Lectures 16&17: Interrupts

- What are interrupts?
- Interrupt design questions
- Interrupts in our board
- Interupt configuration for project 2

Interrupts

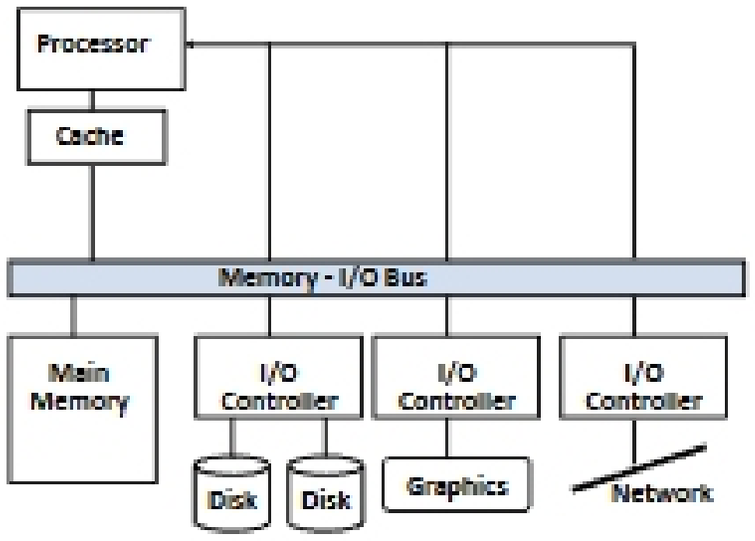
Merriam-Webster:

– "to break the uniformity or continuity of"

Two types of "unexpected" events requiring change in execution flow:

* Exception (or trap)
  – Arises within the CPU
    • e.g., undefined opcode, overflow, syscall, …
* Interrupt
  – From an external I/O controller
  – Informs a program of some external events
    • e.g., timer expires, power failure, system error …

A Typical I/O System



I/O Data Transfer

Two key questions to determine how data is transferred to/from a non-trivial I/O device:

1. How does the CPU know when data is available?
   a. Polling
   b. Interrupts

2. How is data transferred into and out of the device?
   a. Programmed I/O
   b. Direct Memory Access (DMA)

Polling

Our projects 0 and 1 use this scheme

- Periodically check I/O status register
  - If device ready, do operation
  - If error, take action
- Common in small or low-performance real-time embedded systems
  - Predictable timing
  - Low hardware cost
- In other systems, wastes CPU time

Input and Output Devices

- I/O devices are incredibly diverse with respect to
  - Behavior – input, output or storage
  - Partner – human or machine
  - Data rate – the peak rate at which data can be transferred between the I/O device and the main memory or processor

Device	Behavior	Partner	Data rate (Mb/s)
Keyboard	input	human	0.0001
Mouse	input	human	0.0038
Laser printer	output	human	3.2000
Magnetic disk	storage	machine	800.0000-3000.0000
Graphics display	output	human	800.0000-8000.0000
Network/LAN	input or output	machine	100.0000-10000.0000

a order of magnitude range

Interrupts

- When a device is ready (or error occurs)
  - Controller interrupts CPU
  - CPU stops executing the current program and begins executing a special piece of code called an **interrupt handler** or **interrupt service routine (ISR)**
  - The ISR does some work and then resumes the interrupted program
- Interrupts are really glorified procedure calls, except that they:
  - **can occur between any two instructions**
  - are transparent to the running program (usually)
  - call a procedure at an address determined by the type of interrupt, not the program

Polling v.s. Interrupts

- Advantages of using interrupts
  - Relieves the processor from having to continuously poll for an I/O event
  - User program progress is only suspended during the actual transfer of I/O data
- Disadvantage – special hardware is needed to
  - Indicate the I/O device causing the interrupt
  - Save the necessary information prior to servicing the interrupt
  - Resume normal processing after servicing the interrupt

Interrupt handling

Key questions:

- Where do interrupts come from?
  - How do we figure out where to branch to?
- How do we save state for later continuation?
  - How to ensure that we can get back to where we started?
- What if we get another interrupt while we are processing our interrupt?
  - Can we ignore interrupts?
  - Can we prioritize interrupts?

Where to go?

- If you know *what* caused the interrupt then you want to jump to the code that handles that interrupt
- If you number the possible interrupt cases, and an interrupt comes in, you can just branch to a location, using that number as an offset

Back to where you once belonged

- Need to store the return address somewhere
  - Stack *might* be a scary place.
    - *That* would involve a load/store and might cause an interrupt (page fault)!
  - So a dedicated register seems like a good choice
    - But that might cause problems later…

Nested interrupts

If we get one interrupt while handling another what to do?

- Just handle it
  - But what about that dedicated register?
  - What if I'm doing something that can't be stopped?
- Ignore it
  - But what if it is important?
- Prioritize
  - Take those interrupts you care about. Ignore the rest.
  - Devices needing more urgent attention get higher priority.
  - Still have dedicated register problems…