Exceptions and Interrupts

**exception**  
MIPS: an unscheduled event that disrupts program execution

**interrupt**  
MIPS: an exception that comes from outside of the processor

Some examples:
- I/O device request
- OS system call from user program
- arithmetic overflow
- attempt to execute an undefined instruction
- hardware malfunctions

Either; exception or interrupt

Our simple datapath does not admit very many opportunities for exceptions.
- arithmetic overflow
- undefined instruction

Exceptions and Control

The occurrence of an exception will require a response that depends on the source and type of the exception.

Thus, our control logic must be modified.

And, we will need additional hardware support.

This may have an adverse effect on performance.
### Exception Handling on MIPS

MIPS adds two additional registers to the datapath, along with the corresponding control logic to make use of them.

- **EPC**: 32-bit register storing the address of the offending instruction, obtained from the PC.
- **Cause**: 32-bit register storing a code indicating the cause of the exception.

When an exception is detected, control is transferred to a specific address in the operating system.

Then, OS-specific code will execute and respond to the exception. In order to do this, the OS must know what (or where) the offending instruction is, and the cause of the exception. Hence, the hardware support above is added.

### Changes to the Datapath

Summary of additions:

The input to **Cause** is restricted to 0 or 1 because we only need to encode two different exceptions.

Why is the ALU subtracting 4 from the value in the PC?
### Changes to the Control Logic

We need two new states, one for each type of exception.

We now pay attention to the **Overflow** signal from the ALU.

Note: in this design, if an arithmetic overflow occurs the (incorrect) result will have already been written to the register file before the exception is handled. Fixing this (if we want to) requires complicating the control logic further.