

EEC 485 High Performance Arch. (Fall 2006)

Chapter 6 Pipelining

Chansu Yu

Cleveland State University

Table of Contents

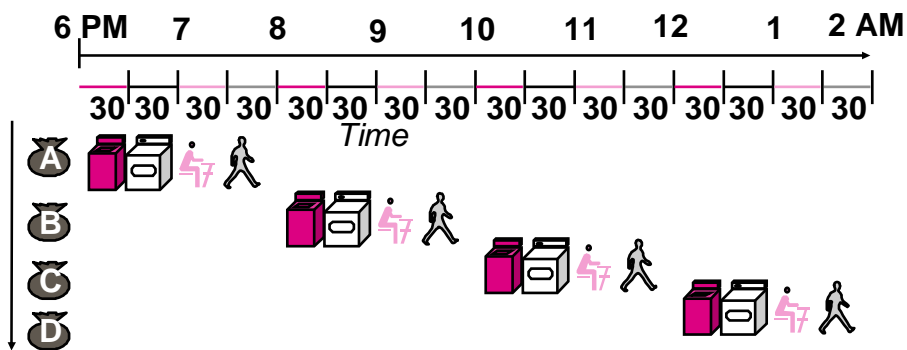
- Ch. 1-2 Introduction & Performance
 - Ch. 3 Instruction: Machine Language
 - 3.1 Introduction
 - 3.2 Operations (arithmetic, memory operations)
 - 3.3 Operands
 - 3.4 Representing instructions
 - 3.5 Control flow operations
 - 3.6 Supporting procedures
 - 3.7 Beyond numbers
 - 3.8 MIPS addressing
 - 3.9 Starting a program
 - 3.10 Example
 - Ch. 4 CPU Implementation: Arithmetic
 - 4.1 Introduction
 - 4.2 Signed and unsigned numbers
 - 4.3 Addition and subtraction
 - 4.4 Logical operations
 - 4.5 Constructing an Arithmetic Logic Unit (ALU)
 - 4.6 Multiplication
 - 4.7 Division
 - 4.8 Floating point
 - 4.9-4.14 Etc.
 - Ch. 5 CPU Implementation: All others
 - 5.1 Introduction
 - 5.2 Building a Datapath
 - 5.3 Implementation Scheme
 - 5.4 A Multicycle Implementation
 - 5.5 Microprogramming: Simplifying Control Design
 - 5.6 Exceptions
 - 5.7-5.12 Etc.
 - Ch. 6-9 Advanced topics
- Software interface*
- Hardware interface*
- Key parts of this course**
- Ch.6 Pipelining*
Ch.7 Cache
=> Issue is "High Performance"

Laundry Example

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



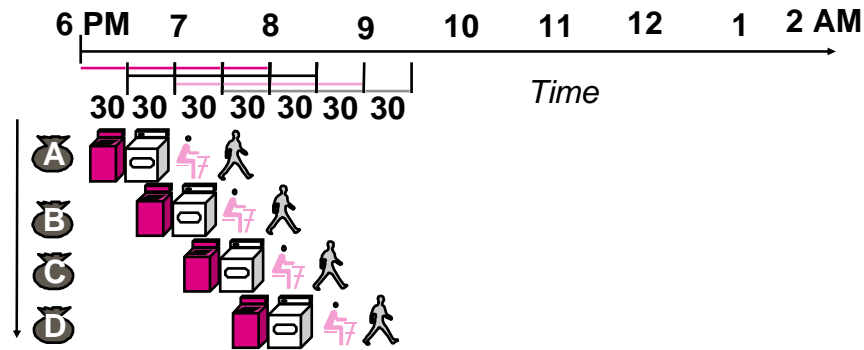
Sequential Laundry



Sequential laundry takes 8 hours for 4 loads

If they learned pipelining, how long would laundry take?

Faster Laundry – Pipelining

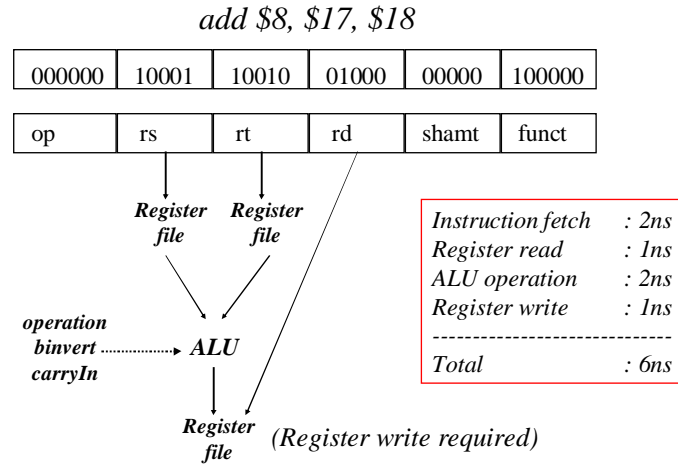


Faster laundry takes 3.5 hours for 4 loads!

5 Stages of MIPS

| Step name | Action for R-type instructions | Action for memory-reference instructions | Action for branches | Action for jumps |
|---|---|--|----------------------------------|---|
| Instruction fetch | $IR = \text{Memory}[PC]$ $PC = PC + 4$ | | | |
| Instruction decode/register fetch | $A = \text{Reg}[\text{IR}[25-21]]$ $B = \text{Reg}[\text{IR}[20-16]]$ $ALUOut = PC + (\text{sign-extend}(\text{IR}[15-0]) \ll 2)$ | | | |
| Execution, address computation, branch/ jump completion | $ALUOut = A \text{ op } B$ | $ALUOut = A + \text{sign-extend}(\text{IR}[15-0])$ | if $(A == B)$ then $PC = ALUOut$ | $PC = \text{PC}[31-28] \parallel (\text{IR}[25-0] \ll 2)$ |
| Memory access or R-type completion | $\text{Reg}[\text{IR}[15-11]] = ALUOut$ | Load: $\text{MDR} = \text{Memory}[ALUOut]$ or Store: $\text{Memory}[ALUOut] = B$ | | |
| Memory read completion | | Load: $\text{Reg}[\text{IR}[20-16]] = \text{MDR}$ | | |

Execution time for “add”

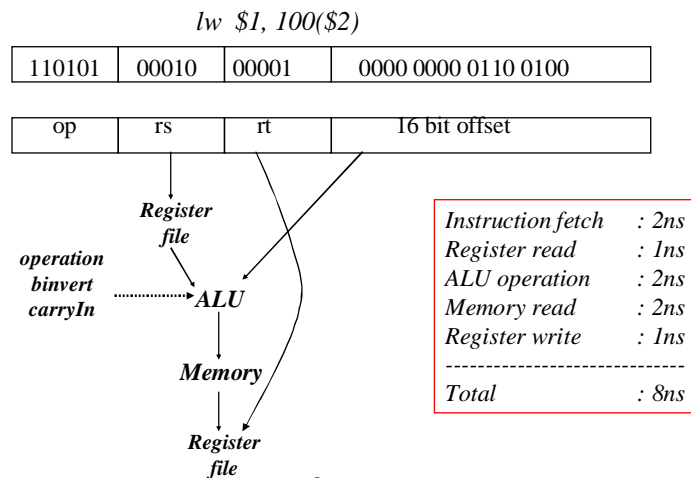


Cleveland State University

7

c.yu91@csuohio.edu

Execution time for “load”

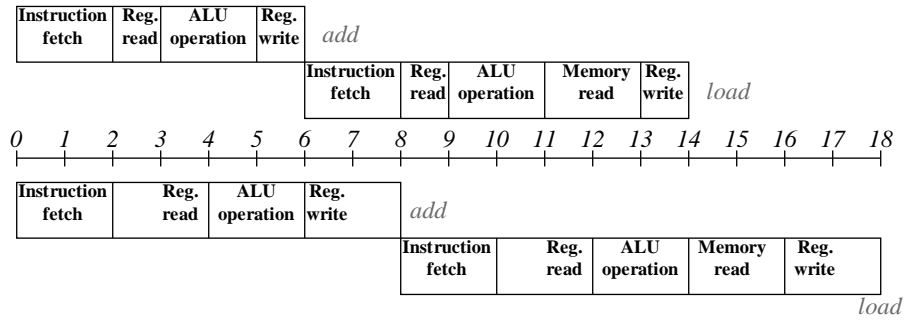


Cleveland State University

8

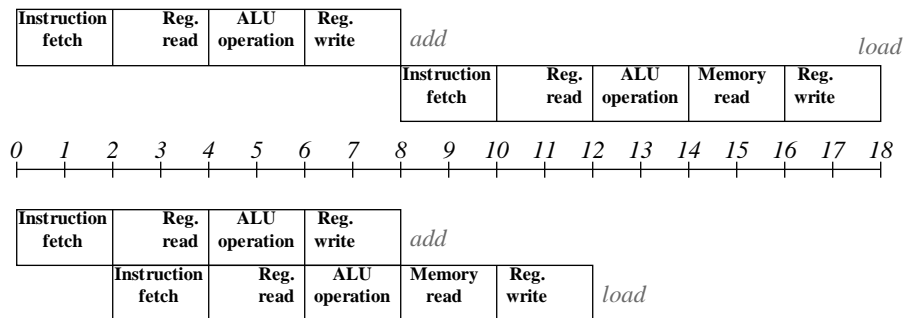
c.yu91@csuohio.edu

Single cycle vs. Multicycle



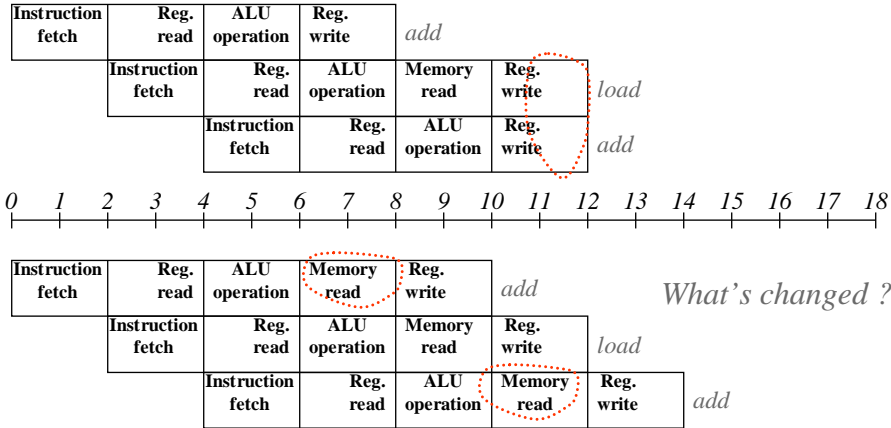
What are the advantages of multicycle implementation ?
What are the disadvantages of multicycle implementation ?

Multicycle vs. Pipelined



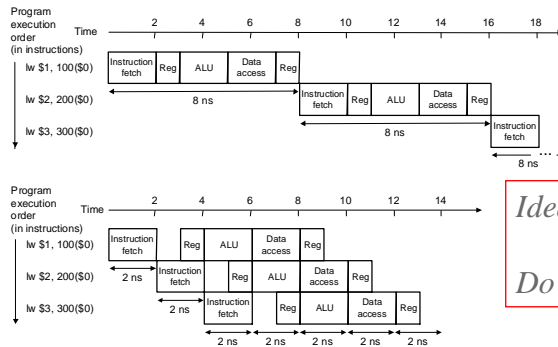
What are the advantages of pipelined implementation ?
What are the disadvantages of pipelined implementation ?

5 Stages in MIPS



Speedup

- ❑ Improve performance by increasing instruction throughput



Stage Length

□ Clock cycle

- Each of 5 stages takes the similar amount of time
- And it must be small as much as possible

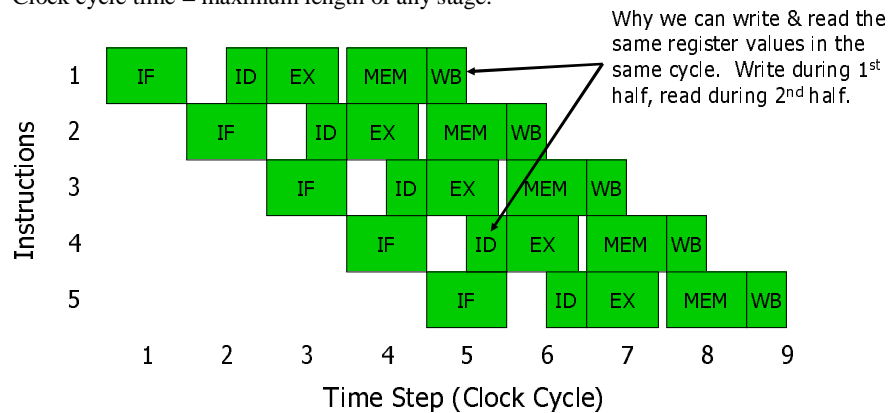
□ What makes it easy

- all instructions are the same length
- just a few instruction formats

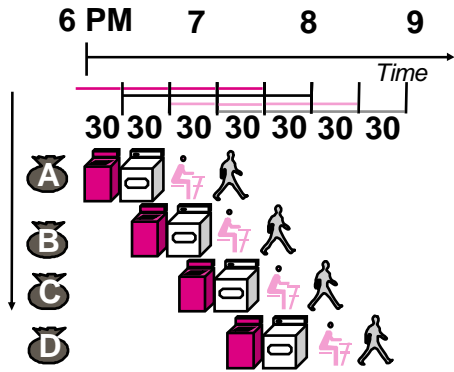
Stage Length

Stages' may require different amounts of time

Clock cycle time = maximum length of any stage.

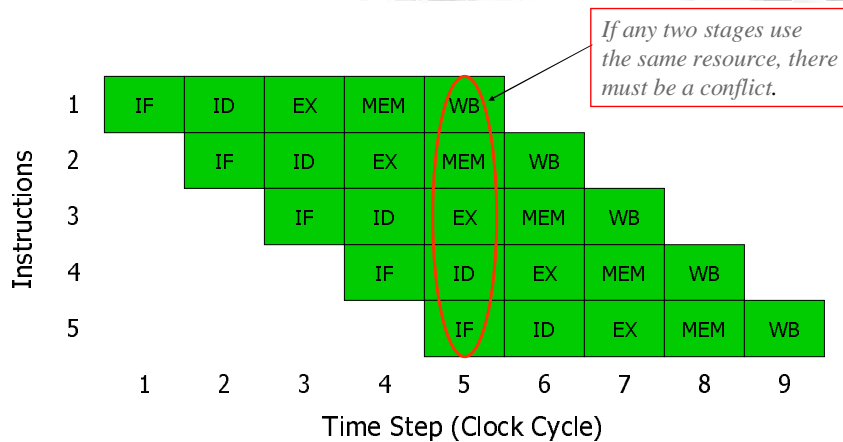


Lessons from Pipelined Laundry



- ✓ Pipelining doesn't help latency of single task, it helps throughput of entire workload
- ✓ Potential speedup = Number pipe stages
- ✓ Pipeline rate limited by slowest pipeline stage
- ✓ Unbalanced lengths of pipe stages reduces speedup
- ✓ Time to "fill" pipeline and time to "drain" it reduces speedup
- ✓ Multiple tasks operating simultaneously using different resources – any dependencies, any conflicts ???

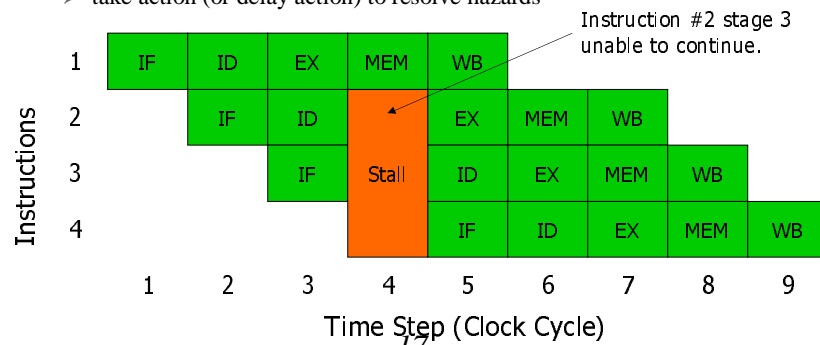
Can pipelining get us into trouble?



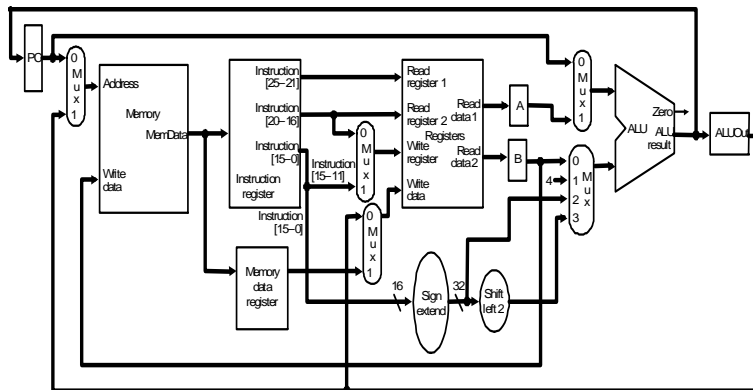
Hazards

Hazard = when an instruction's stage is unable to execute during the current cycle.

- Can always resolve hazards by waiting
 - pipeline control must detect the hazard
 - take action (or delay action) to resolve hazards



Resources in Multicycle MIPS



Hazards

- ❑ Resource conflict
 - At any moment, 5 pipeline stages are all active doing something but with different instructions
 - A resource in each stage must not be used for other stages

- ❑ What makes it easy
 - memory operands appear only in loads and stores

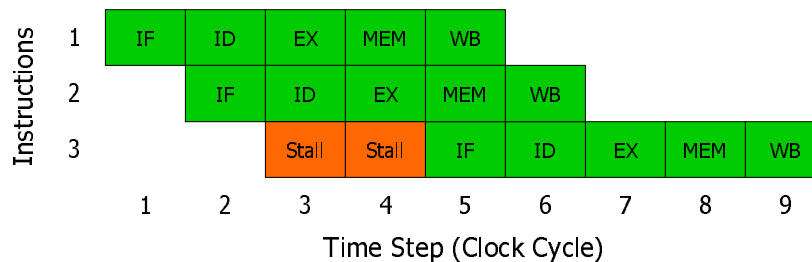
- ❑ What makes it hard
 - structural hazards: suppose we had only one memory (\leq we already remove this type of hazards)
 - control hazards: need to worry about branch instructions
 - data hazards: an instruction depends on a previous instruction

Structural Hazards

A needed functional unit is busy executing a previous instruction
(Attempt to use the same resource two different ways at the same time)

Example:

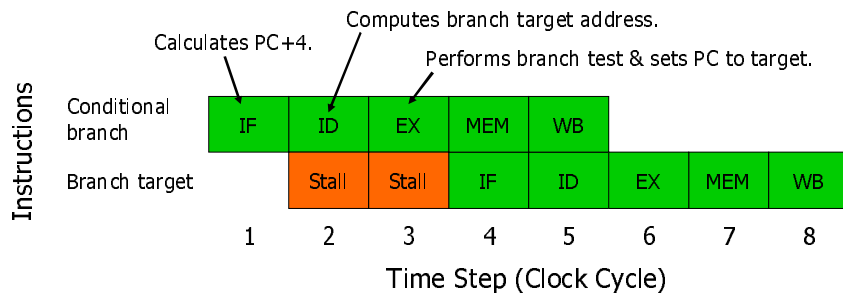
- Our sample MIPS pipeline has none.
- What if PC+4 computation used main ALU instead of separate adder?



Control Hazards

While executing a previous branch, next instruction address might not yet be known.

(attempt to make a decision before condition is evaluated)



Data Hazards

Needed data still being computed by previous instruction.

(attempt to use item before it is ready)

