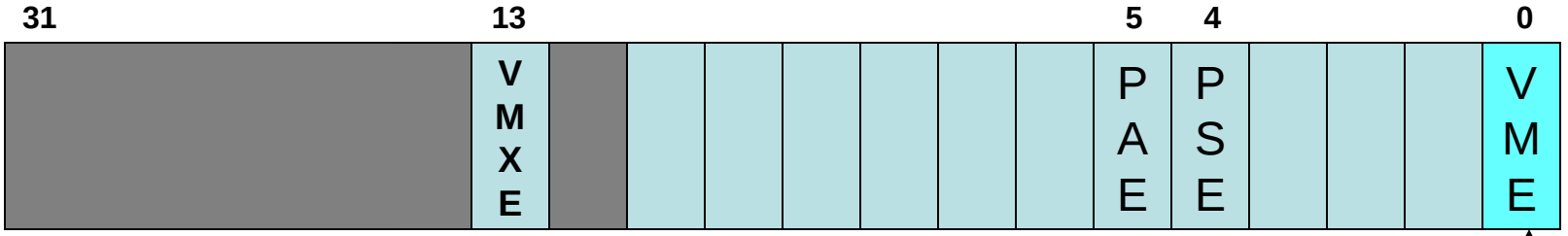
Interrupts in the guest VM

A look at the steps needed to “reflect” hardware interrupts back into the ROM-BIOS for servicing

The VME-bit in CR4

- Our VMX demo-program set the VME-bit (bit #0) in Guest's Control Register CR4



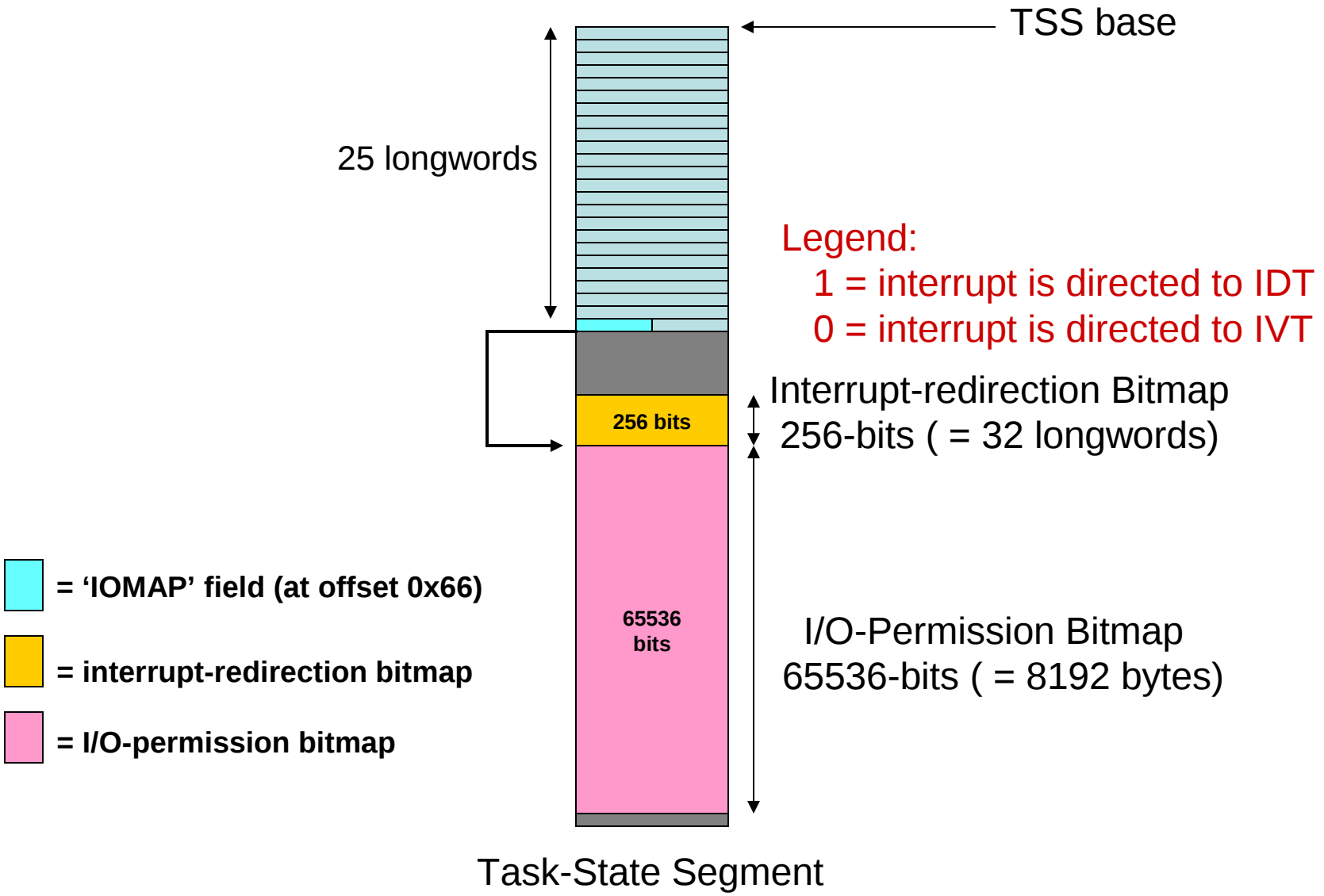
Legend:

VME (Virtual-8086 Extensions): 1=on, 0=off
PSE (Page-Size Extensions): 1=on, 0=off
PAE (Page-Address Extensions): 1=on, 0=off
VMXE (Virtual Machine eXtensions Enabled): 1=yes, 0=no

Virtual-8086 Mode Extensions

- Software interrupt instructions (int $nn) will selectively be directed either to IDT-gates or to IVT-vectors, depending on a ‘bitmap’ located within the Task-State Descriptor
- This ‘interrupt redirection bitmap’ has 256 bits (one for each 8-bit interrupt-number)
- Its location within the TSS is immediately ahead of the I/O Permission Bitmap

Interrupt-redirection Bitmap



Software INTs Only!

- The interrupt-redirection bitmap does NOT affect any ‘hardware’ interrupts – they are serviced by the interrupt-handlers whose entry-points are specified within the gate-descriptors that comprise the IDT
- How can the Guest VM in our VMX demo-program handle the ‘hardware’ interrupts generated by the peripheral devices?

We’ll modify our VMX demo

- One change to ‘vmxstep3.s’:

```
guest_RFLAGS: 0x00023202     # IF=1, IOPL=3
```

- One change to ‘vmxdemo.s’:

```
in  $0x21, %al          # get master-PIC’s mask
or  $0x10, %al          # mask UART interrupt
out %al, $0x21          # set master-PIC’s mask
```

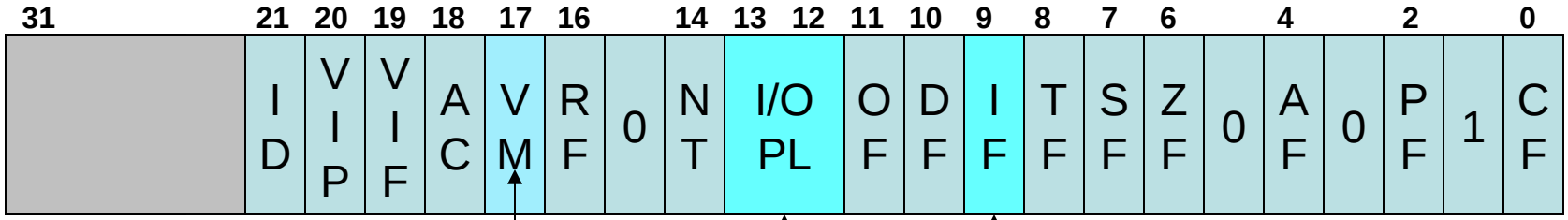
Modify ‘guest_isrGPF’

- We introduce a major modification into the guest’s General Protection Fault-handler, to “reflect” external device-interrupts back to ‘real-mode’ code in the ROM-BIOS that will be executed in ‘Virtual-8086 mode’
- The steps needed to do this are based on ‘emulating’ the CPU’s usual response to an external interrupt in 8086 real-mode

CPU’s interrupt-response

- Push FLAGS register onto the stack
- Clear IF and TF bits in FLAGS register
- Push CS and IP registers onto the stack
- Acquire the device’s interrupt-ID number
- Lookup that ID-number’s interrupt-vector
- Put that vector’s ‘loword’ into IP register
- Put that vector’s ‘hiword’ into CS register
- Then resume CPU’s fetch-execute cycle

EFLAGS



VM (Virtual-8086 Mode):
1=on, 0=off

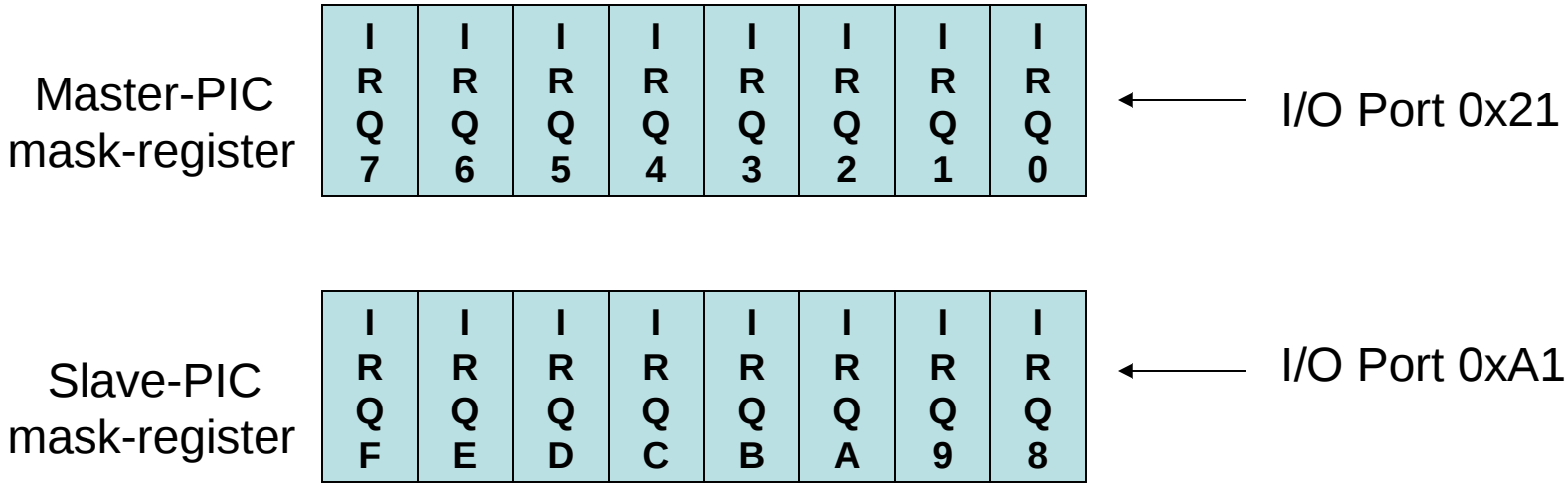
IF (Interrupt-Flag):
1=on, 0=off

IOPL (Input/Output Permission-Level):
=00 (only ring0 can execute ‘in’ and ’out’)
=01 (ring0 and ring1 can execute ‘in’ and ‘out’)
=10 (ring0, ring1, ring2 can execute ‘in’ and ‘out’)
=11 (ring0, ring1, ring2, ring3 can execute ‘in’ and ‘out’)

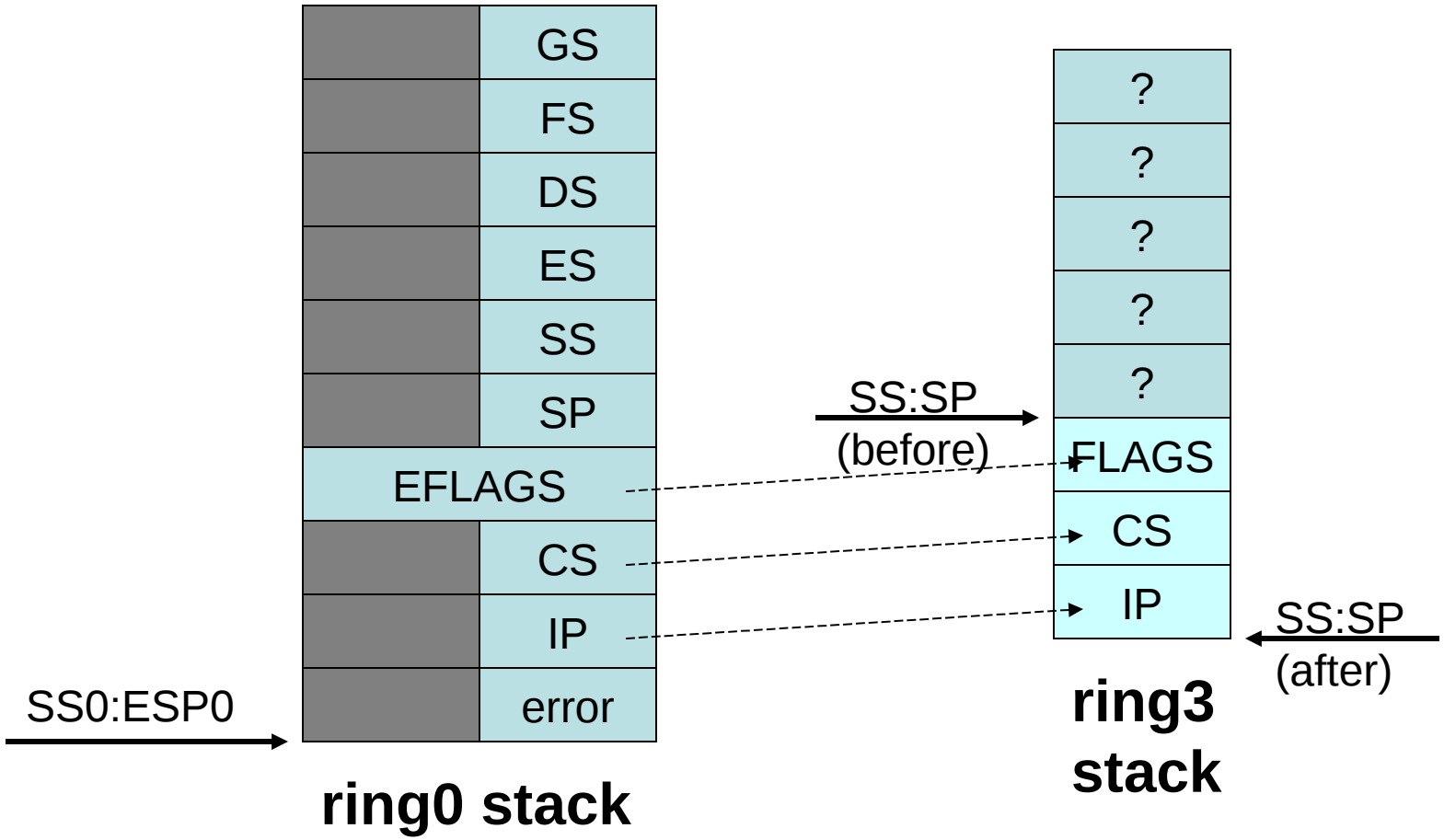
NOTE: Virtual-8086 mode operates at the ‘ring3’ privilege-level

PIC masks

- Each Programmable Interrupt Controller has a ‘mask register’ that allows blocking of the interrupts from specific devices



GPF stack-frame



GPF error-code

Legend:

EXT (External-event): 1=yes, 0=no
INT (Interrupt-table): 1=yes, 0=no
TI (Table-Indicator): 1=LDT, 0=GDT

Index = Table’s element-number

GPF stack-frame