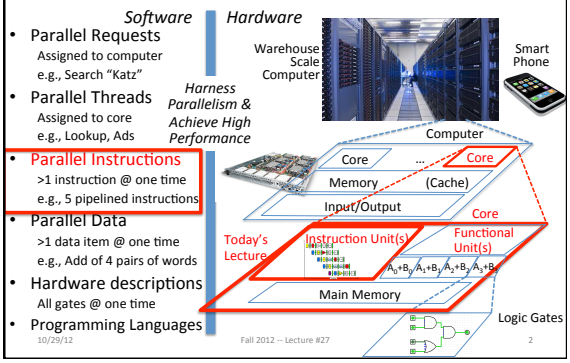
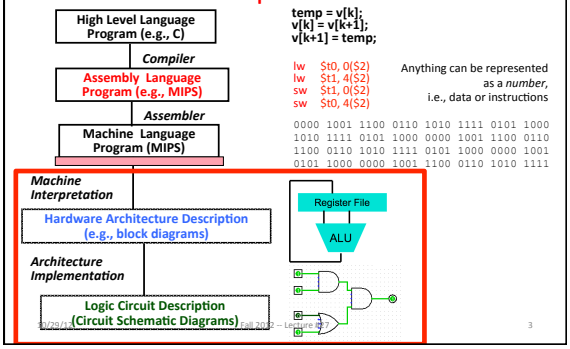
CS 61C:
Great Ideas in Computer Architecture
Control and Pipelining, Part I

Instructors:
Krste Asanovic, Randy H. Katz
http://inst.eecs.Berkeley.edu/~cs61c/fa12

You Are Here!



Levels of Representation/ Interpretation



Agenda

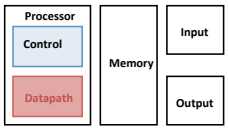
- Pipelined Execution
- Administrivia
- Pipelined Datapath

Agenda

- Pipelined Execution
- Administrivia
- Pipelined Datapath

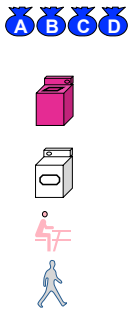
Review: Single-Cycle Processor

- Five steps to design a processor:
  1. Analyze instruction set → datapath requirements
  2. Select set of datapath components & establish clock methodology
  3. Assemble datapath meeting the requirements: re-examine for pipelining
  4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
  5. Assemble the control logic
     - Formulate Logic Equations
     - Design Circuits

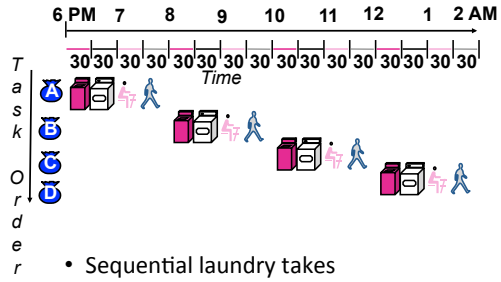


Pipeline Analogy: Doing Laundry

- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, fold, and put away
  - Washer takes 30 minutes
  - Dryer takes 30 minutes
  - "Folder" takes 30 minutes
  - "Stasher" takes 30 minutes to put clothes into drawers

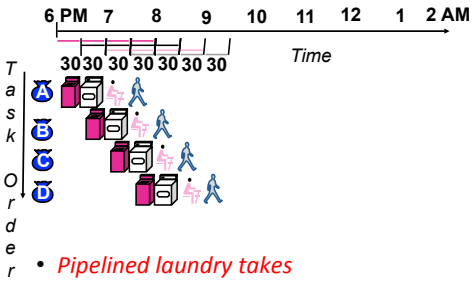


Sequential Laundry



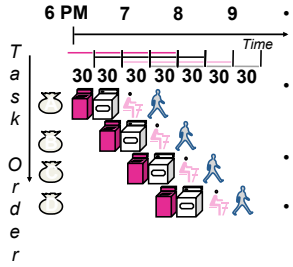
- Sequential laundry takes 8 hours for 4 loads

Pipelined Laundry



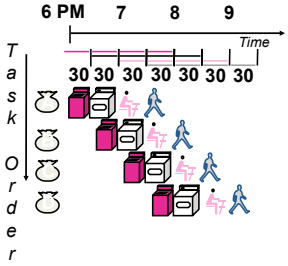
- *Pipelined laundry takes 3.5 hours for 4 loads!*

Pipelining Lessons (1/2)



- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number pipe stages (4 in this case)
- Time to fill pipeline and time to drain it reduces speedup: 8 hours/3.5 hours or 2.3X v. potential 4X in this example

Pipelining Lessons (2/2)



- Suppose new Washer takes 20 minutes, new Stasher takes 20 minutes. How much faster is pipeline?
- Pipeline rate limited by slowest pipeline stage
- Unbalanced lengths of pipe stages reduces speedup

Agenda

- Pipelined Execution
- Administrivia
- Pipelined Datapath

Administrivia

- Project #4, Labs #10 and #11, (Last) HW #6 posted
  - Project due 11/11 (Sunday after next)
  - Project is not difficult, but is long: don't wait for the weekend when it is due! Start early!
  - Look at Logisim labs before you start on Project #4
    - Lots of useful tip and tricks for using Logisim in those labs
- TA Sung Roa will hold extended office hours on the project this coming weekend

Administrivia

- Reweighting of Projects
  - Still 40% of grade overall
    - Project 1: 5%
    - Project 2: 12.5%
    - Project 3: 10%
    - Project 4: 12.5%
    - Optional Extra Credit Project 5: up to 5% extra
      - Gold, Silver, and Bronze medals to the three fastest projects
      - Code size and aesthetics also a criteria for recognition
      - Winning projects as selected by the TAs will be highlighted in class

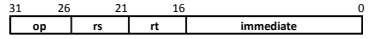
Agenda

- Pipelined Execution
- Administrivia
- Pipelined Datapath

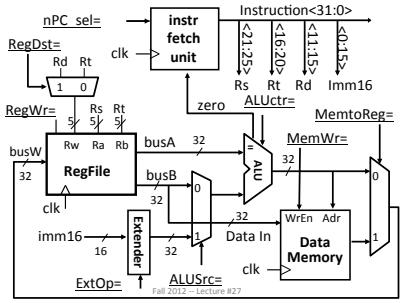
Review: RISC Design Principles

- "A simpler core is a faster core"
- Reduction in the number and complexity of instructions in the ISA → simplifies pipelined implementation
- Common RISC strategies:
  - *Fixed* instruction length, generally a single word (MIPS = 32b); Simplifies process of fetching instructions from memory
  - *Simplified* addressing modes; (MIPS just register + offset) Simplifies process of fetching operands from memory
  - *Fewer* and *simpler* instructions in the instruction set; Simplifies process of executing instructions
  - *Simplified memory access:* only load and store instructions access memory;
  - *Let the compiler do it.* Use a good compiler to break complex high-level language statements into a number of simple assembly language statements

Review: Single Cycle Datapath



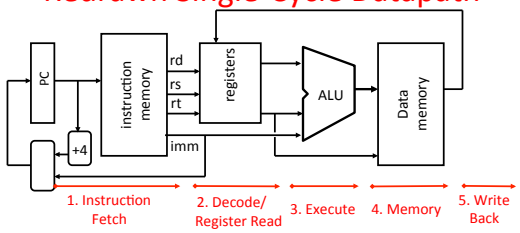
- Data Memory {R[rs] + SignExt[imm16]} = R[rt]



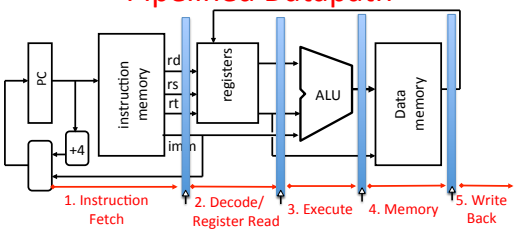
Steps in Executing MIPS

1) IF: Instruction Fetch, Increment PC
2) ID: Instruction Decode, Read Registers
3) EX: Execution
   Mem-ref: Calculate Address
   Arith-log: Perform Operation
4) Mem:
   Load: Read Data from Memory
   Store: Write Data to Memory
5) WB: Write Data Back to Register

Redrawn Single-Cycle Datapath



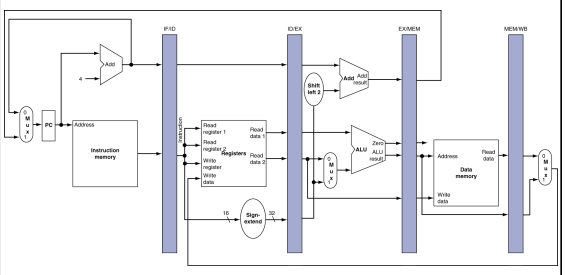
Pipelined Datapath



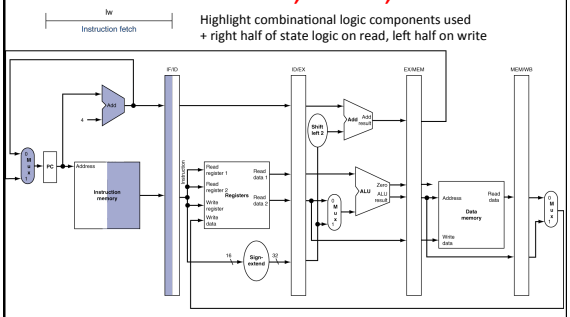
- Add registers between stages
  - Hold information produced in previous cycle
- 5 stage pipeline; clock rate potential 5X faster

More Detailed Pipeline

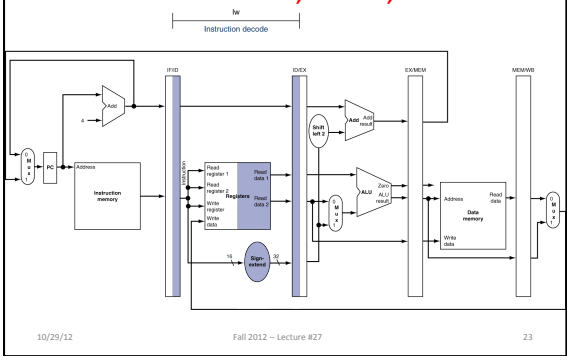
Registers named for adjacent stages, e.g., IF/ID



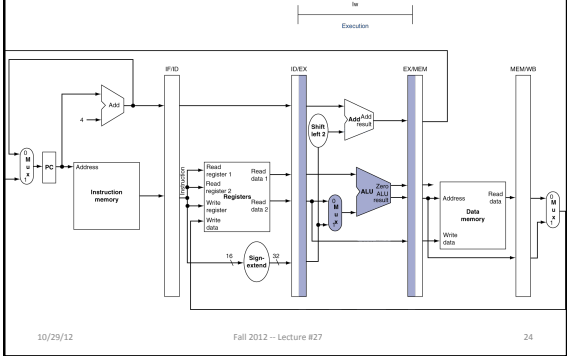
IF for Load, Store, …



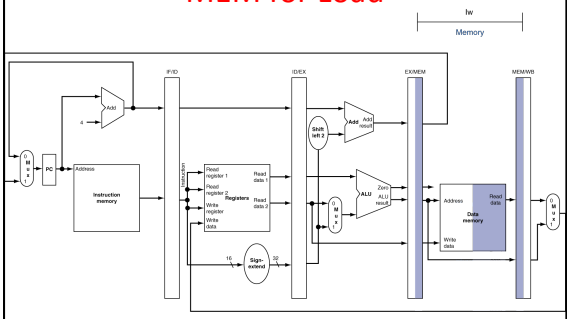
ID for Load, Store, …



EX for Load

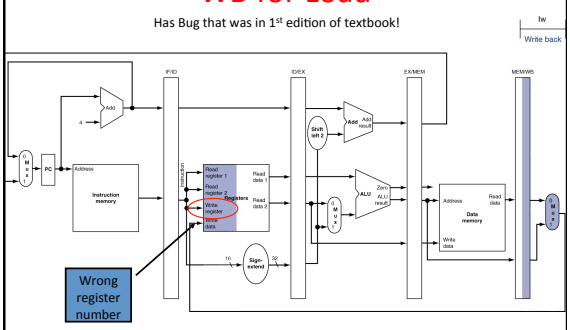


MEM for Load

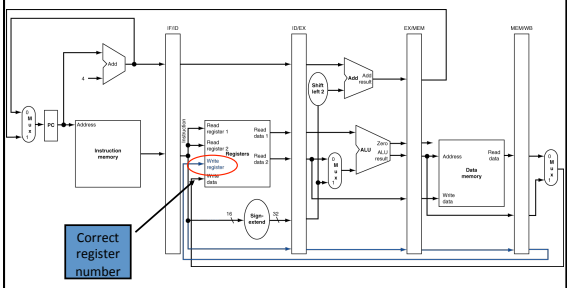


WB for Load

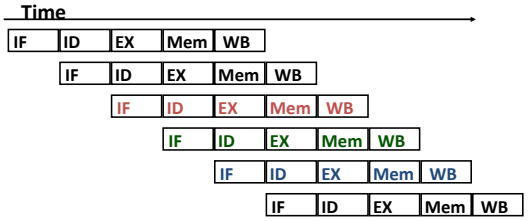
Has Bug that was in 1st edition of textbook!



Corrected Datapath for Load

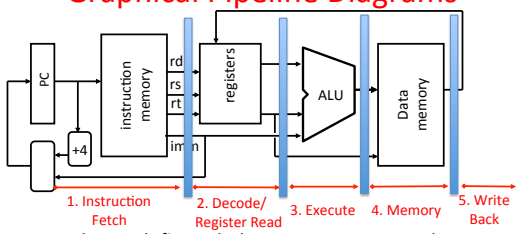


Pipelined Execution Representation

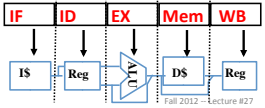


- Every instruction must take same number of steps, also called pipeline stages, so some will go idle sometimes

Graphical Pipeline Diagrams

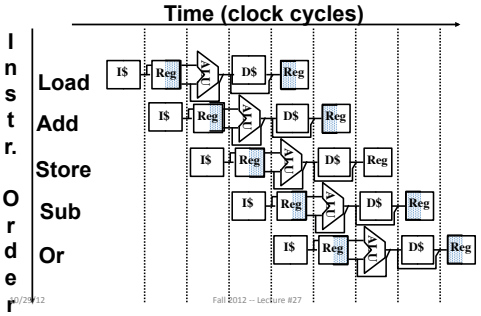


- Use datapath figure below to represent pipeline



Graphical Pipeline Representation

(In Reg, right half highlight read, left half write)



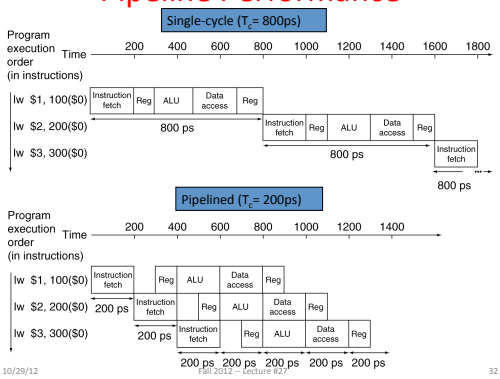
Pipeline Performance

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

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Student Roulette?

Pipeline Performance



Pipeline Speedup

- If all stages are balanced
  - i.e., all take the same time
  - Time between instructions$_{pipelined}$ = $\frac{\text{Time between instructions}_{nonpipelined}}{\text{Number of stages}}$
- If not balanced, speedup is less
- Speedup due to increased throughput
  - Latency (time for each instruction) does not decrease

Instruction Level Parallelism (ILP)

- Another parallelism form to go with Request Level Parallelism and Data Level Parallelism
  - RLP – e.g., Warehouse Scale Computing
  - DLP – e.g., SIMD, Map-Reduce
- *ILP – e.g., Pipelined Instruction Execution*
  - 5 stage pipeline => 5 instructions executing simultaneously, one at each pipeline stage

And in Conclusion, …

The BIG Picture

- Pipelining improves performance by increasing instruction throughput: exploits ILP
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
- Key enabler is placing registers between pipeline stages