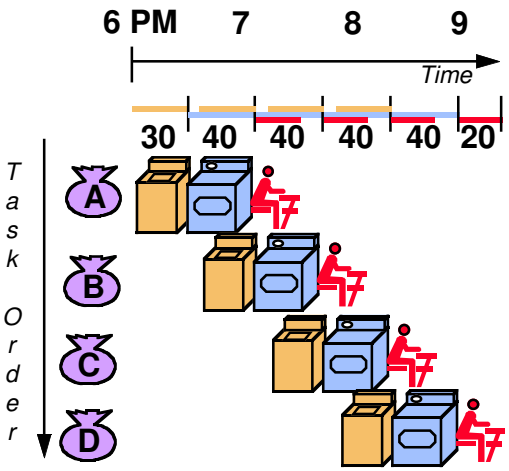
Review From Last Time: Pipelining Lessons



- 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 ɜfill pipeline and time to ɜdrain it reduces speedup

© Patterson 1994

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:

```
LW    Rb,b
LW    Rc,c
ADD   Ra,Rb,Rc
SW    a,Ra
LW    Re,e
LW    Rf,f
SUB   Rd,Re,Rf
SW    d,Rd
```

Fast code:

```
LW    Rb,b
LW    Rc,c
LW    Re,e
ADD   Ra,Rb,Rc
LW    Rf,f
SW    a,Ra
SUB   Rd,Re,Rf
SW    d,Rd
```


© Patterson 1994

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:

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall cycles per instruction}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}$$

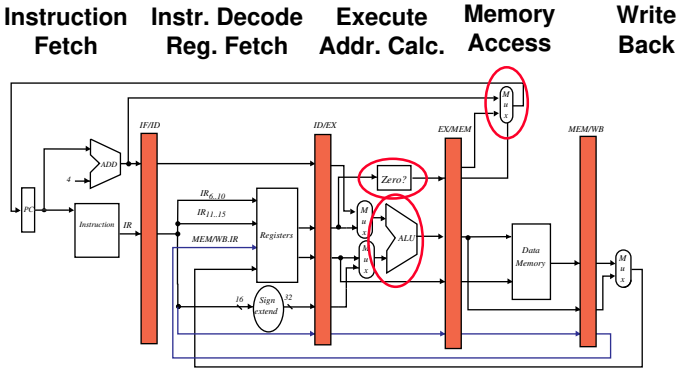
Hazards limit performance on computers:

- **structural: need more HW resources**
- **data: need forwarding, compiler scheduling**
- **control: discuss next time**

Today Branches and Other Difficulties

© Patterson 1994

Pipelined DLX Datapath *(Figure 4.4, page 171 Beta Edition of CA:AQA,2/e)*

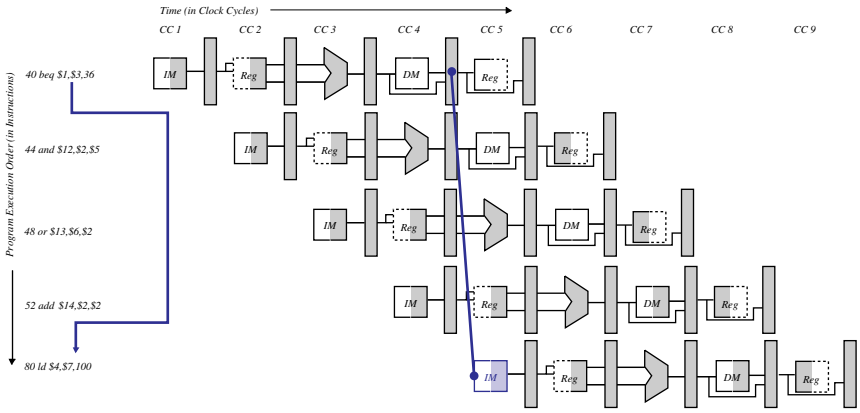


Data stationary control

- **local decode for each instruction phase / pipeline stage**

© Patterson 1994

Control Hazard on Branches: 3 stage stall



© Patterson 1994

Branch Stall Impact

If CPI = 1, 30% branch, Stall 3 cycles => new CPI = 1.9!

2 part solution:

- **Determine branch taken or not sooner, AND**
- **Compute taken branch address earlier**

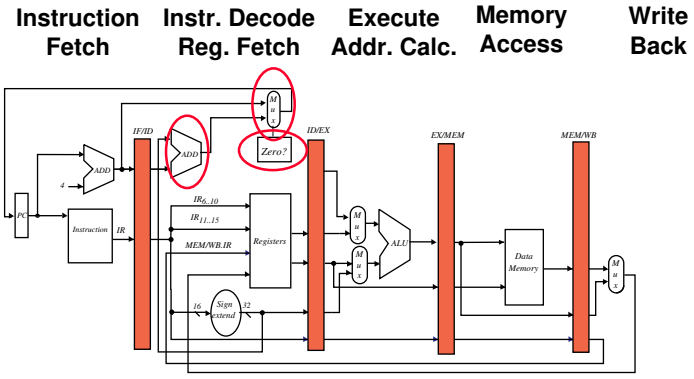
DLX branch tests = 0 or 0

DLX Solution:

- **Move Zero test to ID/RF stage**
- **Adder to calculate new PC in ID/RF stage**
- **1 clock cycle penalty for branch vs. 3**

© Patterson 1994

Pipelined DLX Datapath *(Figure 4.23, page 198 Beta Edition of CA:AQA,2/e)*



© Patterson 1994

Branch Hazard Alternatives

- Stall until branch direction is clear
- Predict not taken
  - ≅ 47% DLX branches not taken on average
  - ≅ PC+4 already calculated
- Predict taken
  - ≅ 53% DLX branches taken on average
  - ≅ But haven t calculated branch target address in DLX
- Delayed Branch
  - ≅ Define branch to take place AFTER a following instruction

```
branch instruction
  sequential successor1
  sequential successor2
  ........
  sequential successorn
branch target if taken
```

  - ≅ 1 slot delay allows proper decision and branch target address in 5 stage pipeline; DLX uses this

© Patterson 1994

Delayed Branch

Where get instructions to fill branch delay slot?

- **Before branch instruction**
- **From the target address: only valuable when branch**
- **From fall through: only valuable when don t branch**

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**

© Patterson 1994

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

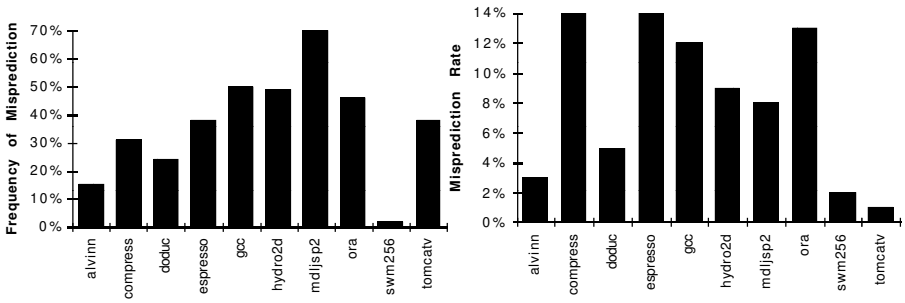
© Patterson 1994

Compiler Prediction of Taken/Untaken Branches

Improves strategy on placing instructions in delay slot

Two strategies

- **Backward branch predict taken, forward branch untaken**
- **Profile-based prediction: record branch behavior, predict branch based on prior run**

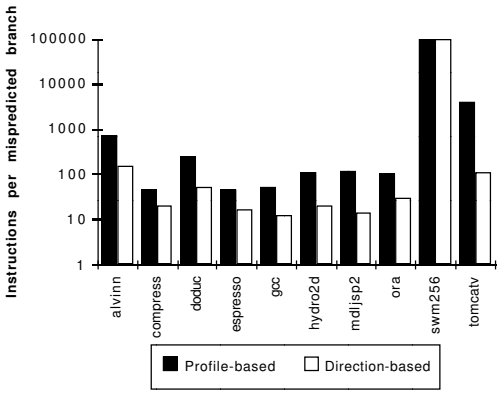


© Patterson 1994

Evaluating branch prediction strategies

Misprediction ignores frequency of branch

Instructions between mispredicted branches a better metric



© Patterson 1994

When is pipelining hard?

Interrupts: 5 instructions executing in 5 stage pipeline

- ≅ How to stop the pipeline?
- ≅ Restrart?
- ≅ 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

Load with data page fault, Add with instruction page fault?

Solution 1: interrupt vector/instruction , check last stage

Solution 2: interrupt ASAP, restart everything incomplete

© Patterson 1994

When is pipelining hard?

Complex Addressing Modes and Instructions

Address modes: Autoincrement causes register change during instruction execution

- ≅ Interrupts?
- ≅ Adds WAR and WAW hazards since writes no longer last stage

Memory-memory Move instructions

- ≅ Multiple page faults
- ≅ make progress?

© Patterson 1994

When is pipelining hard?

Floating Point: long execution time

Also, may pipeline FP execution unit so that 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	

Divide, Square Root take 10X to 30X longer than Add

- ≅ **Interrupts?**
- ≅ **Adds WAR and WAW hazards since pipelines are no longer same length**

© Patterson 1994

Case Study: MIPS R4000 (100 MHz to 200 MHz)

8 stage pipeline:

- ≅ **IF≅first half of instruction fetch; PC selection happens here, together with initiation of instruction cache access.**
- ≅ **IS≅second half of instruction fetch, complete instruction cache access.**
- ≅ **RF≅instruction decode and register fetch, hazard checking and also instruction cache hit detection.**
- ≅ **EX≅execution, which includes effective address calculation, ALU operation, and branch target computation and condition evaluation.**
- ≅ **DF≅data fetch, first half of data cache access.**
- ≅ **DS≅second half of data fetch, compete of data cache access.**
- ≅ **TC≅tag check, determine whether the data cache access hit.**
- ≅ **WB≅write back for loads and register-register operations.**

8 stages & impact on Load delay? Branch delay? Why?

© Patterson 1994

MIPS R4000 Floating Point

FP Adder, FP Multiplier, FP Divider

Last step of FP Multiplier/Divider uses FP Adder HW

8 kinds of stages in FP units:

Stage	*Functional unit*	*Description*
A	FP adder	Mantissa ADD stage
D	FP divider	Divide pipeline stage
E	FP multiplier	Exception test stage
M	FP multiplier	First stage of multiplier
N	FP multiplier	Second stage of multiplier
R	FP adder	Rounding stage
S	FP adder	Operand shift stage
U		Unpack FP numbers

© Patterson 1994

MIPS FP pipe stages

FP Instr	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Add, Subtract	U	S+A	A+R	R+S				
Multiply	U	E+M	M	M	M	N	N+A	R
Divide	U	A	R	D^{28}		D+A	D+R, D+R, D+A, D+R, A, R	
Square root	U	E	$(A+R)^{108}$			A	R	
Negate	U	S						
Absolute value	U	S						
FP compare	U	A	R					

Stages:

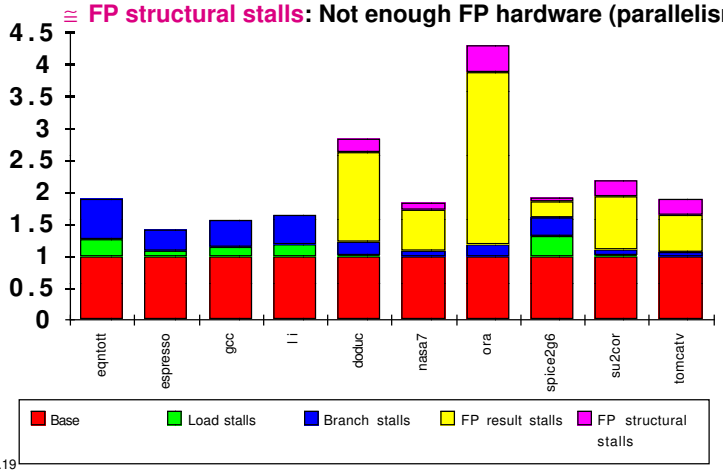
M	*First stage of multiplier*	*A*	*Mantissa ADD stage*
N	*Second stage of multiplier*	*D*	*Divide pipeline stage*
R	*Rounding stage*	*E*	*Exception test stage*
S	*Operand shift stage*		
U	*Unpack FP numbers*		

© Patterson 1994

R4000 Performance

Not ideal CPI of 1:

- ≅ Load stalls (1 or 2 clock cycles)
- ≅ Branch stalls (2 cycles + unfilled slots)
- ≅ FP result stalls: RAW data hazard (latency)
- ≅ FP structural stalls: Not enough FP hardware (parallelism)



© Patterson 1994

Summary of Pipelining Basics

Hazards limit performance on computers:

- ≅ structural: need more HW resources
- ≅ data: need forwarding, compiler scheduling
- ≅ control: discuss next time

Compilers key to reducing cost of data and control hazards

- ≅ load delay slots
- ≅ branch delay slots
- ≅ branch prediction

Interrupts, Instruction Set, FP makes pipelining harder

Increasing length of pipe increases impact of hazards since pipelining helps instruction bandwidth, not latency

Better branch prediction, more instruction parallelism?

© Patterson 1994