CS5375 Computer Systems Organization and Architecture

Lecture 10

Instructor: Yong Chen, Ph.D.
Department of Computer Science
Texas Tech University
Yong.Chen@ttu.edu, 806-834-0284

Announcements

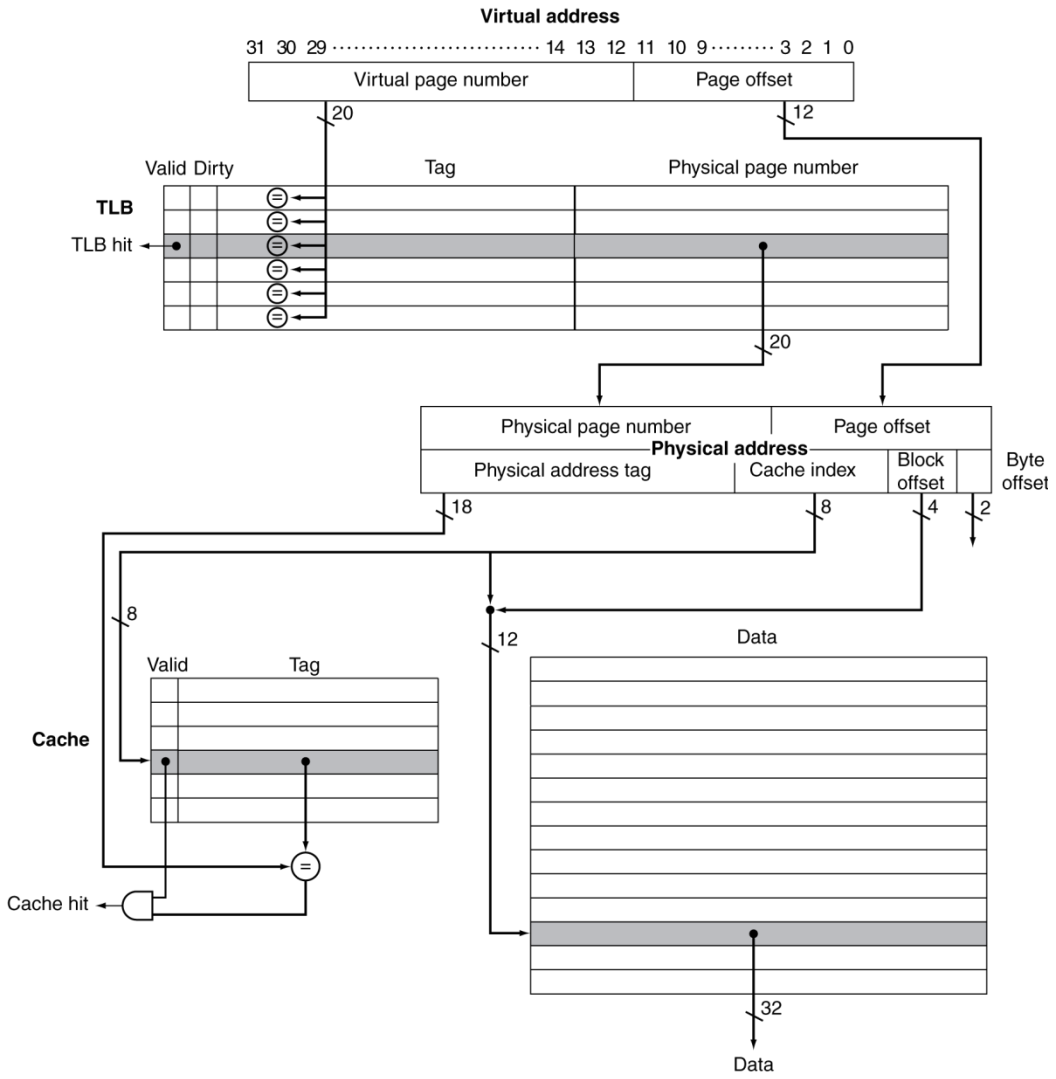
- Live training
  - Introduction to Linux: Wednesday, October 5th, 10 am - 3 pm (1-hour lunch break)
  - New User Training: Wednesday, October 12th, 10 am - 3 pm (1-hour lunch break)

- Register at: https://www.depts.ttu.edu/hpcc/about/training.php

Review of Last Lecture

- Optimizations of Cache Performance (cont.)
  - Reduce Miss Rate: Compiler Optimizations, e.g.,

    Loop Interchange, Blocking
- Virtual Memory and Virtual Machines
  - Use main memory as a “cache” for secondary (disk) storage
    - Managed jointly by CPU hardware and the operating system (OS)
  - Address translation using a page table
  - Fast translation using a TLB



Outline

- Programming project #1 and demo
- Instruction-level parallelism

Demo

- Log in the system following the instruction in Part 1, using your eRaider ID and password
  - E.g., `$ ssh yonchen@login.hpcc.ttu.edu`

- Request an interactive session for you to code, debug, test, etc., instead of overloading the head/login node
  - E.g., `$ interactive -A cs5375 -r cs5375 -p nocona -c 2`

```
-A: The account name that students belong to (cs5375)
-r: The reservation name that cs5375 can access to (cs5375)
-p: The partition that hosts the reserved nodes (nocona)
-c: Number of CPU cores (1-2)
```

- Check out sample code, compile, and run
  - E.g., `$ git clone https://github.com/githubyongchen/CS5375.git`

Outline

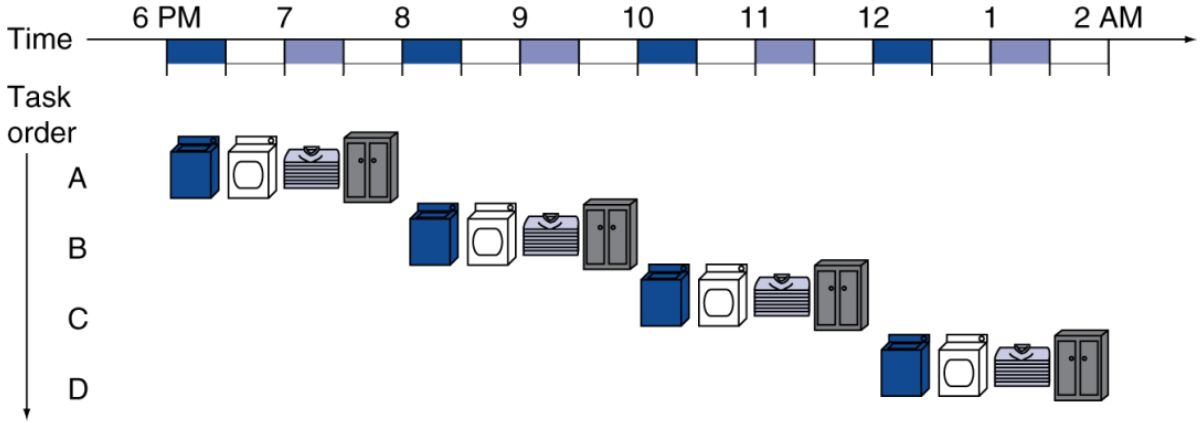
- Programming project #1 and demo
- Instruction-level parallelism
  - Pipelining

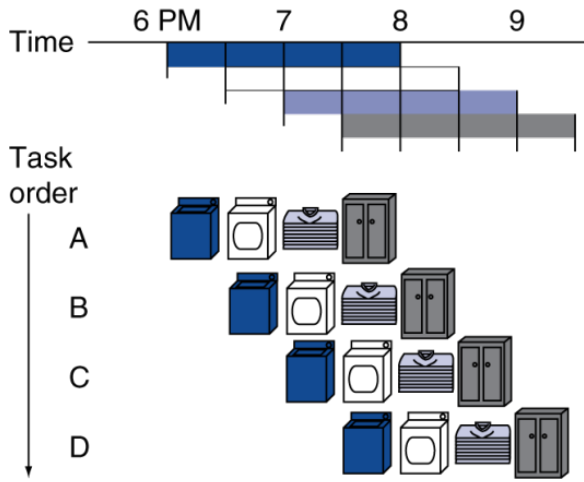
An Overview of Pipelining

- Pipelining
  - An implementation technique in which multiple instructions are overlapped in execution, much like an assembly line
  - Today, pipelining is nearly universal
  - Exploits “Instruction Level Parallelism”

The Laundry Analogy for Pipelining

- Assumptions:
  - **Ann, Brian, Cathy, and Don** each have dirty clothes to be: Washed -> Dried -> Folded -> Put away
  - The washer, dryer, "folder," and "storer" each take 30 minutes for their task
- How Long does it take?
  - Sequential laundry:
    - Takes 8 hours for four loads
  - Pipelined laundry:
    - Takes just 3.5 hours for four loads

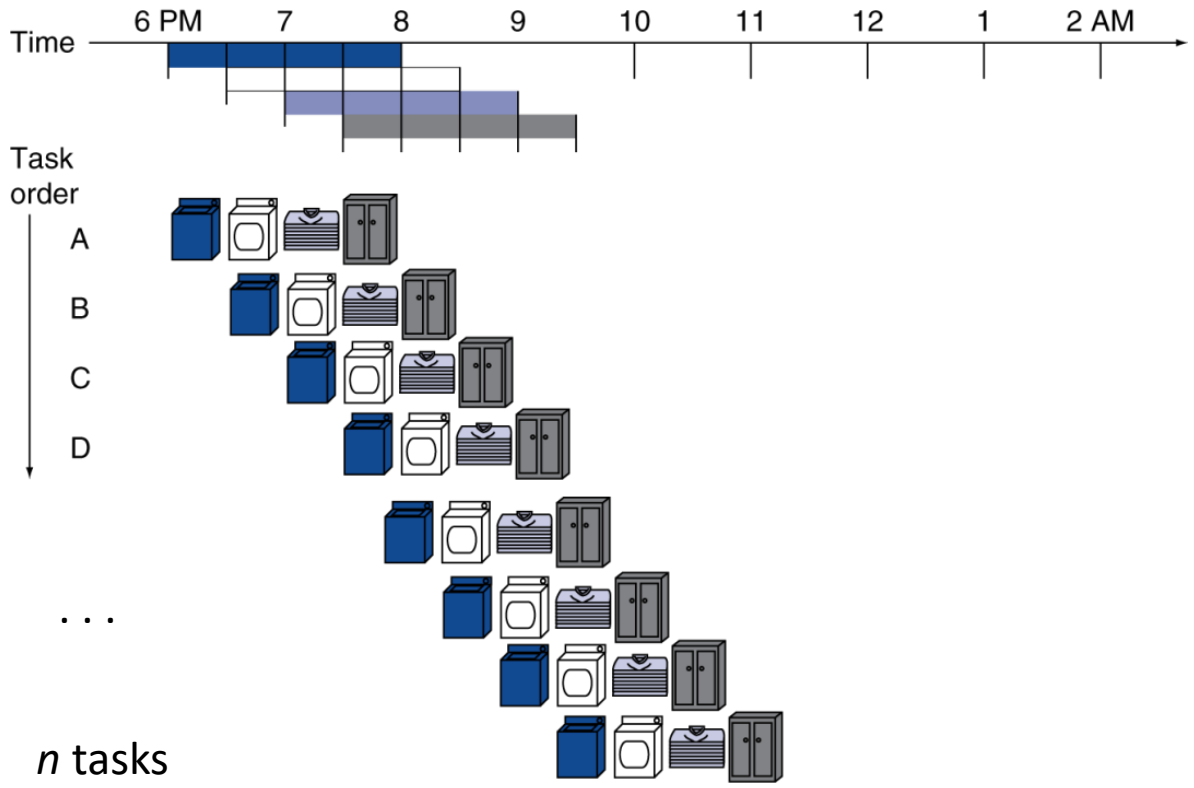




We show the pipeline stage of different loads over time by showing copies of the four resources on this two-dimensional timeline, but we really **have just one of each resource**.

The Laundry Analogy for Pipelining

- Pipelined laundry: overlapping execution
  - Parallelism improves performance



- Four loads:

  Speedup = 8/3.5 = 2.3

- What about *n* loads?

  Sequential laundry: 0.5 X 4 X n = 2n hours

  Pipelined laundry: 0.5 X 3 + 0.5 X n (hours)

  Speedup = 2n / (1.5 + 0.5n)
  ≈ 4 (number of stages) for large *n*

Speedup Analysis

- ***ns***: number of stages
- ***ts***: time length of each stage
- ***ni***: number of instruction (i.e., number of tasks)

- *Non−pipelined (sequential) time = ni×ns×ts*
- *Pipelined time=*$(ns-1)\times ts + ni\times ts = (ns+ni-1)\times ts$
- $Speedup = \frac{\text{Non−pipelined time}}{\text{Pipelined time}} = \frac{ni\times ns\times ts}{(ns+ni-1)\times ts} = \frac{ni\times ns}{ns+ni-1} = \frac{ns}{\frac{ns}{ni}+1-\frac{1}{ni}}$

- For large *ni*:
- $Speedup = \lim_{ni\to\infty} \frac{ns}{\frac{ns}{ni}+1-\frac{1}{ni}} \approx \frac{ns}{0+1-0} \approx ns$

RISC-V Pipeline for Our Mini-CPU

- Five stages, one step per stage
  1. IF: Instruction fetch from memory
  2. ID: Instruction decode & register read
  3. EX: Execute operation or calculate address (ALU operation)
  4. MEM: Access memory operand (either load from memory or store into memory)
  5. WB: Write result back to register

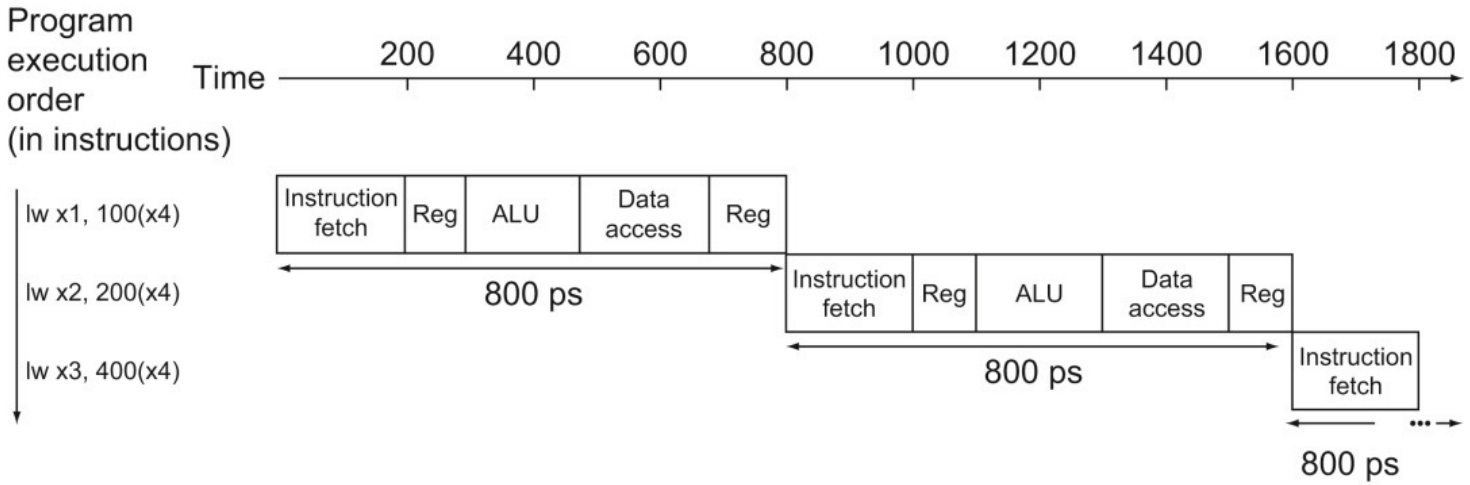
Pipeline Performance

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Compare pipelined datapath with single-cycle datapath

Instruction class	Instruction fetch	Register read	ALU operation	Memory access	Register write	Total time
Load word (lw)	200ps	100 ps	200ps	200ps	100 ps	800ps
Store word (sw)	200ps	100 ps	200ps	200ps		700ps
R-format (add, sub, and, or)	200ps	100 ps	200ps		100 ps	600ps
Branch (beq)	200ps	100 ps	200ps			500ps

Single-cycle, nonpipelined execution (top) versus pipelined execution (bottom)

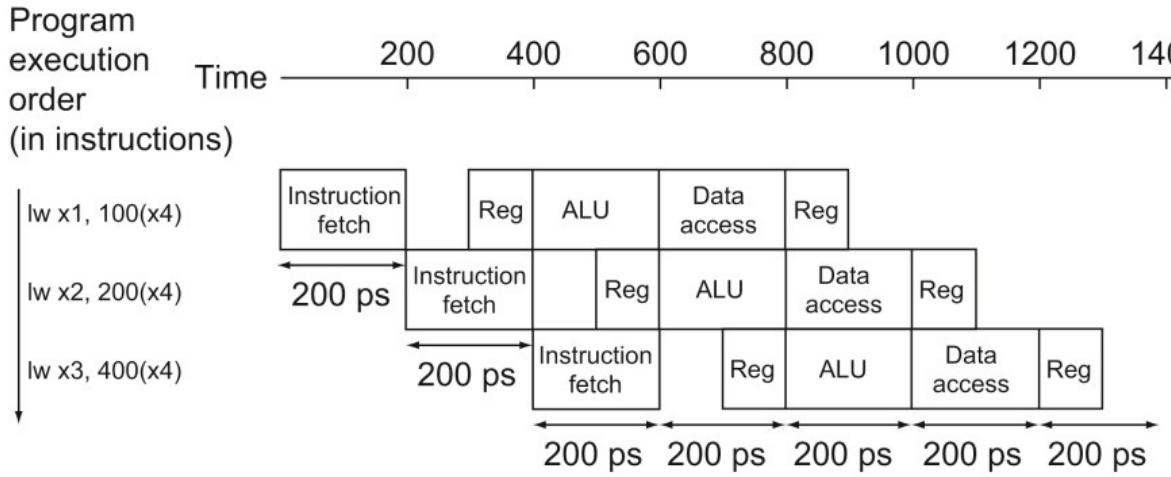
Single-cycle (T_c= 800ps, determined by the slowest instruction)



Nonpipelined time?

$3 \times 800 = 2400\ ps$

Pipelined (T_c= 200ps, cycle time is same as ts)



Pipelined time?

$(\mathrm{ns} + \mathrm{ni} - 1) \times \mathrm{ts}$
$= (5 + 3 - 1) \times 200$
$= 1400\ ps$

$$Speedup = \frac{2400\ ps}{1400\ ps} = 1.71$$

Another View

None-pipelined execution														
Clock Cycle 1					Clock Cycle 2					Clock Cycle 3				
IF	ID	EX	MEM	WB										
					IF	ID	EX	MEM	WB					
										IF	ID	EX	MEM	WB

Pipelined execution														
Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6	Cycle 7	Cycle 8	Cycle 9	Cycle 10	Cycle 11	Cycle 12	Cycle 13	Cycle 14	Cycle 15
IF	ID	EX	MEM	WB										
	IF	ID	EX	MEM	WB									
		IF	ID	EX	MEM	WB								

- Pipelining: overlapping execution
  - Parallelism improves performance

Pipeline Speedup

- If all stages are balanced
  - i.e., all take the same time

$$\text{Time between instructions}_{\text{pipelined}} \approx \frac{\text{Time between instructions}_{\text{nonpipelined}}}{\text{Number of pipe stages}}$$

$$\text{Number of pipe stages} \approx \text{speedup} = \frac{\text{Time between instructions}_{\text{nonpipelined}}}{\text{Time between instructions}_{\text{pipelined}}}$$

- If not balanced, speedup is less
- Speedup due to increased throughput
  - Latency (time for each instruction) does not decrease

Continued from Prior Example

- What would happen if we increased the number of instructions?
  - E.g., extend the previous figures to 1,000,003 instructions (i.e., add 1,000,000 instructions)

- Nonpipelined time?

$$1{,}000{,}003 \times 800 = 800{,}002{,}400\ ps$$

- Pipelined time?

$$(\mathrm{ns} + \mathrm{ni} - 1) \times \mathrm{ts} = (5 + 1{,}000{,}003 - 1) \times 200 = 200{,}001{,}400\ ps$$

- Speedup?

$$Speedup = \frac{800{,}002{,}400\ ps}{200{,}001{,}1400\ ps} \approx 4.0$$

Speedup is not 5, or ns, because stages are not balanced, as we have 100ps for register read or write, and 200ps for other stages

If all 5 stages are balanced, i.e. each stage takes 800 ps / 5 = 160 ps, then

$$Speedup \approx 5.0$$

Readings

- Chapter 3, 3.1