether invoked as a task or as a procedure of the interrupted task, an interrupt handler always returns control to the interrupted procedure in the interrupted task.

- If the NT flag is set, however, the handler is an interrupt task, and the IRET switches back to the interrupted task.

When a task switch is called, the following steps take place:

1.The new TSS descriptor or task gate must have sufficient privilege to allow a task switch.

2.The new TSS descriptor must have its present bit set and have a valid limit field.

3.The state of the current task(also called its context) is saved.

4.The TR is loaded with the selector of the new TSS descriptor.

5.The state of the new task is loaded from its TSS and execution is resumed

Step 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.
- The DPL of the TSS descriptor or task gate must be less than or equal to the maximum of CPL and the RPL of the gate selector.
- Exceptions, interrupts, and IRETs are permitted to switch tasks regardless of the DPL of the target task gate or TSS descriptor.
- The DPL, CPL, and RPL values are compared before any further processing takes place.
- Interrupts and exceptions do not force protection checking.

Step 2

- Checking that the TSS descriptor of the new task is marked present and has a valid limit.
- Any errors up to this point occur in the context of the outgoing task.
- Errors are restartable and can be handled in a way that is transparent to applications procedures.

Step 3

- Saving the state of the current task.
- This involves copying the contents of all processor registers into the TSS for 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.

Step 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, as is the busy bit in the new TSS descriptor.

- The selector is either the operand of a control transfer instruction or is taken from a task gate.

Step 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.
- Any errors detected in this step occur in the context of the incoming task.
- To an exception handler, it appears that the first instruction of the new task has not yet executed.

- The privilege level at which the old task was running has no relation to the privilege level of the new task.
- Because the tasks are isolated by their separate address spaces and task state segments, and because privilege rules control access to a TSS, no privilege checks are needed to perform a task switch.
- The new task begins executing at the privilege level indicated by the RPL of the new contents of the CS register, which are loaded from the TSS.

Note:

- The state of the outgoing task is always saved when a task switch occurs.
- If execution of that task is resumed, it starts after the instruction that caused the task switch.
- The registers are restored to the values they held when the task stopped executing.

11. Task Linking

- The back-link field of the TSS and the NT (nested task) bit of the flag word 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.

- The NT flag indicates whether the back-link field is valid.
- The new task releases control by executing an IRET instruction.
- When interpreting an IRET, the 80386 examines the NT flag.
- If NT is set, the 80386 switches back to the task selected by the back-link field.
- Table (Next) summarizes the uses of these fields.

Table 7-2. 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

Busy Bit Prevents Loops

- The Busy bit of the TSS descriptor prevents re-entrant task switching.
- There is only one saved task context, the context saved in the TSS, therefore a task only may be called once before it terminates.
- The chain of suspended tasks may grow to any length, due to multiple interrupts, exceptions, jumps, and calls.
- The Busy bit prevents a task from being called if it is in this chain.
- A re-entrant task switch would overwrite the old TSS for the task, which would break the chain.

- The processor manages the Busy bit as follows:

1. When switching to a task, the processor sets the Busy bit of the new task.

2. When switching from a task, the processor clears the Busy bit of the old task if that task is not to be placed in the chain (i.e., the instruction causing the task switch is a JMP or IRET instruction). If the task is placed in the chain, its Busy bit remains set.

3. When switching to a task, the processor generates a general-protection exception if the Busy bit of the new task already is set.

- In this way, the processor prevents a task from switching to itself or to any task in the chain, which prevents re-entrant task switching.

- The busy bit is effective even in multiprocessor configurations, because the processor automatically asserts a bus lock when it sets or clears the busy bit.

- This action ensures that two processors do not invoke the same task at the same time.

Modifying Task Linkages

- Any modification of the linkage order of tasks should be accomplished only by software that can be trusted to correctly update the back-link and the busy-bit.
- Such changes may be needed to resume an interrupted task before the task that interrupted it.
- Trusted software that removes a task from the back-link chain must follow one of the following policies:

1. First change the back-link field in the TSS of the interrupting task, then clear the busy-bit in the TSS descriptor of the task removed from the list.

2. Ensure that no interrupts occur between updating the back-link chain and the busy bit.

12. Task Addressing Space

- The LDT selector and PDBR fields of the TSS give software systems designers flexibility in utilization of segment and page mapping features of the 80386.
- By appropriate choice of the segment and page mappings for each task:
  - tasks may share address spaces,
  - may have address spaces that are largely distinct from one another,

    OR
  - may have any degree of sharing between these two extremes.

- The ability for tasks to have distinct address spaces is an important aspect of 80386 protection.

- A module in one task cannot interfere with a module in another task if the modules do not have access to the same address spaces.

- The flexible memory management features of the 80386 allow systems designers to assign areas of shared address space to those modules of different tasks that are designed to cooperate with each other.