Exceptions

Definition:

- all “unexpected” events in the normal flow of execution
- interrupts, traps, faults, aborts, exceptions
  - **exceptions** caused by instruction execution **within the processor**
  - **interrupts** caused by some event **external to the processor**

Some examples:

Exception Handling

Identify the instruction that caused the exception

- pipelines execute several instructions at once:
  how determine which one caused the exception?
- **exception program counter** (EPC in MIPS)

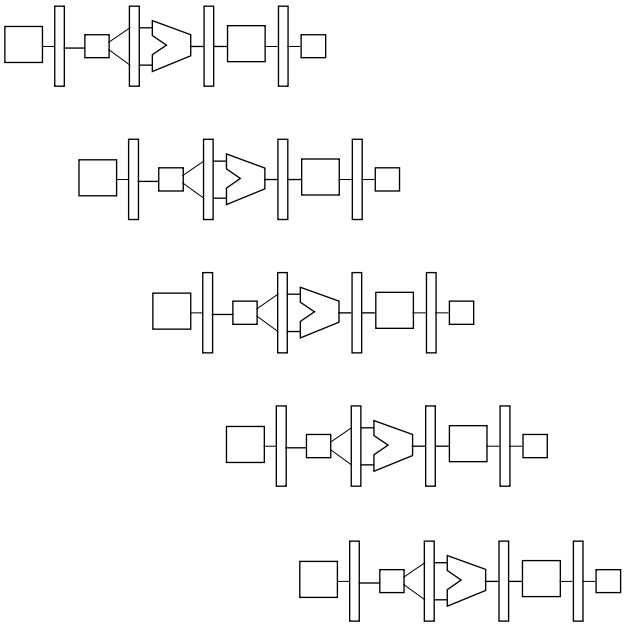
Identify the reason for the exception

- **status register** (Cause register in MIPS)
- **vectored interrupts**: bit vector for causes of interrupts
  - each bit stands for a different interrupt
  - bit is set when the interrupt occurs

Force trap instruction into pipeline

- status register
  - single entry point for all exceptions
  - OS checks the status register to determine which service routine to use
- vectored interrupts
  - table of starting addresses for the interrupt service routines
  - use the set bit in the bit vector to index into the table

A 5-Stage Pipeline



Exception Handling, cont'd.

Execute the trap/interrupt routine

- kill the program (arithmetic overflow, undefined opcode, hardware malfunction)
- service the trap/interrupt & restart the faulting & subsequent instructions (page fault, I/O device request)
  - save the state of the process
    - PC, registers
    - page table address register, TLB contents, PID register
    - EPC, Cause register
  - determine the address that caused the exception
    - usually it's in EPC
    - if a data page fault, the base register + offset of the instruction whose address is in EPC
  - if servicing the exception requires using the disk, context switch this process out & another one in
    context switch the first process back in when the I/O has completed
  - restore the state of the process
  - reexecute the excepting instruction

Exceptions in a Pipeline

Example exception: arithmetic overflow

Save the restart address:

- an exception PC
- trap handler has to decrement it when used (by now it's PC+4)

Record the cause of the exception:

- Cause register

Flush the faulting instruction:

- EX.FLUSH control signal
- MUXes to choose between passing values from ID/EX to EX/MEM pipeline registers or zeroing them

Flush the subsequent instructions in IF and ID

- **IF.Flush** control signal is already there (why?)
- **ID.Flush** control signal and OR gate
  hardwired zero lines & MUX to choose them already there (why?)

Jam in the address for the exception handler:

- hardwired input to PC MUX

Prioritizing Exceptions

What if simultaneous interrupts?

- match the exception to the instruction
- handle the exception for the earlier instruction first
- restart: second will reoccur
- now handle it

Some asynchronous exceptions can be handled later

Some exceptions have higher priority

- hardware malfunction
- arithmetic overflow
- I/O device request