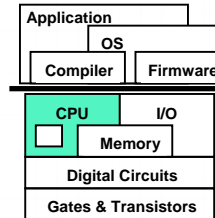


ECE 152
Introduction to Computer Architecture
Pipelining
Copyright 2008 Daniel J. Sorin
Duke University

Slides are derived from work by
Amir Roth (U. Penn)
Spring 2008

1

This Unit: Pipelining



- Basic Pipelining
 - Pipeline control
- Data Hazards
 - Software interlocks and scheduling
 - Hardware interlocks and stalling
 - Bypassing
- Control Hazards
 - Fast and delayed branches
 - Branch prediction
- Multi-cycle operations
- Exceptions

© 2008 Daniel J. Sorin from Roth

ECE 152

2

Readings

- P+H
 - Chapter 6

© 2008 Daniel J. Sorin from Roth

ECE 152

3

Quick Review

- Datapath and control
 - **Fetch**: get insn at current PC from IMem
 - **Decode**: translate insn into control signals for datapath
 - **Execute**: control signals implement insn on datapath
- Two implementations: differentiated by control
 - **Single-cycle**: hardwired control
 - **Multi-cycle**: hardwired or micro-programmed control

© 2008 Daniel J. Sorin from Roth

ECE 152

4

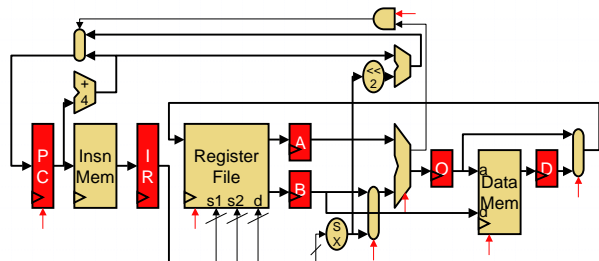
What About Performance?

- Single-cycle
 - + Low cycles per instruction (CPI), i.e., $CPI=1$
 - Long clock period (to accommodate slowest insn)
- Multi-cycle
 - + Short clock period
 - High CPI
- Can we have both low CPI and short clock period?
 - Not if datapath executes only one insn at a time
 - No good way to make a single insn go faster
 - Each insn has to go through certain number of gates, and that's it
 - Can't reduce the amount of "work" per insn

Pipelining

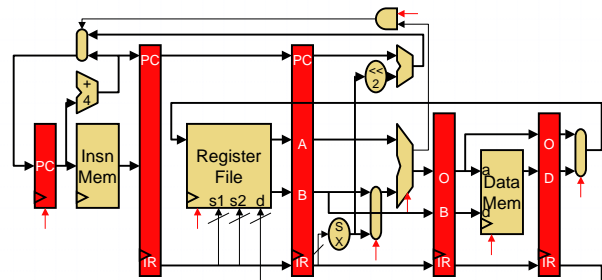
- Important performance technique
 - **Improves insn throughput (rather than insn latency)**
- Begin with multi-cycle design
 - When insn advances from stage 1 to 2
 - Allow next insn to enter stage 1
 - Etc.
- Individual insn takes the same number of stages
 - + **But insns enter and leave at a much faster rate**
- Laundry / Subway analogy

5 Stage Multi-Cycle Datapath



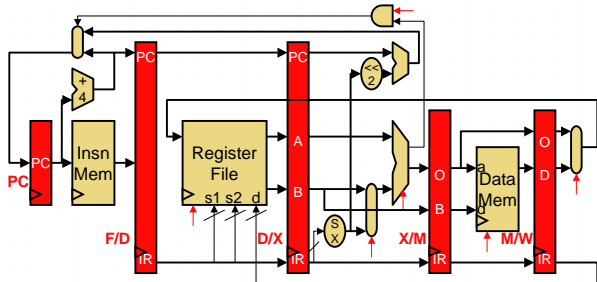
- Ignoring the following for purposes of this explanation
 - Jump (j) insn
 - Destination register mux (R-type vs. I-type)

5 Stage Pipelined Datapath



- Temporary values (PC, IR, A, B, O, D) re-latched every stage
 - Why? 5 insns may be in pipeline at once, they share a single PC?
 - Notice, PC not re-latched after ALU stage (why not?)

Pipeline Terminology



- Five stage: **F**etch, **D**ecode, **eX**ecute, **M**emory, **W**riteback
 - Latches (pipeline registers) named by stages they separate
 - **PC, F/D, D/X, X/M, M/W**

© 2008 Daniel J. Sorin from Roth

ECE 152

9

Aside: Not All Pipelines Have 5 Stages

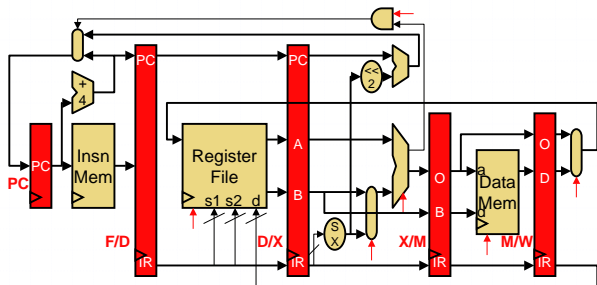
- H&P textbook uses well-known 5-stage pipe != all pipes have 5 stages
- Some examples
 - OpenRISC 1200: 4 stages
 - Sun UltraSPARC T1/T2 (Niagara/Niagara2): 6/8 stages
 - AMD Athlon: 10 stages
 - Pentium 4: 20 stages
- **ICQ**: why does Pentium 4 have so many stages?
- **ICQ**: how can you possibly break "work" to do single insn into that many stages?
- Moral of the story: in ECE 152, we focus on H&P 5-stage pipe, but don't forget that this is just one example

© 2008 Daniel J. Sorin from Roth

ECE 152

10

Pipeline Example: Cycle 1



add \$3,\$2,\$1

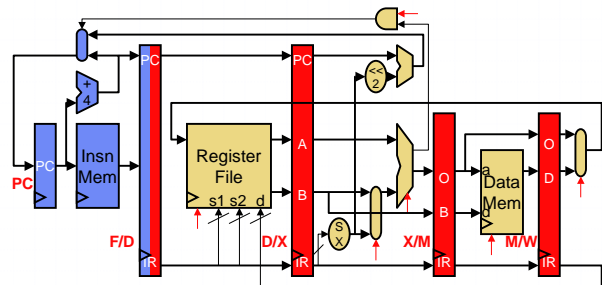
- 3 instructions

© 2008 Daniel J. Sorin from Roth

ECE 152

11

Pipeline Example: Cycle 2



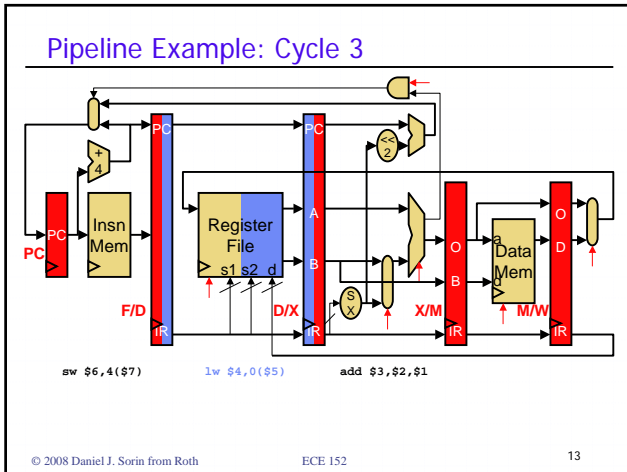
lw \$4,0(\$5) add \$3,\$2,\$1

© 2008 Daniel J. Sorin from Roth

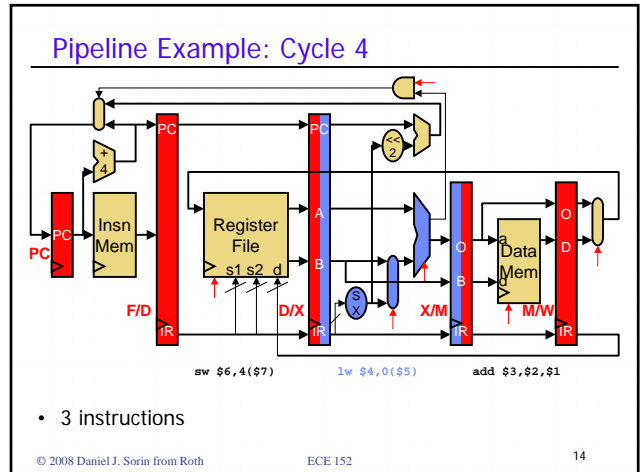
ECE 152

12

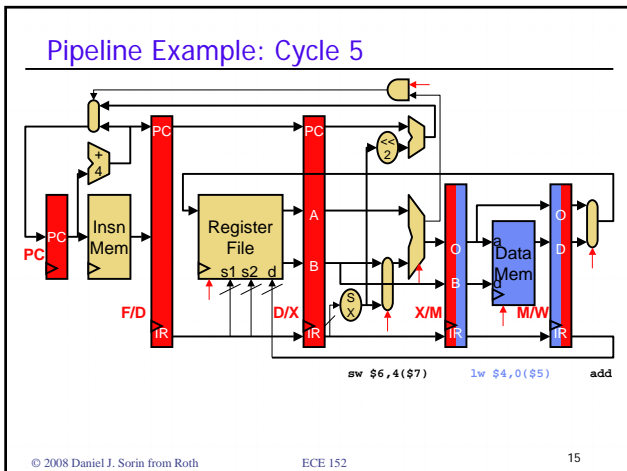
Pipeline Example: Cycle 3



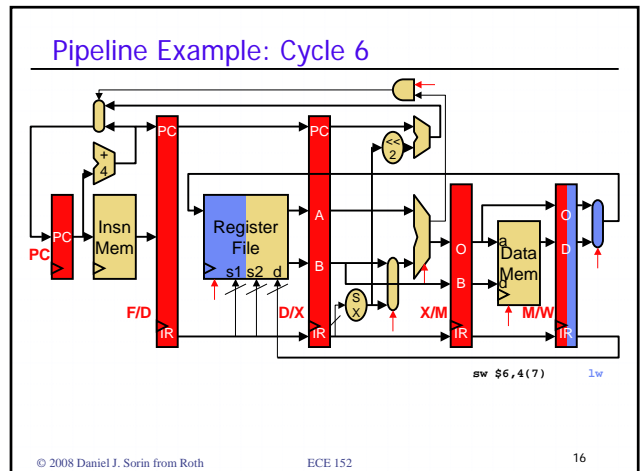
Pipeline Example: Cycle 4



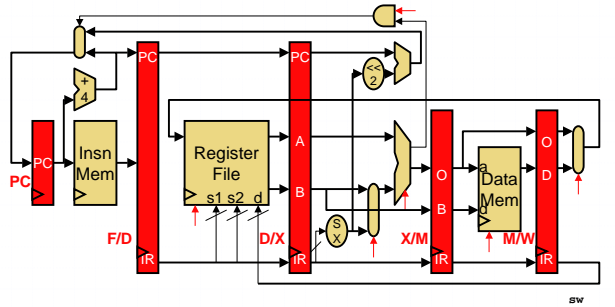
Pipeline Example: Cycle 5



Pipeline Example: Cycle 6



Pipeline Example: Cycle 7



Pipeline Diagram

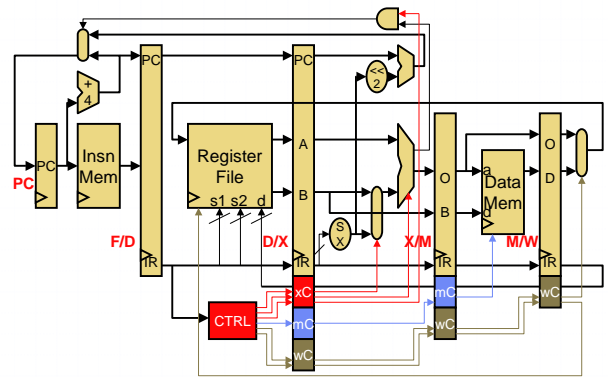
- **Pipeline diagram:** shorthand for what we just saw
 - Across: cycles
 - Down: insns
 - Convention: **X** means `lw $4, 0($5)` finishes execute stage and writes into X/M latch at end of cycle 4

	1	2	3	4	5	6	7	8	9
<code>add \$3, \$2, \$1</code>	F	D	X	M	W				
<code>lw \$4, 0(\$5)</code>		F	D	X	M	W			
<code>sw \$6, 4(\$7)</code>			F	D	X	M	W		

What About Pipelined Control?

- Should it be like single-cycle control?
 - But individual insn signals must be staged
- Should it be like multi-cycle control?
 - But all stages are simultaneously active
- How many different control units do we need?
 - One for each insn in pipeline?
- Solution: use simple single-cycle control, but pipeline it
 - Single controller
 - Pass control signals with instruction through pipeline

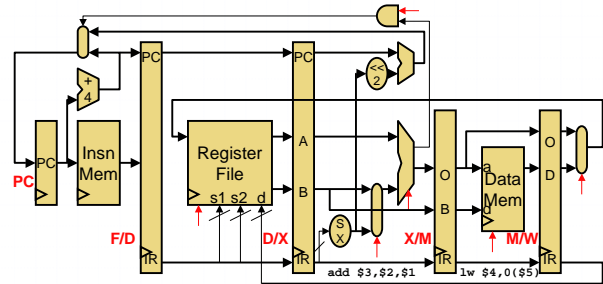
Pipelined Control



Pipeline Performance Calculation

- “Back of the envelope” calculation
 - Branch: 20%, load: 20%, store: 10%, other: 50%
- Single-cycle
 - Clock period = 50ns, CPI = 1
 - Performance = 50ns/insn
- Multi-cycle
 - Clock period = 12ns, CPI = $(0.2 \cdot 3 + 0.2 \cdot 5 + 0.6 \cdot 4) = 4$
 - Performance = 48ns/insn
- Pipelined
 - Clock period = **12ns**
 - CPI = **1** (each insn takes 5 cycles, but 1 completes each cycle)
 - Performance = **12ns/insn**

Why Does Every Insn Take 5 Cycles?

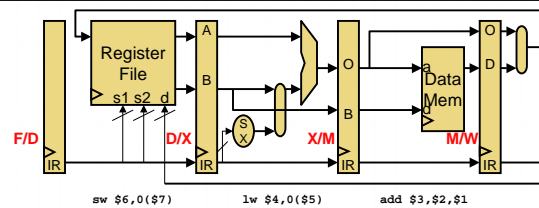


- Why not let **add** skip M and go straight to W?
 - It wouldn't help: peak fetch still only 1 insn per cycle
 - **Structural hazards**: not enough resources per stage for 2 insns

Pipeline Hazards

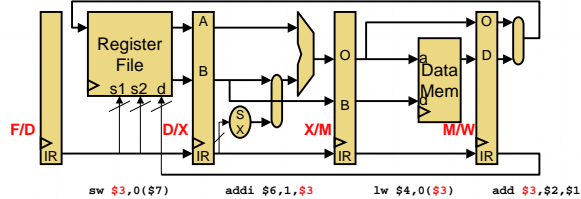
- **Hazard**: condition leads to incorrect execution if not fixed
 - “Fixing” typically increases CPI
 - Three kinds of hazards
- **Structural hazards**
 - Two insns trying to use same circuit at same time
 - E.g., structural hazard on RegFile write port
 - Fix by proper ISA/pipeline design: 3 rules to follow
 - Each insn uses every structure exactly once
 - For at most one cycle
 - Always at same stage relative to F
- **Data hazards** (next)
- **Control hazards** (a little later)

Data Hazards



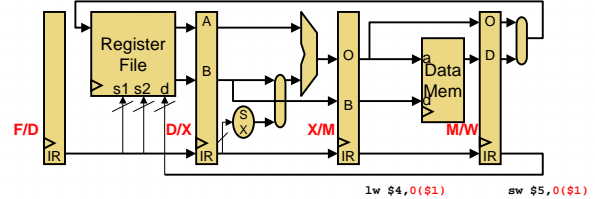
- Let's forget about branches and control for a while
- The sequence of 3 insns we saw earlier executed fine...
 - But it wasn't a real program
 - Real programs have **data dependences**
 - They pass values via registers and memory

Data Hazards



- Would this “program” execute correctly on this pipeline?
 - Which insns would execute with correct inputs?
 - **add** is writing its result into **\$3** in current cycle
 - **lw** read **\$3** 2 cycles ago → got wrong value
 - **addi** read **\$3** 1 cycle ago → got wrong value
 - **sw** is reading **\$3** this cycle → OK (regfile timing: write first half)

Memory Data Hazards



- What about data hazards through memory? No
 - **lw** following **sw** to same address in next cycle, gets right value
 - Why? DMem read/write take place in same stage
- Data hazards through registers? Yes (previous slide)
 - Occur because register write is 3 stages after register read
 - Can only read a register value 3 cycles after writing it

Fixing Register Data Hazards

- Can only read register value 3 cycles after writing it
- One way to enforce this: make sure programs don't do it
 - Compiler puts two **independent** insns between write/read insn pair
 - If they aren't there already
 - Independent means: “do not interfere with register in question”
 - Do not write it: otherwise meaning of program changes
 - Do not read it: otherwise create new data hazard
 - **Code scheduling**: compiler moves around existing insns to do this
 - If none can be found, must use **NOPs**
- This is called **software interlocks**
 - **MIPS**: Microprocessor w/out Interlocking Pipeline Stages

Software Interlock Example

```
add $3, $2, $1
lw $4, 0($3)
sw $7, 0($3)
add $6, $2, $8
addi $3, $5, 4
```

- Can any of last 3 insns be scheduled between first two?
 - **sw \$7, 0(\$3)**? No, creates hazard with **add \$3, \$2, \$1**
 - **add \$6, \$2, \$8**? OK
 - **addi \$3, \$5, 4**? No, **lw** would read **\$3** from it
 - Still need one more insn, use **nop**

```
add $3, $2, $1
add $6, $2, $8
nop
lw $4, 0($3)
sw $7, 0($3)
addi $3, $5, 4
```

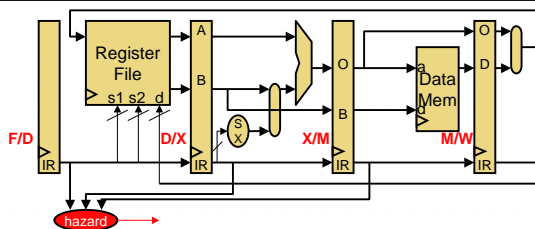
Software Interlock Performance

- Assume same distribution of instructions
 - Branch: 20%, load: 20%, store: 10%, other: 50%
- Software interlocks
 - 20% of insns require insertion of 1 `nop`
 - 5% of insns require insertion of 2 `nops`
- CPI is still 1 technically
- But now there are more insns
- #insns = $1 + 0.20 * 1 + 0.05 * 2 = 1.3$
- **30% more insns (30% slowdown) due to data hazards**

Hardware Interlocks

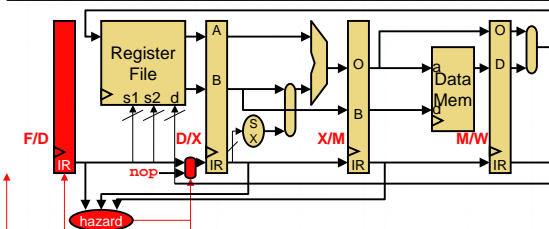
- Problem with software interlocks? Not compatible
 - Where does **3** in "read register 3 cycles after writing" come from?
 - From structure (depth) of pipeline
 - What if next MIPS version uses a 7 stage pipeline?
 - Programs compiled assuming 5 stage pipeline will break
- A better (more compatible) way: **hardware interlocks**
 - Processor detects data hazards and fixes them
 - Two aspects to this
 - Detecting hazards
 - Fixing hazards

Detecting Data Hazards



- Compare F/D insn input register names with output register names of older insns in pipeline
- Hazard =
- $$(F/D.IR.RS1 == D/X.IR.RD) \parallel (F/D.IR.RS2 == D/X.IR.RD) \parallel (F/D.IR.RS1 == X/M.IR.RD) \parallel (F/D.IR.RS2 == X/M.IR.RD)$$

Fixing Data Hazards



- Prevent F/D insn from reading (advancing) this cycle
 - Write `nop` into D/X.IR (effectively, insert `nop` in hardware)
 - Also clear the datapath control signals
 - Disable F/D latch and PC write enables (why?)
- Re-evaluate situation next cycle