Lab 3 Primer

Patrick Lowry, T.A.
CS439H, Dr. Elnozahy
October 3, 2012

x86 Interrupts and Exceptions

- Both transfer control to an “interrupt service routine (ISR)”
    - ISRs can see the state of the machine’s registers as they were at the time the event occurred.
- What is the difference between an Interrupt and an Exception?
    - Interrupts see the state of the machine after the instruction that was executing when the event occurred. Can also be “masked” by processor.
    - Exceptions see the state of the machine before the instruction that caused the event to occur. Exceptions cannot be masked.

x86 Interrupts and Exceptions

- What causes them?
    - Interrupts are caused by:
        - External devices requesting OS attention.
        - Application software requesting OS attention.
    - Exceptions are caused by:
        - Software (application or OS) doing something wrong.
        - Intel refers to these as “faults.”

x86 Interrupts and Exceptions

- Masking Interrupts:
    - CLI instruction: clears the “Interrupt Flag (IF)” in the FLAGS register
        - Clearing has the effect of masking interrupts
    - STI instruction: sets the “Interrupt Flag (IF)” in the FLAGS register
        - Setting has the effect of allowing interrupts
- Both are privileged instructions that can only be executed in ring 0.

x86 Interrupts and Exceptions

- How are ISRs set up?
    - In real mode: through the Interrupt Vector Table (IVT)
    - In protected mode: through the Interrupt Descriptor Table (IDT)
- In both the IVT and the IDT, the first 32 entries are reserved for exceptions
- The rest of the entries (indices 32 - 255) are available for use with interrupts

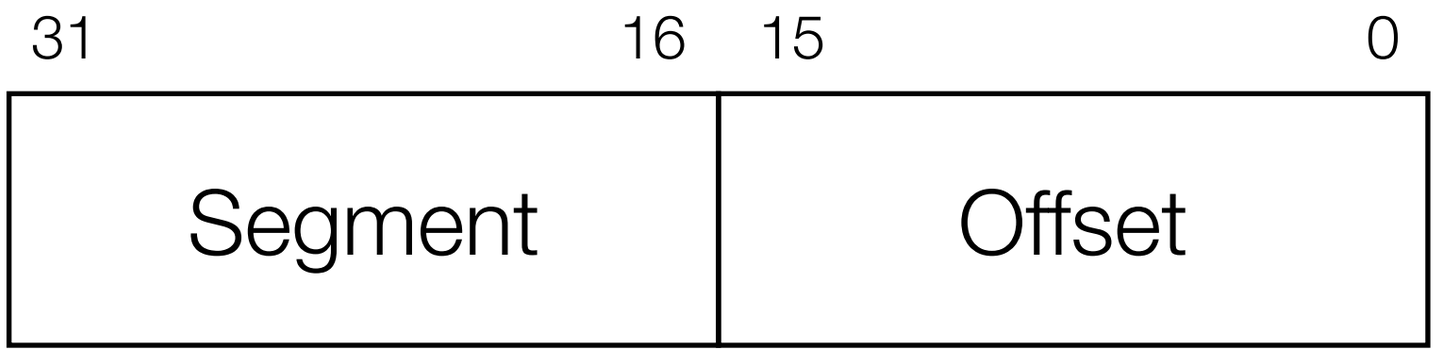
x86 Interrupts and Exceptions

- Notable “Hard-Wired” Exceptions:

Number	Description
0	Divided by zero
6	Undefined instruction
8	Double Fault
11	Not Present
13	General Protection
14	Page Fault

x86 Interrupt Vector Table (IVT)

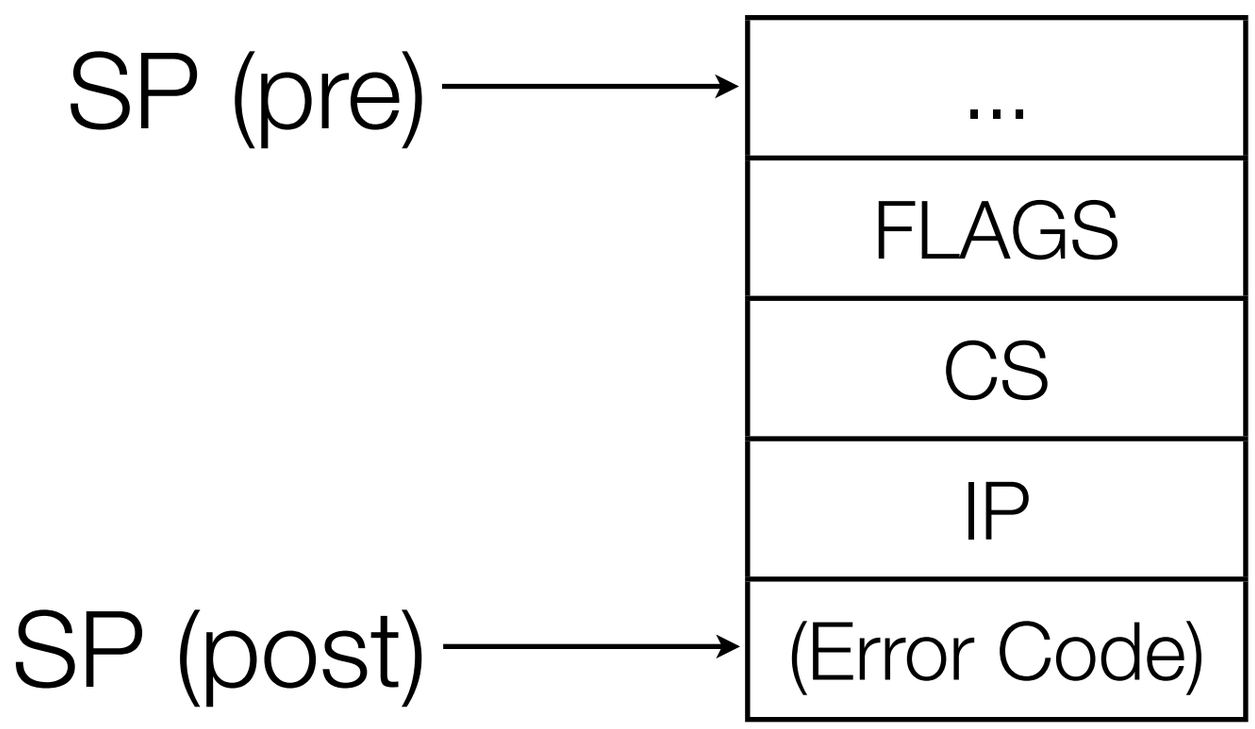
- Starts at 0000:0000
- Each entry is four bytes, specifying the real address (seg : off) of the ISR



x86 Real Mode Interrupt / Exception Handling

- Step 1: Save state

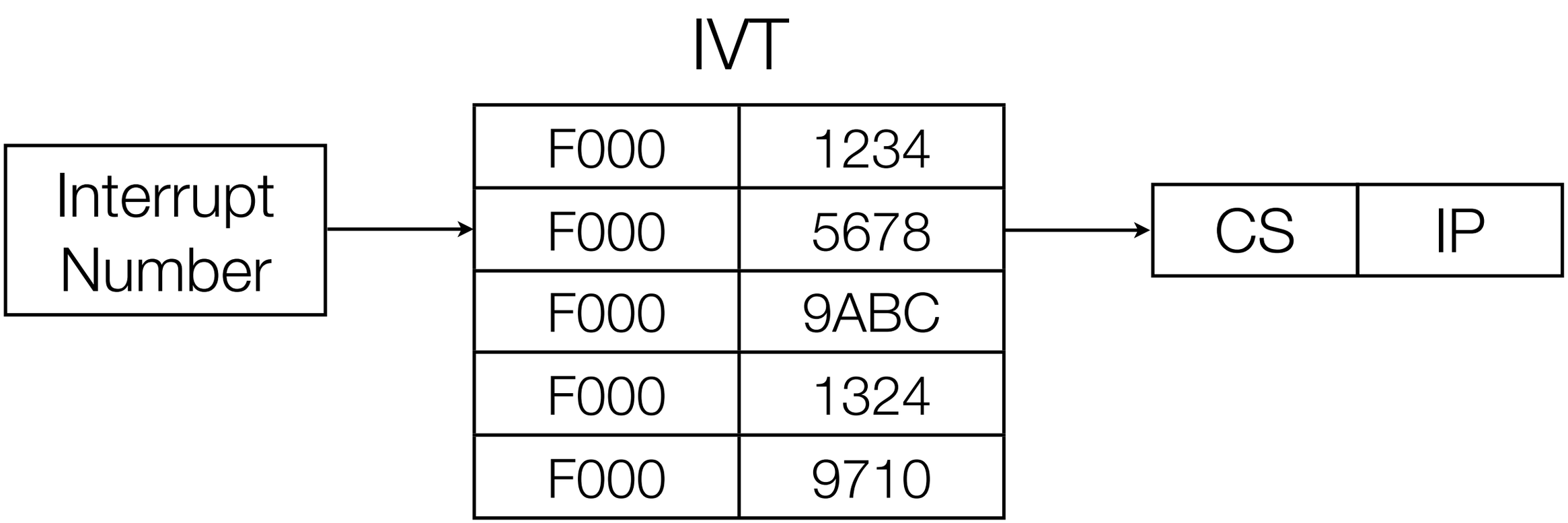
- PUSH (on the normal stack): FLAGS, CS, IP, and if necessary, an Error Code



x86 Real Mode Interrupt / Exception Handling

- Step 2: Jump to ISR

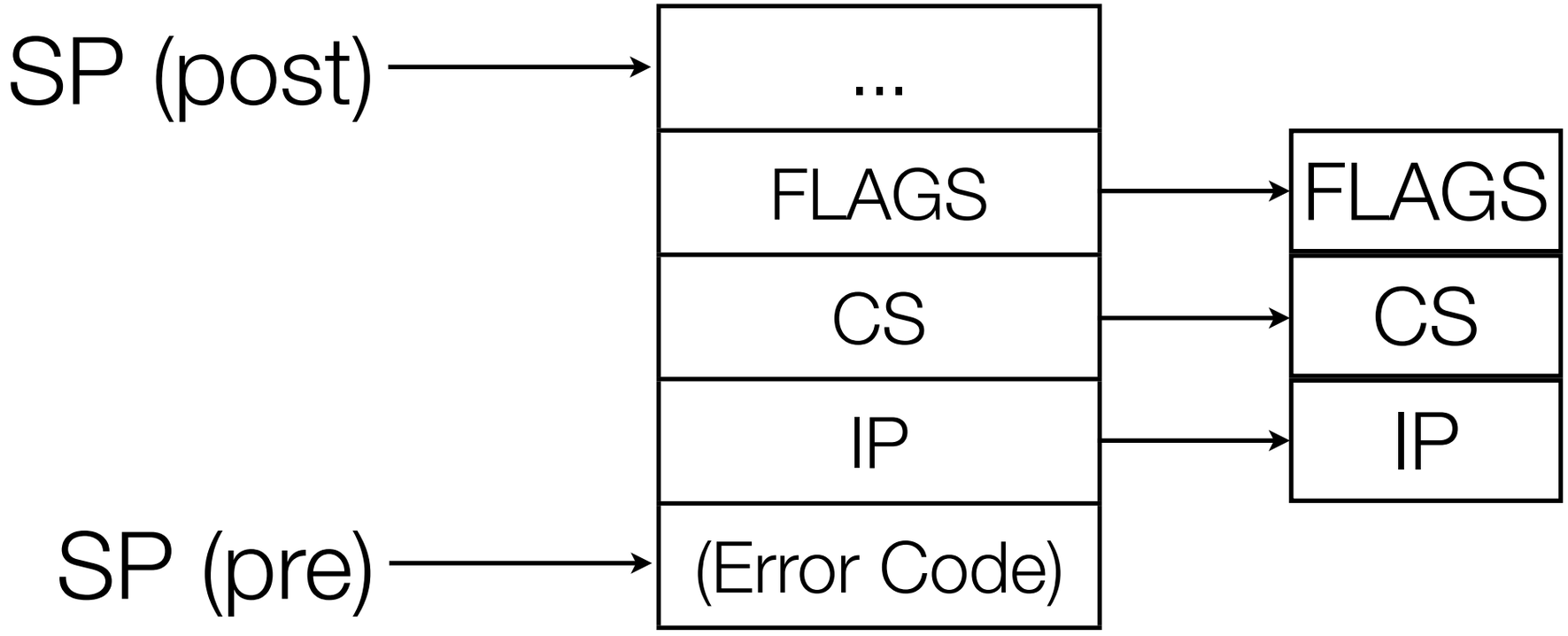
- Index IVT; load CS and IP



x86 Real Mode Interrupt / Exception Handling

- Returning from ISR: IRET instruction

- POP: Error Code (if necessary), IP, CS, FLAGS



x86 Software-Generated Interrupt

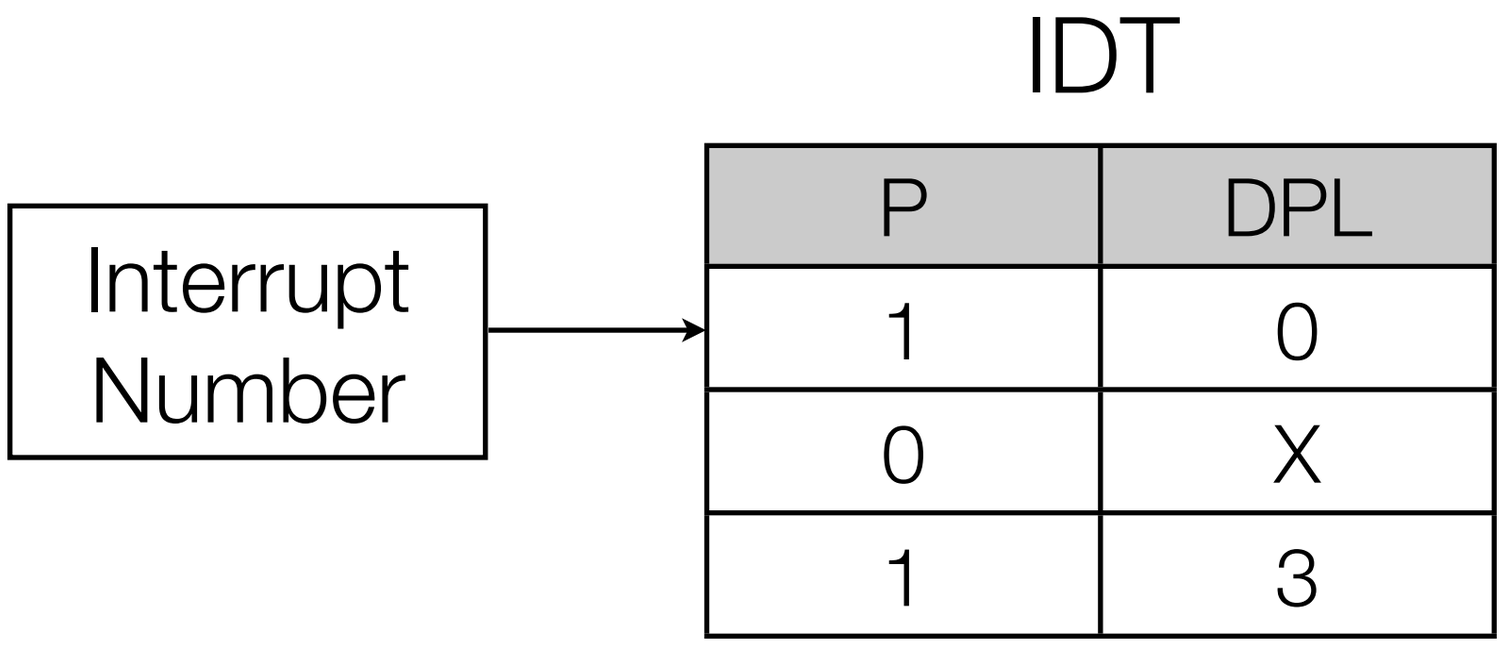
- INT instruction
  - Takes a constant operand: the interrupt number you wish to generate
  - Register / memory operands are not allowed (assembler error)
    - Privileged instruction (can only be executed in ring 0)
- Examples:
  - Valid: int $0x80
  - Invalid: int (%eax)

Interrupt Descriptor Table (IDT)

- Each entry contains:
    - 32-bit address of ISR
    - 16-bit CS segment selector to use when running the ISR
        - Note that this includes the CPL that will be used when executing
    - “Present” bit
    - Required privilege level to call this ISR via the INT instruction
    - Bit indicating whether or not to clear IF when entering the ISR

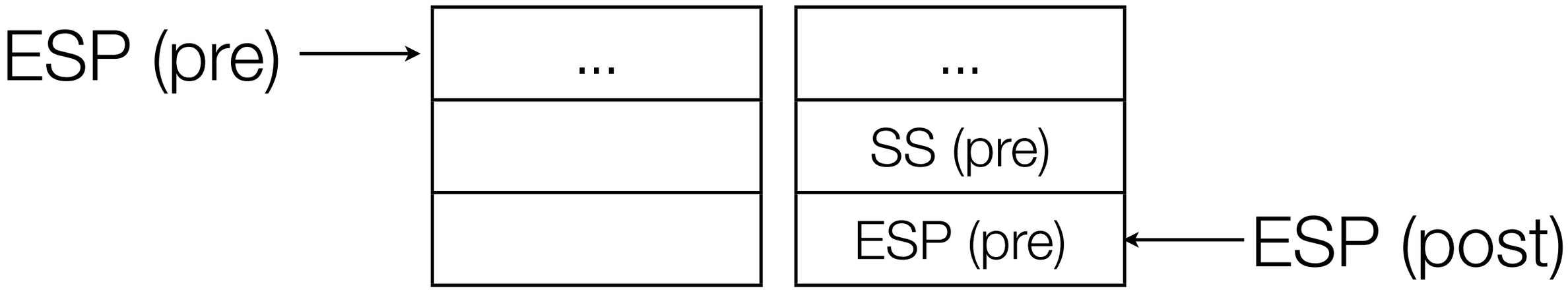
x86 Protected Mode Interrupt / Exception Handling

- Step 1: Index the IDT and make sure the corresponding entry is Present.
    - What if it’s not?
- Step 2: If this is a software-generated interrupt, check that the current privilege level is sufficient for this interrupt.
    - What if it’s not?



x86 Protected Mode Interrupt / Exception Handling

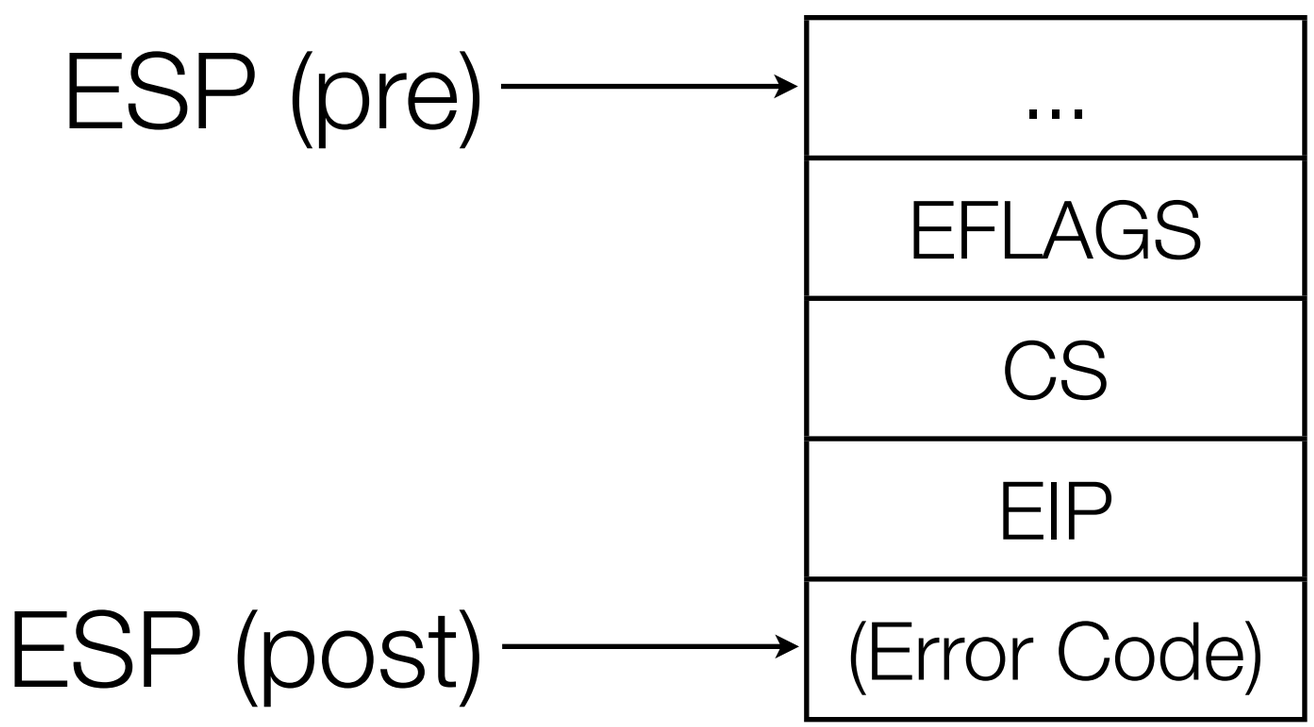
- Step 3: If there will be a change of privilege by jumping to the ISR, switch to the new privilege level's SS and ESP, and PUSH the old SS and ESP on the new stack.
    - Where are each privilege level's SS and ESP stored?



x86 Protected Mode Interrupt / Exception Handling

- Step 4: Save state.

- PUSH: EFLAGS, CS, EIP, and if necessary, an Error Code



x86 Protected Mode Interrupt / Exception Handling

- Step 5: Jump to ISR.

- Index into IDT and load CS & EIP