Input/Output, Exceptions and Interrupts

Mr. Sumit Shinde
Assistant Professor
Computer Engineering Department

Pune Institute of Computer Technology

Exceptions & Interrupts

- Special kinds of control transfer
- They alter the normal program flow to handle external events or to report errors or exceptional conditions.
- Interrupts are used to handle asynchronous events external to the processor, but exceptions handle conditions detected by the processor itself in the course of executing instructions.

- **Interrupts-** Maskable (INTR), Nonmaskable (NMI)
- **Exceptions-**
- Processor detected. These are further classified as faults, traps, and aborts.
- Programmed. The instructions INTO, INT 3, INT n, and BOUND can trigger exceptions. These instructions are often called "software interrupts", but the processor handles them as exceptions.

Identifying Interrupts

- The NMI and the exceptions recognized by the processor are assigned predetermined identifiers in the range 0 through 31.
- The identifiers of the maskable interrupts are determined by external interrupt controllers (such as Intel's 8259A Programmable Interrupt Controller) and communicated to the processor during the processor's interrupt-acknowledge sequence.

Identifier	Description
0	Divide error
1	Debug exceptions
2	Nonmaskable interrupt
3	Breakpoint (one-byte INT 3 instruction)
4	Overflow (INTO instruction)
5	Bounds check (BOUND instruction)
6	Invalid opcode
7	Coprocessor not available
8	Double fault
9	(reserved)
10	Invalid TSS
11	Segment not present
12	Stack exception
13	General protection
14	Page fault
15	(reserved)
16	Coprecessor error
17-31	(reserved)
32-255	Available for external interrupts via INTR pin

Classification of Exceptions

- **Faults**
- Faults are either detected before the instruction begins to execute, or during execution of the instruction.
- permits the instruction to be restarted.
- **Traps**
- reported at the instruction boundary immediately after the instruction in which the exception was detected.
- **Aborts**
- used to report severe errors, such as hardware errors and inconsistent or illegal values in system tables.
- Wont permit instruction to be restarted

Enabling and Disabling Interrupts

- **NMI Masks Further NMIs**
- **IF Masks INTR**
- **RF Masks Debug Faults**
- **MOV or POP to SS Masks Some Interrupts and Exceptions**

Priority Among Simultaneous Interrupts and Exceptions

Priority	Class of Interrupt or Exception
HIGHEST	Faults except debug faults
	Trap instructions INTO, INT n, INT 3
	Debug traps for this instruction
	Debug faults for next instruction
	NMI interrupt
LOWEST	INTR interrupt

Interrupt Descriptor Table

Figure 9-1. IDT Register and Table

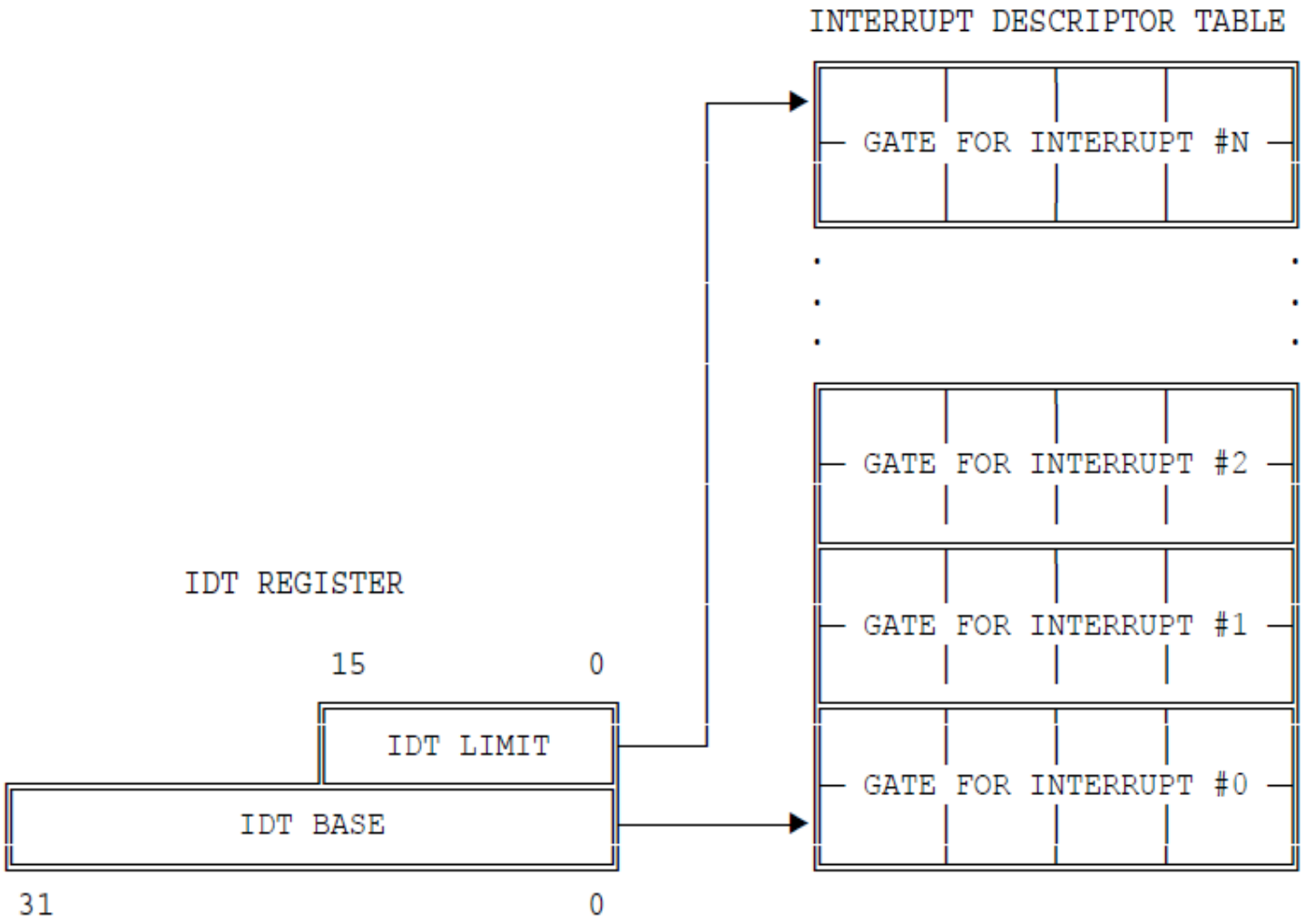
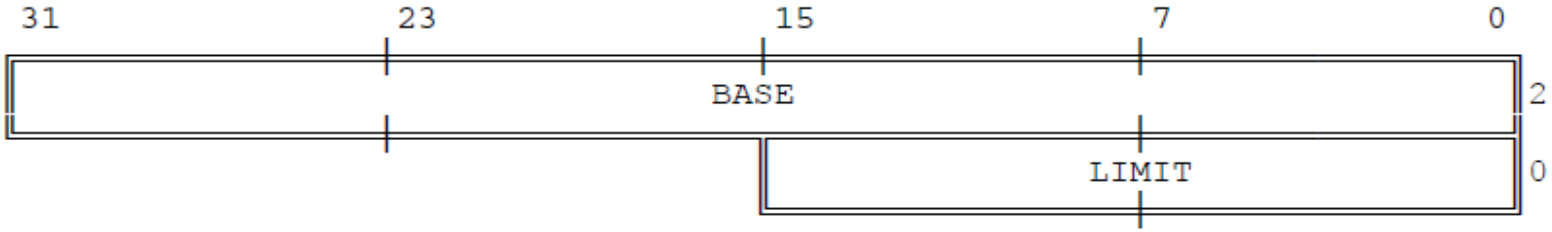


Figure 9-2. Pseudo-Descriptor Format for LIDT and SIDT

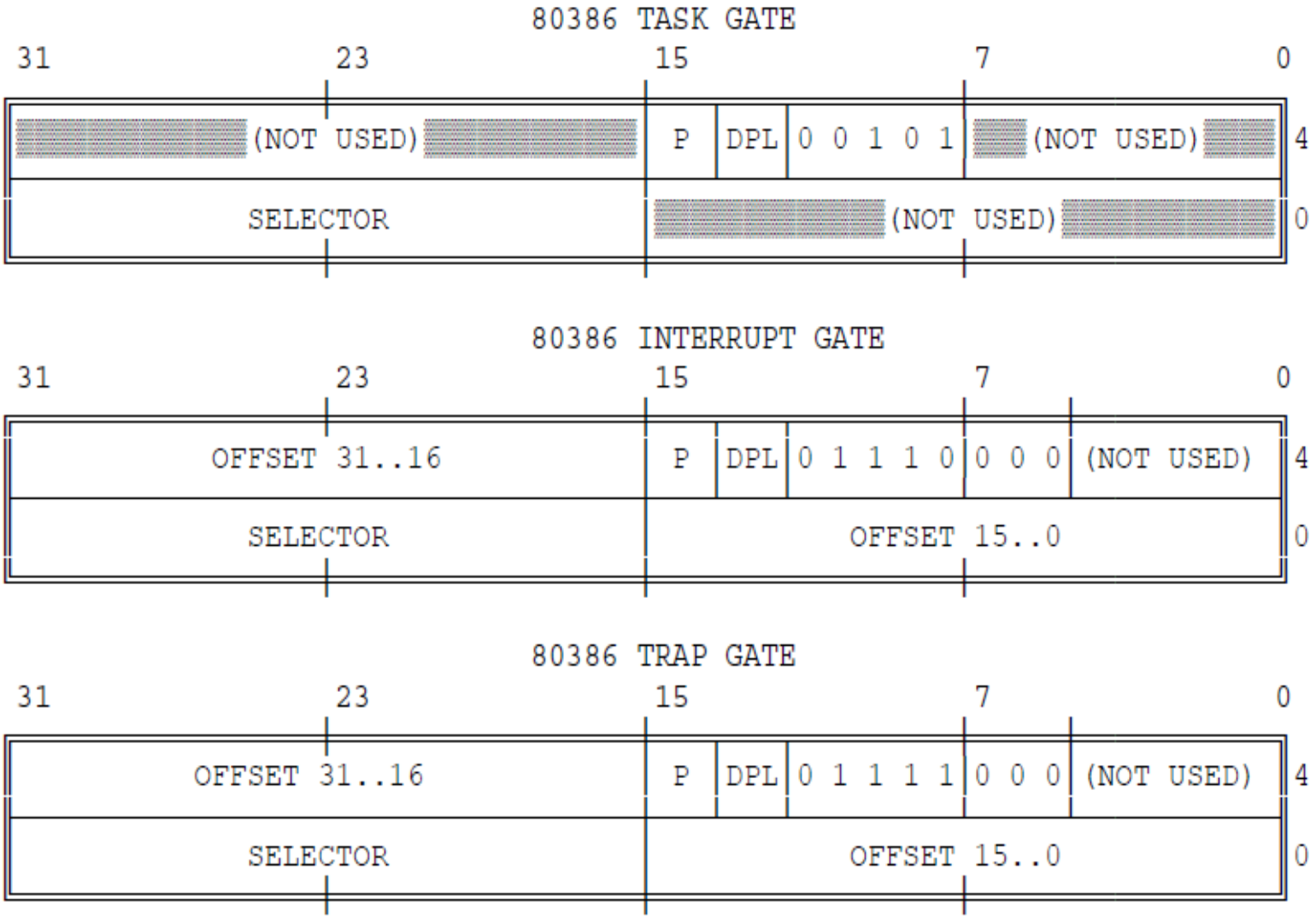


- The interrupt descriptor table (IDT) associates each interrupt or exception identifier with a descriptor for the instructions that service the associated event.
- As there are only 256 identifiers, the IDT need not contain more than 256 descriptors.
- Processor locates the IDT by means of the IDT register (IDTR).
- Instructions LIDT and SIDT operate on the IDTR.
- Both instructions have one explicit operand: the address in memory of a 6-byte area.

- **LIDT (Load IDT register)**
- loads the IDT register with the linear base address and limit values contained in the memory operand.
- Executed when CPL=0
- **SIDT (Store IDT register)**
- Copies the base and limit value stored in IDTR to a memory location.
- This instruction can be executed at any privilege level.

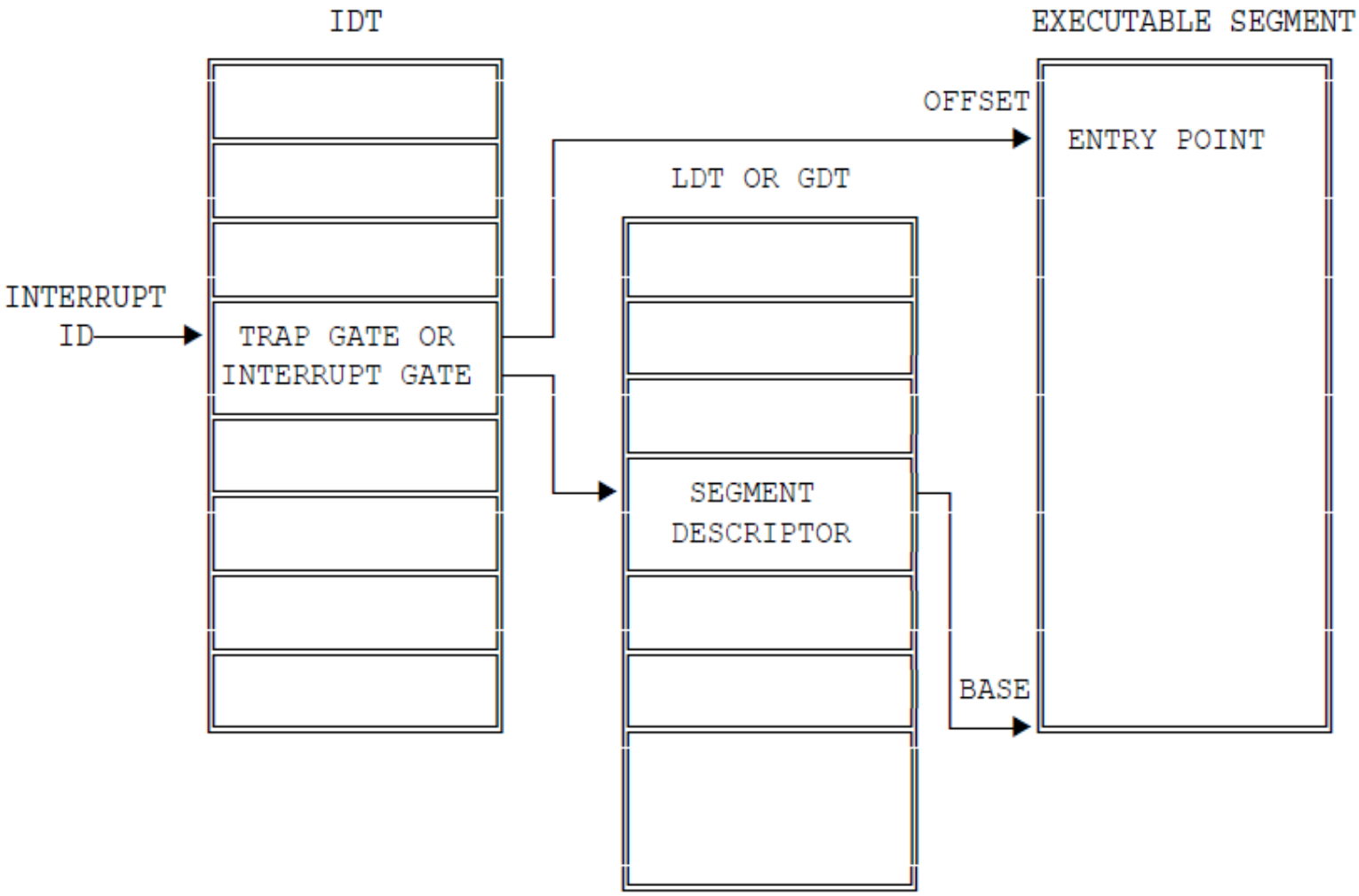
IDT Descriptors

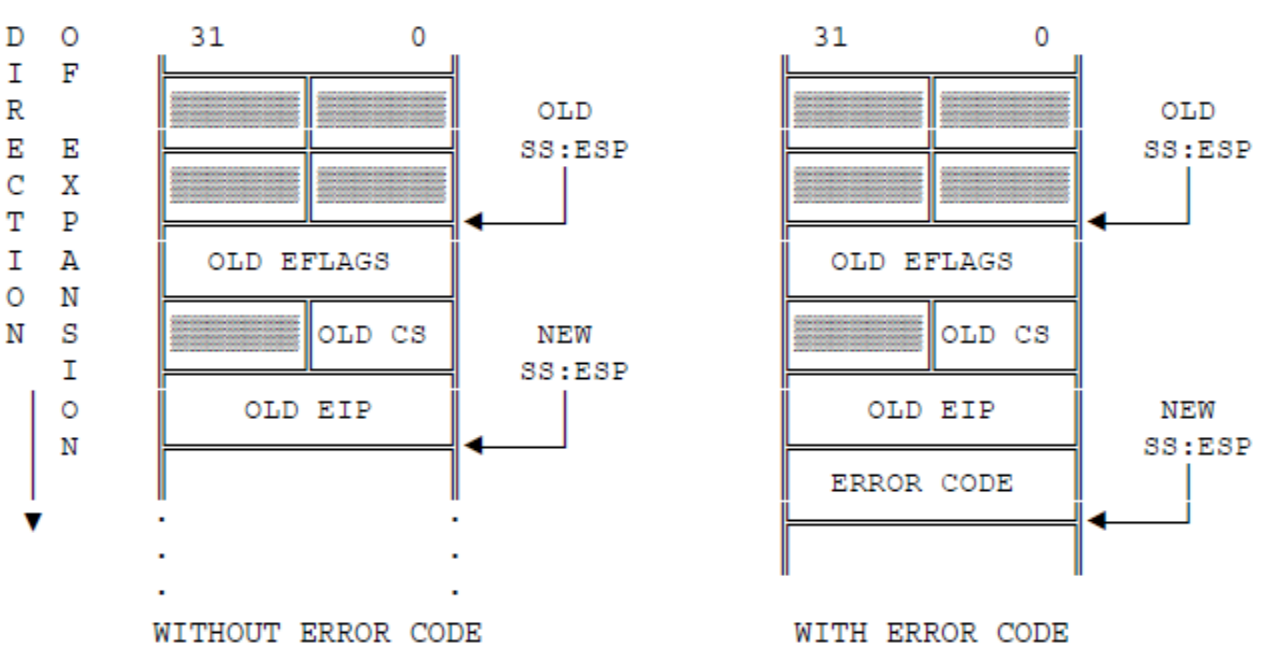
Figure 9-3. 80306 IDT Gate Descriptors



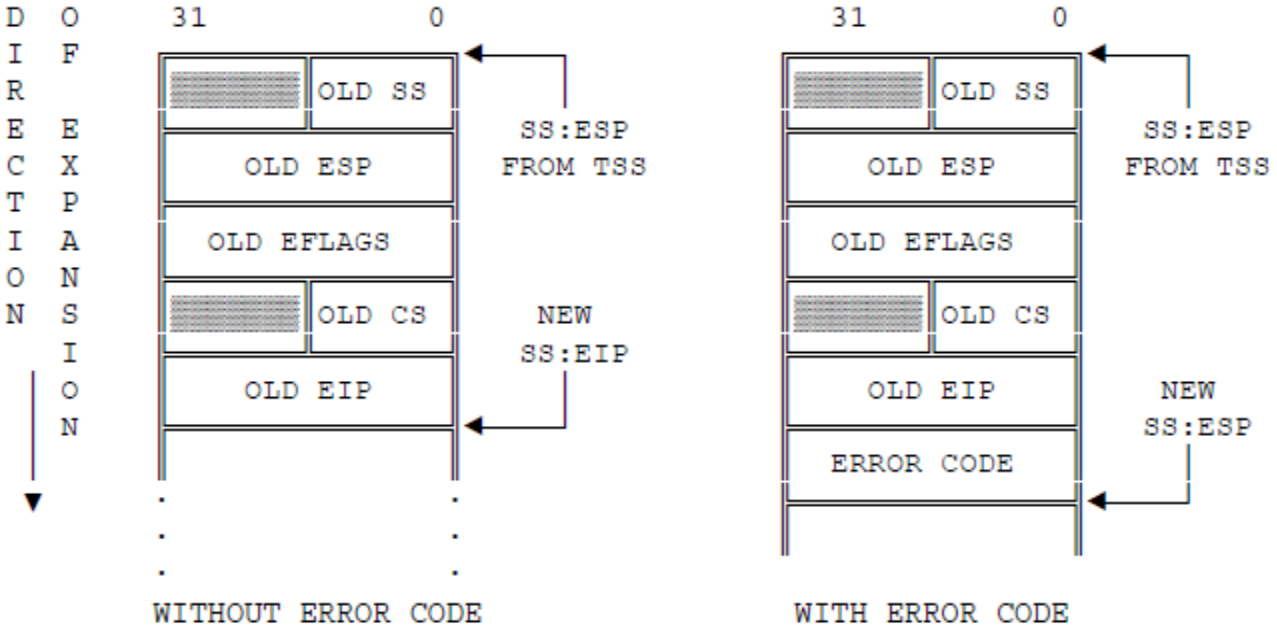
Interrupt Procedures

Figure 9-4. Interrupt Vectoring for Procedures



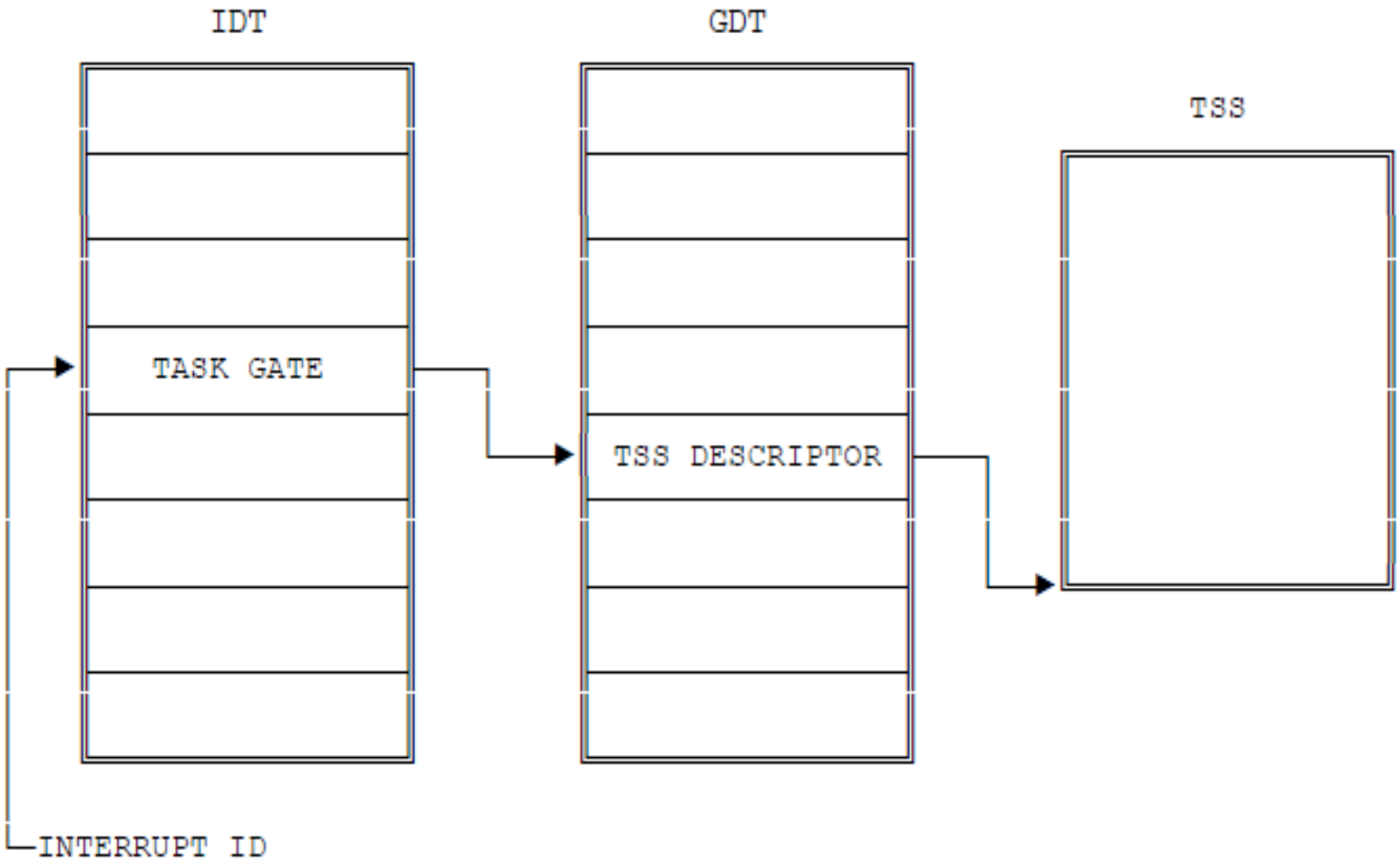


WITH PRIVILEGE TRANSITION



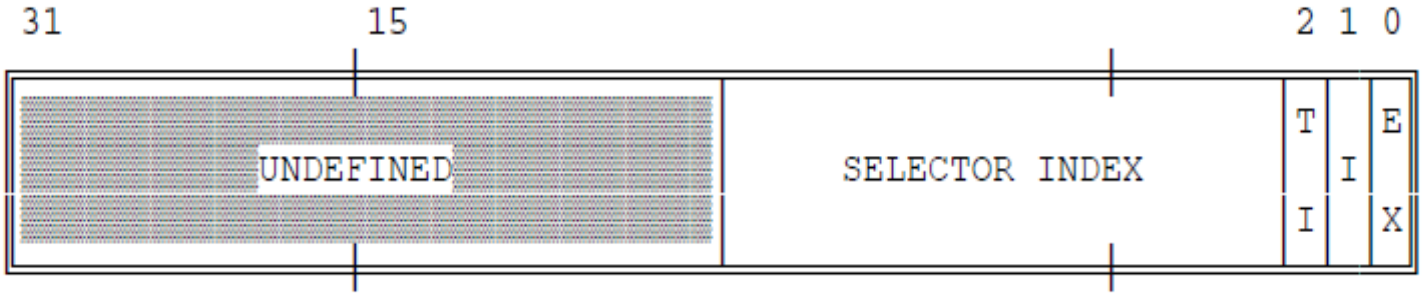
Interrupt Tasks

Figure 9-6. Interrupt Vectoring for Tasks



Error Code Format

Figure 9-7. Error Code Format



Exception Conditions

- **Interrupt 0 —— Divide Error**
- The divide-error fault occurs during a DIV or an IDIV instruction when the divisor is zero.
- **Interrupt 1 —— Debug Exceptions**
- The processor triggers this interrupt for any of a number of conditions.
- An exception handler can examine the debug registers to determine which condition caused the exception.

- **Interrupt 2 —— NMI**
- This is the only hardware interrupt assigned permanent vector.
- **Interrupt 3 —— Breakpoint**
- The INT 3 instruction causes this trap.
- Debuggers typically use breakpoints as a way of displaying registers, variables, etc., at crucial points in a task.
- **Interrupt 4 —— Overflow**
- This trap occurs when the processor encounters an INTO instruction and the OF (overflow) flag is set.
- When doing arithmetic on signed operands, careful programmers and compilers either test OF directly or use the INTO instruction.

- **Interrupt 5 —— Bounds Check**
- This fault occurs when the processor, while executing a BOUND instruction, finds that the operand exceeds the specified limits.
- **Interrupt 6 —— Invalid Opcode**
- This fault occurs when an invalid opcode is detected by the execution unit.
- No error code is pushed on the stack. The exception can be handled within the same task.
- This exception also occurs when the type of operand is invalid for the given opcode.

- **Interrupt 7 —— Coprocessor Not Available**
- The processor encounters an ESC (escape) instruction, and the EM (emulate) bit ofCR0 (control register zero) is set.
- The processor encounters either the WAIT instruction or an ESC instruction, and both the MP (monitor coprocessor) and TS (task switched) bits of CR0 are set.
- **Interrupt 8 —— Double Fault**
- when the processor detects an exception while trying to invoke the handler for a prior exception, processor cannot handle them serially, it signals the double-fault exception instead.
- The error code is always zero.

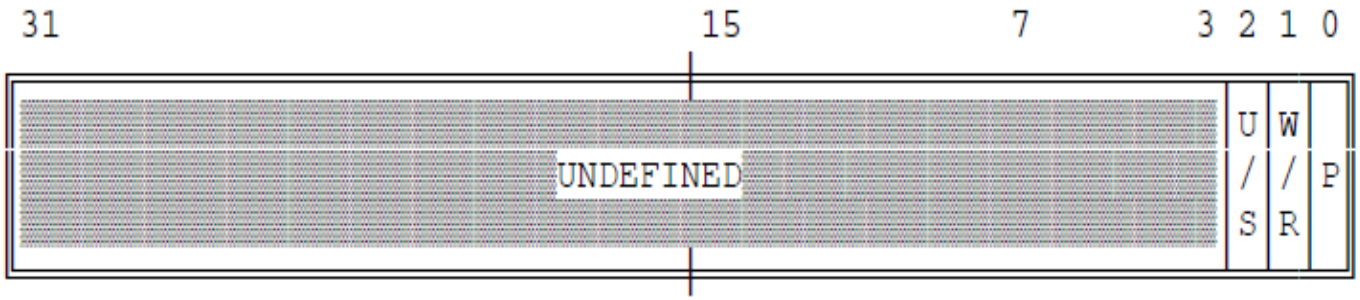
- The faulting instruction may not be restarted. If any other exception occurs while attempting to invoke the double-fault handler, the processor shuts down.
- **Interrupt 9 —— Coprocessor Segment Overrun**
- Equivalent to a general protection fault.
- Trying to load or store a floating point number beyond the limit of a data segment.
- **Interrupt 10 —— Invalid TSS**
- occurs if during a task switch the new TSS is invalid.
- An error code is pushed onto the stack to help identify the cause of the fault.

- **Interrupt 11 — Segment Not Present**
- While attempting to load the CS, DS, ES, FS, or GS registers; loading the SS register, however, causes a stack fault.
- While attempting loading the LDT register with an LLDT instruction; loading the LDT register during a task switch operation, however, causes the "invalid TSS" exception.
- While attempting to use a gate descriptor that is marked not-present.
- This fault is restartable. If the exception handler makes the segment present and returns, the interrupted program will resume execution.

- **Interrupt 12 —— Stack Exception**
- As a result of a limit violation in any operation that refers to the SS register.
- When attempting to load the SS register with a descriptor that is marked not-present but is otherwise valid.
- **Interrupt 13 —— General Protection Exception**
- All protection violations that do not cause another exception cause a general protection exception.
- The general protection exception is a fault.
- **Interrupt 14 —— Page Fault**
- This exception occurs when paging is enabled (PG=1)
- The page-directory or page-table entry needed for the address translation has zero in its present bit.
- The current procedure does not have sufficient privilege to access the indicated page.

Figure 9-8. Page-Fault Error Code Format

Field	Value	Description
U/S	0	The access causing the fault originated when the processor was executing in supervisor mode.
	1	The access causing the fault originated when the processor was executing in user mode.
W/R	0	The access causing the fault was a read.
	1	The access causing the fault was a write.
P	0	The fault was caused by a not-present page.
	1	The fault was caused by a page-level protection violation.



- **Interrupt 15 — Reserved**
- **Interrupt 16 — Coprocessor Error**
- The 80386 reports this exception when it detects a signal from the 80287 or 80387 on the 80386's ERROR# input pin.
- The 80386 tests this pin only at the beginning of certain ESC instructions and when it encounters a WAIT instruction while the EM bit of the MSW is zero (no emulation).