



Topic Notes: Exceptions and Interrupts

Exceptions and Interrupts

We have seen how to build a relatively simple datapath and control system. The same ideas and techniques can be used to build computers that implement a more complete instruction set, such as the full MIPS ISA.

Our implementation so far assumes that everything will go as planned.

- all instructions are valid
- arithmetic operations complete correctly
- all data is immediately available
- the operating system has not yet become involved

Conditions that cause an “unusual” operation – one that does not simply proceed to the next instruction specified by the PC – are called *exceptions* or *interrupts*.

Our text uses the term “exception” for an event generated by the processor and “interrupt” for one coming from outside. However, different texts and ISAs use different conventions. We will use the terms pretty much interchangeably.

So far, there are only two things we have seen that can cause an exception:

1. arithmetic overflow
2. attempted execution of an undefined instruction

We will see others soon and consider them when they arise.

Data Path Augmentation for Exceptions

The MIPS approach to handling exceptions involves several steps:

1. the address of the instruction that caused the exception is stored in a new register: the *exception program register* or EPC

2. control is transferred to the operating system to take appropriate action by loading the PC with a specified address
3. the OS handles the situation, which may involve
 - (a) providing a service to the user program (e.g., reading some information from an I/O device)
 - (b) handling an overflow
 - (c) stopping a user program that has executed an illegal instruction
4. if the offending program is allowed to continue, the OS restarts its execution based on the value in the EPC

In the MIPS ISA, the OS is informed of the exception's cause by storing a value in an additional register *Cause*.

Other systems use an *interrupt vector*, which specifies different target addresses to be used by the OS based on the reason for the interrupt/exception.

We will consider the MIPS approach, and look at how to augment our data path and control to handle the two exception types that could happen in our MIPS subset.

The two additional registers are, as mentioned above:

1. EPC: holds the 32-bit address of the instruction that was being executed when the exception occurred
2. *Cause*: records the reason for the exception

For our simplified system, there are only two causes. We will store a 10 in *Cause* for an undefined instruction and a 12 for an arithmetic overflow.

This also requires some new data flow:

- EPC is loaded with the value of PC+4 from the **ID/EX** pipeline registers. This is where the exception will
- a 3rd option on the MUX that loads the PC with the OS exception handling code address: 80000180_{16} .
- the ability to flush subsequent instructions from the pipeline – something we already know how to from the handling of control hazards (in fact, from this perspective, exceptions are just another type of control hazard)

Figure 4.66 shows the pipelined datapath with these enhancements.

Figure 4.67 shows an example of dealing with an arithmetic overflow exception.

We will return to the ideas of exception handling as they come up in our remaining topics.