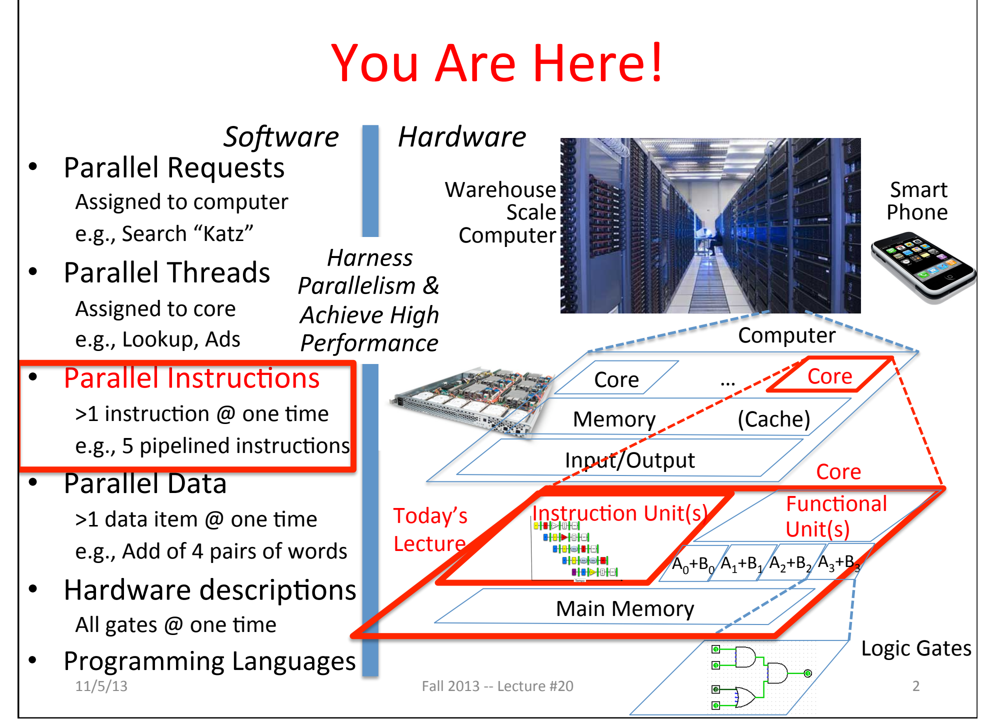
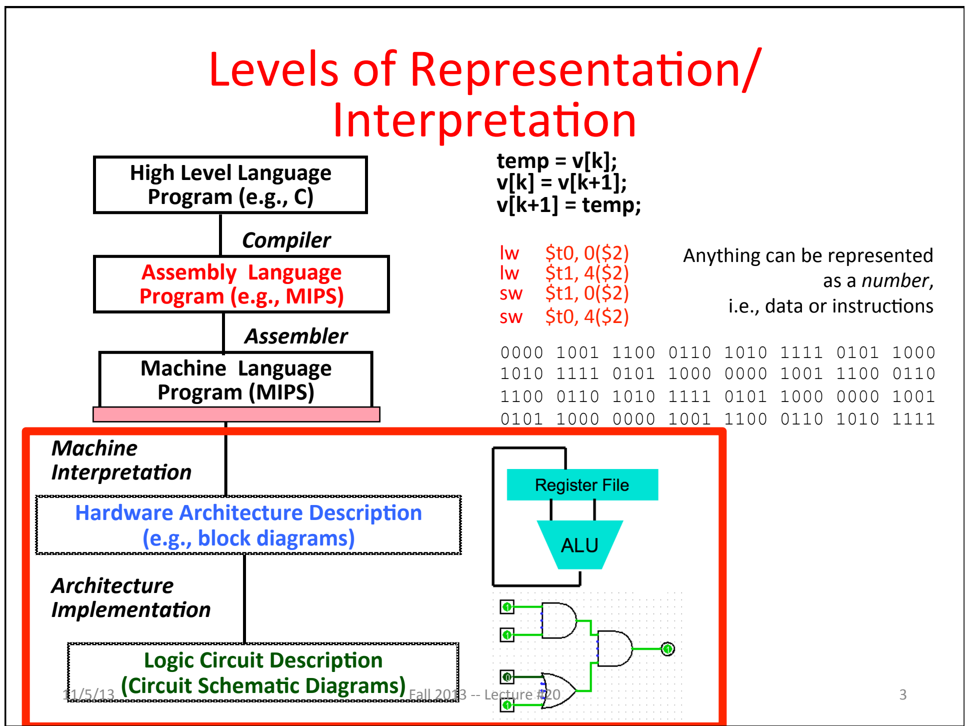
CS 61C:
Great Ideas in Computer Architecture
Control and Pipelining

Instructor:
Randy H. Katz
http://inst.eecs.Berkeley.edu/~cs61c/fa13

You Are Here!





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

11/5/13 Fall 2013 -- Lecture #20 4

Agenda

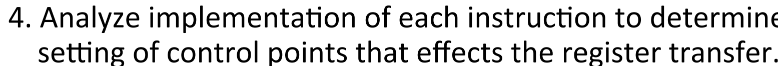
- Pipelined Execution
- Pipelined Datapath
- Structural and Data Hazards
- Control Hazards

Agenda

- Pipelined Execution
- Pipelined Datapath
- Structural and Data Hazards
- Control Hazards

Review: Single-Cycle Processor

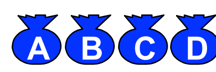
- Five steps to design a processor:
  1. Analyze instruction set → datapath requirements
  2. Select set of datapath components & establish clock methodology
  3. Assemble datapath meeting the requirements: re-examine for pipelining

  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

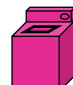
Processor | Control | Datapath | Memory | Input | Output

Pipeline Analogy: Doing Laundry

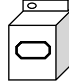
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, fold, and put away



  – Washer takes 30 minutes

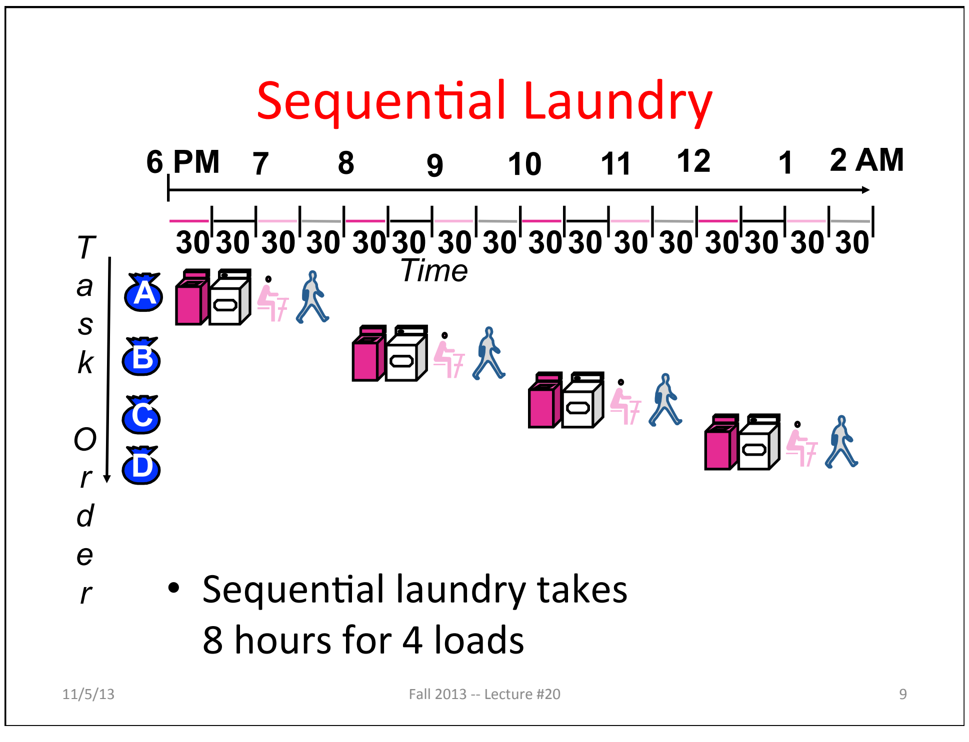


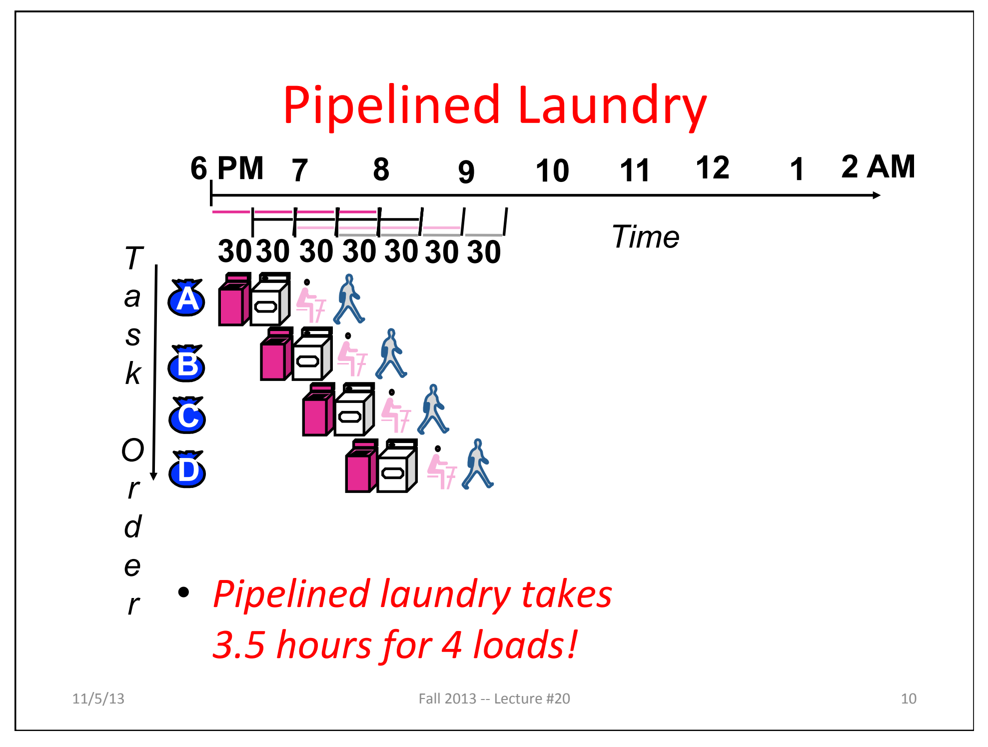
  – Dryer takes 30 minutes



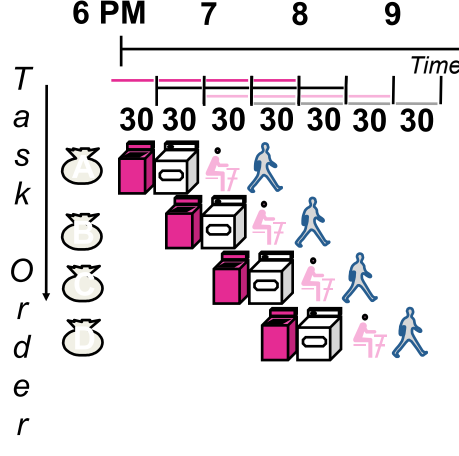
  – "Folder" takes 30 minutes

  – "Stasher" takes 30 minutes to put clothes into drawers


Sequential Laundry
6 PM
7
8
9
10
11
12
1
2 AM
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
Time
Task Order
A
B
C
D
• Sequential laundry takes 8 hours for 4 loads
11/5/13
Fall 2013 -- Lecture #20
9


Pipelined Laundry
6 PM
7
8
9
10
11
12
1
2 AM
30 30 30 30 30 30 30
Time
Task Order
A
B
C
D
• Pipelined laundry takes 3.5 hours for 4 loads!
11/5/13
Fall 2013 -- Lecture #20
10

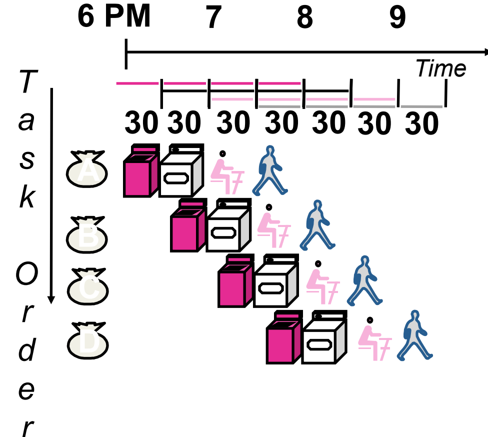
Pipelining Lessons (1/2)



- 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

11/5/13 Fall 2013 -- Lecture #20 11

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

11/5/13 Fall 2013 -- Lecture #20 12

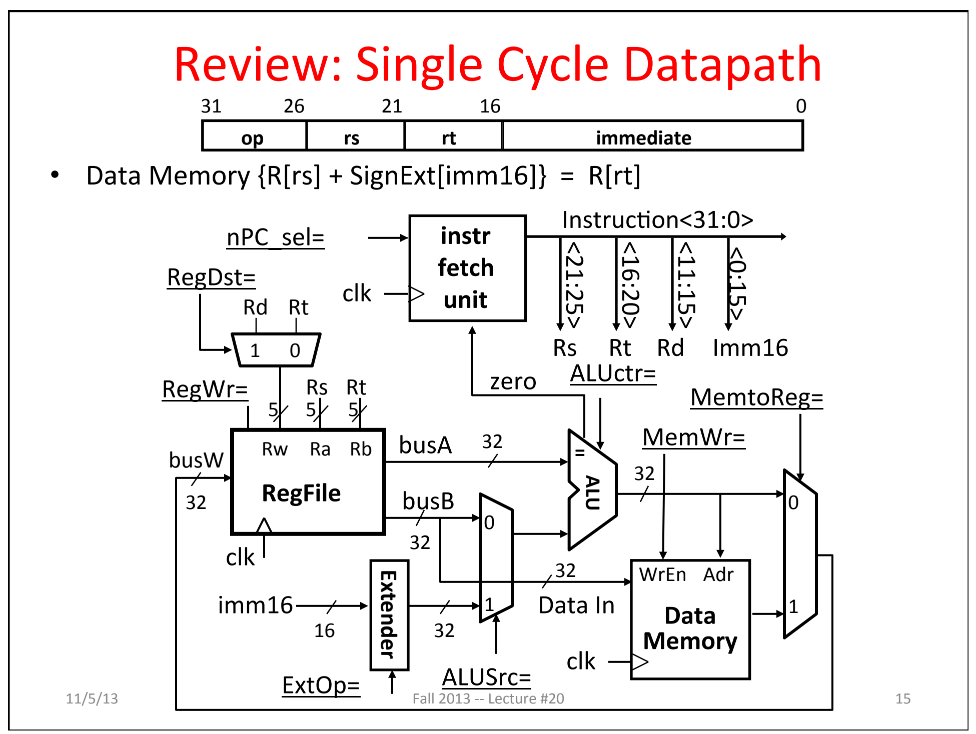
Agenda

- Pipelined Execution
- Pipelined Datapath
- Structural and Data Hazards
- Control Hazards

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



Steps in Executing MIPS

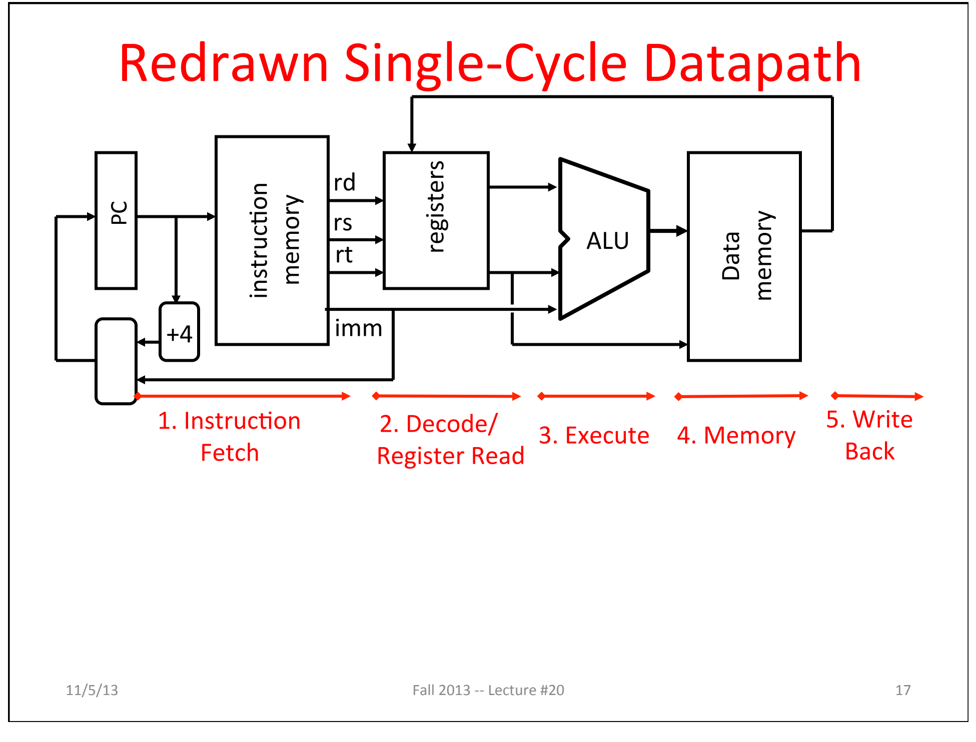
1) IF: Instruction Fetch, Increment PC

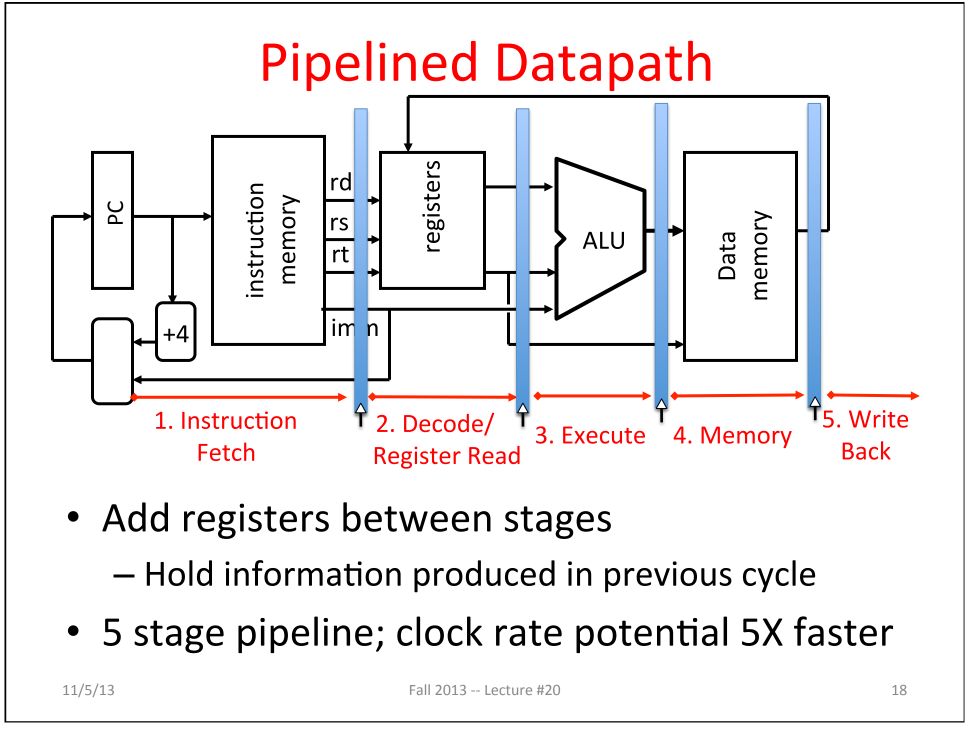
2) ID: Instruction Decode, Read Registers

3) EX: Execution
   Mem-ref: Calculate Address
   Arith-log: Perform Operation

4) Mem:
   Load: Read Data from Memory
   Store: Write Data to Memory

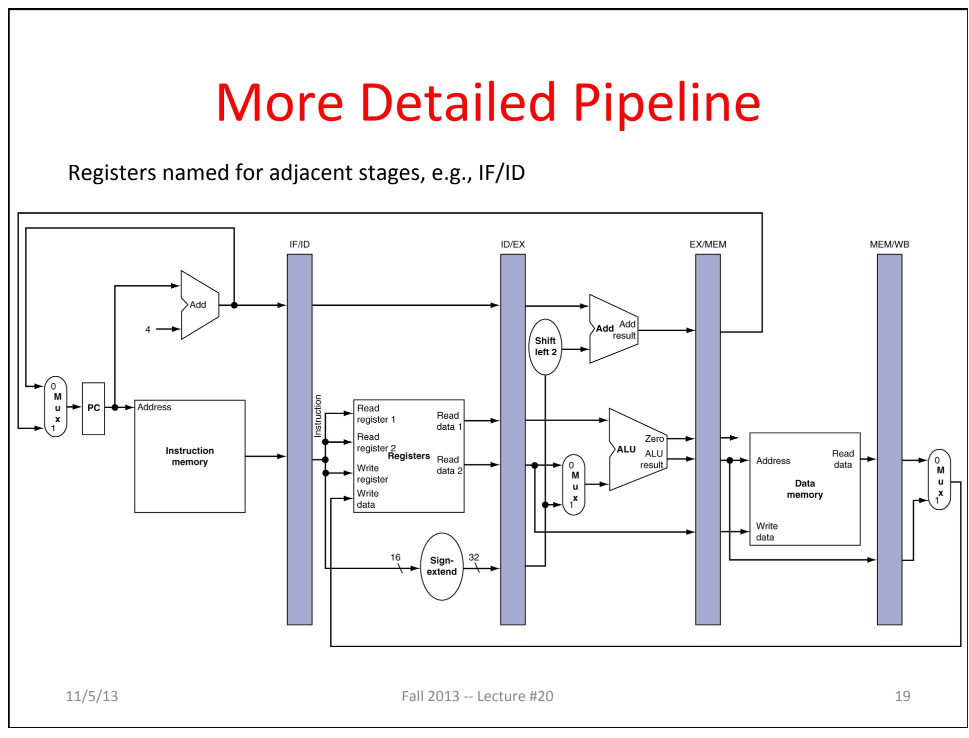
5) WB: Write Data Back to Register


Redrawn Single-Cycle Datapath
PC
+4
instruction memory
rd
rs
rt
imm
registers
ALU
Data memory
1. Instruction Fetch
2. Decode/ Register Read
3. Execute
4. Memory
5. Write Back
11/5/13
Fall 2013 -- Lecture #20
17


Pipelined Datapath
PC
+4
instruction memory
rd
rs
rt
im n
registers
ALU
Data memory
1. Instruction Fetch
2. Decode/ Register Read
3. Execute
4. Memory
5. Write Back
• Add registers between stages
– Hold information produced in previous cycle
• 5 stage pipeline; clock rate potential 5X faster
11/5/13
Fall 2013 -- Lecture #20
18

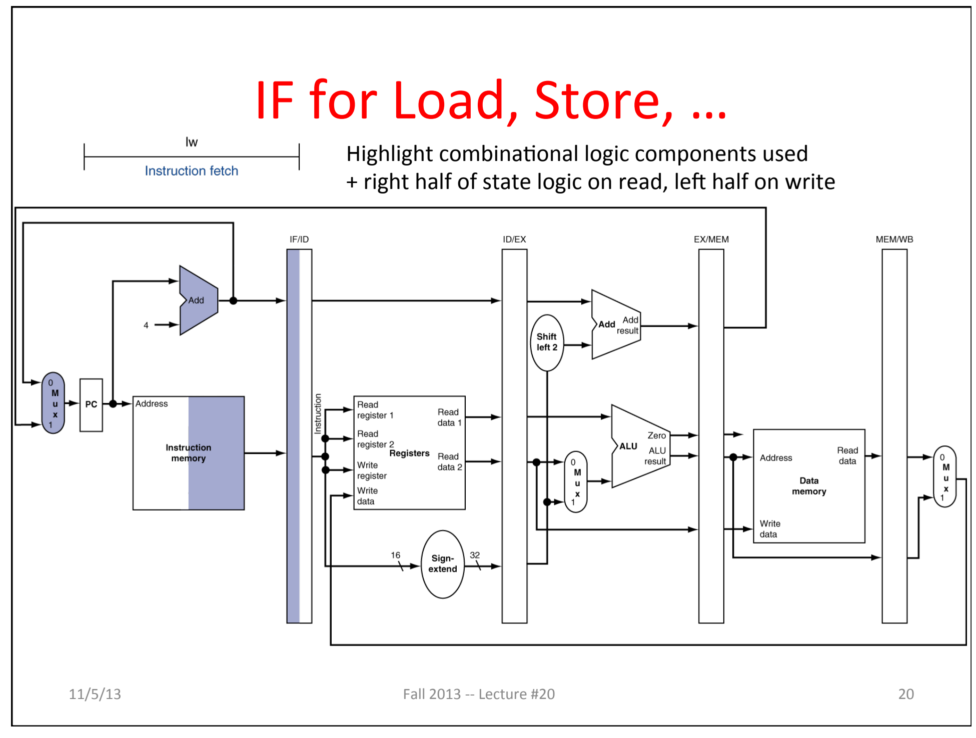
More Detailed Pipeline

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

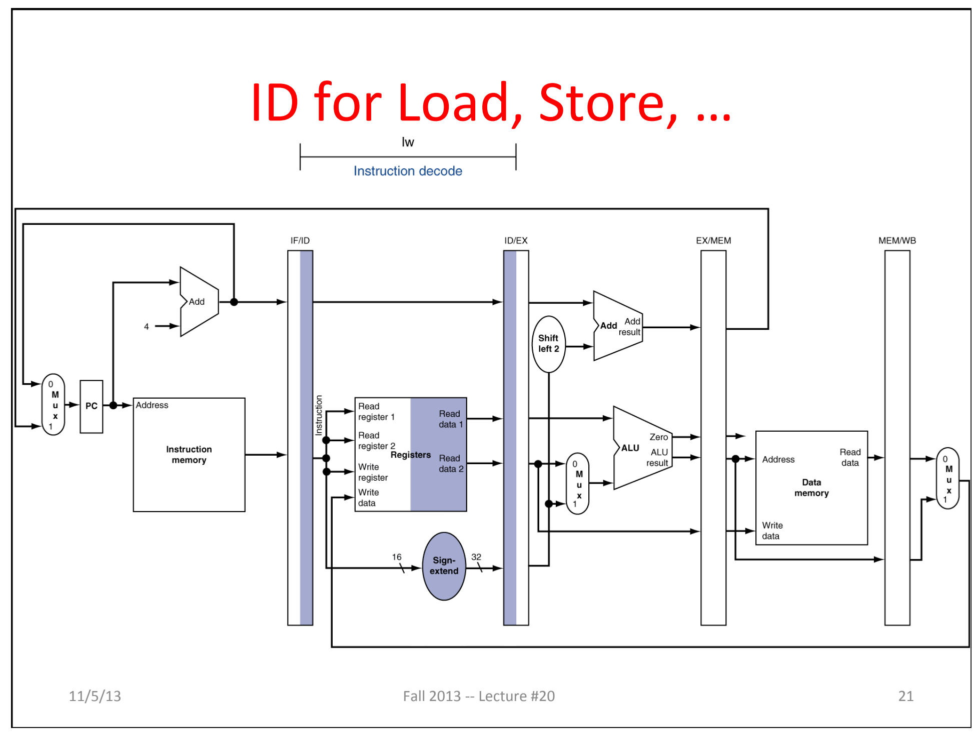


IF for Load, Store, …

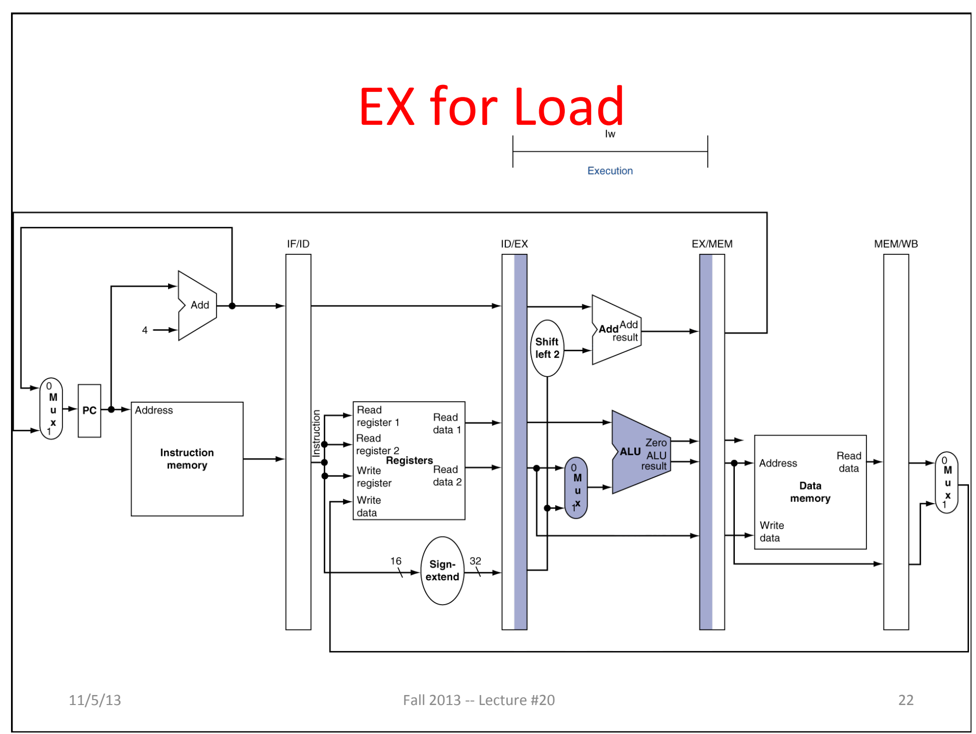
Highlight combinational logic components used
+ right half of state logic on read, left half on write

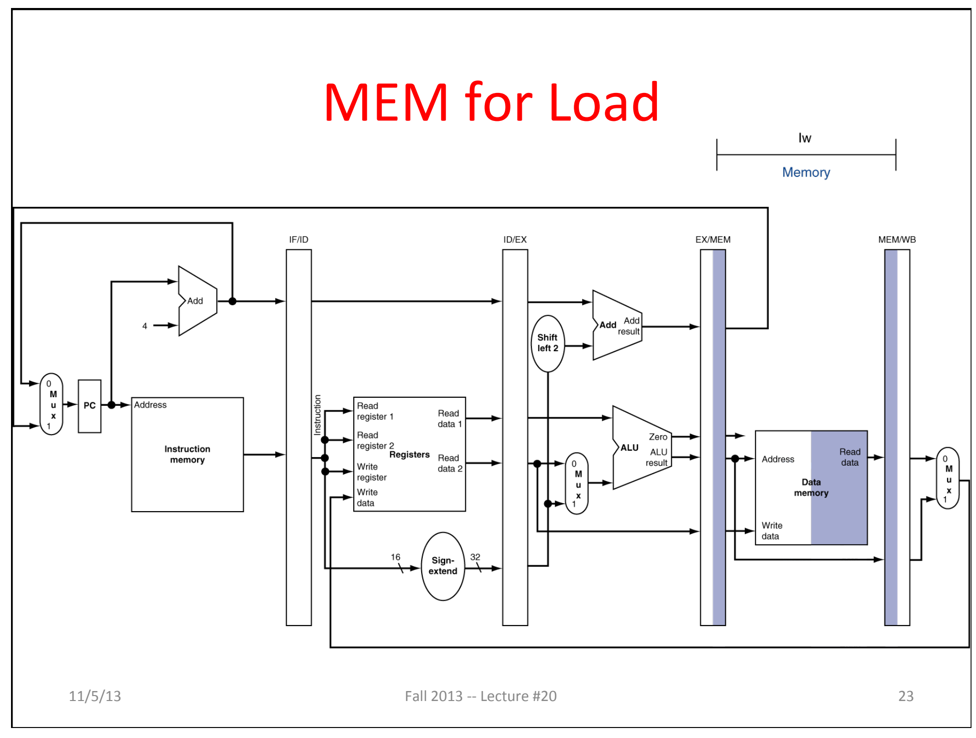


ID for Load, Store, …

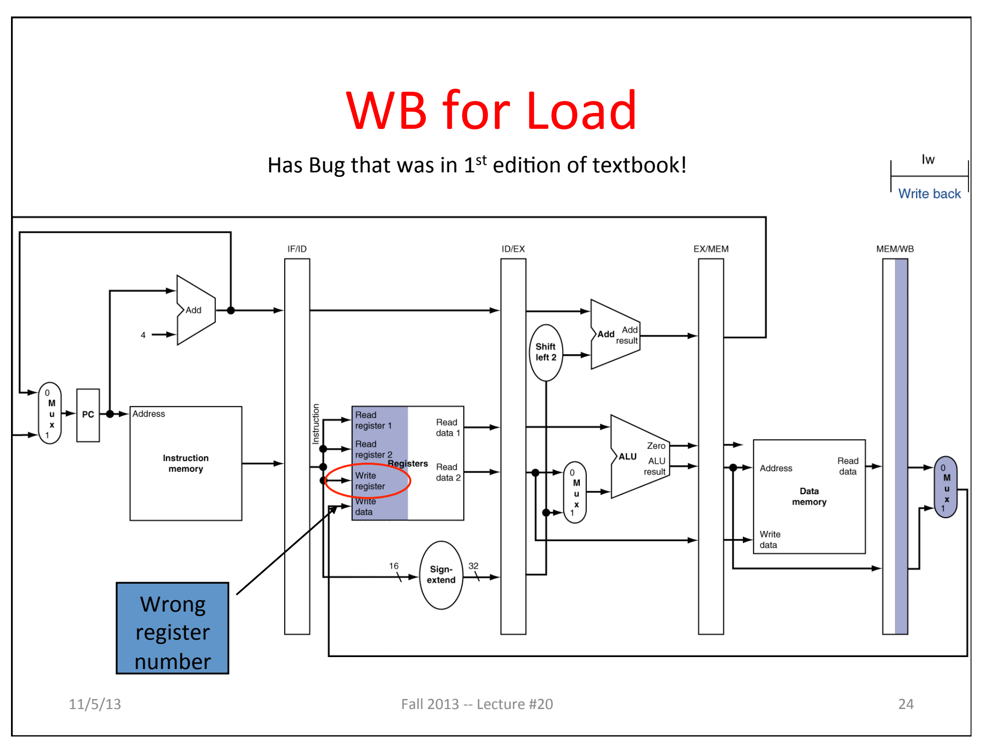


EX for Load

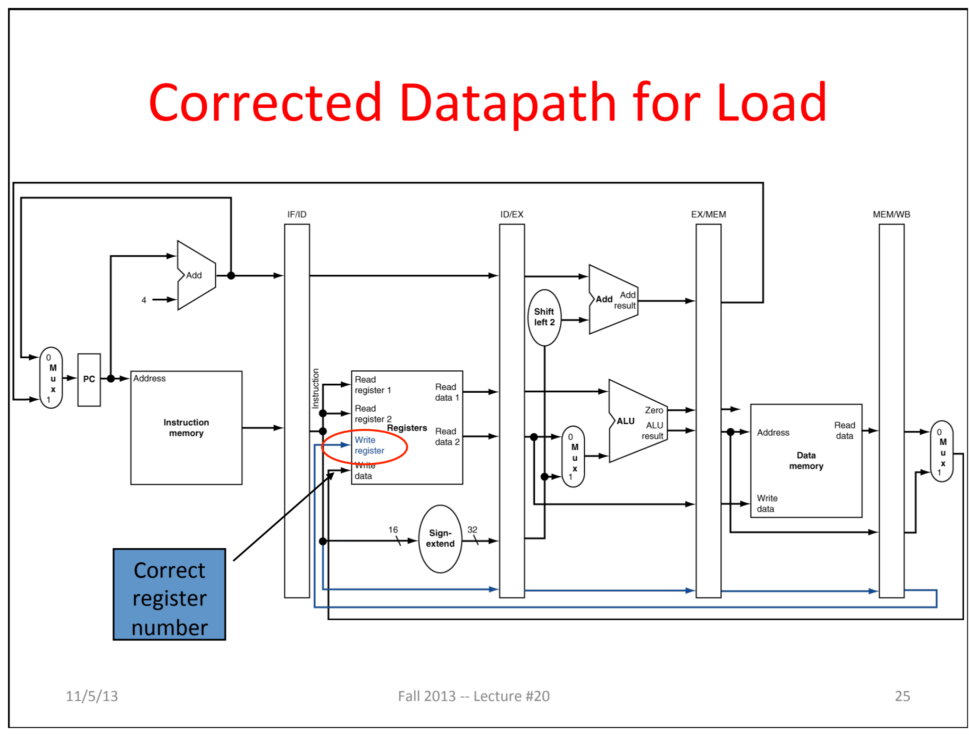


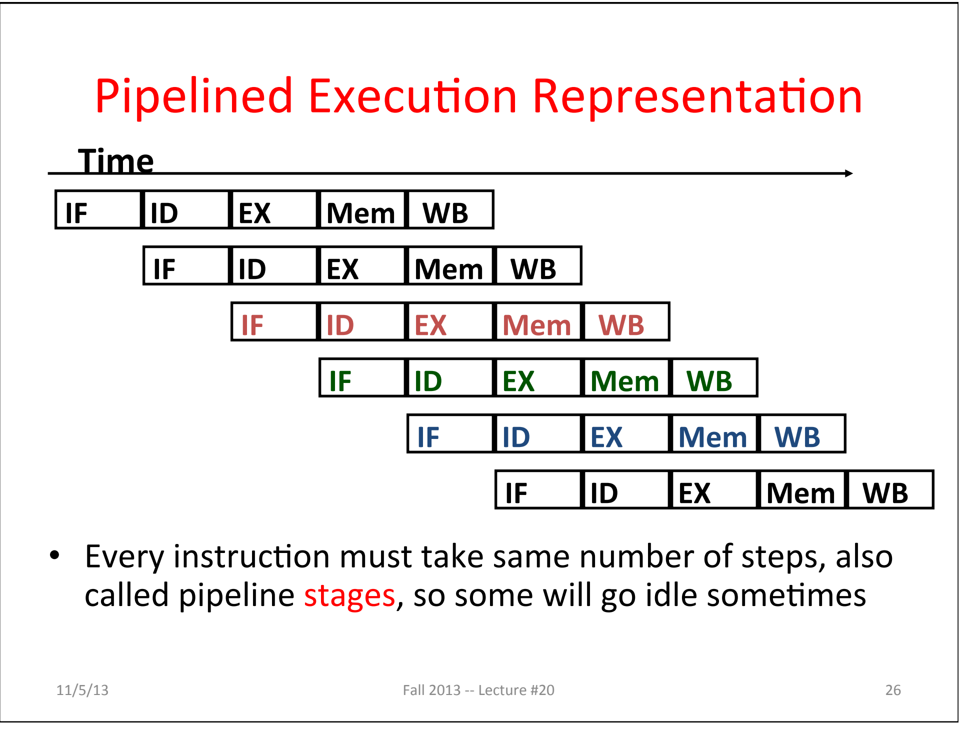


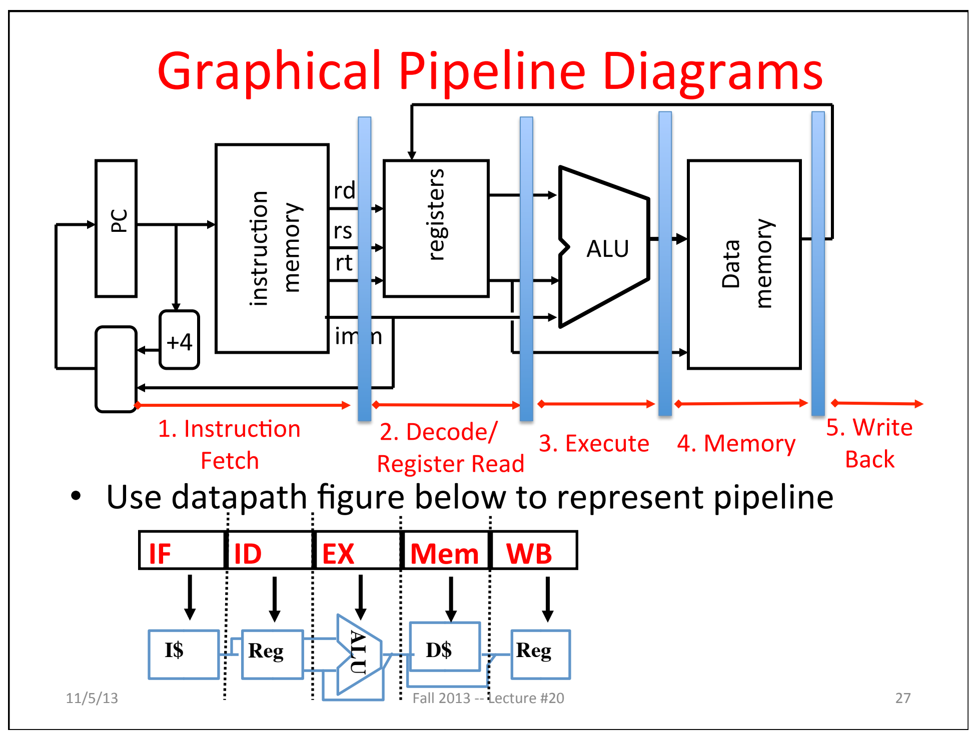
MEM for Load
lw
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
11/5/13
Fall 2013 -- Lecture #20
23

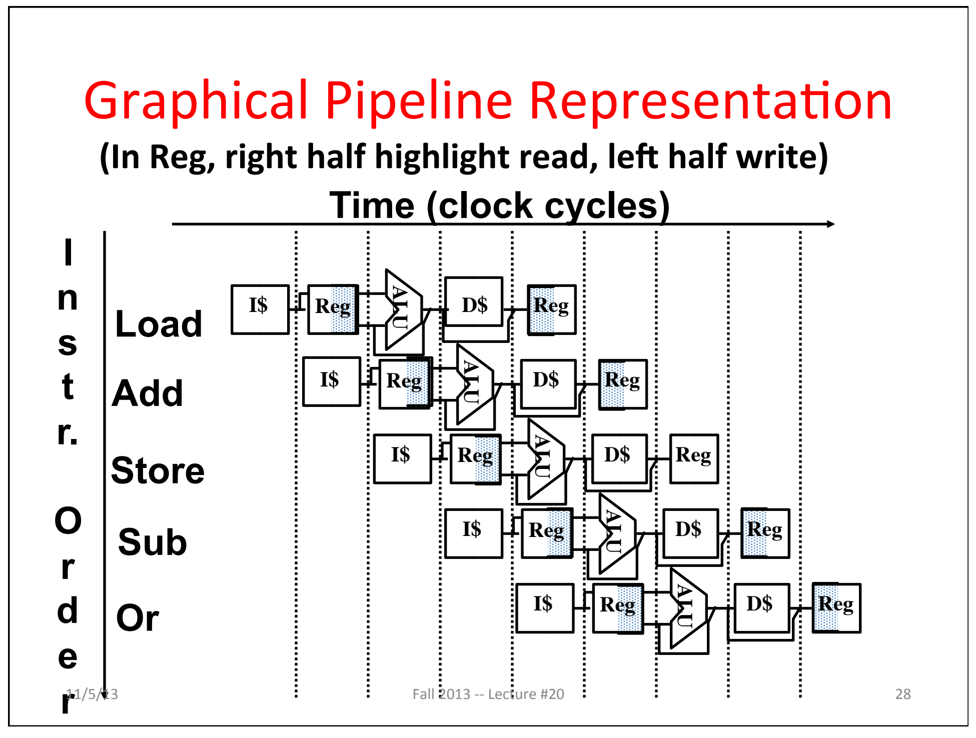


WB for Load
Has Bug that was in 1st edition of textbook!
lw
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
Wrong register number
11/5/13
Fall 2013 -- Lecture #20
24


Corrected Datapath for Load
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign-extend
32
Correct register number
11/5/13
Fall 2013 -- Lecture #20
25


Pipelined Execution Representation
Time
IF ID EX Mem WB
IF ID EX Mem WB
IF ID EX Mem WB
IF ID EX Mem WB
IF ID EX Mem WB
IF ID EX Mem WB
• Every instruction must take same number of steps, also called pipeline stages, so some will go idle sometimes
11/5/13
Fall 2013 -- Lecture #20
26


Graphical Pipeline Diagrams
PC
+4
instruction memory
rd
rs
rt
imm
registers
ALU
Data memory
1. Instruction Fetch
2. Decode/ Register Read
3. Execute
4. Memory
5. Write Back
• Use datapath figure below to represent pipeline
IF
ID
EX
Mem
WB
I$
Reg
ALU
D$
Reg
11/5/13
Fall 2013 -- Lecture #20
27


Graphical Pipeline Representation
(In Reg, right half highlight read, left half write)
Time (clock cycles)
Instr. Order
Load
Add
Store
Sub
Or
I$
Reg
ALU
D$
Reg
11/5/13
Fall 2013 -- Lecture #20
28

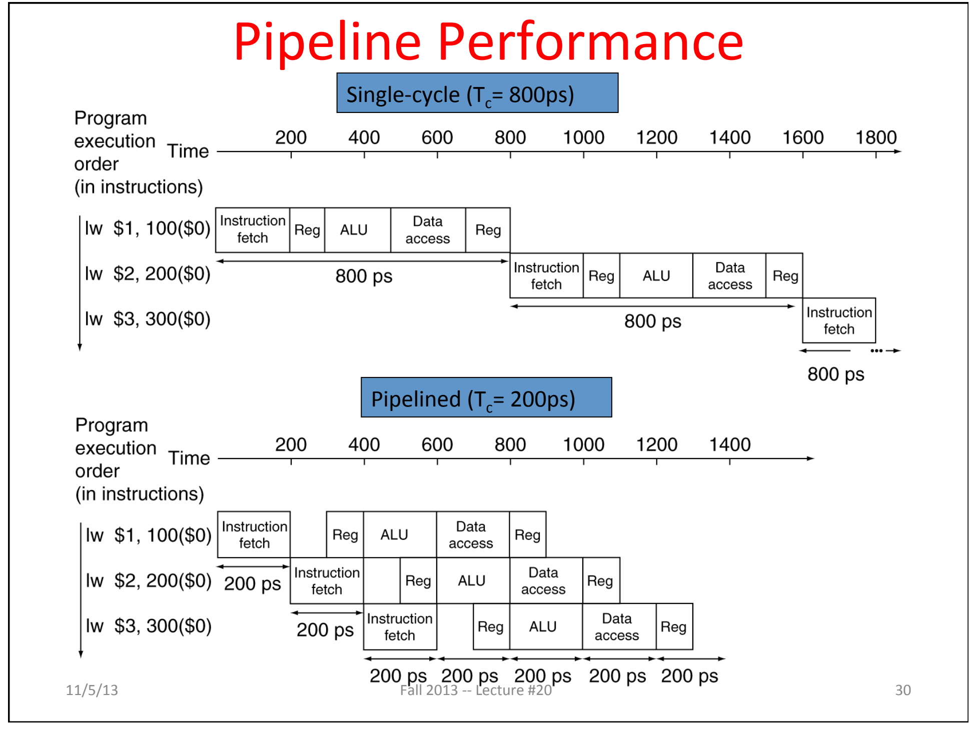
Pipeline Performance

- 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$_{pipelined}$ = Time between instructions$_{nonpipelined}$ / Number of stages

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

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

Agenda

- Pipelined Execution
- Pipelined Datapath
- Structural and Data Hazards
- Control Hazards

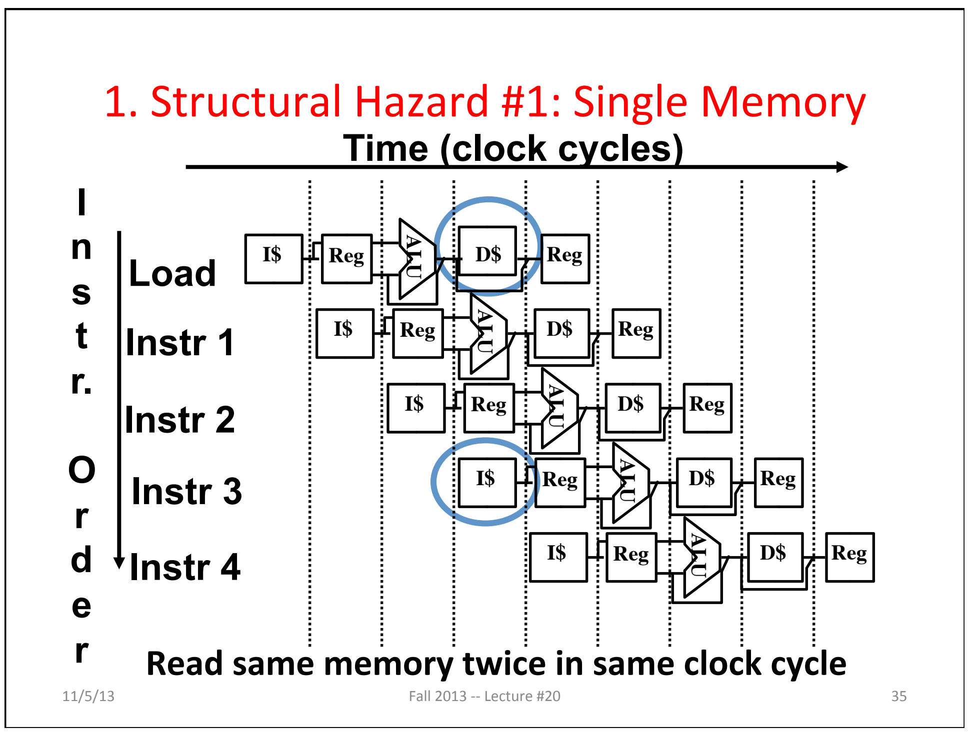
Hazards

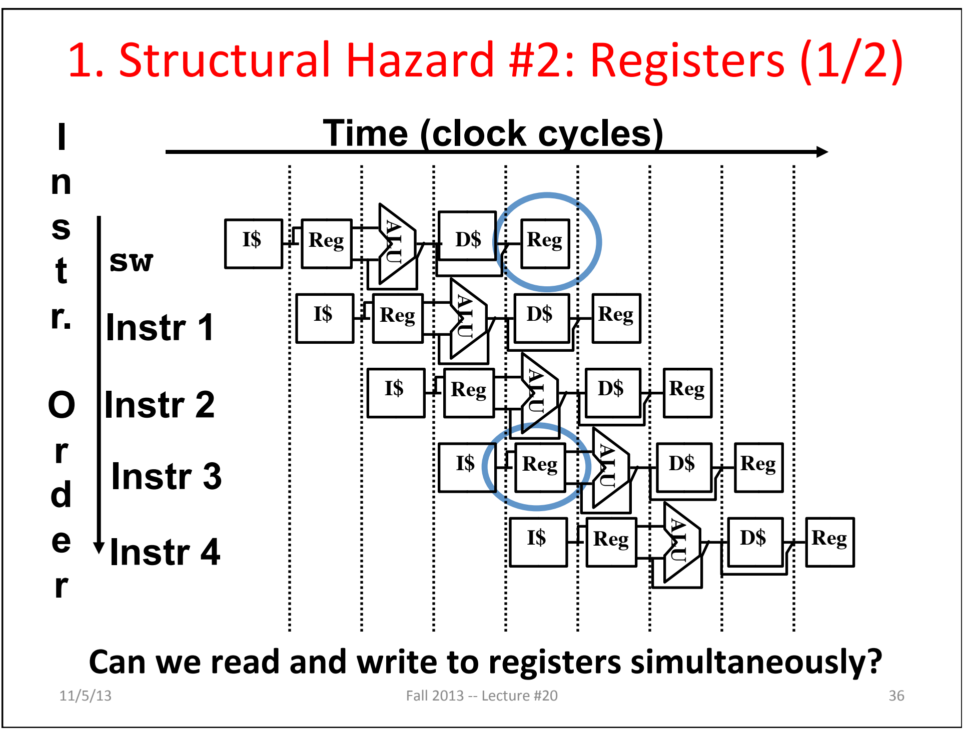
Situations that prevent starting the next logical instruction in the next clock cycle

1. Structural hazards
   – Required resource is busy (e.g., stasher is studying)
2. Data hazard
   – Need to wait for previous instruction to complete its data read/write (e.g., pair of socks in different loads)
3. Control hazard
   – Deciding on control action depends on previous instruction (e.g., how much detergent based on how clean prior load turns out)

1. Structural Hazards

- Conflict for use of a resource
- In 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
  – In reality, provide separate L1 instruction cache and L1 data cache


1. Structural Hazard #1: Single Memory
Time (clock cycles)
Instr. Order
Load
Instr 1
Instr 2
Instr 3
Instr 4
I$
Reg
ALU
D$
Reg
Read same memory twice in same clock cycle
11/5/13
Fall 2013 -- Lecture #20
35


1. Structural Hazard #2: Registers (1/2)
Time (clock cycles)
Instr. Order
sw
Instr 1
Instr 2
Instr 3
Instr 4
I$
Reg
ALU
D$
Reg
Can we read and write to registers simultaneously?
11/5/13
Fall 2013 -- Lecture #20
36

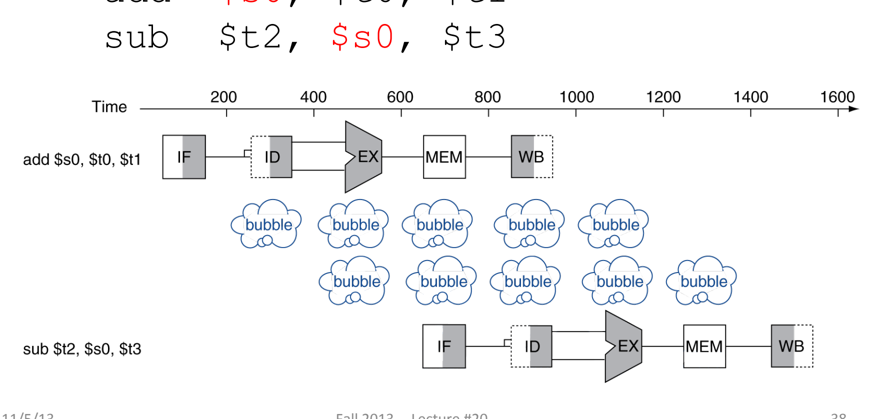
1. Structural Hazard #2: Registers (2/2)

- Two different solutions have been used:
  1) RegFile access is *VERY* fast: takes less than half the time of ALU stage
     - Write to Registers during first half of each clock cycle
     - Read from Registers during second half of each clock cycle
  2) Build RegFile with independent read and write ports
- Result: can perform Read and Write during same clock cycle

2. Data Hazards

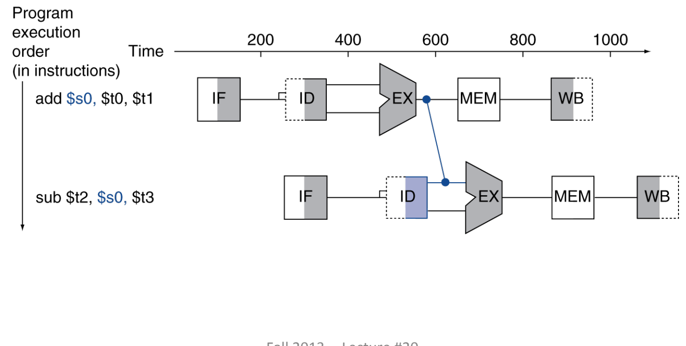
- An instruction depends on completion of data access by a previous instruction

```
add  $s0, $t0, $t1
sub  $t2, $s0, $t3
```

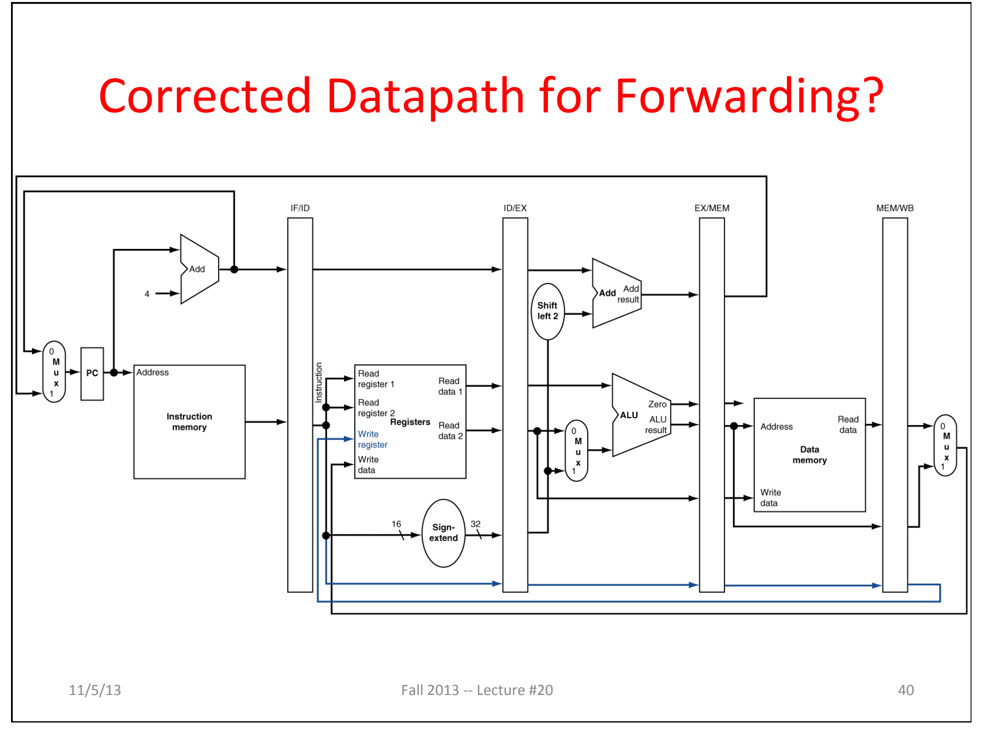


Forwarding (aka Bypassing)

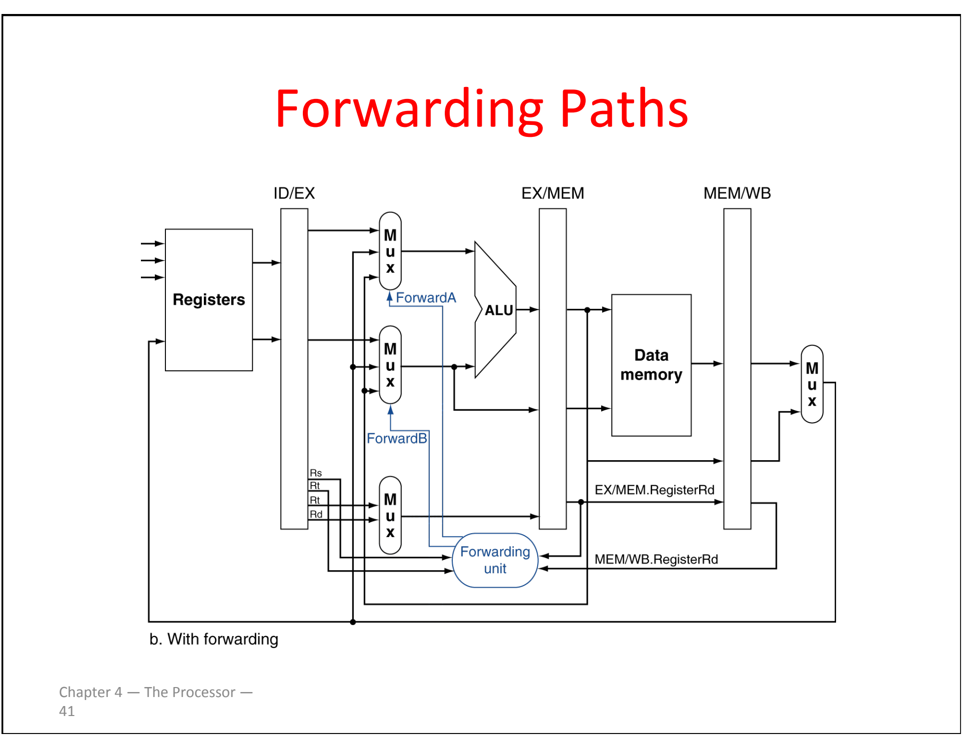
- Use result when it is computed
  - Don't wait for it to be stored in a register
  - Requires extra connections in the datapath



Corrected Datapath for Forwarding?



Forwarding Paths

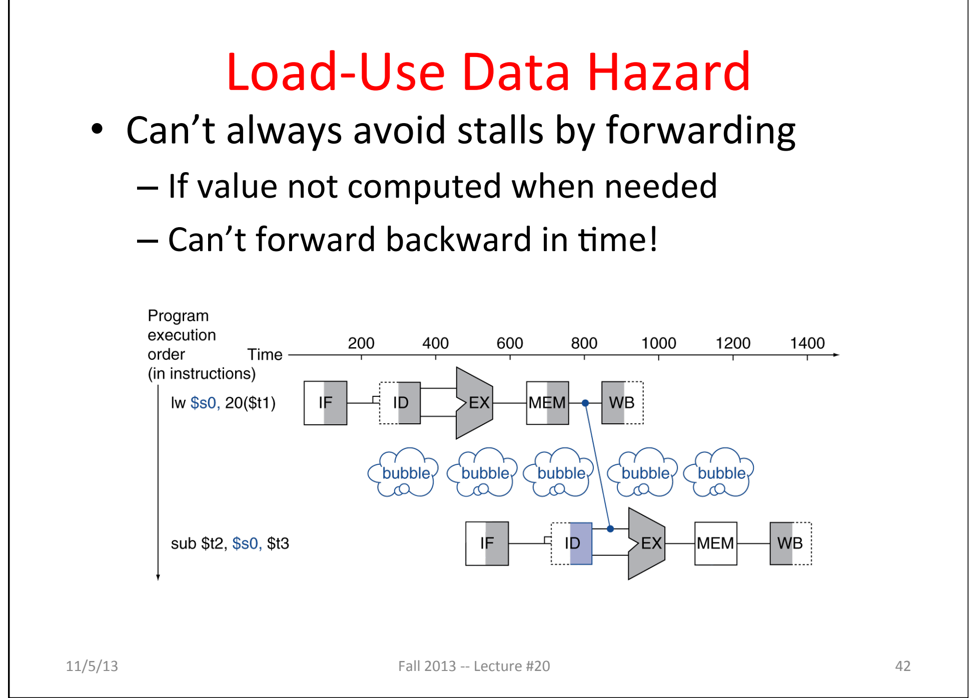


b. With forwarding

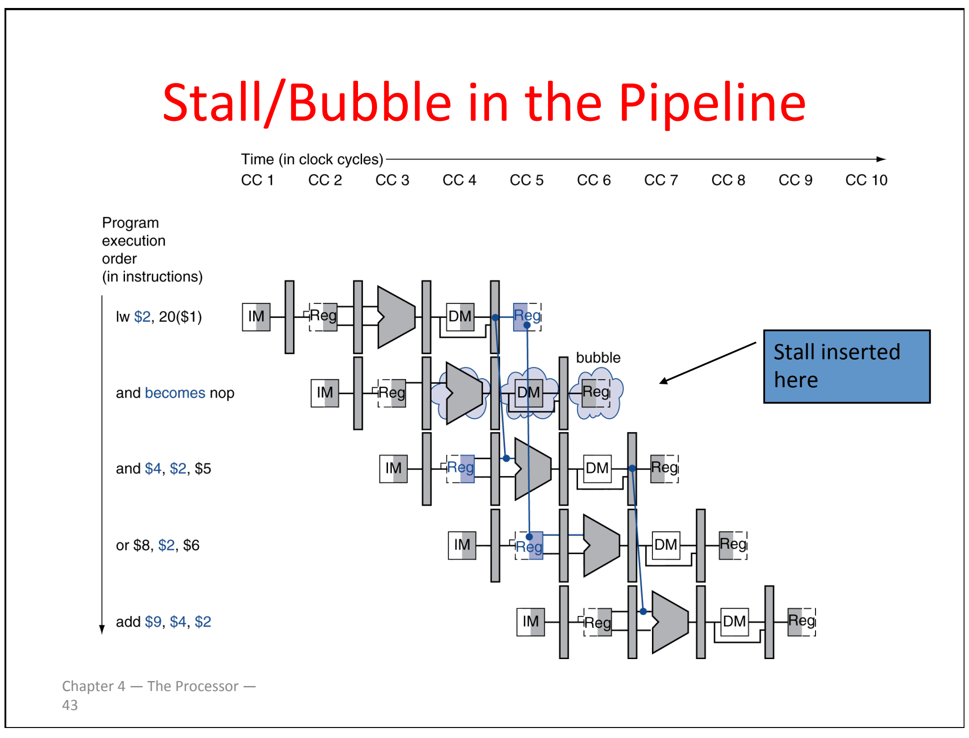
Chapter 4 — The Processor — 41

Load-Use Data Hazard

- Can't always avoid stalls by forwarding
  - If value not computed when needed
  - Can't forward backward in time!



Stall/Bubble in the Pipeline



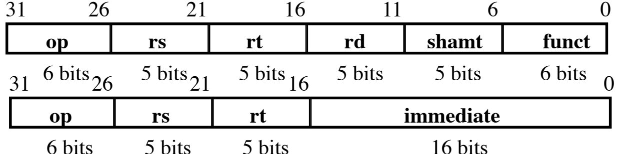
Chapter 4 — The Processor — 43

Pipelining and ISA Design

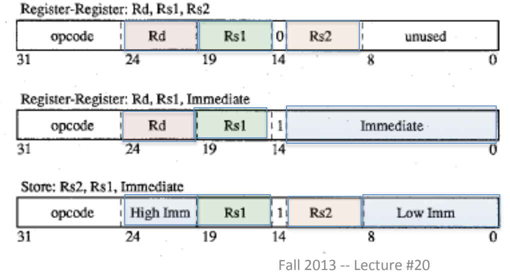
- MIPS Instruction Set designed for pipelining
- All instructions are 32-bits
  - Easier to fetch and decode in one cycle
  - x86: 1- to 17-byte instructions

  (x86 HW actually translates to internal RISC instructions!)
- Few and regular instruction formats, 2 source register fields always in same place
  - Can decode and read registers in one step
- Memory operands only in Loads and Stores
  - Can calculate address 3rd stage, access memory 4th stage
- Alignment of memory operands
  - Memory access takes only one cycle

Why Isn't the Destination Register Always in the Same Field in MIPS ISA?



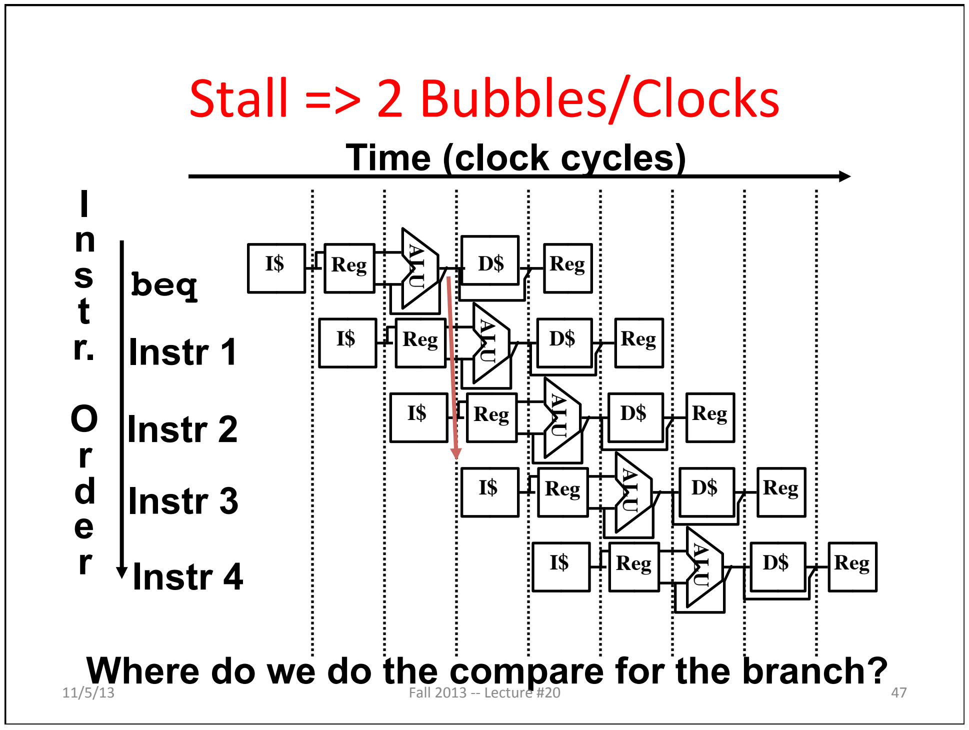
- Need to have 2 part immediate if 2 sources and 1 destination always in same place



SPUR processor (A project Dave Patterson and Randy worked on together)

3. Control Hazards

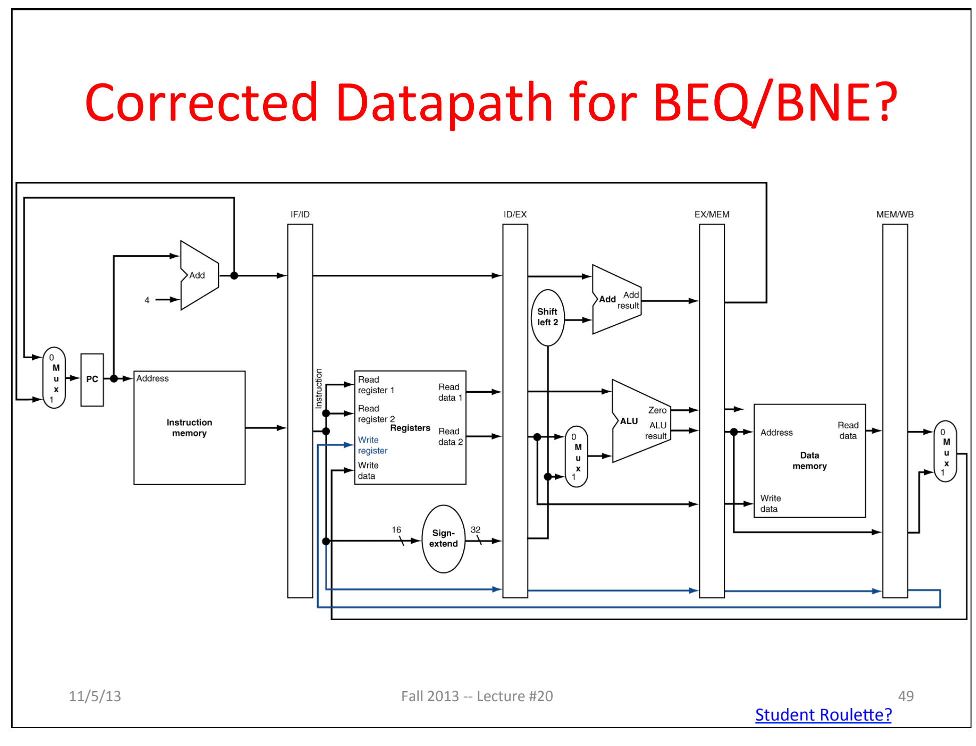
- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can't always fetch correct instruction
    - Still working on ID stage of branch
- BEQ, BNE in MIPS pipeline
- Simple solution Option 1: *Stall* on every branch until have new PC value
  - Would add 2 bubbles/clock cycles for every Branch! (~ 20% of instructions executed)

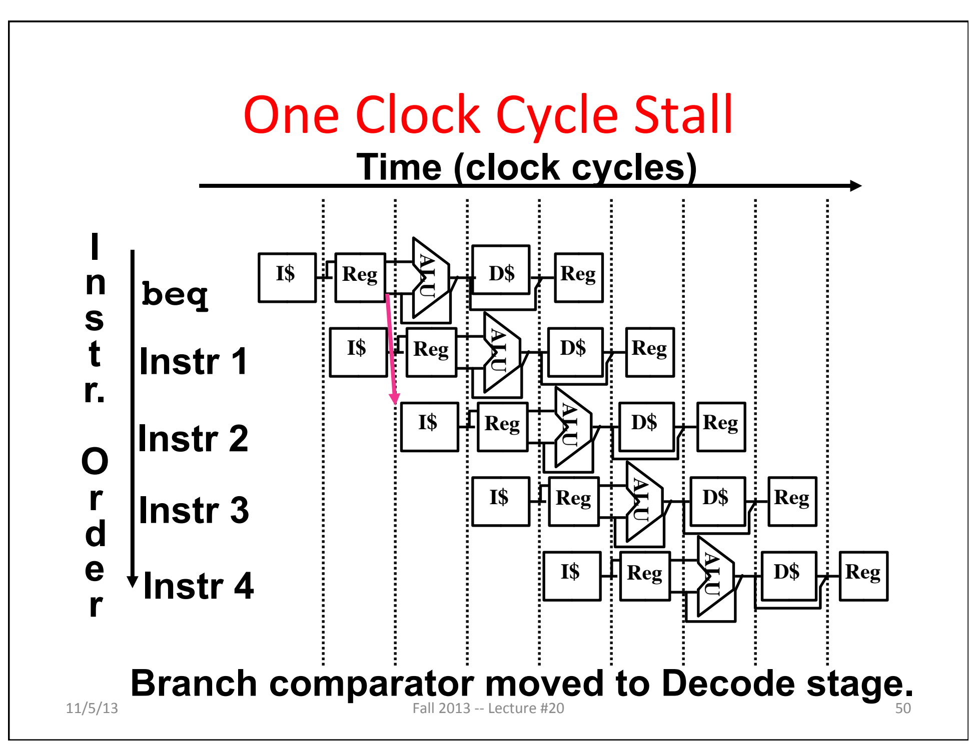


3. Control Hazard: Branching

- Optimization #1:
  - Insert special branch comparator in Stage 2
  - As soon as instruction is decoded (Opcode identifies it as a branch), immediately make a decision and set the new value of the PC
  - Benefit: since branch is complete in Stage 2, only one unnecessary instruction is fetched, so only one no-op is needed
  - Side Note: means that branches are idle in Stages 3, 4 and 5

11/5/13 Fall 2013 -- Lecture #20 48


Corrected Datapath for BEQ/BNE?
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
Add
Add result
Shift left 2
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
11/5/13
Fall 2013 -- Lecture #20
49
Student Roulette?


One Clock Cycle Stall
Time (clock cycles)
Instr. Order
beq
Instr 1
Instr 2
Instr 3
Instr 4
I$
Reg
ALU
D$
Reg
Branch comparator moved to Decode stage.
11/5/13
Fall 2013 -- Lecture #20
50

Agenda

- Pipelined Execution
- Pipelined Datapath
- Structural and Data Hazards
- Control Hazards

3. Control Hazards

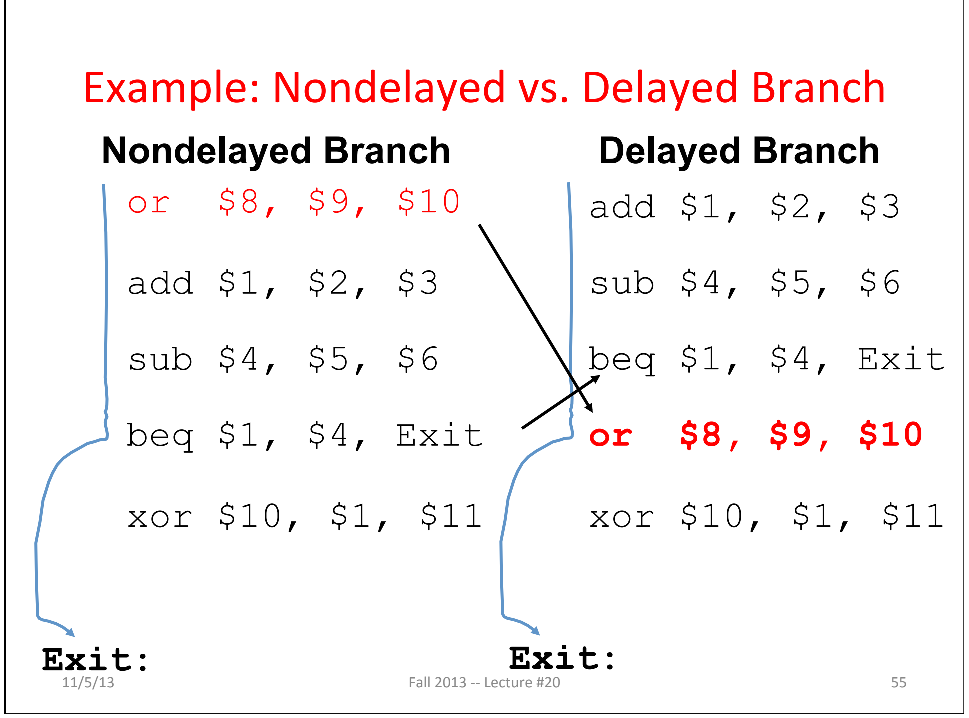
- Option 2: *Predict* outcome of a branch, fix up if guess wrong
  – Must cancel all instructions in pipeline that depended on guess that was wrong
- Simplest hardware if we predict that all branches are NOT taken
  – Why?

3. Control Hazard: Branching

- Option #3: Redefine branches
  - Old definition: if we take the branch, none of the instructions after the branch get executed by accident
  - New definition: whether or not we take the branch, the single instruction immediately following the branch gets executed (the *branch-delay slot*)
- *Delayed Branch* means *we always execute inst after branch*
- This optimization is used with MIPS

3. Control Hazard: Branching

- Notes on Branch-Delay Slot
  - Worst-Case Scenario: put a no-op in the branch-delay slot
  - Better Case: place some instruction preceding the branch in the branch-delay slot—as long as the changed doesn't affect the logic of program
    - Re-ordering instructions is common way to speed up programs
    - Compiler usually finds such an instruction 50% of time
    - Jumps also have a delay slot …

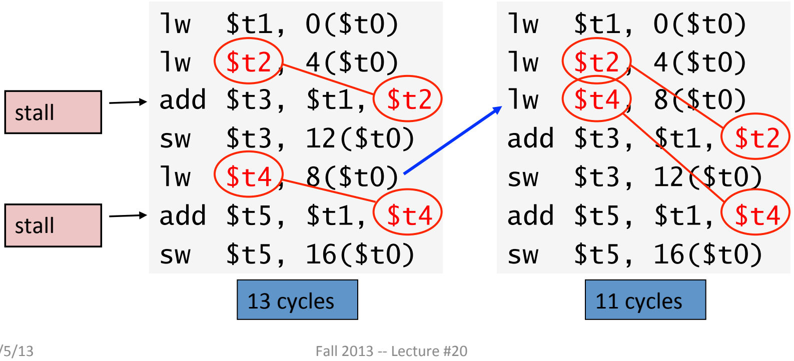


Delayed Branch/Jump and MIPS ISA?

- Why does JAL put PC+8 in register 31?

Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction
- C code for `A = B + E; C = B + F;`



Peer Instruction

I. **Thanks to pipelining, I have reduced the time it took me to wash my one shirt.**

II. **Longer pipelines are always a win (since less work per stage & a faster clock).**

A)(orange) I is True and II is True

B)(green) I is False and II is True

C)(pink) I is True and II is False

D)(yellow) I is False and II is False

And, in Conclusion, …

- 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
- Subject to hazards
  - Structure, data, control
- Stalls reduce performance
  - But are required to get correct results
- Compiler can arrange code to avoid hazards and stalls
  - Requires knowledge of the pipeline structure