Lecture Topics

- Programmable interrupt controller (PIC)
  - motivation & design
  - hardware for x86
  - Linux abstraction of PIC

ECE391 EXAM 1

- **EXAM 1 – March 2 (Tuesday);**
  - UIUC students: 6:00pm to 8:00pm; Illinois time (or CST)
  - ZJUI students: 8:00pm to 10:00pm; China time (which is 6:00am to 8:00am Illinois time)
  - Detailed instructions will be provided soon

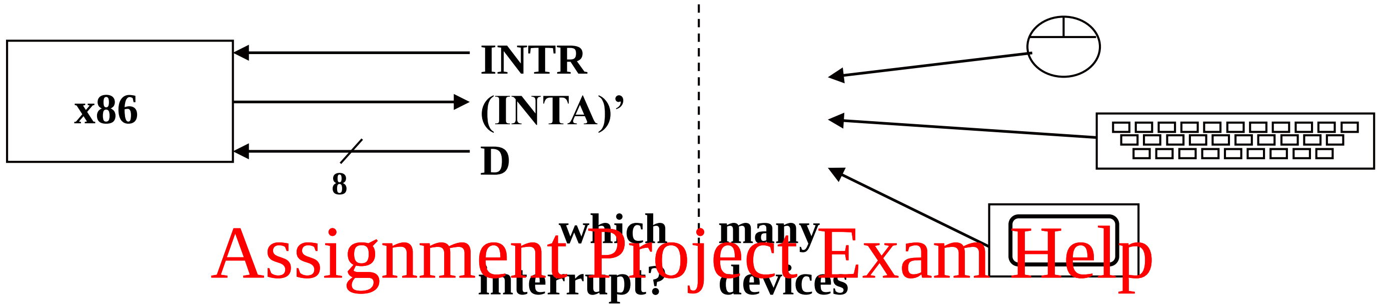
- **Conflict Exam**
  - Deadline to request conflict exam: Friday February 26 (by email to: *kalbarcz@Illinois.edu*)
- **Exam 1 Synchronous Review Session in collaboration with HKN**
  - Saturday February 27; 8:00pm; (Illinois time)
  - Zoom link will be provided later this week

ECE391 EXAM 1

- **Topics covered by EXAM 1**
  - Materia covered in lectures (Lecture1 – Lecture10)
    - x86 Assembly
    - C-Calling Convention
    - Synchronization
    - Interrupt control (using PIC)
  - Material covered in discussions
  - MP1

- **NO Lecture on Tuesday, March 2**

PIC Motivation and Design



- How do we connect devices to the processor's interrupt input?

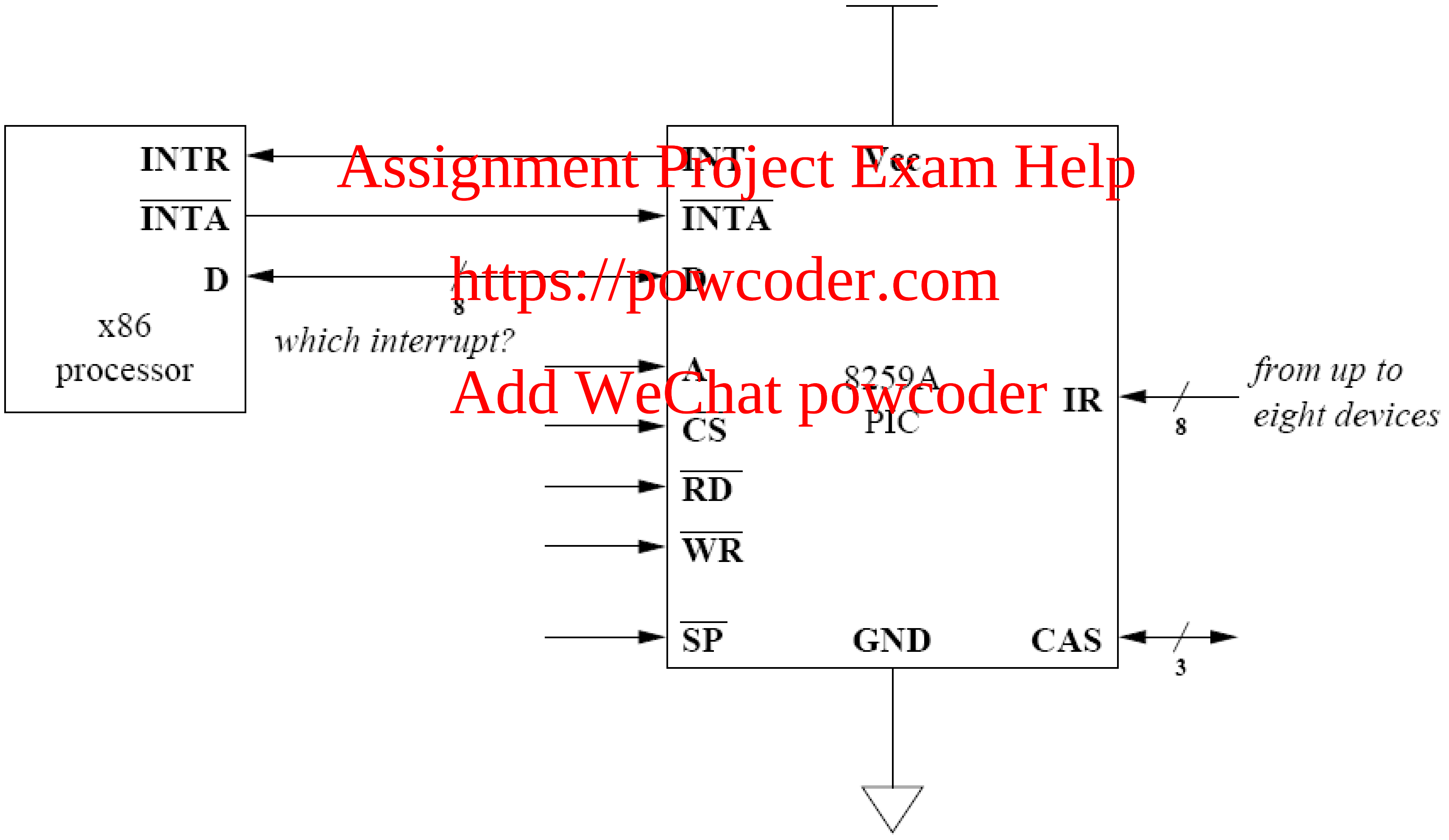
- An OR gate? why not?

  - who writes the vector # ?
    - possible to build arbiter, but…
    - what if more than one raised interrupt?
  - extra work for processor to query all devices
    - must execute interrupt code for device that raised interrupt
    - could have been more than one device
    - no way to tell with OR gate

PIC Motivation and Design (cont.)

- not all devices support query
  - many devices too simplistic to support query
  - and operations (e.g., reading from port) may not be idempotent
- nice to have concept of priority and preemption, i.e., interrupting an interrupt handler

8259A
Programmable Interrupt Controller (PIC)



Logical Model of PIC Behavior

- Watch for interrupt signals
  - from up to eight devices
  - one interrupt line each

- Using internal state
  - track which devices/input lines
  - are currently in service by processor
  - i.e., processor is executing interrupt handler for that interrupt

Logical Model of PIC Behavior (cont)

- When a device raises an interrupt
  - check if priority is higher than those of in-service interrupts
  - if not, do nothing
  - if so
    - report the highest priority raised to the processor
    - mark that device as being in service

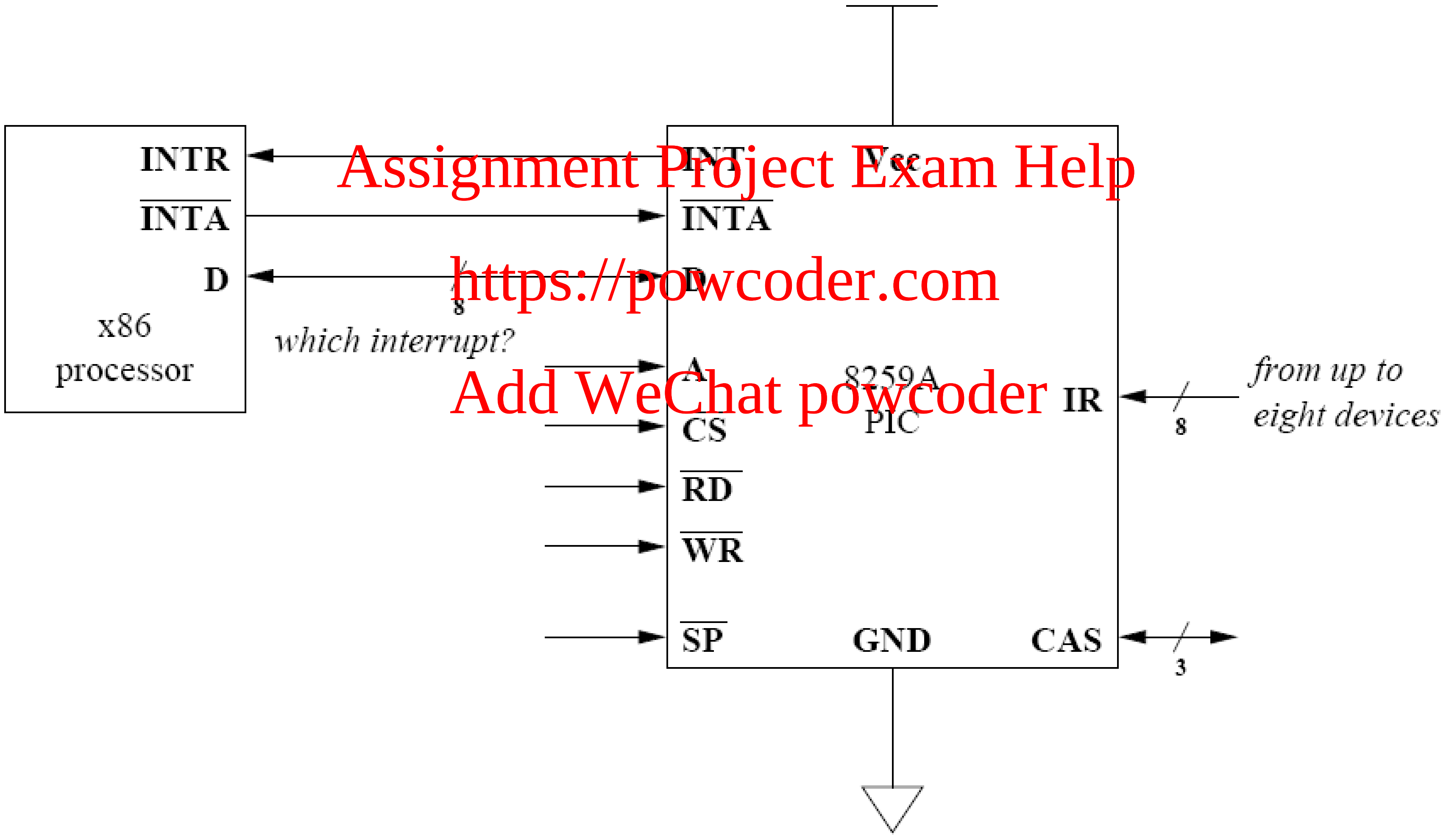
Logical Model of PIC Behavior (cont.)

- When processor reports EOI (end of interrupt) for some interrupt
  - remove the interrupt from the in-service mask
  - check for raised interrupt lines
    - that should be reported to processor

Logical Model of PIC Behavior (cont.)

- Protocol for reporting interrupts
  - PIC raises INTR
  - processor strobes INTA’ (active low) repeatedly
    - creates cycles for PIC to write vector to data bus
    - (must follow spec timing! PIC is not infinitely fast!)
  - processor sends EOI with specific combinations of A & D inputs (A is from address bus, D is from data bus)

8259A Programmable Interrupt Controller (PIC)

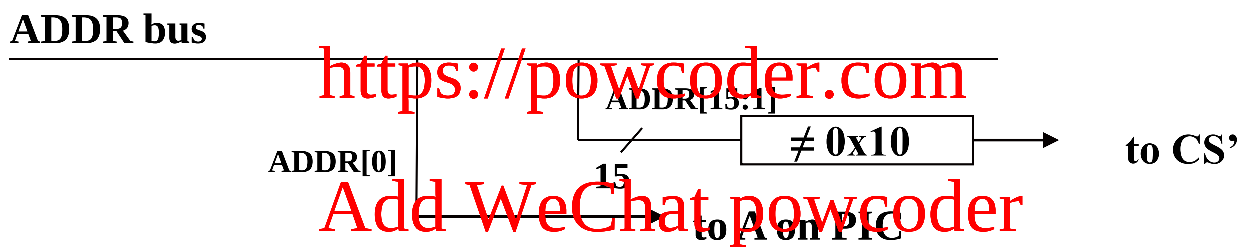


Logical Model of PIC Behavior (cont.)

- What about A, CS’, RD’, and WR’ ?
  - A = address match for ports
  - CS’ = chip select (does processor want PIC to read/write?)
  - RD’ and WR’ defined from <u>processor’s</u> point of view
    - RD’ = processor will read data (vector #) from PIC
    - WR’ = processor will write data (command, EOI) to PIC

Logical Model of PIC Behavior (cont.)

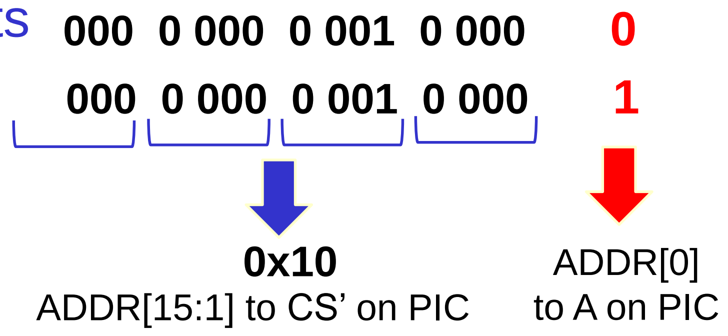
- Map to ports 0x20 & 0x21
  - given ADDR bus
  - logic to form A and CS' inputs to PIC



- Remember
  - the PIC is asynchronous!
  - all interactions have timing constraints
  - see specifications for details

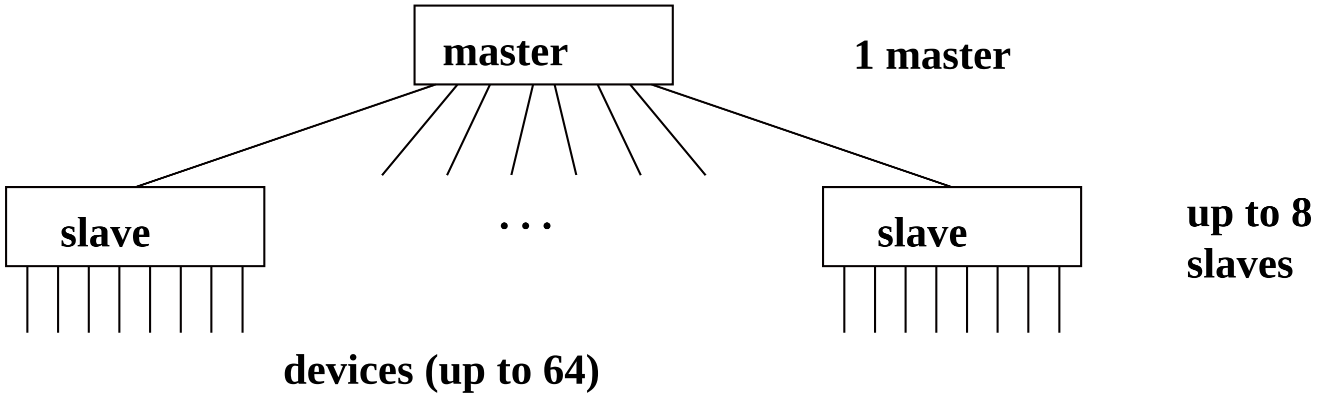
0x20 => 0000 0000 0010 000 0

0x21 => 0000 0000 0010 000 1

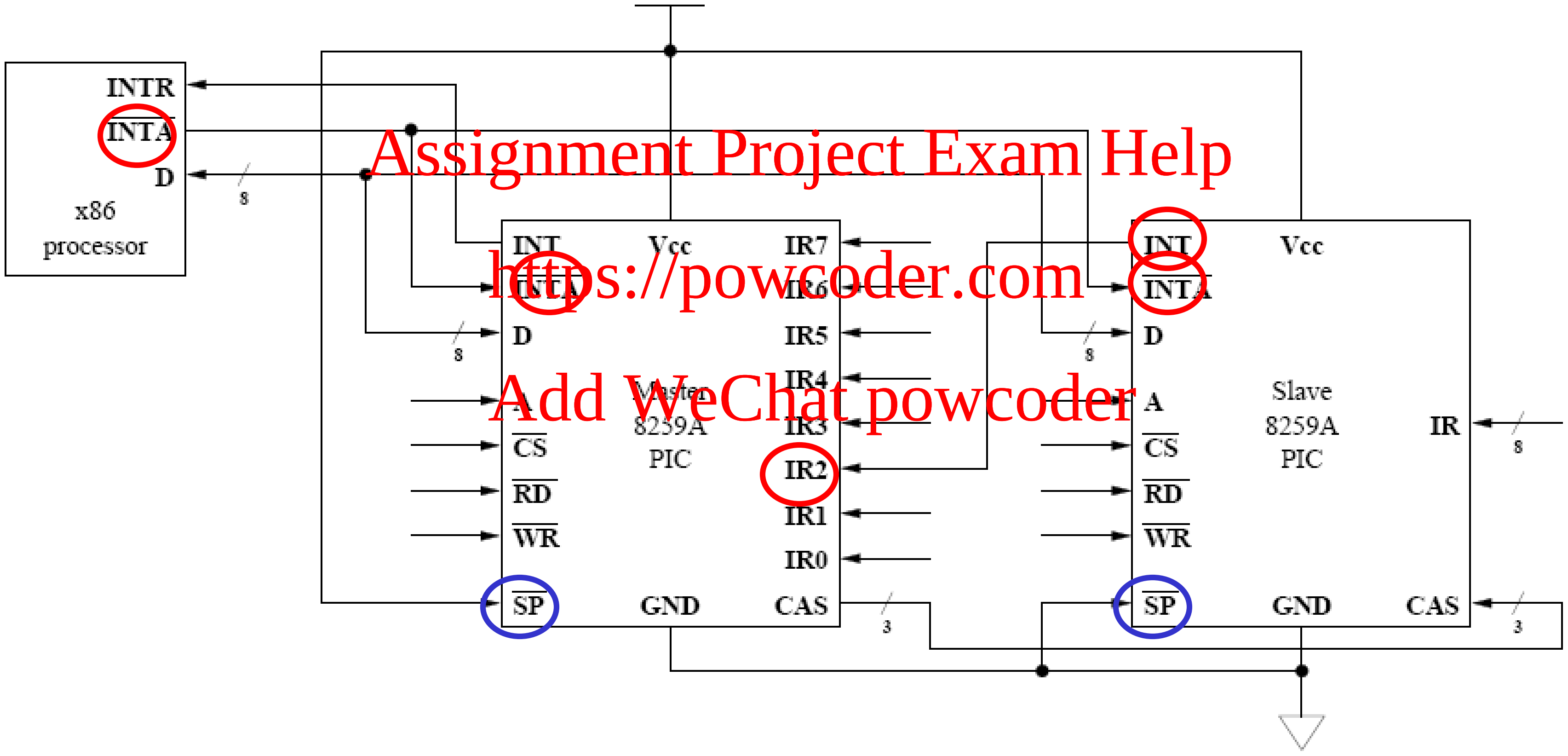


Logical Model of PIC Behavior (cont.)

- What about SP’ & CAS?
- Are eight devices enough? No? What then?
  - hook 2, 3, or 4 PICs to processor?
    [same problem as before!]
  - design a big PIC?
    [waste of transistors; not cheap in 8259A era]
  - design several PICs?
    [waste of humans! (and production costs)]
- Better answer: cascade



Cascade Configuration of PICs



PIC (cont.)

- Previous figure showed x86 configuration of two 8259A's
  - master 8259A mapped to ports 0x20 & 0x21
  - slave 8259A mapped to ports 0xA0 & 0xA1
  - slave connects to IR2 on master

- Question
  - prioritization on 8259A: 0 is high, 7 is low
  - what is prioritization across all 15 pins in x86 layout?
  - (highest) M0…M1…S0…S7…M3…M7 (lowest)

PIC (cont.)

- In Linux (initialization code to be seen shortly)
  - master IR's mapped to vector #'s 0x20 – 0x27
  - slave IR's mapped to vector #'s 0x28 – 0x2F
  - remember the IDT?