Interrupts

Introduction

- Interrupts provide an efficient way to handle **unanticipated events** and improve **processor utilization**
- Interrupts alter a program's flow of control
  - Interrupt causes transfer of control to an *interrupt service routine* (ISR)
    - ISR is also called a *handler*
  - When the ISR is completed, the original program resumes execution
  - Behavior is similar to a procedure call
    - Some significant differences between the two

Interrupts vs. Procedures

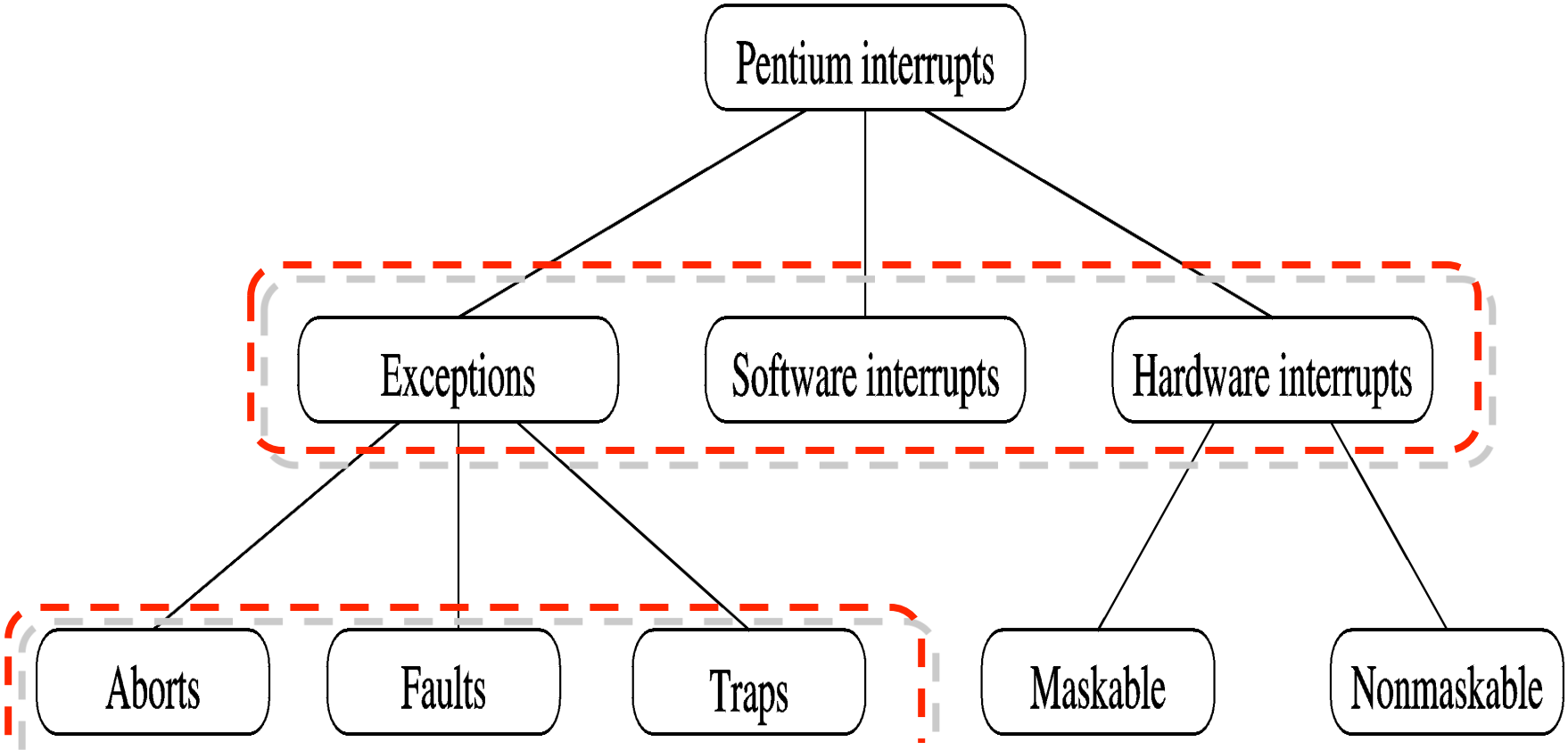
Interrupts

- Initiated by both *software* and *hardware*
- Can handle *anticipated* and *unanticipated* internal as well as external events
- ISRs or interrupt handlers are memory resident
- Use numbers to identify an interrupt service
- **eflags** register is saved automatically

Procedures

- Can only be initiated by *software*
- Can handle *anticipated* events that are coded into the program
- Typically loaded along with the program
- Use meaningful names to indicate their function
- Do not save the **eflags** register

A Taxonomy of Pentium Interrupts



Difference:

- Depending on the way they are reported
- Whether or not the interrupted instruction is restarted

Exceptions: Faults, Traps, and Aborts

- Faults
  - Instruction boundary before the instruction during which the exception was detected
  - Restarts the instruction
- Examples:
  - Page fault
  - Segment-not-found fault

Exceptions: Faults, Traps, and Aborts

- Traps
  - Instruction boundary immediately after the instruction during which the exception was detected
  - No instruction restart
- Examples:
  - Overflow exception (interrupt 4) is a trap
  - User defined interrupts are also examples of traps

Exceptions: Faults, Traps, and Aborts

- Aborts
  - No precise location of the instruction that caused the exception
  - No instruction restarting
  - Reporting severe errors such as hardware errors and inconsistent values in system tables
- Examples:
  - Machine check
  - Double fault

Dedicated Interrupts

- Several Pentium predefined interrupts --- called dedicated interrupts
- These include the first five interrupts:

interrupt type	Purpose
0	Divide error
1	Single-step
2	Non-maskable interrupt (NMI)
3	Breakpoint
4	Overflow

Dedicated Interrupts (cont'd)

- Single-Step Interrupt
  - Useful in debugging
  - To single step, Trap Flag (TF) should be set
  - CPU automatically generates a type 1 interrupt after executing each instruction if TF is set
  - Type 1 ISR can be used to present the system state to the user

Dedicated Interrupts (cont'd)

- Breakpoint Interrupt
  - Useful in debugging
  - CPU generates a **`type 3`** interrupt
  - Generated by executing a special single-byte version of **`int 3`** instruction (opcode CCH)

Interrupt Taxonomy

- Exceptions
- Software Interrupts
- Hardware Interrupts

Software Interrupts

- Initiated by executing an ***int*** instruction, where the interrupt number is an integer between 0 and 255
- Each interrupt can be parameterized to provide several services.
  - For example, Linux interrupt service **`int 0x80`** provides a large number of services (more than 330 system calls!)
    - EAX register is used to identify the required service under int 0x80

Hardware Interrupts

- Software interrupts are synchronous events
  - Caused by executing the `int` instruction
- Hardware interrupts are asynchronous in nature
  - Typically caused by applying an electrical signal to the processor chip
- Hardware interrupts can be
  - Maskable
  - Non-maskable

How Are Hardware Interrupts Triggered?

- Maskable interrupt is triggered by applying an electrical signal to the INTR (INTerrupt Request) pin of Pentium
  - Processor recognizes this interrupt only if IF (interrupt enable flag) is set
  - Interrupts can be masked or disabled by clearing IF
- Non-maskable interrupt is triggered by applying an electrical signal to the NMI pin of processor
  - Processor always responds to this signal
  - Cannot be disabled under program control

How Does the CPU Know the Interrupt Type?

- Interrupt invocation process is common to all interrupts
  - Whether originated in software or hardware
- For hardware interrupts, processor initiates an interrupt acknowledge sequence
  - processor sends out interrupt acknowledge (INTA) signal
  - In response, interrupting device places interrupt vector on the data bus
  - Processor uses this number to invoke the ISR that should service the device (as in software interrupts)

How Can More Than One Device Interrupt?

- Processor has only one INTR pin to receive interrupt signal
- Typical system has more than one device that can interrupt --- keyboard, hard disk, etc.
- Use a special chip to prioritize the interrupts and forward only one interrupt to the CPU
  - 8259 Programmable Interrupt Controller chip performs this function

Interrupt Processing

- How many interrupts can be supported?
  - Up to 256 interrupts
- Interrupt number is used as an index into the **Interrupt Descriptor Table** (IDT)
  - This table stores the addresses of all ISRs
  - Each descriptor entry is 8 bytes long
    - Interrupt number is multiplied by 8 to get byte offset into IDT

Detailed Steps in Interrupt Processing

- Step 1: Save the current machine state
- Step 2: Load the machine state for interrupt handling
- Step 3: Invoke the corresponding ISR
- Step 4: Resume the program execution

Question:

Why do we need to save the current machine states?

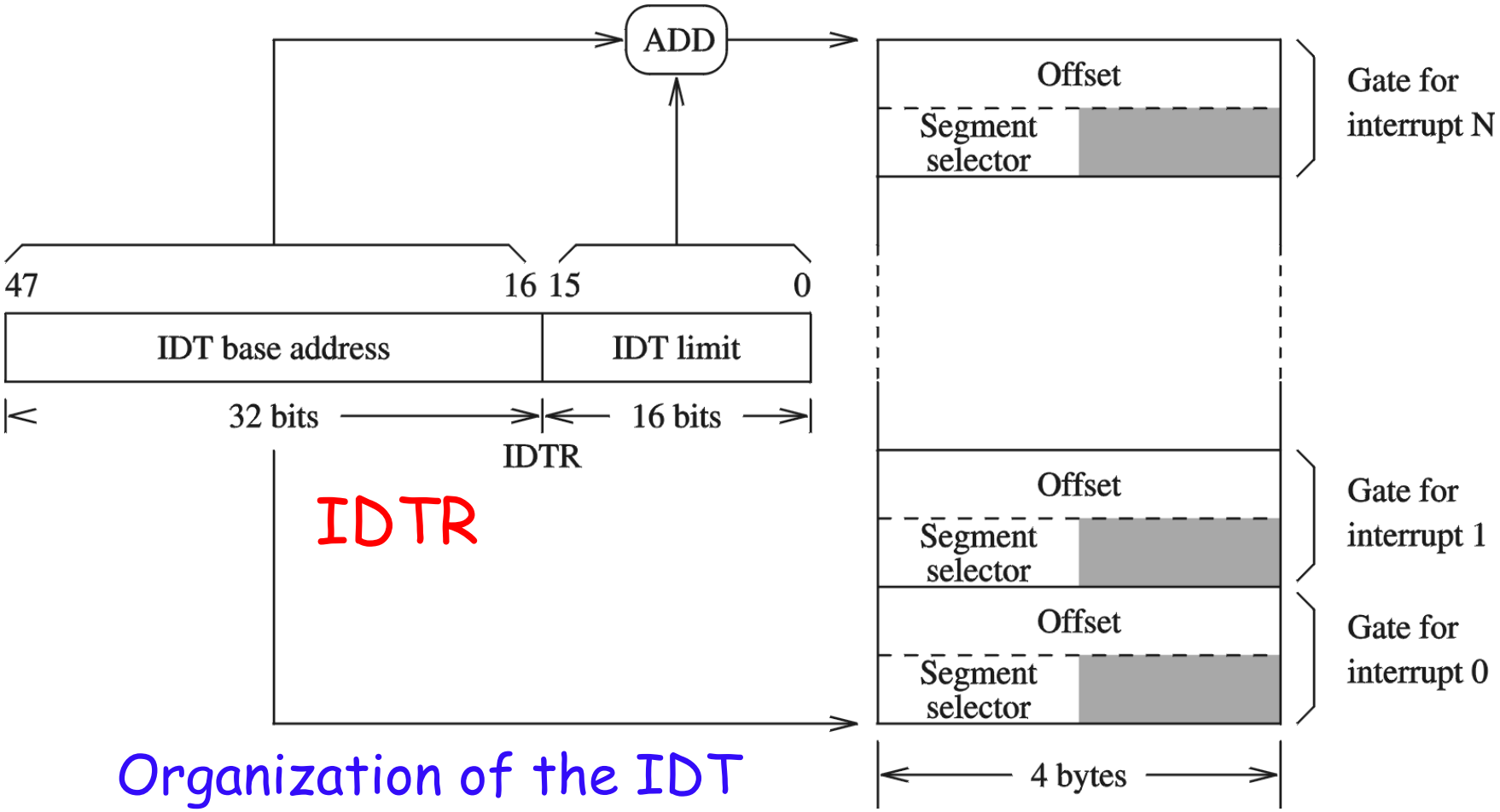
Step 1: Save the Current Machine State

- Push the EFLAGS register onto the stack
- Clear interrupt enable and trap flags
  - This disables further interrupts
  - Use `cli` to clear interrupts
  - Use `sti` to enable interrupts
- Push CS and EIP registers onto the stack

Step 2: Load the Machine State for Interrupt Handling

- Load CS (code segment register) with the 16-bit segment selector from the interrupt gate
- Load EIP (instruction pointer register) with the 32-bit offset value from the interrupt gate

Protected Mode Interrupt Processing

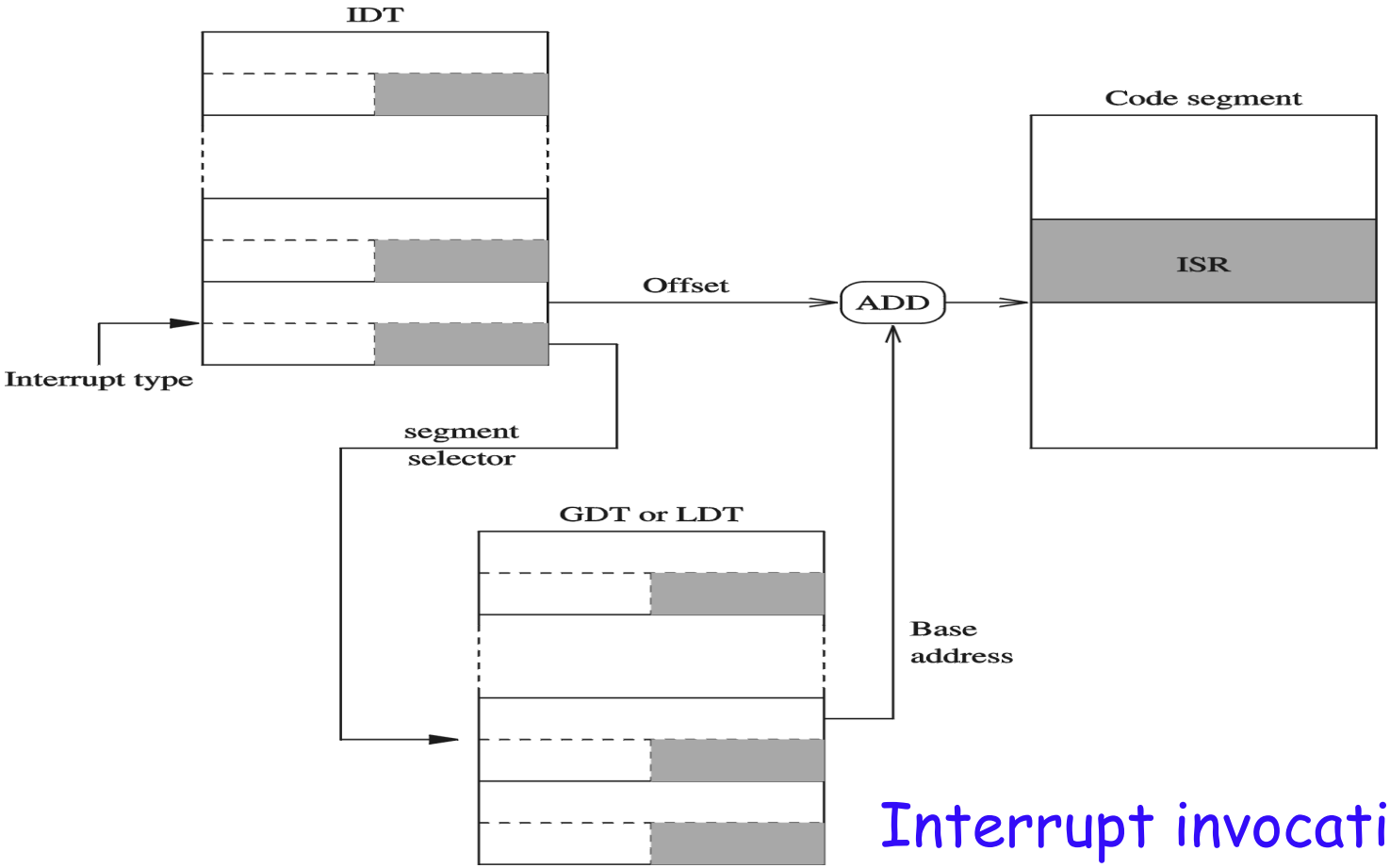


Organization of the IDT

Protected Mode Interrupt Processing

- IDTR contains the memory location of IDT
- IDTR is a 48-bit register
  - 32 bits for IDT base address
  - 16 bits for IDT limit value
    - IDT requires only 2048 (11 bits)
    - A system may have smaller number of descriptors
      - Set the IDT limit to indicate the size in bytes
- Two special instructions to load (**`lidt`**) and store (**`sidt`**) IDT
  - Both take the address of a 6-byte memory as the operand

Protected Mode Interrupt Processing



Interrupt invocation

Step 3: Invoke the ISR

- ISR: Interrupt-specific service routine
- Examples:
  - Single-step
  - Breakpoint
  - Timer
  - Page fault
  - …

Step 4: Resume the Program Execution

- What is the last instruction in an ISR:
  - **`iret`**
- The actions taken on **`iret`** are:
  - pop the 32-bit value on top of the stack into EIP
  - pop the 16-bit value on top of the stack into CS
  - pop the 32-bit value on top of the stack into the EFLAGS register
- As in procedures, make sure that your ISR does not leave any data on the stack
  - Match your push and pop operations within the ISR

An Example:

- Timer interrupt handler
  - Related files: sys/clkint.S sys/clkinit.c
  - Interrupt rate – based on clock timer
    - ctr1000: 1ms
  - Scheduling rate:
    - Interrupt rate * QUANTUM
- You will be familiar with page fault handler in PA 3!