Introduction to Pipelining

Tom Kelliher, CS 220

Nov. 21, 2011

1 Administrivia

Announcements

Date for second exam?

Assignment

From Last Time

Single-cycle implementation.

Outline

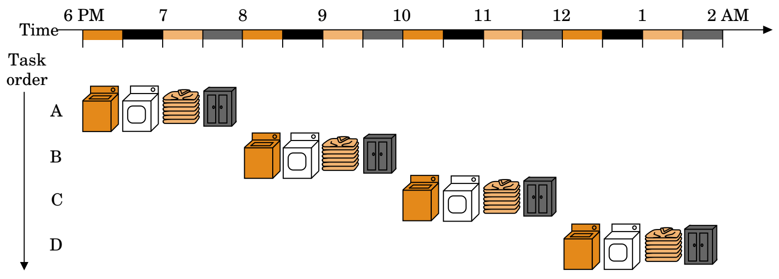
1. Introduction to pipelining; comparison with single-cycle implementation.
2. Architectural features encouraging pipelining.

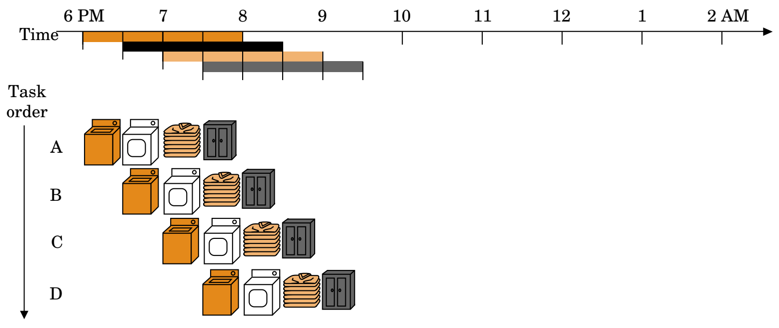
Coming Up

Pipeline hazards.

2 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.

2.1 Comparison of Single-Cycle and Pipelined Performance

Assume:

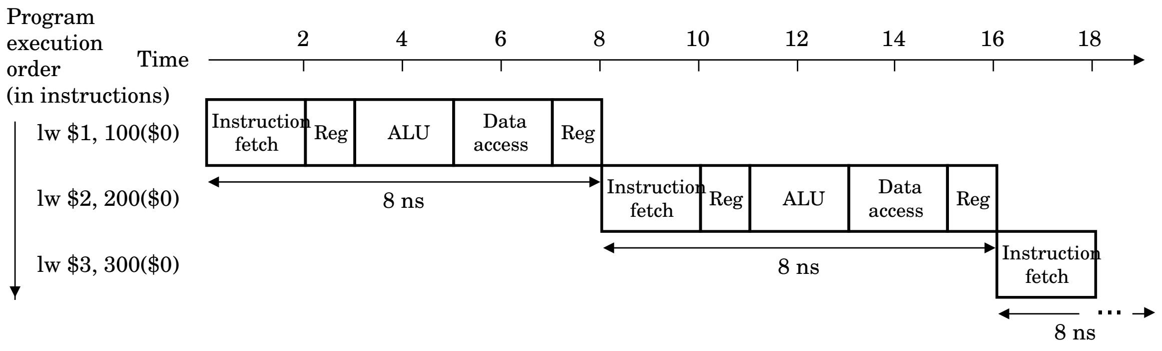
1. Memory access is 200 ps.
2. ALU use is 200 ps.
3. Register file access is 100 ps.

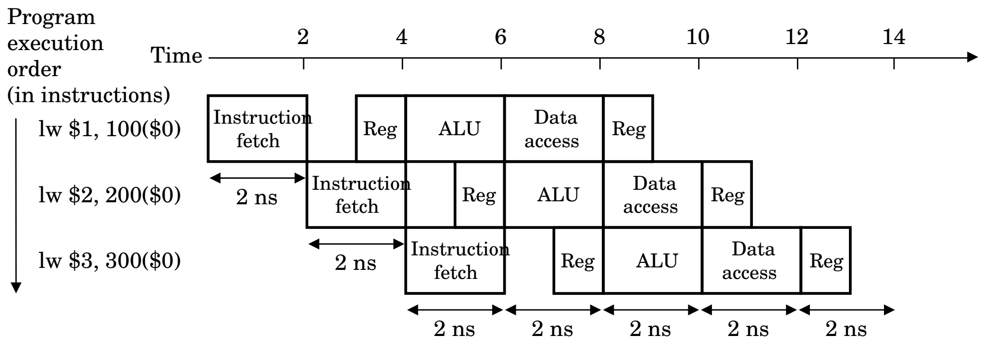
Instruction class times:

Instruction Class	**Instruction fetch**	**Register read**	**ALU operation**	**Data access**	**Register write**	**Total time**
lw	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
sw	200 ps	100 ps	200 ps	200 ps		700 ps
R-format	200 ps	100 ps	200 ps		100 ps	600 ps
beq	200 ps	100 ps	200 ps			500 ps

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:

- How does the speedup occur: shortened instruction execution time or higher instruction bandwidth?

3 Architectural Features Encouraging Pipelining

1. A single instruction size: simplifies instruction fetch.

2. A small number of instruction formats, with register fields in common locations for all formats: simplifies instruction decode and allows register fetch to proceed in parallel.

3. Memory operands occur only in `sw/lw` instructions: simplifies pipeline design and decreases pipeline depth.

4. Memory data is aligned: a memory operation requires only a single memory read or write.