Exceptions and Interrupts

November 15

CSC201 Section 002

Fall, 2000

Concepts from Operating Systems

- Many services are provided automatically to programs
  - Some services must be requested explicitly, via a subroutine call
- Tasks, or processes
- State of a task: executing, ready, blocked
- Scheduling: determine which task is executing
- Multitasking: interleave the execution of a group of ready tasks
  - each gets a "time-slice"

Operating System Dependence

- Programs compiled under one O.S. (e.g., Windows) will not run under another O.S. (e.g., Linux)
- Multiple operating systems can run on the same hardware, although not simultaneously
- Java Virtual Machine provides platform (hardware+O.S.) independence

Types of Exceptions

- Synchronous (program controlled) vs. asynchronous (external) exceptions
- Traps and OS calls are synchronous exceptions
- Trap = user program created some unusual condition requiring attention. Examples:
  - Attempt to execute illegal instruction
  - Stack overflow detected
  - Divide by zero detected
  - Attempt to access "protected" memory
- Traps are conditions detected by hardware

Types of Exceptions (cont.)

- OS calls are needed to invoke services. Examples:
  - I/O request
  - Process communication
- Asynchronous exceptions are due to external (asynchronous) causes: interrupts! Examples:
  - I/O device needs attention
  - Time-slice expiration
  - User input detected

Interrupt Execution (Basic Steps)

1. (A user program is running)
2. Hardware determines an interrupt is needed
3. During each instruction fetch, the processor checks if...
   - an interrupt has been requested
   - interrupts are enabled
4. If so, the processor sends an acknowledgment signal to the source of the interrupt
   - This tells the source to stop requesting service

Interrupt Execution (Basic Steps)

5. The processor saves the state of the running task (user program) on a special stack

6. Hardware loads the address of an interrupt service routine (ISR) into the program counter (EIP register for Pentium)
   - This forces a jump to the interrupt service routine

7. ISR determines the cause of the interrupt and executes the appropriate response

8. ISR restores user program state, including EIP value
   - forces a return to the user program

Interrupt "Masking"

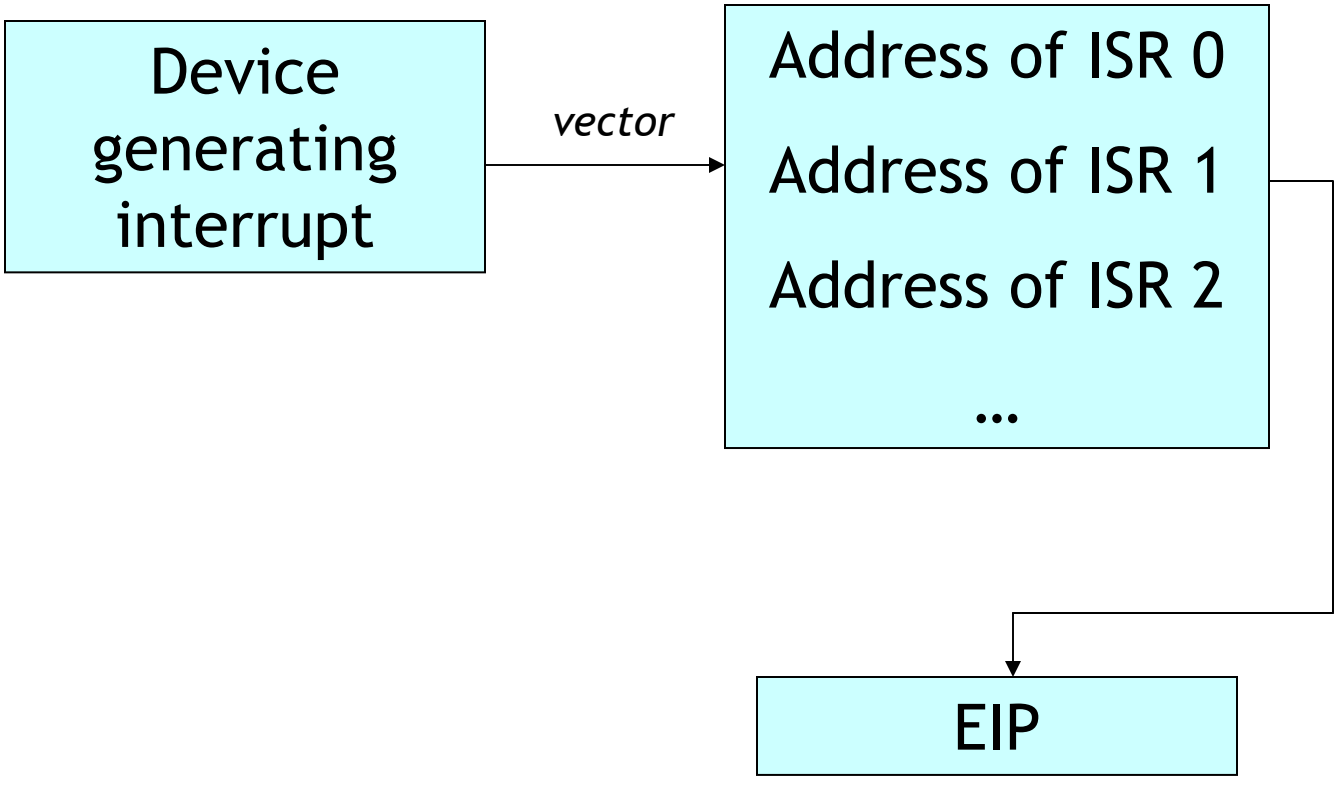
- A bit (interrupt flag, or IF) is used to indicate that interrupts will *not* be processed
  - Interrupts "enabled" (=1) or "disabled" (=0)
- CLI and STI instructions to disable (clear) and enable (set) the IF
- Why disable interrupts?
  - Some things are more important than others
  - Some things must be finished if they are started; should not be partially completed
- But, don't disable for too long! Some devices can't wait

Determining The Reason for an Interrupt

- Two possibilities (not used in Pentium)
  - Use multiple interrupt request lines (wires)
  - Interrupt service routine queries the possible interrupt sources (polling)
- Another possibility (used in Pentium): "vectored" interrupts
  - the interrupt source provides an identifier of which interrupt service routine to use
  - The identifier = index in a table of ISR starting addresses

Interrupt Descriptor Table

- Located in memory at an address indicated by the IDTR register
- Each table entry (8 bytes long) contains
  - Address of the start of an interrupt service routine (ISR)
  - Privilege level of the service routine (discussed later)



Interrupt Descriptor Table

- The table can be up to 256 entries long. Certain entries are preassigned for the Pentium
  - 0 = divide by 0 exception
  - 3 = breakpoint detected
  - 6 = invalid opcode exception
  - 13 = general protection fault exception
  - ...

Saving the Task State

- There is a stack used exclusively for exceptions
  - not the same as stack defined in a user's program
- Essential to save
  - EIP
  - EFLAGS
  - ESP
  - Segment registers (CS and SS) and descriptors

Hardware Interrupt Initiation

- First, disable interrupts!
- Save the task state
- Acknowledge the interrupt (turn off interrupt request)
- Disable lower priority interrupts (see below)
- Read ISR address from the Interrupt Descriptor Table
- Transfer control to the ISR

Interrupt Service Routine

- (no input parameters, no value returned)
- Save user-level registers
- Enable interrupts
  - ISRs can be interrupted!
  - Like nested procedure calls
- Performs actions appropriate to the cause of the interrupt

Returning from the ISR

- Disable interrupts
- Restore registers
- Execute "iret" instruction
  - restore task state (including interrupt flag)
  - return to interrupted instruction