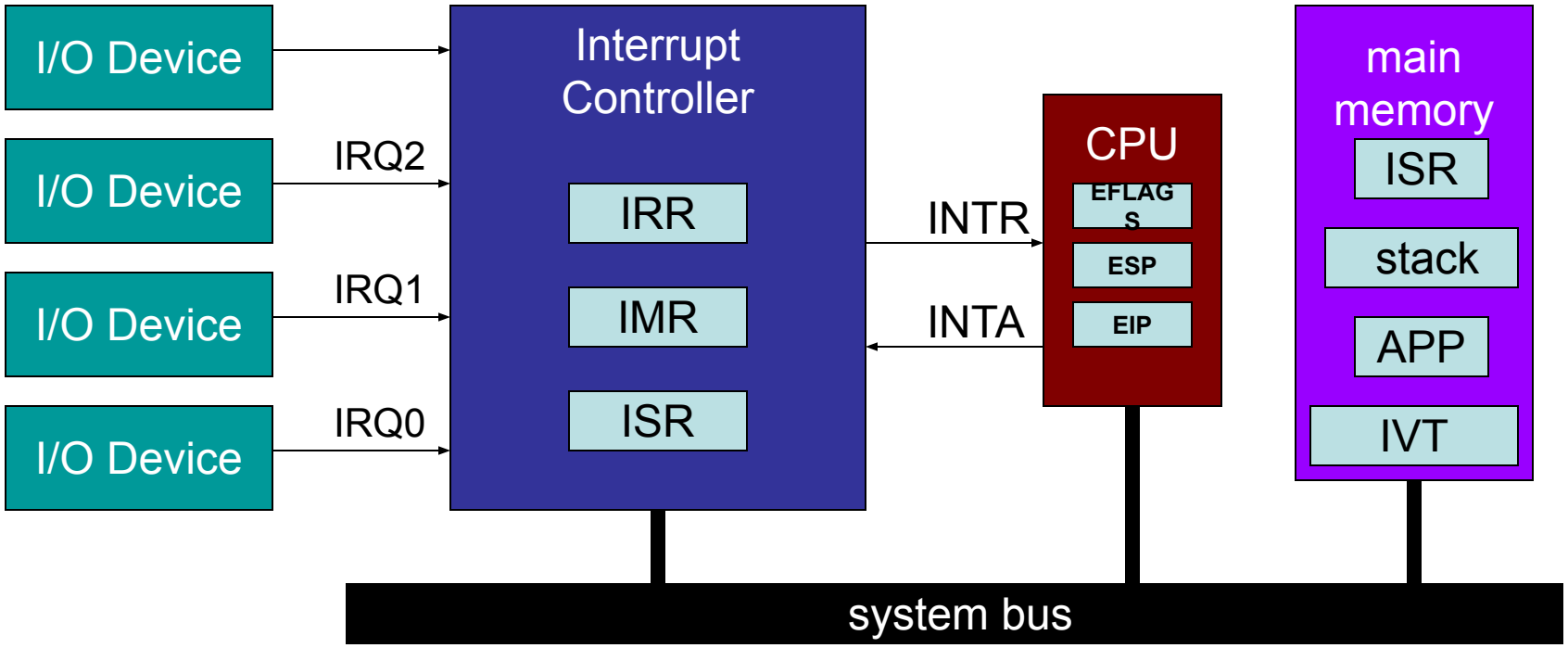
Message Signaled Interrupts

A look at our network controller's optional capability to utilize Message Signaled Interrupts

The ‘old’ way

- In order to appreciate the benefits of using Message Signaled Interrupts, let’s first see how devices do interrupts in a legacy PC



Multi-step communication

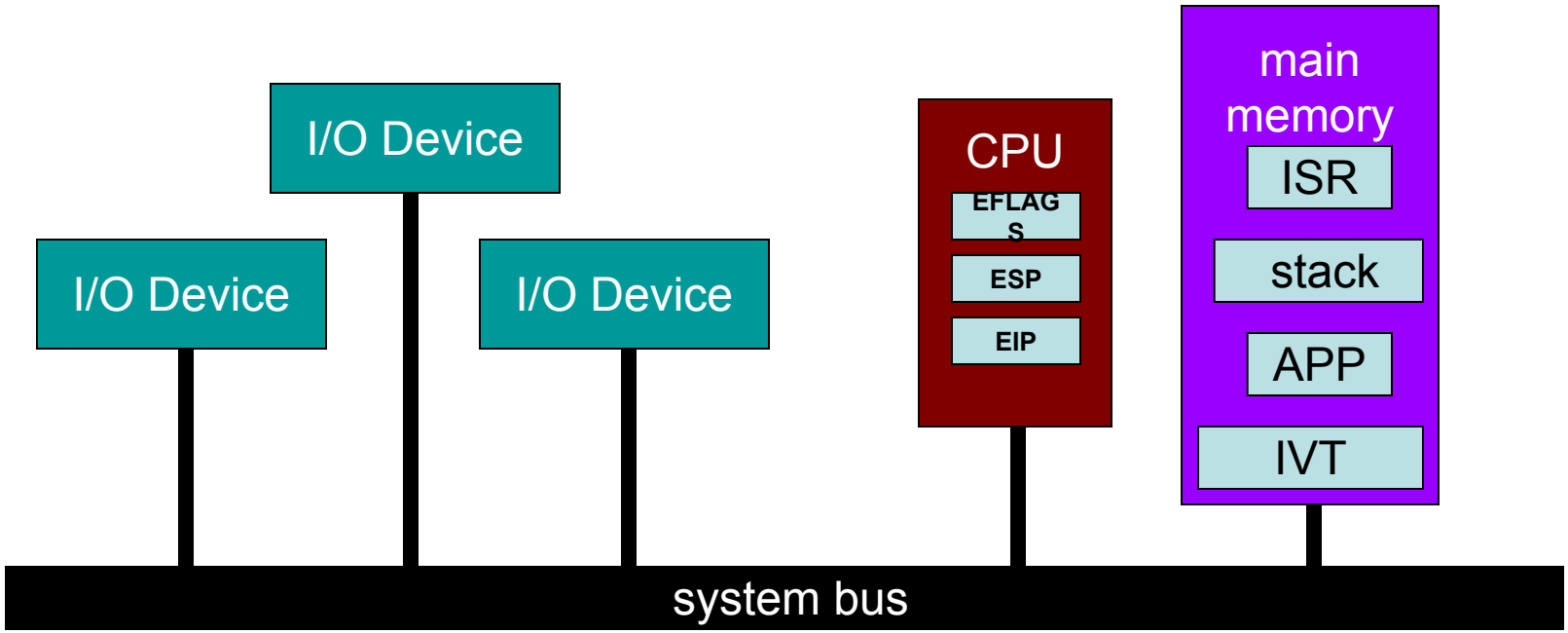
- A device signals that it needs CPU service
- The Interrupt Controller signals the CPU
- The CPU responds with two INTA cycles
  - First INTA causes bit-changes in IRR and ISR
  - Second INTA puts ID-number on system bus
- CPU uses ID-number to lookup IVT entry
- CPU saves minimum context on its stack, adjusts eflags, and jumps to specified ISR

Faster, cheaper, and more

- Faster response to interrupts is possible if the old multi-step communication scheme can be replaced by a single-step protocol
- Less expensive PCs can be manufactured if their total number of signal pins and the physical interconnections can be reduced
- More devices can have their own ‘private’ interrupt(s) if signal lines aren’t required

The ‘new’ way

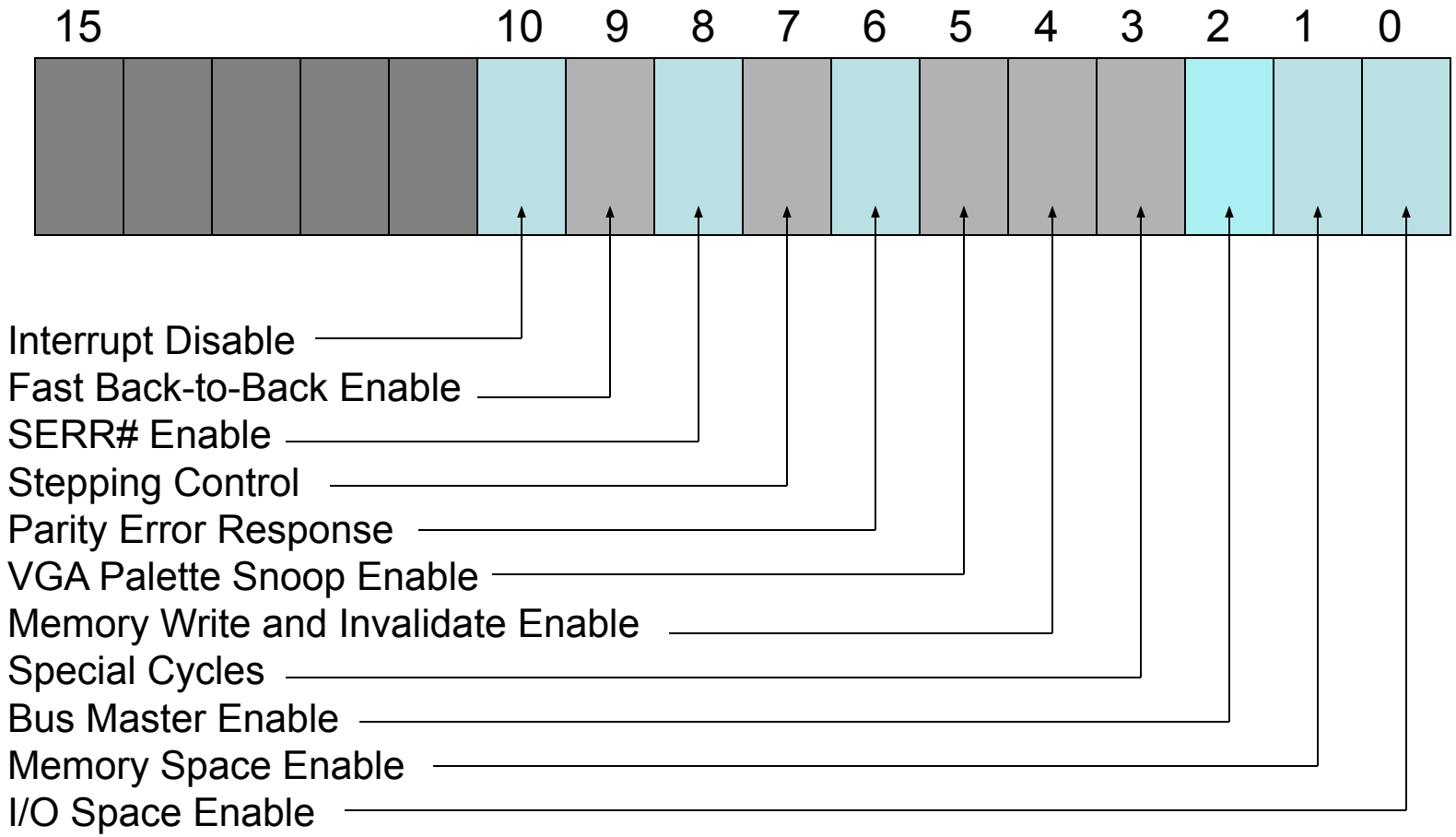
- Message Signaling allows all the needed information to arrive in a single package, and go directly from a device to the CPU



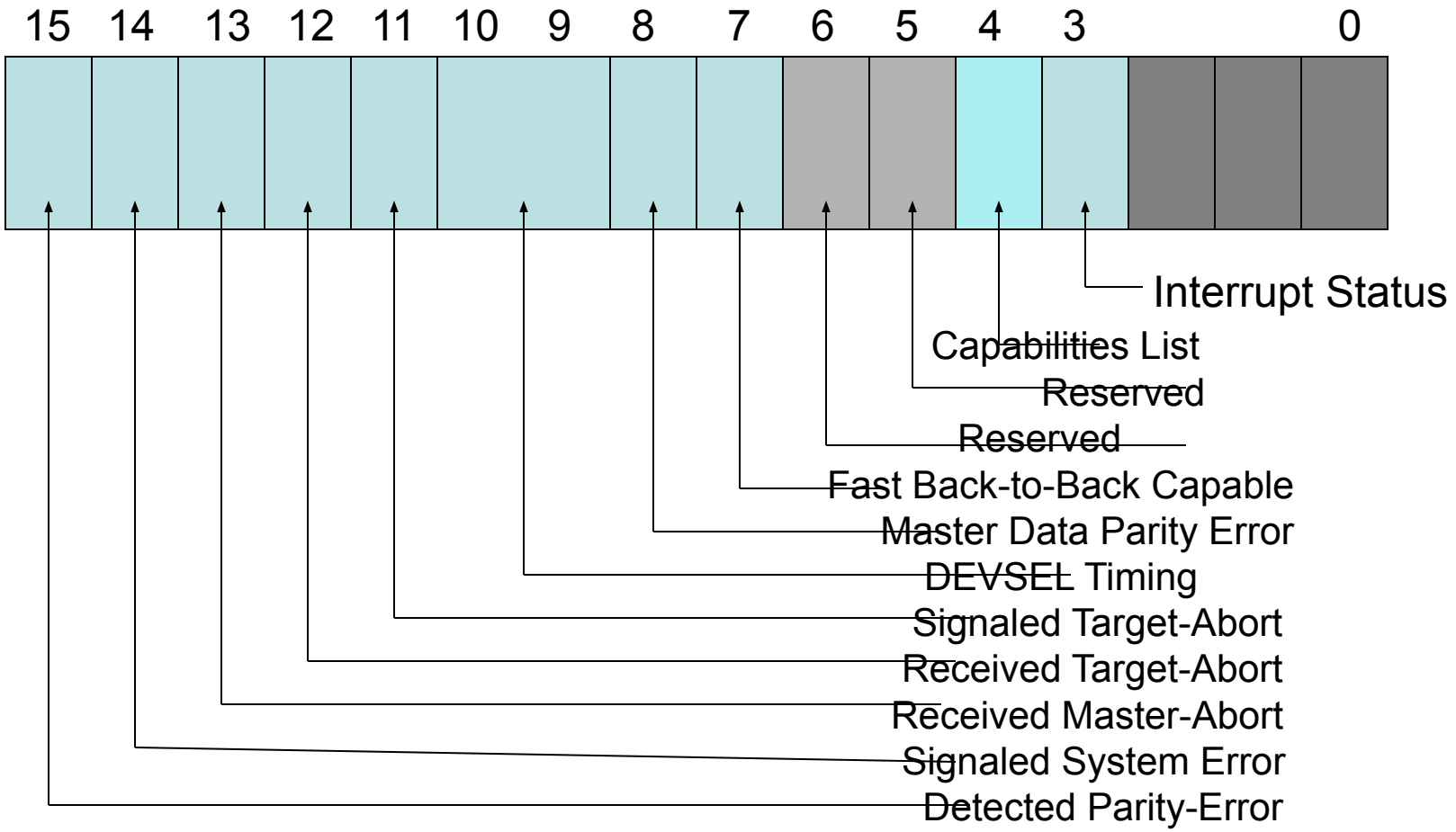
Implementation

- The customary PCI Configuration Space is modified to accommodate three additional registers, which collectively are known as the **MSI Capability Register Set**:
  - An MSI Control Register (16 bits)
  - An MSI Address Register (32 bits/64 bits)
  - An MSI Data Register (32 bits)
- (In fact these additions fit within a broader scheme of so-called “new capabilities”)

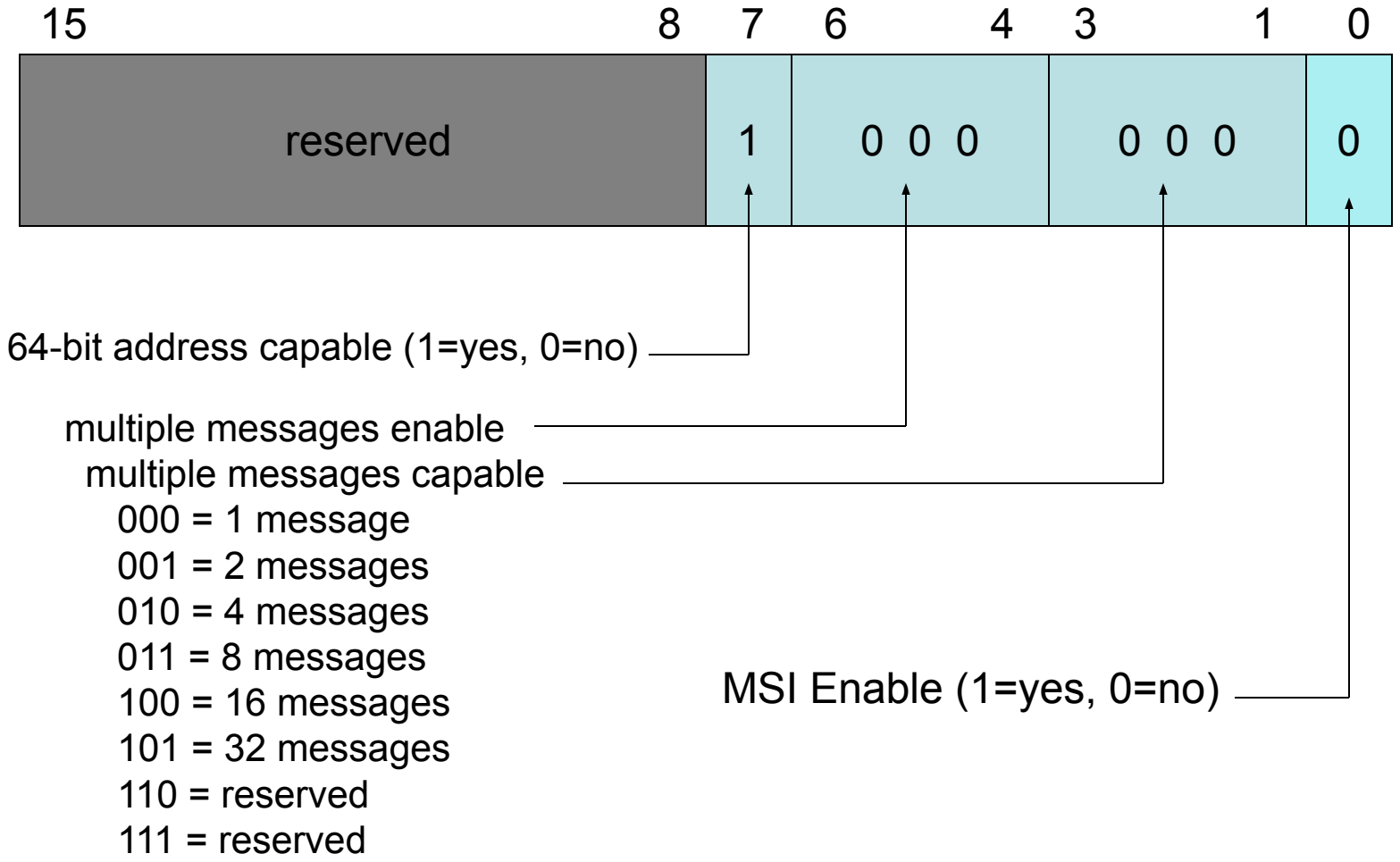
PCI Command Register



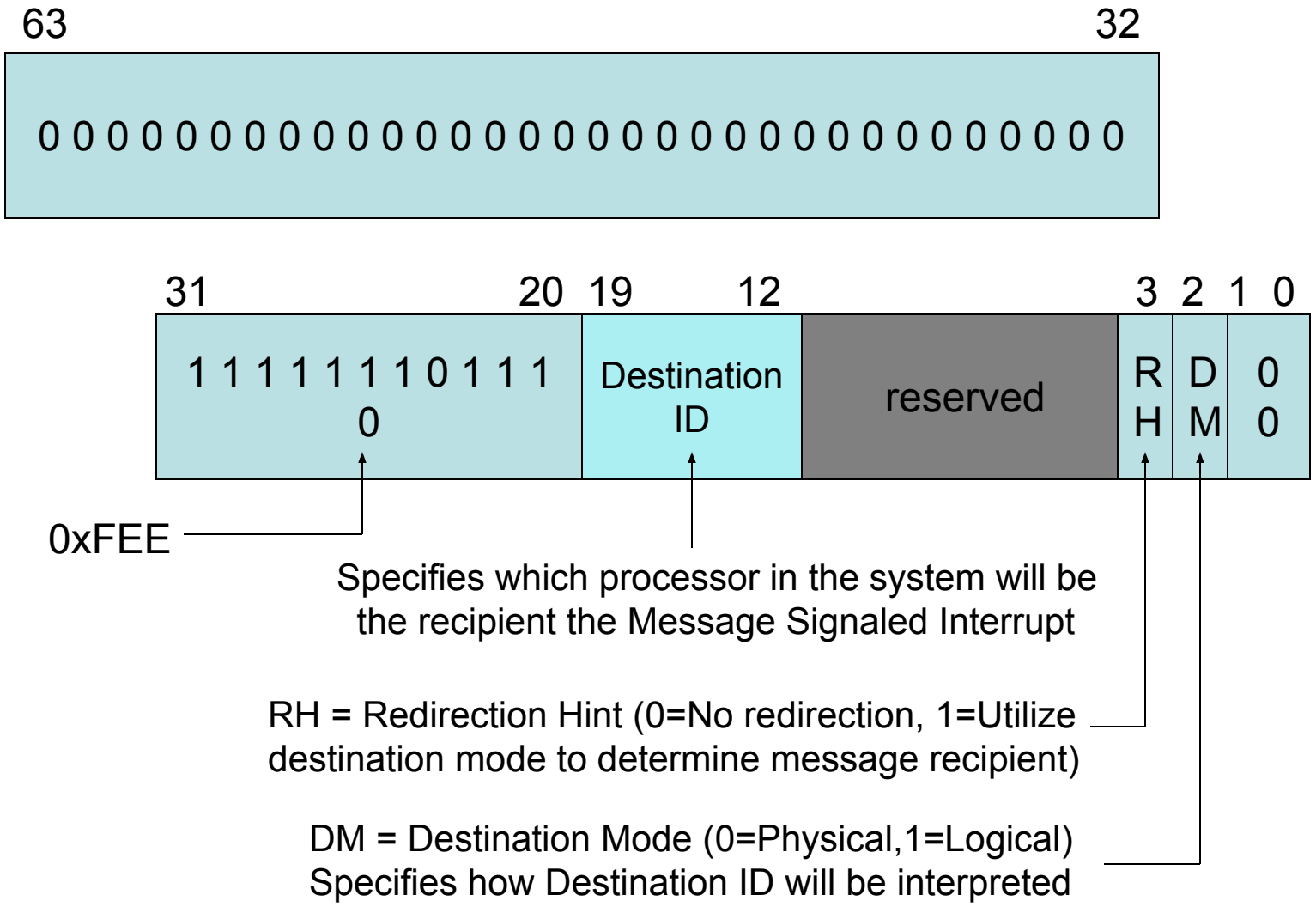
PCI Status Register



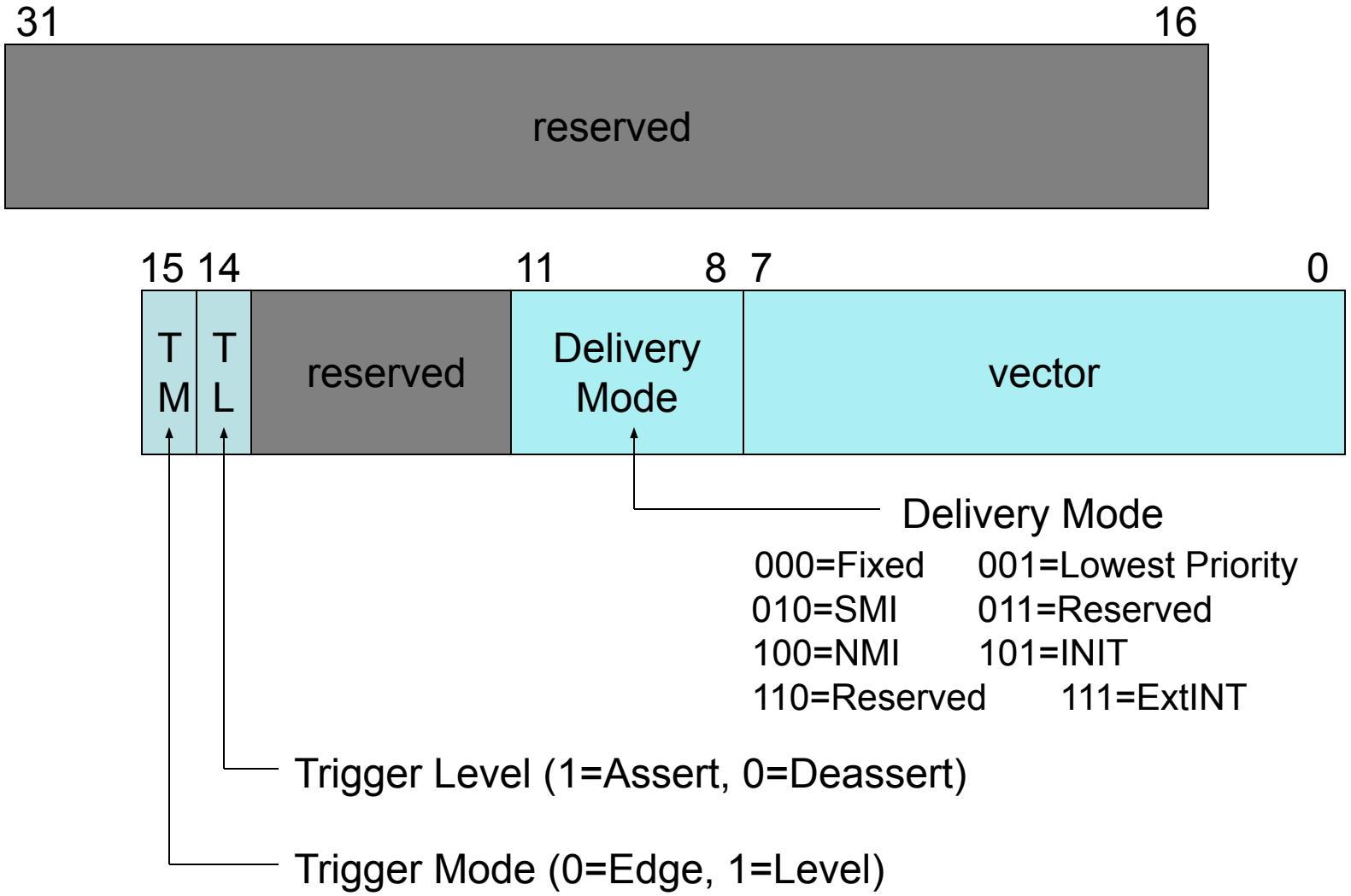
MSI Control Register



MSI Address Register



MSI Data Register



Recall NIC's interrupt registers

```
enum    {
      E1000_ICR   = 0x00C0,     // Interrupt Cause Read
      E1000_ICS   = 0x00C8,     // Interrupt Cause Set
      E1000_IMS   = 0x00D0,     // Interrupt Mask Set
      E1000_IMC   = 0x00D8,     // Interrupt Mask Clear
      };
```

Registers' usage

You use Interrupt Mask Set to selectively enable the NIC's various interrupts;
You use Interrupt Mask Clear to selectively disable any of the NIC's interrupts;
You use Interrupt Cause Read to find out which events have caused the NIC to generate an interrupt (and then you can 'clear' those bits by writing to ICR);
You can write to the Interrupt Cause Set register to selectively trigger the NIC to generate any of its various interrupts -- provided they have been 'enabled' by bits being previously set in the NIC's Interrupt Mask Register.

Demo module: ‘msidemo.c’

- This module installs an interrupt-handler for an otherwise unused interrupt-vector
- It initializes the MSI Capability Registers residing in our Intel Pro1000 controller’s PCI Configuration Space, to enable the NIC to issue Message Signaled Interrupts
- It creates a pseudo-file (‘/proc/msidemo’) that triggers an interrupt when it’s read

Tools

- The ‘unused’ interrupt-number is selected by examining the settings in the IOAPIC’s Redirection Table (e.g., for serial-UART)
- Our NIC’s PCI Configuration Space can be viewed by installing our ‘82573.c’ module and reading its pseudo-file (‘/proc/82573’)
- We can watch interrupts being generated with our ‘smpwatch’ application-program