Exceptions; Introduction to Pipelining

Tom Kelliher, CS 220

Nov. 14, 2007

1 Administrivia

Announcements

Homework due, exam Friday.

Assignment

Read 6.2.

From Last Time

Controlling the multi-cycle implementation.

Outline

1. Exceptions, syscalls, and interrupts.
2. Introduction to pipelining; comparison with single-cycle implementation.

Coming Up

Pipelined datapath.

2 Exceptions

1. Exception examples:

   (a) Divide-by-zero.

   (b) Privileged instruction.

   (c) Memory protection.

   (d) Hardware faults.

   What action should be taken? Can the hardware carry it out?

2. Software interrupts (syscalls):

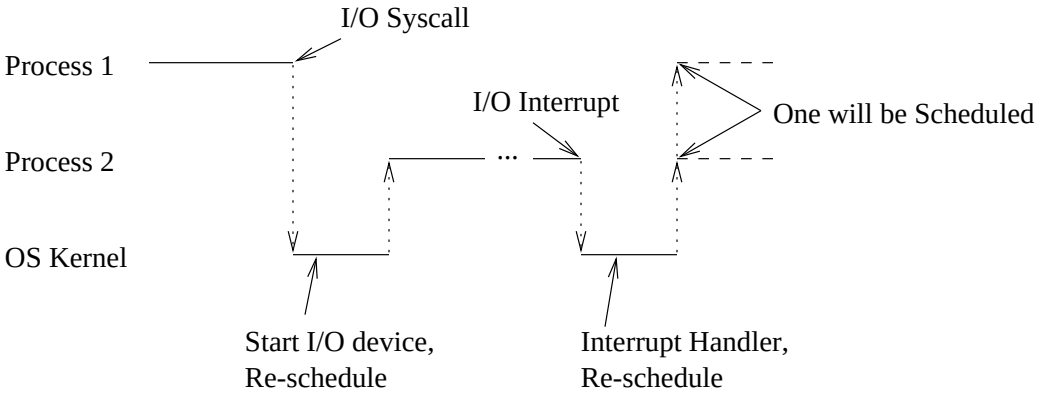
   (a) Used to make transition from user mode to kernel mode.

3. Interrupts are a mechanism allowing a CPU to "disconnect" from active I/O devices to continue doing useful work.

   (a) CPU *much* faster than most I/O devices.

   (b) Polling wastes CPU cycles if there's other work to be done — multiprogrammed system.

   Conceptually, how do interrupts work?



"Unplanned" function calls setup by planned function calls (syscalls, etc.).

Why does the kernel initiate the I/O operation?

Components of an interrupt system:

(a) Interrupt request line(s). Priorities, arbitration within level, masking.

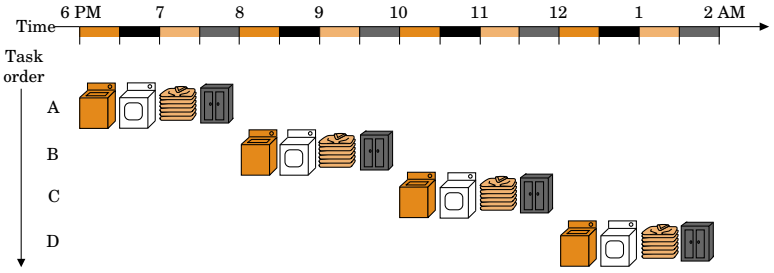
(b) Interrupt acknowledge line(s).

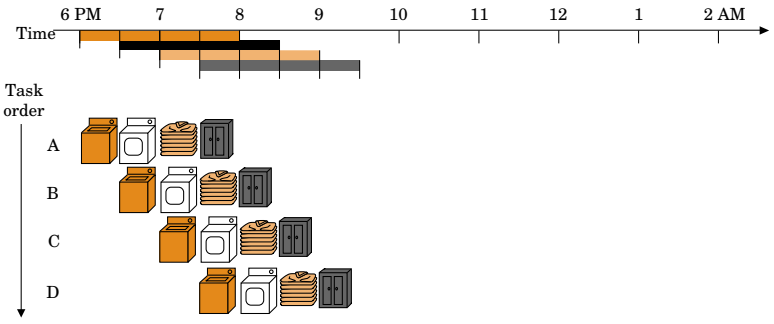
(c) Interrupt handlers (service routines).

(d) A mechanism for indicating what device interrupted and why.

3 Introduction to Pipelining

The laundry analogy:





The five stage MIPS pipeline:

1. Instruction fetch.

2. Decode and read registers.

   The consistent placement of the source registers permits this.

3. Execute ALU operation or calculate an address.

4. Access memory.

5. Result write-back.

3.1 Comparison of Single-Cycle and Pipelined Performance

Assume:

1. Memory access is 2 ns.

2. ALU use is 2 ns.

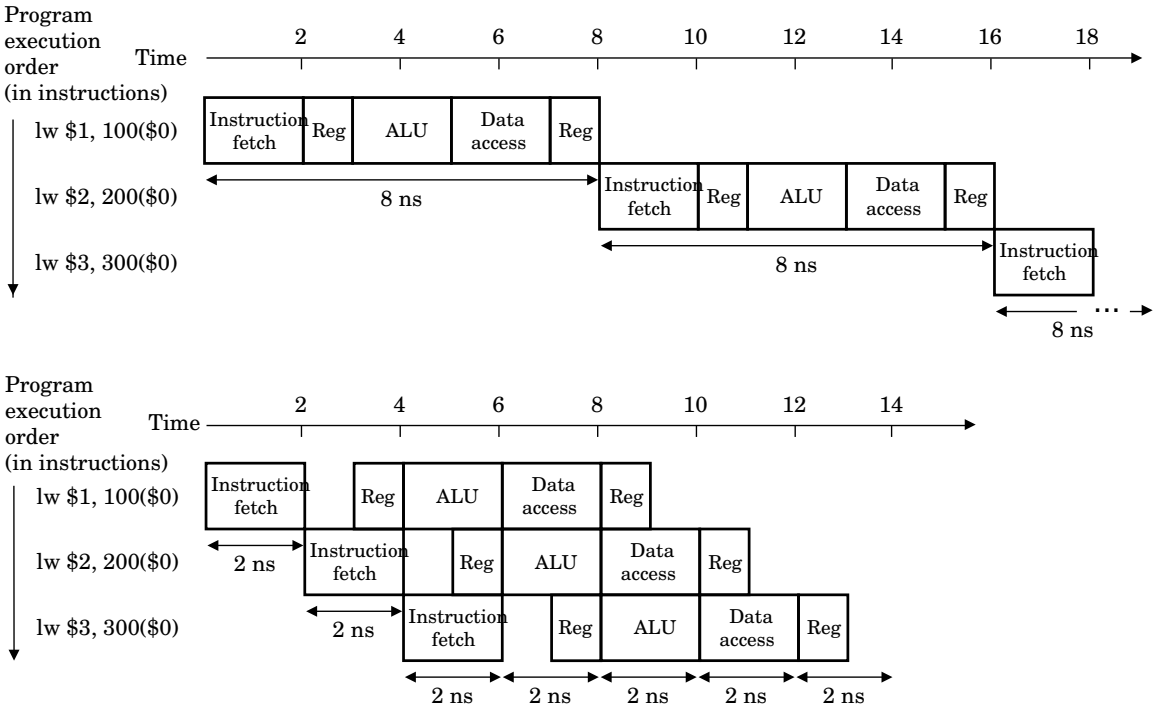
3. Register file access is 1 ns.

Instruction class times:

Instruction Class	**Instruction fetch**	**Register read**	**ALU operation**	**Data access**	**Register write**	**Total time**
lw	2 ns	1 ns	2 ns	2 ns	1 ns	8 ns
sw	2 ns	1 ns	2 ns	2 ns		7 ns
R-format	2 ns	1 ns	2 ns		1 ns	6 ns
beq	2 ns	1 ns	2 ns			5 ns

Clock periods for the two implementations?

Execution example:



Note that pipelined register file reads are done during the second half of the clock cycle and writes are done during the first half. Why?

Consider the speedup:

1. Assumption: Stages are of equal length. What if they aren't?
2. Speedup is at most the number of pipeline stages.

   Do we achieve that?

   Consider the execution of 1,000 instructions and compute the actual speedup.

   What happened? The cost of the pipeline registers.

Consider:

1. How does the speedup occur?
2. Shortened instruction execution time?
3. Higher instruction bandwidth?