

# CS 61C: Great Ideas in Computer Architecture

## Pipelining Hazards

Guest Lecturer: Justin Hsia

4/12/2013

Spring 2013 -- Lecture #31

1

## Great Idea #4: Parallelism

**Software**

- Parallel Requests  
Assigned to computer  
e.g. search "Garcia"
- Parallel Threads  
Assigned to core  
e.g. lookup, ads
- Parallel Instructions  
> 1 instruction @ one time  
e.g. 5 pipelined instructions
- Parallel Data  
> 1 data item @ one time  
e.g. add of 4 pairs of words
- Hardware descriptions  
All gates functioning in parallel at same time

**Hardware**

Warehouse Scale Computer

Smart Phone

Leverage Parallelism & Achieve High Performance

4/12/2013

Spring 2013 -- Lecture #31

2

## Review of Last Lecture

- Implementing controller for your datapath
  - Take decoded signals from instruction and generate control signals
  - Use "AND" and "OR" Logic scheme
- Pipelining improves performance by exploiting Instruction Level Parallelism
  - 5-stage pipeline for MIPS: IF, ID, EX, MEM, WB
  - Executes multiple instructions in parallel
  - What can go wrong???

4/12/2013

Spring 2013 -- Lecture #31

3

## Agenda

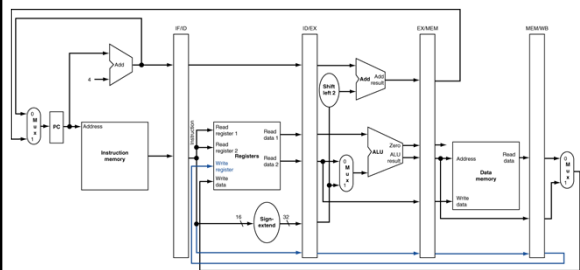
- Pipelining Performance
- Structural Hazards
- Administrivia
- Data Hazards
  - Forwarding
  - Load Delay Slot
- Control Hazards

4/12/2013

Spring 2013 -- Lecture #31

4

## Review: Pipelined Datapath

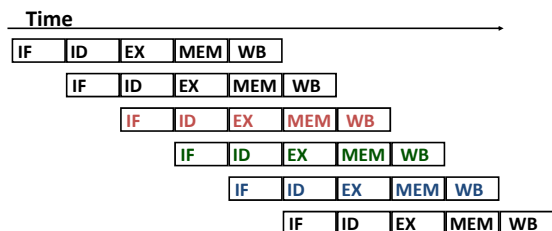


4/12/2013

Spring 2013 -- Lecture #31

5

## Pipelined Execution Representation

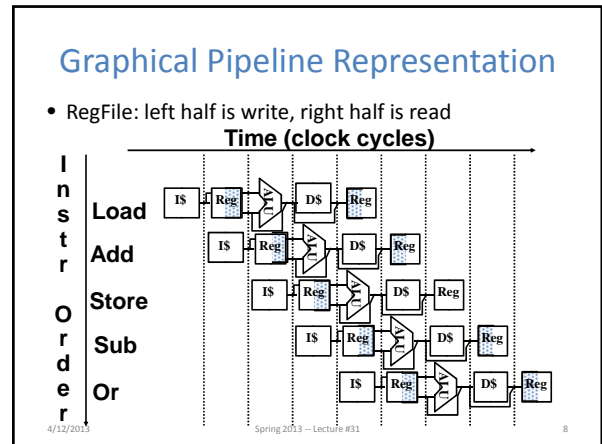
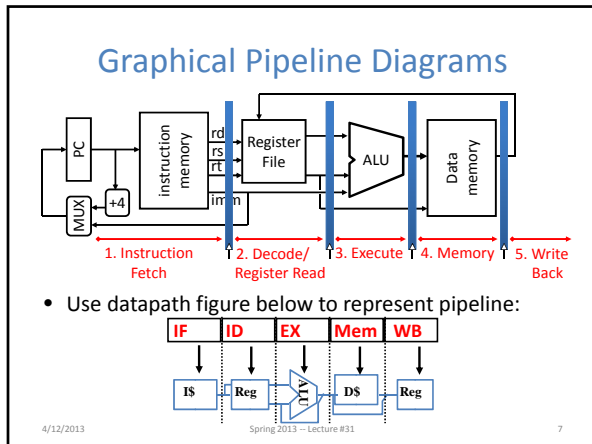


- Every instruction must take same number of steps, so some stages will idle
  - e.g. MEM stage for any arithmetic instruction

4/12/2013

Spring 2013 -- Lecture #31

6



### Pipelining Performance (1/3)

- Use  $T_c$  ("time between completion of instructions") to measure speedup
  - $T_{c, \text{pipelined}} \geq \frac{T_{c, \text{single-cycle}}}{\text{Number of stages}}$
  - Equality only achieved if stages are *balanced* (i.e. take the same amount of time)
- If not balanced, speedup is reduced
- Speedup due to increased *throughput*
  - *Latency* for each instruction does not decrease

7/24/2012 Summer 2012 -- Lecture #21 9

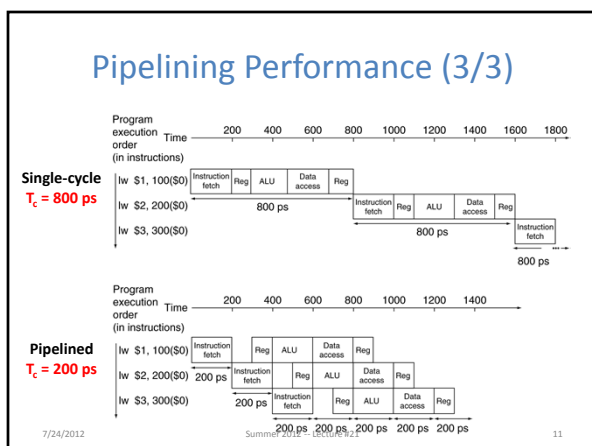
### Pipelining Performance (2/3)

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages

| 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      |

- What is pipelined clock rate?
  - Compare pipelined datapath with single-cycle datapath

7/24/2012 Summer 2012 -- Lecture #21 10



### Pipelining Hazards

A *hazard* is a situation that prevents starting the next instruction in the next clock cycle

- 1) *Structural hazard*
  - A required resource is busy (e.g. needed in multiple stages)
- 2) *Data hazard*
  - Data dependency between instructions
  - Need to wait for previous instruction to complete its data read/write
- 3) *Control hazard*
  - Flow of execution depends on previous instruction

4/12/2013 Spring 2013 -- Lecture #31 12

## Agenda

- Pipelining Performance
- **Structural Hazards**
- Administrivia
- Data Hazards
  - Forwarding
  - Load Delay Slot
- Control Hazards

4/12/2013

Spring 2013 -- Lecture #31

13

## 1. Structural Hazards

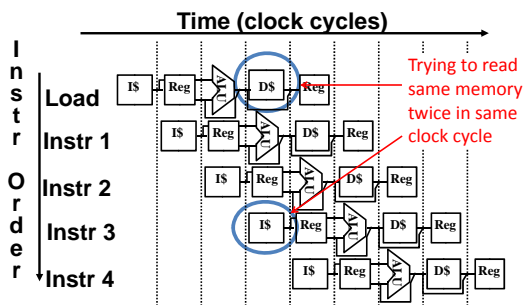
- Conflict for use of a resource
- MIPS pipeline with a single memory?
  - Load/Store requires memory access for data
  - Instruction fetch would have to *stall* for that cycle
    - Causes a pipeline “*bubble*”
- Hence, pipelined datapaths require separate instruction/data memories
  - Separate L1 I\$ and L1 D\$ take care of this

4/12/2013

Spring 2013 -- Lecture #31

14

## Structural Hazard #1: Single Memory

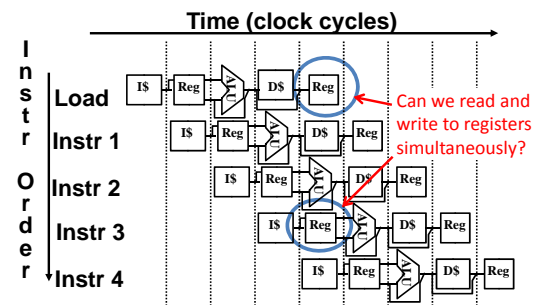


4/12/2013

Spring 2013 -- Lecture #31

15

## Structural Hazard #2: Registers (1/2)



4/12/2013

Spring 2013 -- Lecture #31

16

## Structural Hazard #2: Registers (2/2)

- Two different solutions have been used:
  - 1) Split RegFile access in two: Write during 1<sup>st</sup> half and Read during 2<sup>nd</sup> half of each clock cycle
    - Possible because RegFile access is *VERY* fast (takes less than half the time of ALU stage)
  - 2) Build RegFile with independent read and write ports
- **Conclusion:** Read and Write to registers during same clock cycle is okay

4/12/2013

Spring 2013 -- Lecture #31

17

## Agenda

- Pipelining Performance
- Structural Hazards
- **Administrivia**
- Data Hazards
  - Forwarding
  - Load Delay Slot
- Control Hazards

4/12/2013

Spring 2013 -- Lecture #31

18

## Administrivia

- Project 2: Performance Optimization
  - Part 1 due Sunday (4/14)
  - Part 2 released by Sunday night, due 4/21
  - Built-in performance competition for Part 2!
- Never too early to start looking at past exams!
- Dan has moved his OH because of traveling
  - See Piazza post @1492

4/12/2013

Spring 2013 – Lecture #31

19

## Agenda

- Pipelining Performance
- Structural Hazards
- Administrivia
- **Data Hazards**
  - Forwarding
  - Load Delay Slot
- Control Hazards

4/12/2013

Spring 2013 – Lecture #31

20

## 2. Data Hazards (1/2)

- Consider the following sequence of instructions:

```

add $t0, $t1, $t2
sub $t4, $t0, $t3
and $t5, $t0, $t6
or $t7, $t0, $t8
xor $t9, $t0, $t10
    
```

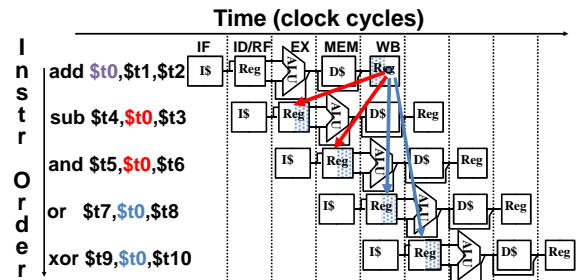
4/12/2013

Spring 2013 – Lecture #31

21

## 2. Data Hazards (2/2)

- Data-flow *backwards* in time are hazards



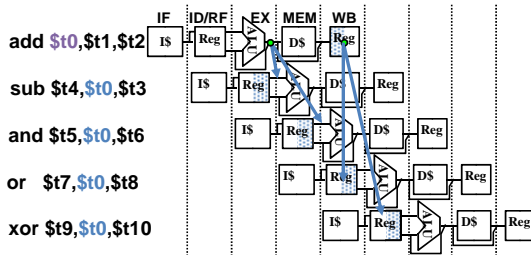
4/12/2013

Spring 2013 – Lecture #31

22

## Data Hazard Solution: Forwarding

- Forward result as soon as it is available
  - OK that it's not stored in RegFile yet



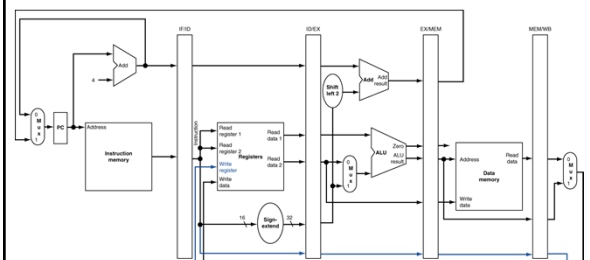
4/12/2013

Spring 2013 – Lecture #31

23

## Datapath for Forwarding (1/2)

- What changes need to be made here?



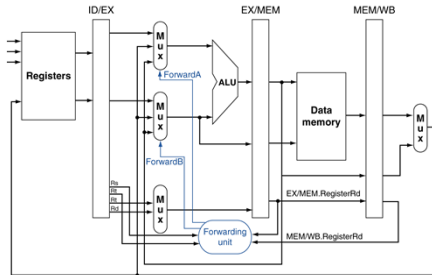
4/12/2013

Spring 2013 – Lecture #31

24

## Datapath for Forwarding (2/2)

- Handled by *forwarding unit*



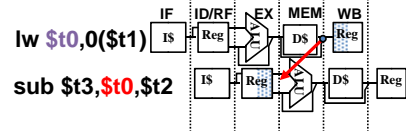
4/12/2013

Spring 2013 -- Lecture #31

25

## Data Hazard: Loads (1/4)

- Recall:** Dataflow backwards in time are hazards



- Can't solve all cases with forwarding
  - Must *stall* instruction dependent on load, then forward (more hardware)

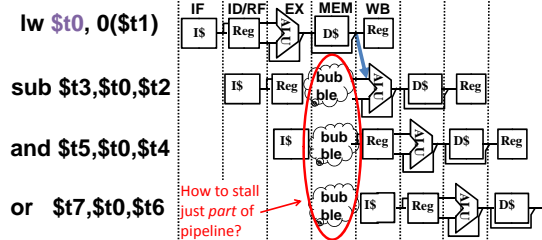
4/12/2013

Spring 2013 -- Lecture #31

26

## Data Hazard: Loads (2/4)

- Hardware stalls pipeline
  - Called "hardware interlock"



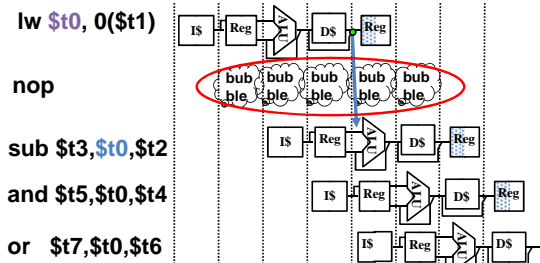
4/12/2013

Spring 2013 -- Lecture #31

27

## Data Hazard: Loads (3/4)

- Stall is equivalent to `nop`



4/12/2013

Spring 2013 -- Lecture #31

28

## Data Hazard: Loads (4/4)

- Slot after a load is called a *load delay slot*
  - If that instruction uses the result of the load, then the hardware interlock will stall it for one cycle
  - Letting the hardware stall the instruction in the delay slot is equivalent to putting a `nop` in the slot (except the latter uses more code space)
- Idea:** Let the compiler put an unrelated instruction in that slot → no stall!

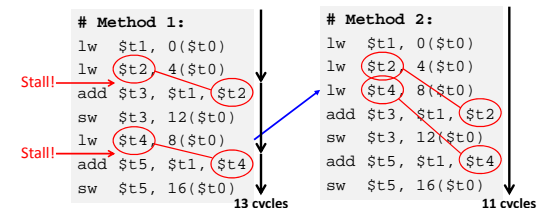
4/12/2013

Spring 2013 -- Lecture #31

29

## Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction!
- MIPS code for  $D=A+B$ ;  $E=A+C$ ;



4/12/2013

Spring 2013 -- Lecture #31

30

## Agenda

- More Pipelining
- Structural Hazards
- Administrivia
- Data Hazards
  - Forwarding
  - Load Delay Slot
- **Control Hazards**

4/12/2013

Spring 2013 -- Lecture #31

31

## 3. Control Hazards

- Branch (`beq`, `bne`) determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can't always fetch correct instruction
    - Still working on ID stage of branch
- **Simple Solution:** Stall on *every* branch until we have the new PC value
  - How long must we stall?

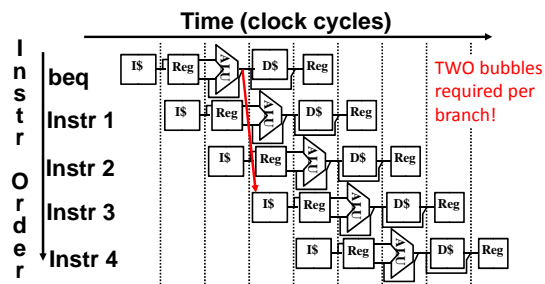
4/12/2013

Spring 2013 -- Lecture #31

32

## Branch Stall

- When is comparison result available?



4/12/2013

Spring 2013 -- Lecture #31

33

## Summary

- Hazards reduce effectiveness of pipelining
  - Cause stalls/bubbles
- Structural Hazards
  - Conflict in use of datapath component
- Data Hazards
  - Need to wait for result of a previous instruction
- Control Hazards
  - Address of next instruction uncertain/unknown
  - *More to come next lecture!*

4/12/2013

Spring 2013 -- Lecture #31

34

**Question:** For each code sequences below, choose one of the statements below:

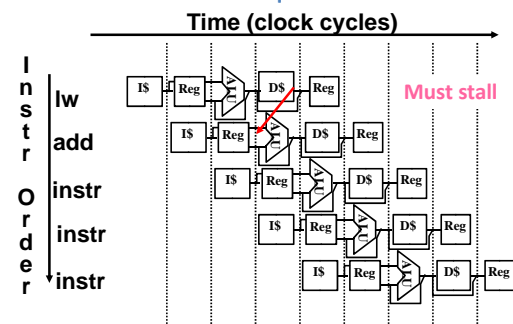
- 1: `lw $t0,0($t0)`  
`add $t1,$t0,$t0`
- 2: `add $t1,$t0,$t0`  
`addi $t2,$t0,5`  
`addi $t4,$t1,5`
- 3: `addi $t1,$t0,1`  
`addi $t2,$t0,2`  
`addi $t3,$t0,2`  
`addi $t3,$t0,4`  
`addi $t5,$t1,5`

- A) No stalls as is
- B) No stalls with forwarding
- C) **Must stall**



35

## Code Sequence 1



7/25/2012

Summer 2012 -- Lecture #22

36

