10/13: Lecture Topics

- Slides from last time:
  - Execution cycle
  - Pipelining
  - Hazards
- Exceptions

Exceptions and Interrupts

- So far, we've assumed that the assembled code can always be executed
- Lots of ways for unexpected things to happen:
  - Undefined instruction
  - Arithmetic overflow
  - System call
  - I/O device request

Exceptions

- An *exception* is an internal event
  - The unexpected condition was caused by something the program did
  - Undefined instructions and arithmetic overflows are examples
  - If you ran the program again, the exception would (probably) happen again at the same point in the program's execution

Interrupts

- An *interrupt* is an external event
  - The unexpected condition was not caused by the program
  - An I/O device request is an example
  - If you ran the program again, the interrupt would probably *not* happen at the same point

What should happen?

- These events result in an *unnatural change in the flow of control*
- Normally, the next instruction executed is _____
- When one of these events takes place, something else happens
  - The system must respond to the event
  - The response depends on the type of event

Exception Handling

- Loosely, the following steps are taken:
  1. Save the address of the offending instruction in a register
  2. Make the reason for the exception known
     - Set the value of the *status register*, or
     - Use *vectored interrupts* to do step 3
  3. Transfer control to the operating system
  4. Operating system decides what to do:
     - May report the error to the user
     - May terminate the program

Exception/Pipelining Interface

- Suppose an add instruction overflows, causing an exception
- Instructions after the add are already in the pipeline
  - The partially computed instructions must be *flushed*
- Exception must be caught before register contents have changed
  - Pipeline designers must be wary of exception handling