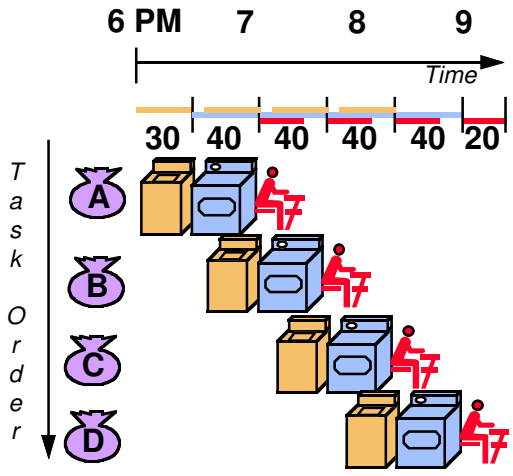
# Lecture 8: Pipeline Complications—Control Hazards, Branches and Interrupts

**Professor Randy H. Katz**
**Computer Science 252**
**Spring 1996**



## Review From Last Time Pipelining Lessons




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

# Review From Last Time: Software Scheduling to Avoid Load Hazards

Try producing fast code for

a = b + c;

d = e – f;

assuming a, b, c, d ,e, and f in memory.

| Slow code: | | Fast code: | |
|---|---|---|---|
| LW | Rb,b | LW | Rb,b |
| LW | Rc,c | LW | Rc,c |
| ADD | Ra,Rb,Rc | LW | Re,e |
| SW | a,Ra | ADD | Ra,Rb,Rc |
| LW | Re,e | LW | Rf,f |
| LW | Rf,f | SW | a,Ra |
| SUB | Rd,Re,Rf | SUB | Rd,Re,Rf |
| SW | d,Rd | SW | d,Rd |



# Review From Last Time: Pipelining Summary

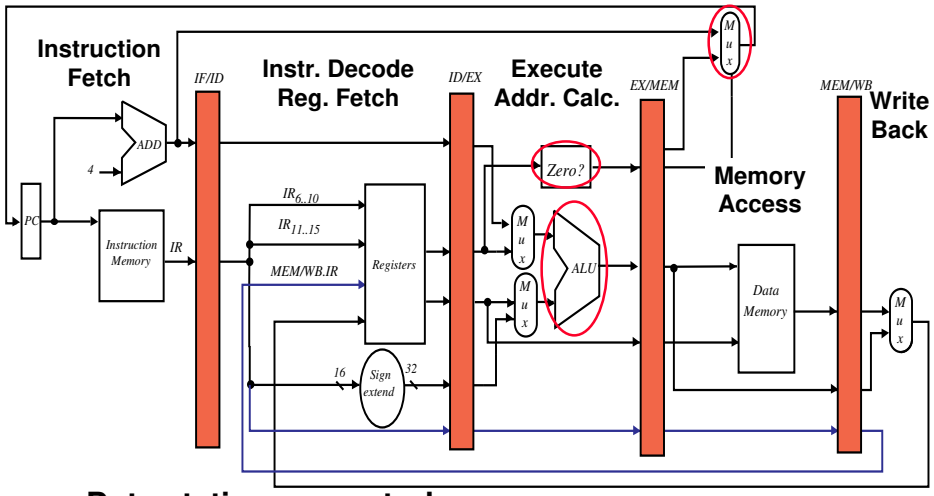
- Just overlap tasks, and easy if tasks are independent
- Speed Up ≤ Pipeline Depth; if ideal CPI is 1, then:

$$\text{Speedup} = \frac{\text{Pipeline Depth}}{\text{1 + Pipeline stall CPI}} \times \frac{\text{Clock Cycle Unpipelined}}{\text{Clock Cycle Pipelined}}$$

- Hazards limit performance on computers:
  - Structural: need more HW resources
  - Data: need forwarding, compiler scheduling
  - Control: discuss next time
- Today Branches and Other Difficulties

# Pipelined DLX Datapath

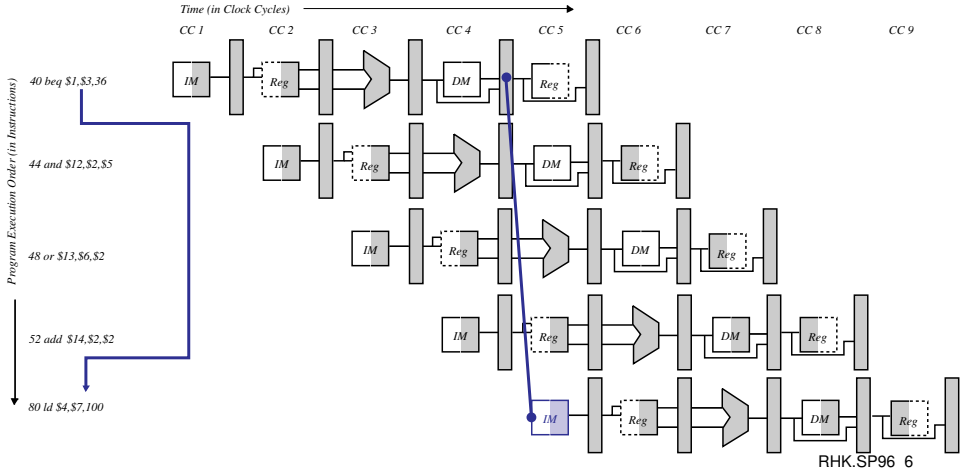
Figure 3.4, page 134




- **Data stationary control**
  - **Local decode for each instruction phase / pipeline stage**

RHK.SP96 5

# Control Hazard on Branches Three Stage Stall




RHK.SP96 6

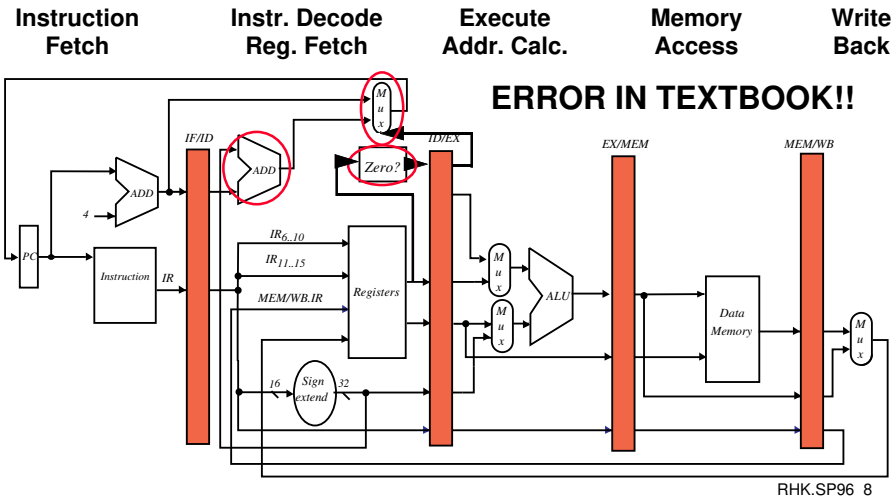
# Branch Stall Impact

- If CPI = 1, 30% branch, Stall 3 cycles => new CPI = 1.9!
- Two part solution:
  - Determine branch taken or not sooner, AND
  - Compute taken branch address earlier
- DLX branch tests if register = 0 or ≠ 0
- DLX Solution:
  - Move Zero test to ID/RF stage
  - Adder to calculate new PC in ID/RF stage
  - 1 clock cycle penalty for branch versus 3

RHK.SP96 7

# Pipelined DLX Datapath

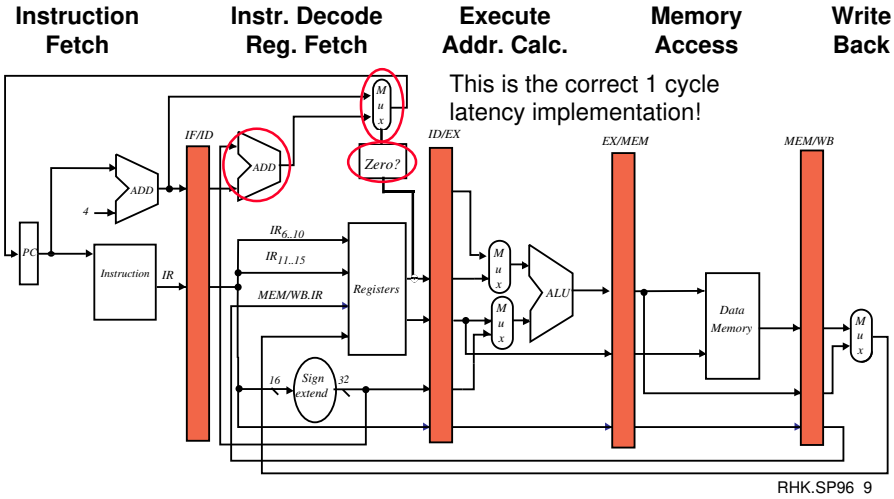
Figure 3.22, page 163




RHK.SP96 8

# Pipelined DLX Datapath

Figure 3.22, page 163




RHK.SP96 9

# Four Branch Hazard Alternatives

**#1: Stall until branch direction is clear**

**#2: Predict Branch Not Taken**

- Execute successor instructions in sequence
- "Squash" instructions in pipeline if branch actually taken
- Advantage of late pipeline state update
- 47% DLX branches not taken on average
- PC+4 already calculated, so use it to get next instruction

**#3: Predict Branch Taken**

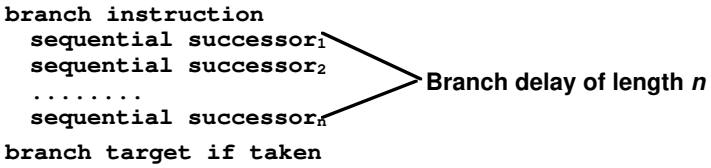
- 53% DLX branches taken on average
- But haven't calculated branch target address in DLX
  - » DLX still incurs 1 cycle branch penalty
  - » Other machines: branch target known before outcome

RHK.SP96 10

# Four Branch Hazard Alternatives

### #4: Delayed Branch

– Define branch to take place AFTER a following instruction




– 1 slot delay allows proper decision and branch target address in 5 stage pipeline

– DLX uses this



# Delayed Branch

- **Where to get instructions to fill branch delay slot?**
  - Before branch instruction
  - From the target address: only valuable when branch taken
  - From fall through: only valuable when branch not taken
  - Cancelling branches allow more slots to be filled
- **Compiler effectiveness for single branch delay slot:**
  - Fills about 60% of branch delay slots
  - About 80% of instructions executed in branch delay slots useful in computation
  - About 50% (60% x 80%) of slots usefully filled

# Evaluating Branch Alternatives

$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

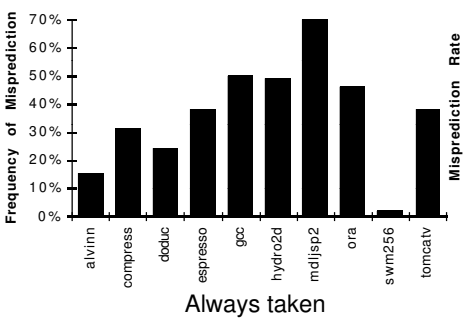
| *Scheduling scheme* | *Branch penalty* | *CPI* | *speedup v. unpipelined* | *speedup v. stall* |
|---|---|---|---|---|
| **Stall pipeline** | **3** | **1.42** | **3.5** | **1.0** |
| **Predict taken** | **1** | **1.14** | **4.4** | **1.26** |
| **Predict not taken** | **1** | **1.09** | **4.5** | **1.29** |
| **Delayed branch** | **0.5** | **1.07** | **4.6** | **1.31** |

**Conditional & Unconditional = 14%, 65% change PC**

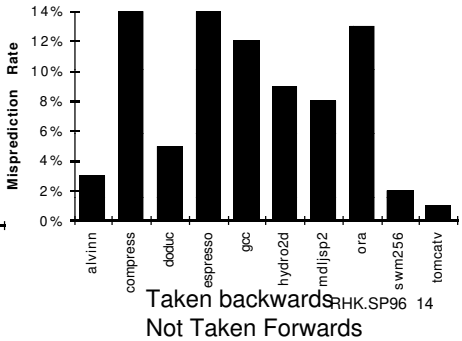


# Compiler "Static" Prediction of Taken/Untaken Branches

- **Improves strategy for placing instructions in delay slot**
- **Two strategies**
  - **Backward branch predict taken, forward branch not taken**
  - **Profile-based prediction: record branch behavior, predict branch based on prior run**



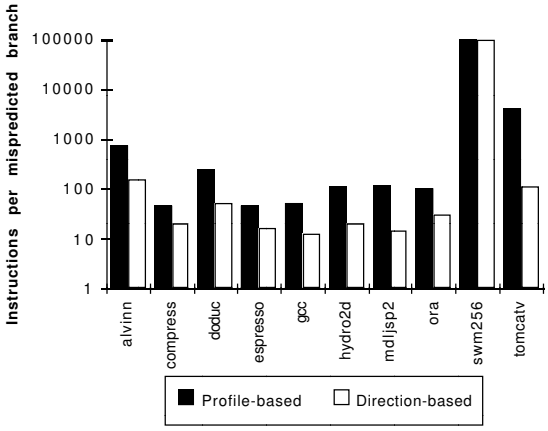

Always taken




Taken backwards
Not Taken Forwards

# Evaluating Static Branch Prediction Strategies

- **Misprediction ignores frequency of branch**
- **"Instructions between mispredicted branches" is a better metric**






# Pipelining Complications

- **Interrupts: 5 instructions executing in 5 stage pipeline**
  - **How to stop the pipeline?**
  - **How to restart the pipeline?**
  - **Who caused the interrupt?**

| *Stage* | *Problem interrupts occurring* |
|---|---|
| **IF** | **Page fault on instruction fetch; misaligned memory access; memory-protection violation** |
| **ID** | **Undefined or illegal opcode** |
| **EX** | **Arithmetic interrupt** |
| **MEM** | **Page fault on data fetch; misaligned memory access; memory-protection violation** |

# Pipelining Complications

- Simultaneous exceptions in more than one pipeline stage, e.g.,
  - Load with data page fault in MEM stage
  - Add with instruction page fault in IF stage
  - Add fault will happen BEFORE load fault
- Solution #1
  - Interrupt status vector per instruction
  - Defer check til last stage, kill state update if exception
- Solution #2
  - Interrupt ASAP
  - Restart everything that is incomplete

Another advantage for state update late in pipeline!



# Pipelining Complications

- Complex Addressing Modes and Instructions
- Address modes: Autoincrement causes register change during instruction execution
  - Interrupts? Need to restore register state
  - Adds WAR and WAW hazards since writes no longer last stage
- Memory-Memory Move Instructions
  - Must be able to handle multiple page faults
  - Long-lived instructions: partial state save on interrupt
- Condition Codes

# Pipelining Complications

- **Floating Point: long execution time**
- **Also, may pipeline FP execution unit so they can initiate new instructions without waiting full latency**

| *FP Instruction* | *Latency* | *Initiation Rate* | *(MIPS R4000)* |
|---|---|---|---|
| **Add, Subtract** | **4** | **3** | |
| **Multiply** | **8** | **4** | |
| **Divide** | **36** | **35** | |
| **Square root** | **112** | **111** | |
| **Negate** | **2** | **1** | |
| **Absolute value** | **2** | **1** | |
| **FP compare** | **3** | **2** | |
| | Cycles before use result | Cycles before issue instr of same type | |



# Pipelining Complications

- **Divide, Square Root take ≈10X to ≈30X longer than Add**
  - **Interrupts?**
  - **Adds WAR and WAW hazards since pipelines are no longer same length**

# Summary of Pipelining Basics

- **Hazards limit performance**
  - **Structural: need more HW resources**
  - **Data: need forwarding, compiler scheduling**
  - **Control: early evaluation & PC, delayed branch, prediction**
- **Increasing length of pipe increases impact of hazards; pipelining helps instruction bandwidth, not latency**
- **Interrupts, Instruction Set, FP makes pipelining harder**
- **Compilers reduce cost of data and control hazards**
  - **Load delay slots**
  - **Bbranch delay slots**
  - **Branch prediction**
- **Next time: Longer pipelines (R4000) => Better branch prediction, more instruction parallelism?**