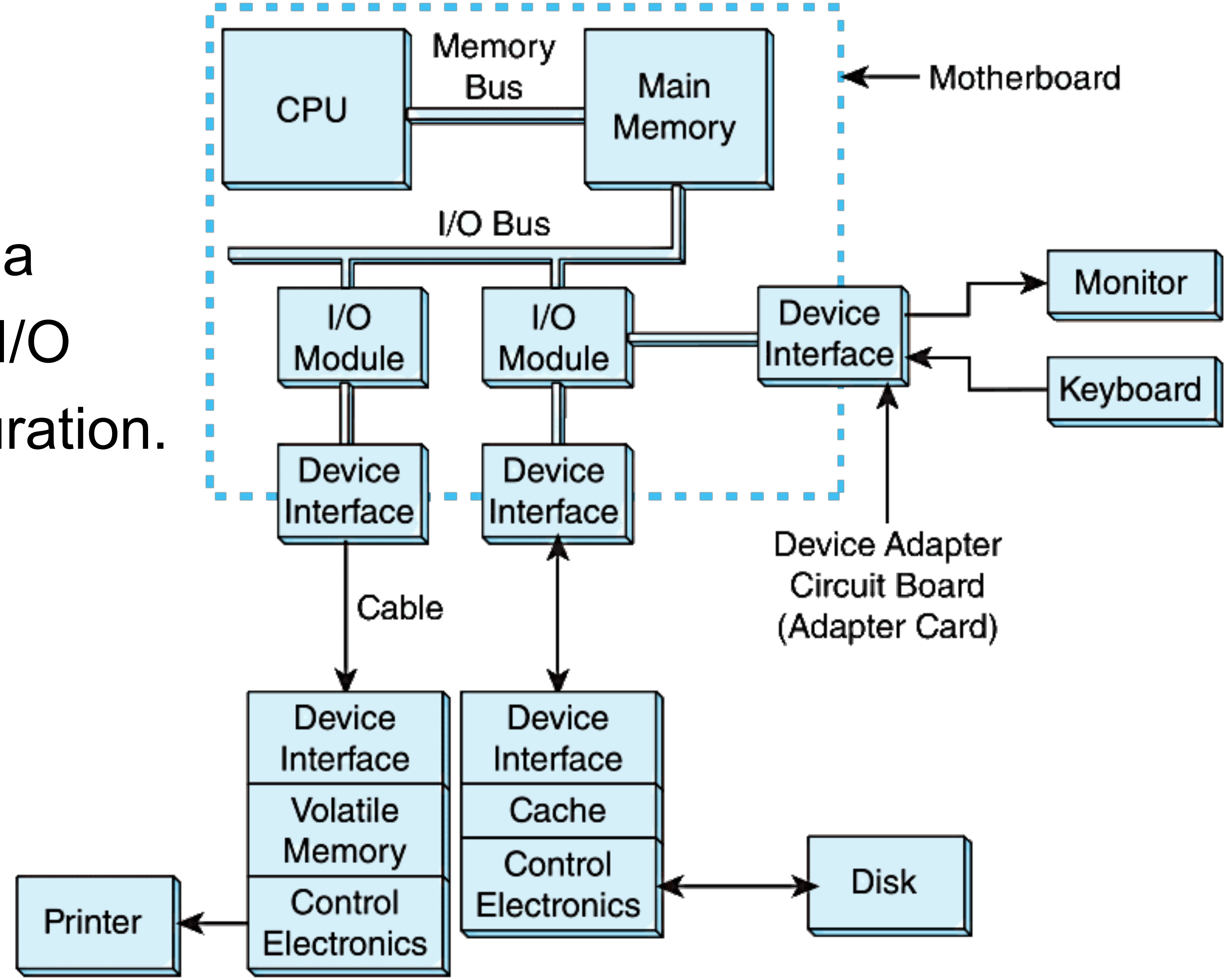
CMSC 313

I/O, Interrupts and Exceptions

Input/ Output (I/O) Architecture

- A system that facilitates communication between a host system and external devices
- System usually include:
  - Blocks of main memory dedicated to I/O functions
  - Buses that move data around
  - Control modules in host and external devices
  - Interfaces to external components such as keyboards]
  - Communication links between host and peripherals

This is a model I/O configuration.



I/O Architecture

- I/O can be controlled in the following general ways
  - Programmed I/O:
    - Reserves a register for each device
    - Each registers is continually polled for data arrived
  - Interrupt driven I/O
  - Memory Mapped I/O
    - Shares memory address space between I/O and program memory
  - Direct Memory Access
    - Off loads I/O to a special chip that handles the specifics
  - Channel IO
    - Uses dedicated I/O Processors

Memory Mapped I/O

- I/O and main memory share address space
  - Each device has its own reserved block of memory
  - From the CPU point of view i/o access looks like memory access
  - Typically can use the same instructions to move data in and out of memory and i/o
- In smaller systems the low level details are offloaded to the i/o controllers in the i/o devices

Direct Memory Access (DMA) I/O

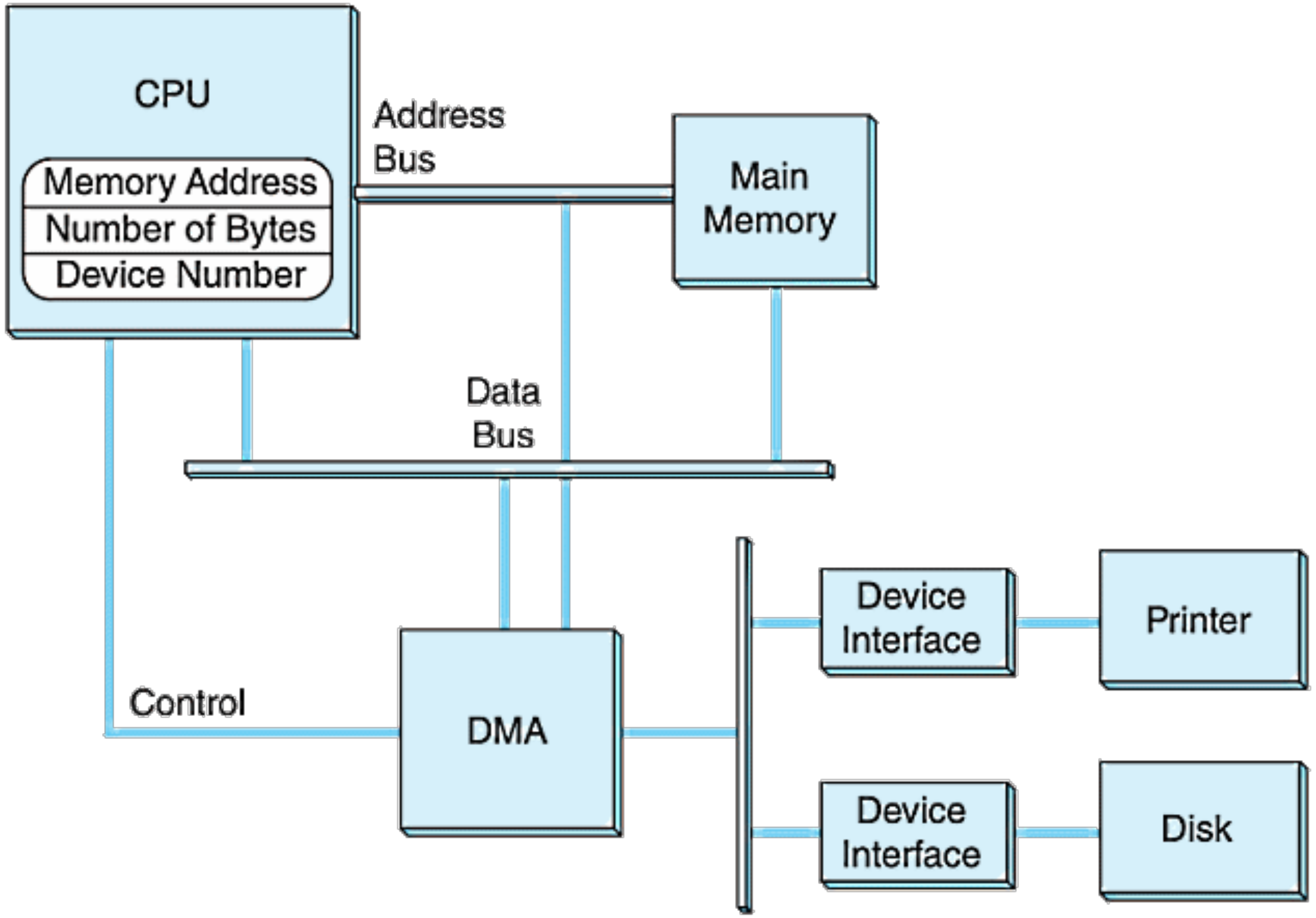
This is a DMA configuration.

CPU initializes data transfer

DMA handles the transfer and lets CPU know when done

Notice that the DMA and the CPU share the bus.

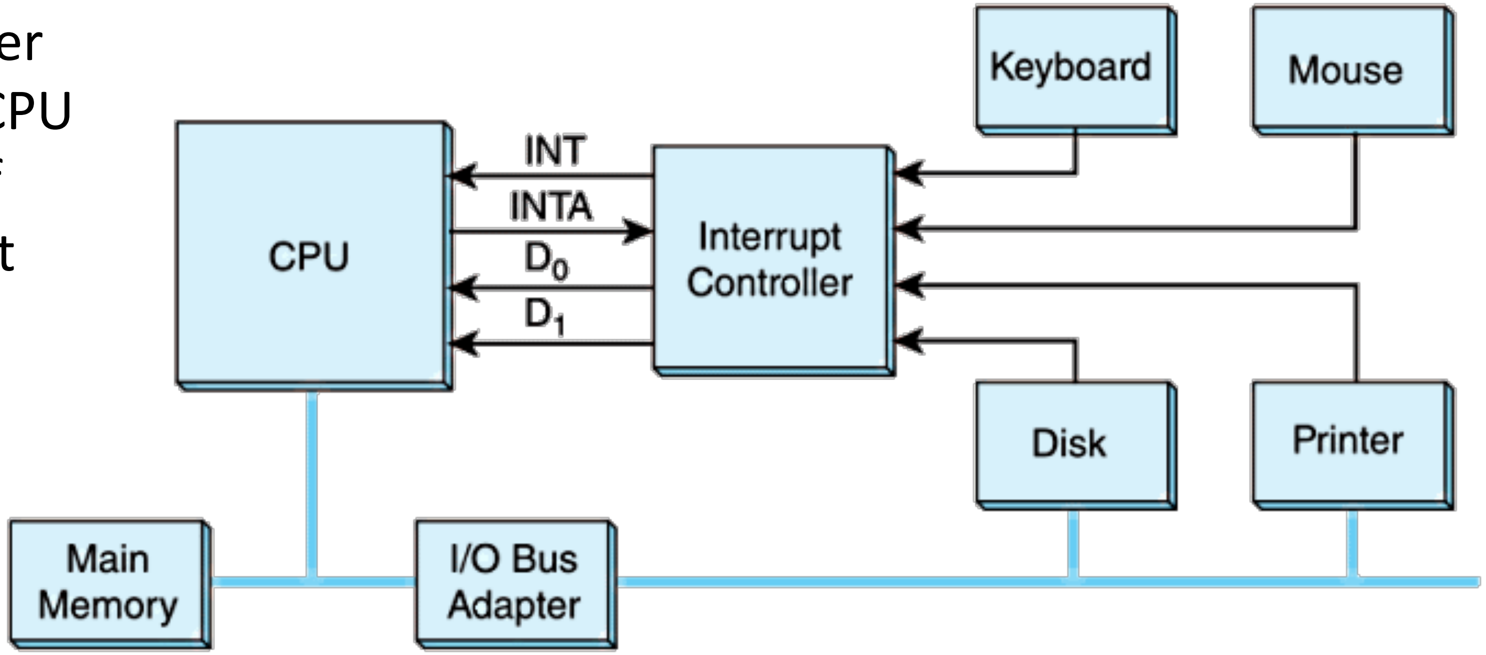
The DMA runs at a higher priority and steals memory cycles from the CPU.



This is an idealized I/O subsystem that uses interrupts.

Each device connects its interrupt line to the interrupt controller.

The controller signals the CPU when any of the interrupt lines are asserted.



Interrupts

Why Interrupts?

```
        mov  RDX, 0x378          ;Printer data port
        mov  RCX, 0              ;Loop counter
Label:  mov RAX, [ABC+RCX]       ;ABC is beginning of
                                 ; memory where
                                 ;characters are to be
                                 printed from

        OUT [RDX], RAX           ;send character to printer
        INC RCX
        CMP RCX, 100000
        JL Label
```

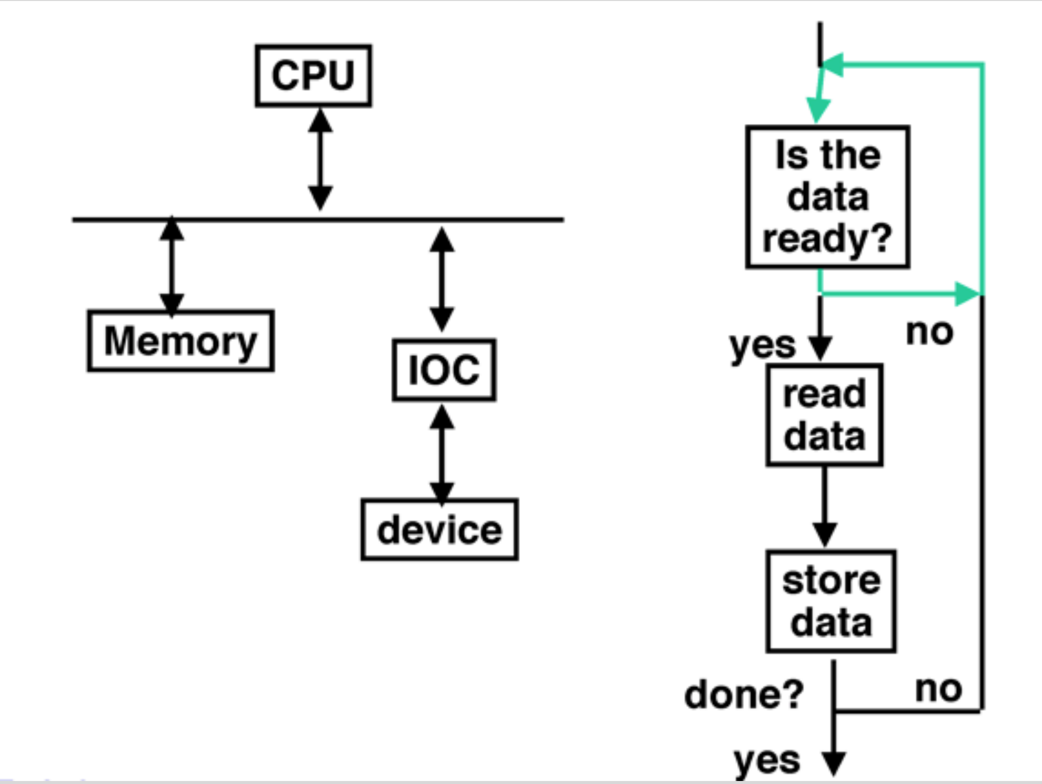
- Assembly program for printing data
- Issues:
  - Speed between processor and printer
  - Printer buffer size

Why Interrupts?

- OS needs to know when:
  - The I/O device has completed an operation
  - The I/O operation has encountered an error
- This can be done in two ways:
  - Polling
    - Information concerning the I/O device is kept in a status register
    - OS checks that register periodically
  - I/O Interrupt
    - Asynchronous, externally stimulated event
    - Does not prevent instruction completion
    - I/O device interrupts processor when it needs attention

Polling

- Polling is simple to implement and processor is in control
- Overhead from polling can consume a lot of CPU time



Polling

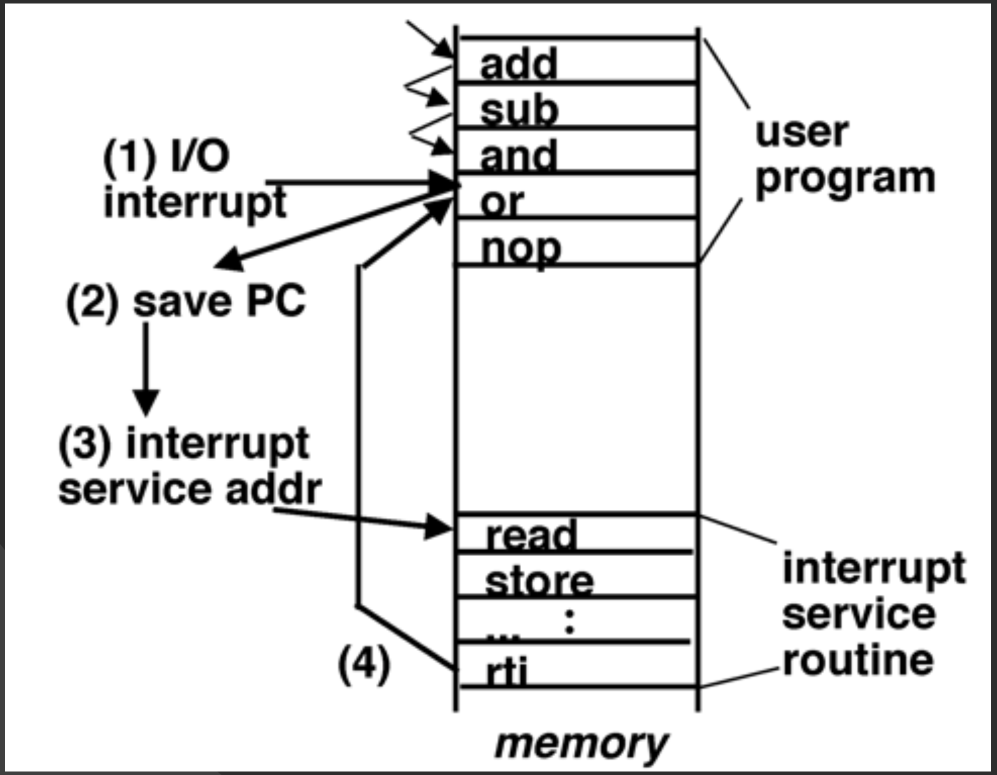
```
    MOV RDX, 0x379            ;Printer status port
    MOV  RCX, 0
Label:
    IN AL, [DX]               ;Ask printer if it is ready
    CMP AL, 1                 ; 1 means it is ready
    JNE Label                 ;if not try again
    MOV AL, [ABC+RCX]
    DEC RDX                   ;Data port is 0x378
    OUT [DX], AL              ;send one byte
    INC RCX
    INC RDX                   ;change back to  status port
    CMP RCX, 100000
    JL Label
```

External Interrupt

- In addition to programs, computers are also hardware driven
- When a device needs attention, it triggers an I/O interrupt
- An I/O interrupt is an externally stimulated event
  - Asynchronous to instruction execution
  - Does not prevent instructions from completing
- Processors typically have one or multiple interrupt pins for device interface

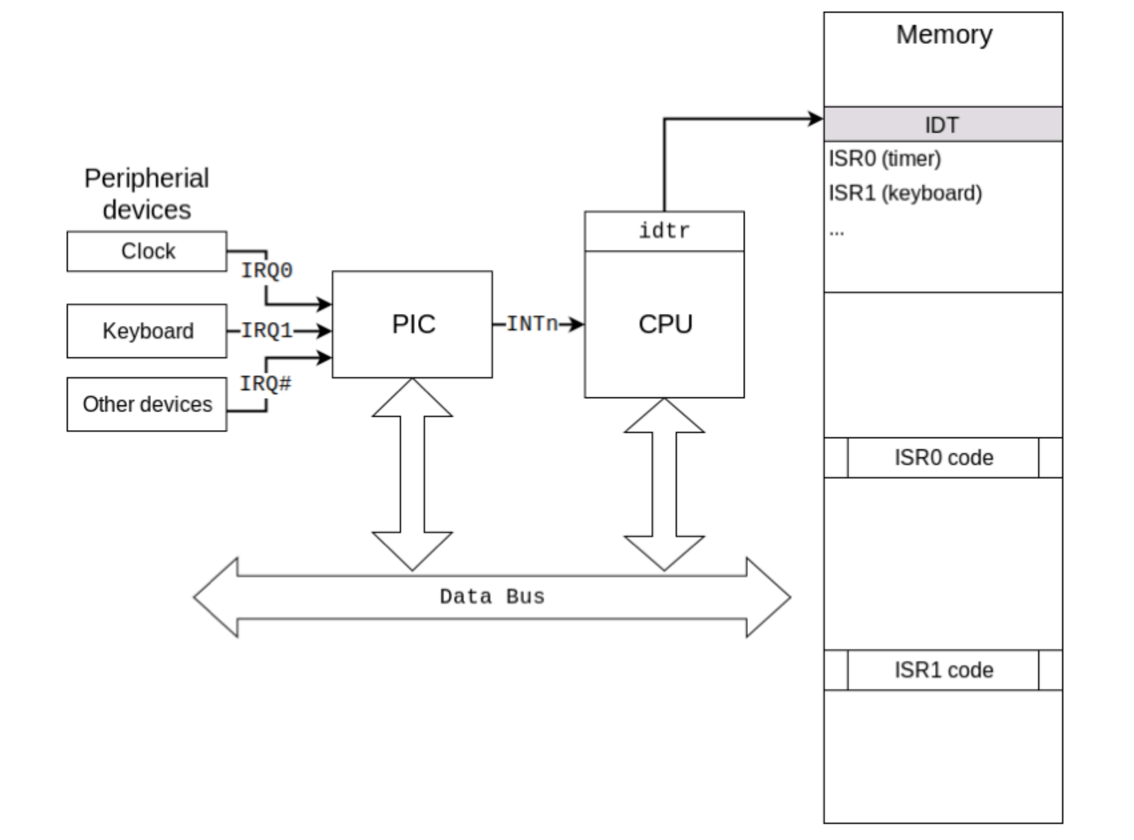
External Interrupt

- Advantage: User program progress is only halted during actual data transfer
- Disadvantage: Special hardware is required to support interrupts:
  - Hardware to cause an interrupt
  - Detect the interrupt
  - Save the processor state and handle the interrupt



x86 Interrupt Handling

- Intel has only one interrupt pin
- Relies on a programmable interrupt controller (PIC)
- Interrupts are handled in the following way:
  - PIC is configured to receive interrupt requests (IRQs)
    - IRQs are numbered according to priority
  - CPU is configured to receive IRQs and invoke correct interrupt handler
    - Interrupt handlers are described in the Interrupt Descriptor Table (IDT)
  - OS system kernel must provide Interrupt Service Routines (ISRs) to handle interrupts
    - OS needs to enable interrupts in PIC and CPU



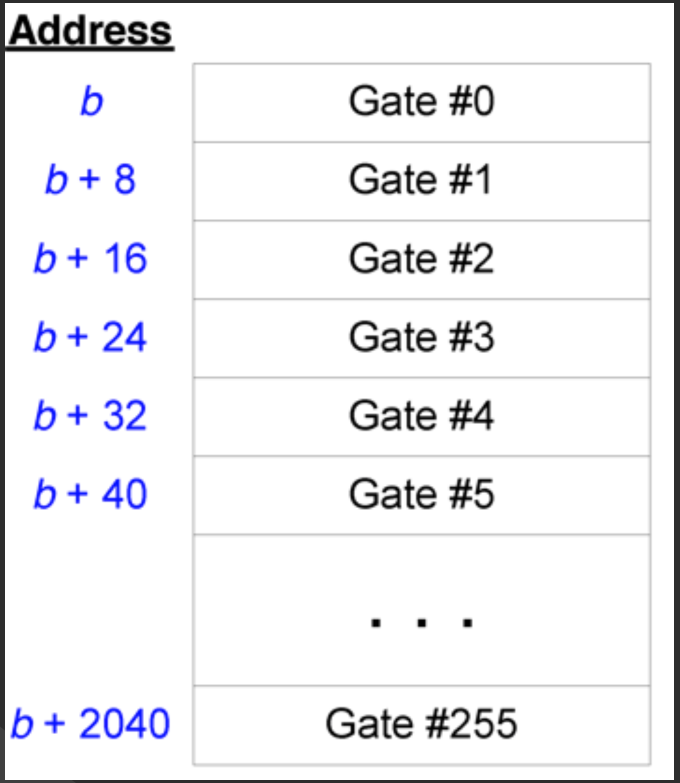
https://alex.dzyoba.com/blog/os-interrupts/

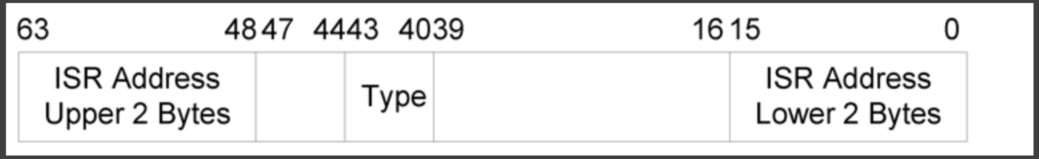
Interrupt Service Routines

- IDT stores pointer to the ISRs
- ISR is code describing what to do in the event of an interrupt
- Basic ISR:
  - Save the state of interrupted procedure
  - Save previous data segment
  - Reload data segment registers with kernel data descriptors
  - Acknowledge interrupt to PIC
  - Do the work
  - Restore data segment
  - Restore the state of interrupted procedure
  - Enable interrupts
  - Exit interrupt handler with iret

Interrupt Descriptor Table (IDT)

- Table that holds addresses and descriptions for Interrupt Service Routines (ISRs)
  - Each entry is 8 bytes
- Table is pointed to by the IDT register
  - Value loaded by the OS



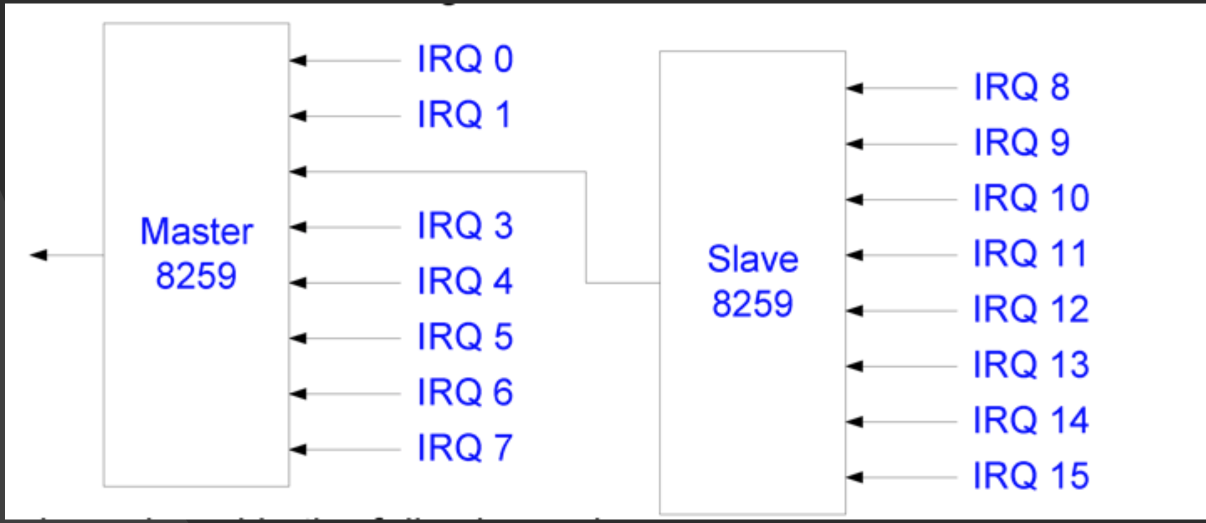


Programmable Interrupt Controller (PIC)

- x86 has one interrupt pin, PIC is needed to handle multiple inputs
- PIC is essentially a multiplexor that saves CPU pins
- Can handle disabling particular interrupts and queuing interrupts
- Programming the PIC takes place at boot time using the OUT commands
- The number of the IRQ dictates its priority
- The PIC can be used in a two tier configuration
  - One becomes the master and the other a slave

The ISA Architecture

- This architecture standardizes:
  - Interrupt controller circuitry
  - IRQ assignments
  - I/O port assignments
  - Connections available to expansion cards
- Dictates one master – one slave configuration
- Priority is assigned to IRQs by their number with 0 having highest priority



IRQs

IRQ (Master)	Description
IRQ0	system timer
IRQ1	keyboard controller
IRQ2	slave IRQs
IRQ3	serial port (available)
IRQ4	serial port (available)
IRQ5	parallel port (sound card)
IRQ6	floppy disk controller
IRQ7	parallel port

IRQ (Slave)	Description
IRQ8	real time clock
IRQ9	ACPI
IRQ10	open
IRQ11	open
IRQ12	mouse on ps2
IRQ13	CPU co-processor
IRQ14	ATA channel
IRQ15	secondary ATA

Software Interrupts

- Interrupts can also be triggered in software
- This is done using the INT instruction in code
  - INT imm where imm indicates the interrupt number
  - IRET returns from a software interrupt
- Before the syscall command in x86-64 system calls were done using INT
  - INT 80H