

Computer Architecture

2006 Comprehensive Exam Review

Jon Su, Philip Guo, Ewen Cheslack-Postava

Exam Review Outline

- Main Topics from 2000-2006 Comps.
 - MIPS Instruction Set Architecture
 - Pipelining
 - Caches

MIPS ISA

| Name | # | Usage | Callee Preserved |
|------------|--------|----------------|------------------|
| \$zero | 0 | Always zero | N/A |
| \$at | 1 | Assembler temp | No |
| \$v0..\$v1 | 2...3 | Return values | No |
| \$a0..\$a3 | 4...7 | Arguments | No |
| \$t0..\$t7 | 8...15 | Temporaries | No |
| \$s0..\$s7 | 16..23 | Saved | Yes |
| \$t8..\$t9 | 24..25 | Temporaries | No |
| \$k0..\$k1 | 26..27 | OS use | No |
| \$gp | 28 | Global pointer | Yes |
| \$sp | 29 | Stack pointer | Yes |
| \$fp | 30 | Frame pointer | Yes |

| | |
|---------------------------------|----------------|
| Register addressing | R-type add |
| Base or displacement addressing | I-type ld/st |
| Immediate addressing | I-type addi |
| PC-relative addressing | I-type beq/bne |
| Pseudo-direct addressing | J-type j/jal |

R-type

| OpCode | SrcReg1 | SrcReg2 | DestReg | Shift amt | Function |
|--------|---------|---------|---------|-----------|----------|
| 6 | 5 | 5 | 5 | 5 | 6 |

I-type

| OpCode | SrcReg | DestReg | Imm/Offset |
|--------|--------|---------|------------|
| 6 | 5 | 5 | 16 |

J-type

| OpCode | Address-offset |
|--------|----------------|
| 6 | 26 |

Design Principles

- Simplicity favors regularity
- Smaller is faster
- Good design demands good compromises
- Make the common case fast

Performance

- Performance = $1 / \text{Execution Time}$
- Speed up = $\text{Time}(B) / \text{Time}(A)$
- Instruction Count, Cycles Per Instruction, Clock Cycle time
- Amdahl's Law

Performance Pitfalls & Fallacies

- Forgetting Amdahl's law
- Hardware independent metrics (e.g. Code size)
- Using MIPS as performance metric
- Synthetic benchmarks
- Using arithmetic mean for normalized execution times

Performance Sample Problem

Given the following frequencies and CPIs for each type of instruction supported by the processor:

| Type | Frequency | CPI |
|-----------------------|-----------|-----|
| Load | 30% | 1.6 |
| Store | 20% | 1.4 |
| Arithmetic operations | 35% | 1 |
| Branches & jumps | 15% | 1.5 |

50% of the stores are used to push values onto the stack, 1/3 of the loads are used to pop values from the stack. All pushes and pops operate on 32b integer numbers. Push and pop are to be implemented in the ISA. They will increase the clock cycle of the processor by 5%. Calculate the overall performance improvement if the new instructions are added.

Performance Sample Problem, Solution

$$\text{Speedup} = \text{Time}_{\text{old}} / \text{Time}_{\text{new}} = (\text{IC}_{\text{old}} * \text{CPI}_{\text{old}} * \text{CC}_{\text{old}}) / (\text{IC}_{\text{new}} * \text{CPI}_{\text{new}} * \text{CC}_{\text{new}})$$

$$\text{CPI}_{\text{old}} = \text{Sum}_{\text{weighted}}(\text{CPIs}) = (.3*1.6 + .2*1.4 + .35*1 + .15*1.5) = 1.335$$

$$\text{CC}_{\text{new}} = 1.05 * \text{CC}_{\text{old}}$$

$$\text{IC}_{\text{new}} = (1 - \text{pushes-pops}) * \text{IC}_{\text{old}} = (1 - .5*.2 - .33*.3) * \text{IC}_{\text{old}} = .80 \text{ IC}_{\text{old}}$$

$$\text{CPI}_{\text{new}} = (.3*1.6 + .2*1.4 + .15*1 + .15*1.5) / .8 = 1.41875$$

$$\begin{aligned} \text{Speedup} &= (\text{IC}_{\text{old}} * \text{CPI}_{\text{old}} * \text{CC}_{\text{old}}) / (\text{IC}_{\text{new}} * \text{CPI}_{\text{new}} * \text{CC}_{\text{new}}) \\ &= (\text{IC}_{\text{old}} * 1.335 * \text{CC}_{\text{old}}) / (.8 * \text{IC}_{\text{old}} * 1.41875 * 1.05 * \text{CC}_{\text{old}}) \\ &= 1.12 \end{aligned}$$

12% speedup

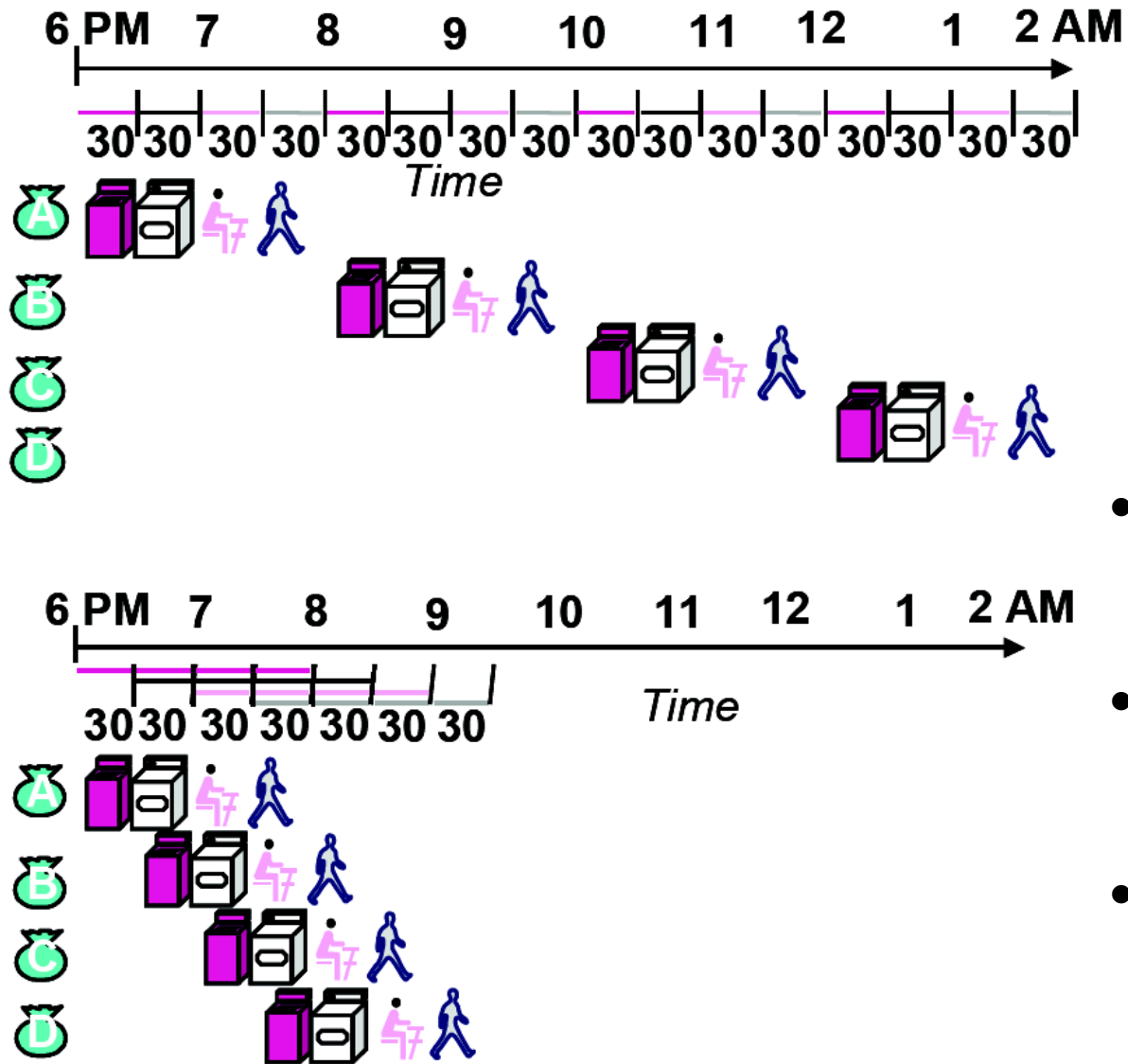
Pipelining

- Pipelining overview
- MIPS 5-stage pipeline
- Pipeline hazards
 - Structural hazards
 - Control hazards
 - Data hazards

These slides provide an overview; the handout contains much more details.

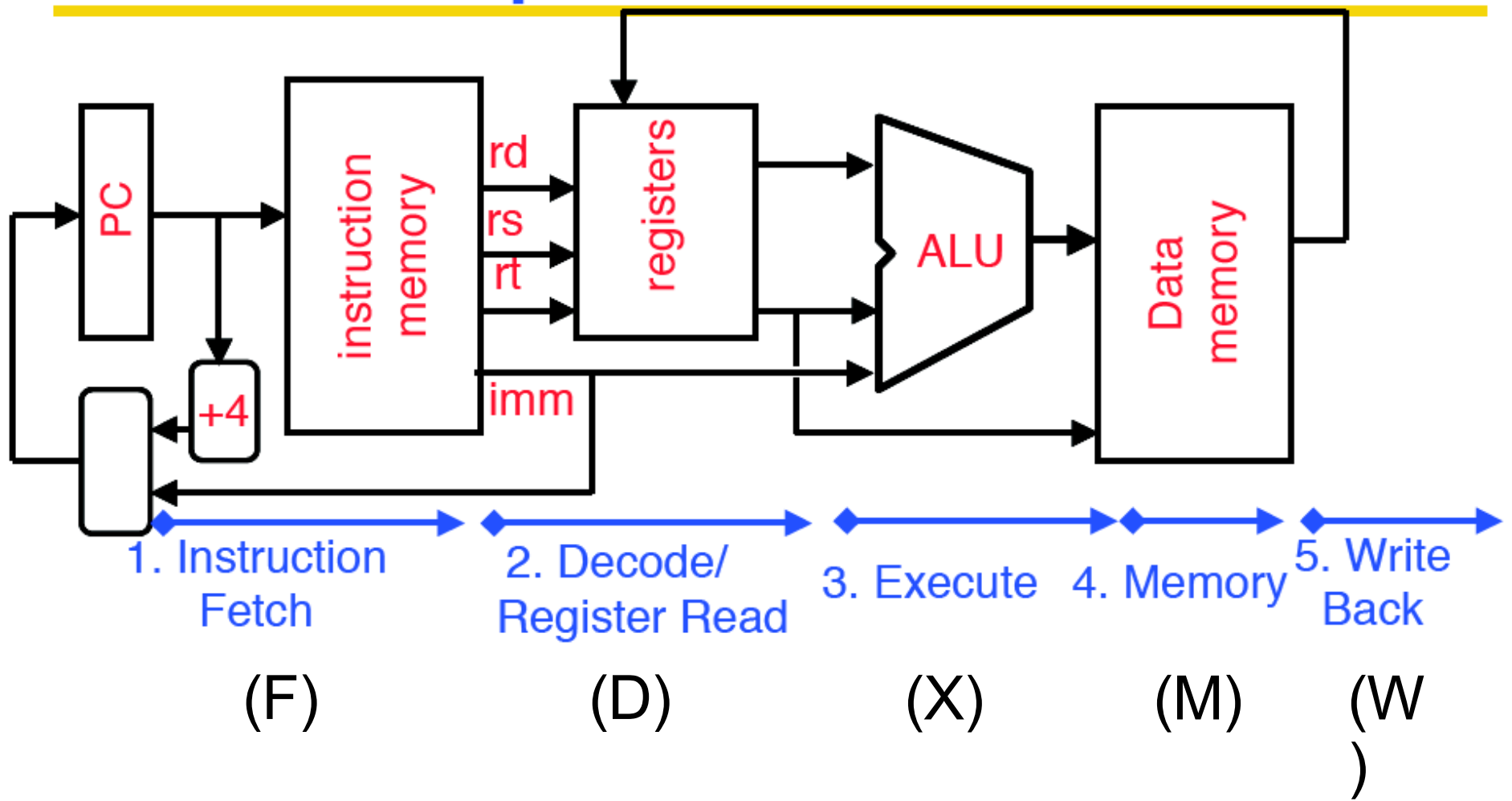
Diagrams taken from Dan Garcia's Fall 2004 CS61C lecture slides (U.C. Berkeley)

Pipelining overview

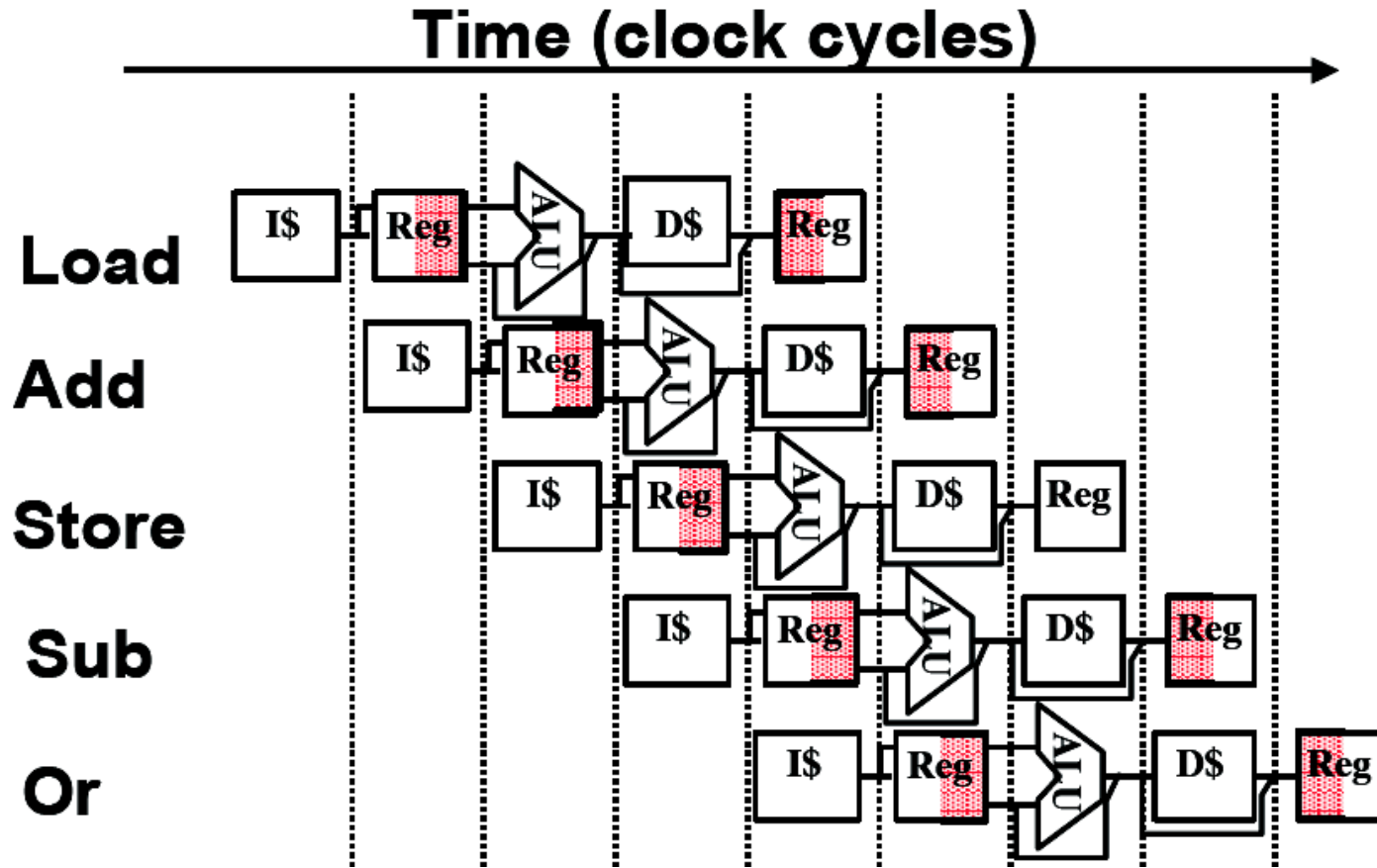


- Divide work into discrete stages
- Maximize utilization of each stage
- Improves *throughput*, NOT *latency*

MIPS 5-stage pipeline

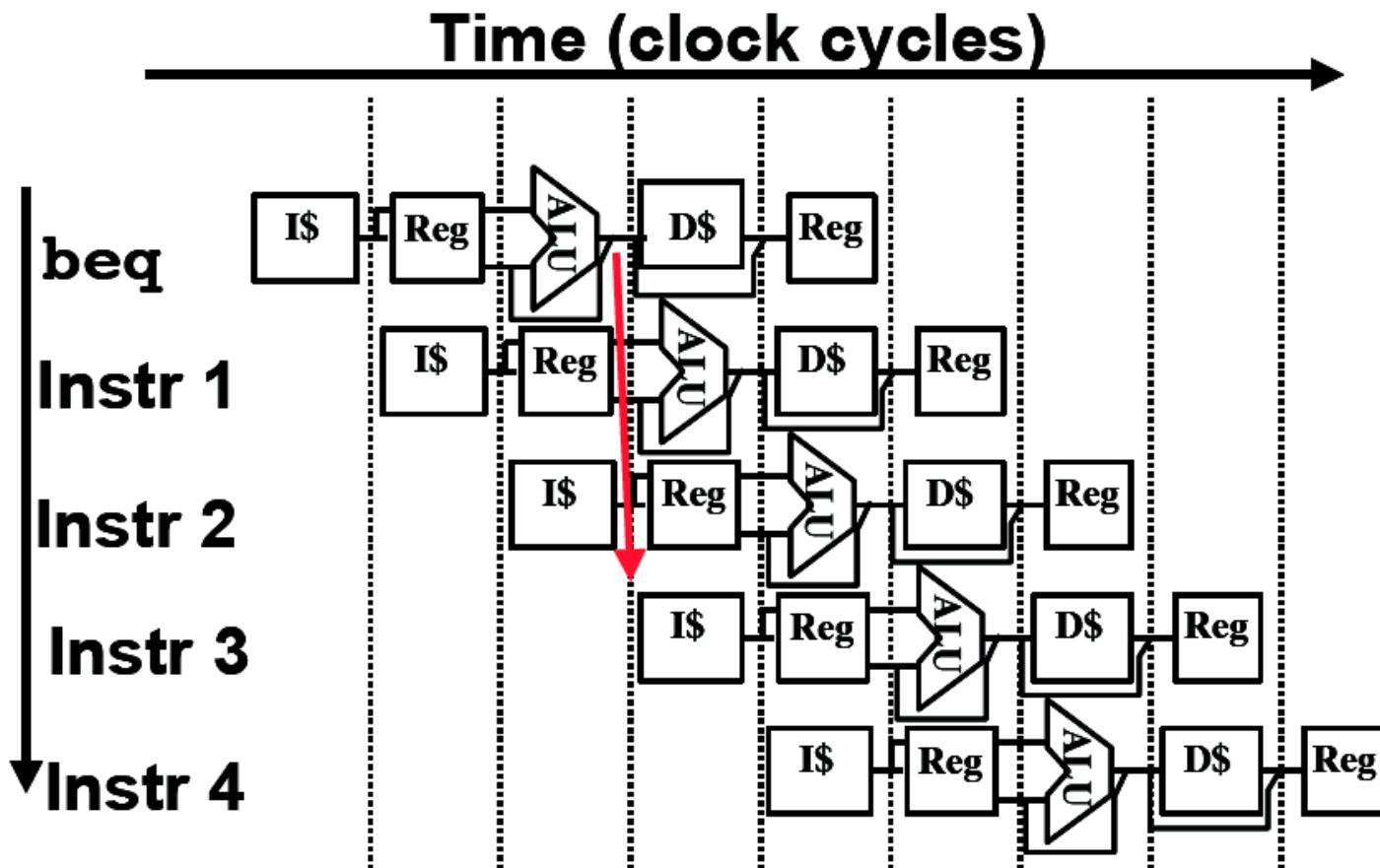


Pipeline hazards - structural



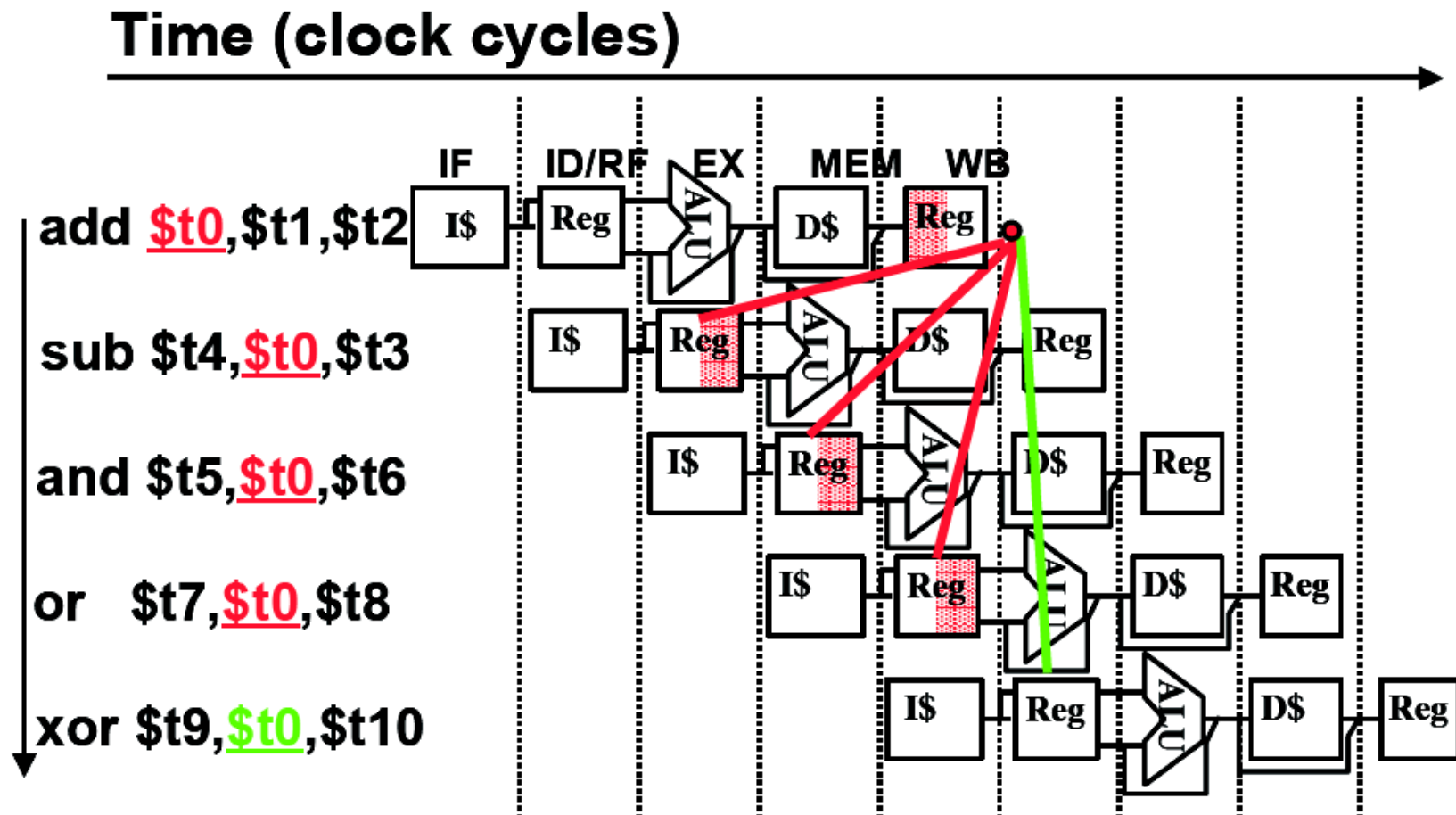
- Reading from 2 different slots in memory
- Reading from and writing to the register file

Pipeline hazards - control



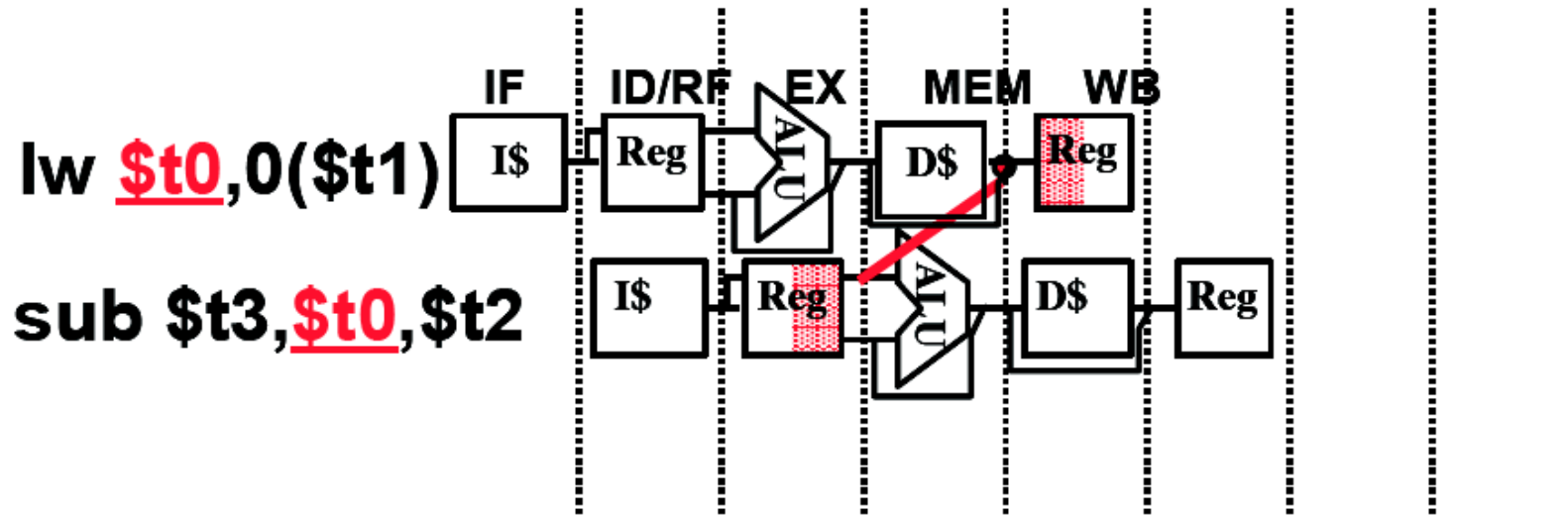
- Branches take 3 cycles to complete
- Only take 2 cycles if comparator moved to D stage
- Can average between 1 and 2 cycles by using *delayed branching* (always execute the instruction after the branch)
- Jumps also 2 cycles; can lower with same trick

Pipeline hazards - data (1 of 2)



- Solution: Add forwarding (bypass) from end of execute (X) stage to beginning of X stage and from end of memory (M) stage to the beginning of X stage

Pipeline hazards - data (2 of 2)



- Forwarding can't solve load data hazard problem because data isn't available until after Memory stage
- Still need 1 delay cycle, implemented as a hardware interlock or a NOP instruction
- Compiler can try to fill load delay slot with a useful instruction to reduce CPI to between 1 and 2

Caches

- Goal: Make memory access fast, while keeping cost down
- Principle of Locality
 - Only a small part of a program is used at any time
- Locality exhibited in programs
 - Temporal Locality (for loops)
 - Spatial Locality (sequential execution of code)

Cache Performance

- Why have memory hierarchies – make more memory accessible but *fast*
- How to evaluate performance of caches:
 - **Hit rate** - % of accesses that are found in the cache
 - **Hit time** – accessing a block
 - **Miss time** – retrieving a block and replacing it

Direct Mapped Caches

- Each memory address maps to **exactly** one location in the cache
- Layout
 - Low order bits *index* into cache
 - Remaining bits form *tag*
 - *Valid* bit
- Read miss – straightforward update of cache
- Write miss – *write through vs. write back*

Multiword Caches

- Direct mapped caches don't take advantage of spatial locality
- **Multiword Caches** – each block contains multiple words
 - Improves *spatial locality*, but only to a point
 - Read misses are resolved the same
 - Write misses now force a fetch before overwriting

Associativity

- Flexible word placement – allow words to be placed in different blocks
- **Fully associative** – a word can be placed in any block in the cache
- **Set associative** – a word can be placed in one of a fixed **n** number of locations
 - *set associative vs. fully associative*

Associativity

- Indexing into set associative caches
 - Low bits are still used to index into cache, but now index into a set of blocks
 - Each block of the cache still has a tag
 - Tags are checked in parallel
- Resolving misses:
 - Use a replacement scheme like LRU to choose which block in a set to replace

What We've Covered

- Main Topics from 2000-2006 Comps.
 - MIPS Instruction Set Architecture
 - Pipelining
 - Caches

Any Questions?