

CS152 – Computer Architecture and Engineering

Lecture 11 – Pipeline Control 2003-09-29

Dave Patterson

(www.cs.berkeley.edu/~patterson)

www-inst.eecs.berkeley.edu/~cs152/



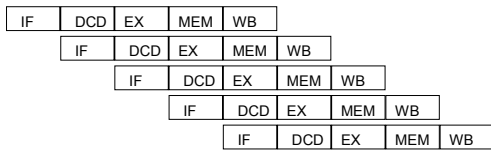
Review: Pipelining

- Reduce CPI by overlapping many instructions
 - Average throughput of approximately 1 CPI with fast clock
- Utilize capabilities of the Datapath
 - start next instruction while working on the current one
 - limited by length of longest stage (plus fill/flush)
 - detect and resolve hazards
- What makes it easy
 - all instructions are the same length
 - just a few instruction formats
 - memory operands appear only in loads and stores
- What makes it hard?
 - structural hazards: suppose we had only one memory
 - control hazards: need to worry about branch instructions
 - data hazards: an instruction depends on a previous instruction



Recap: Ideal Pipelining

Assume instructions are completely independent!

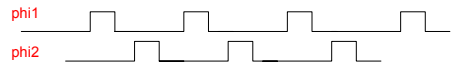


Maximum Speedup \leq Number of stages
 Speedup \leq Time for unpipelined operation / Time for longest stage

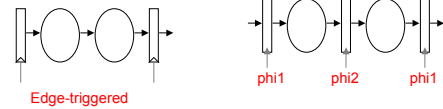
Example: 40ns data path, 5 stages, Longest stage is 10 ns, Speedup \leq 4



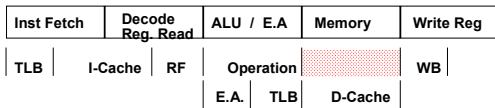
FYI: MIPS R3000 clocking discipline



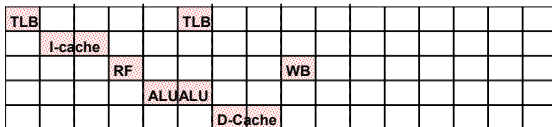
- 2-phase non-overlapping clocks
- Pipeline stage is two (level sensitive) latches



MIPS R3000 Instruction Pipeline



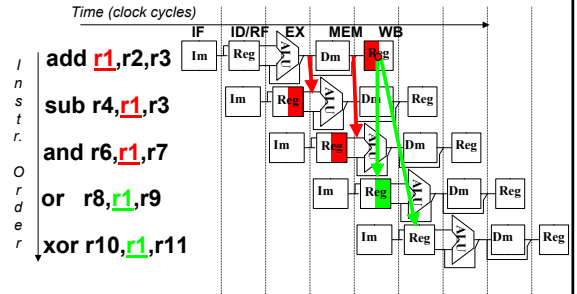
Resource Usage



Write in phase 1, read in phase 2 => eliminates bypass from WB



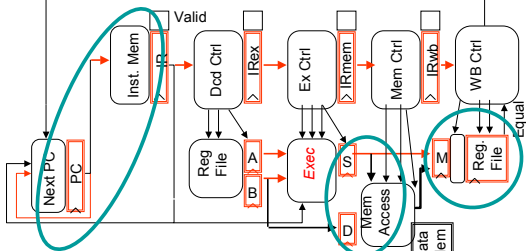
Recall: Data Hazard on r1



With MIPS R3000 pipeline, no need to forward from WB stage



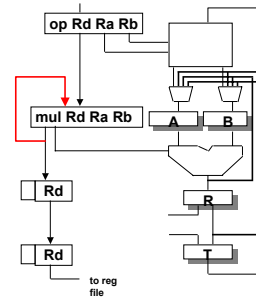
Clarification about clock edges in lab4!



- Since Register have edge-triggered write:
 - Must have everything set up at *end* of memory stage
 - This means that “M” register here is not necessary!
- Also, Memories will be *synchronous*
- Need to setup addresses and values in advance



MIPS R3000 Multicycle Operations



Use control word of local stage to step through multicycle operation

Stall all stages above multicycle operation in the pipeline

Drain (bubble) stages below it

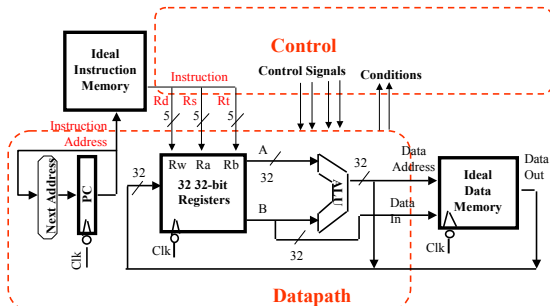
Alternatively, launch multiply/divide to autonomous unit, only stall pipe if attempt to get result before ready

- This means stall mfl0/mfhi in decode stage if multiply/divide still executing
- Extra credit in Lab 5 does this

Ex: Multiply, Divide, Cache Miss

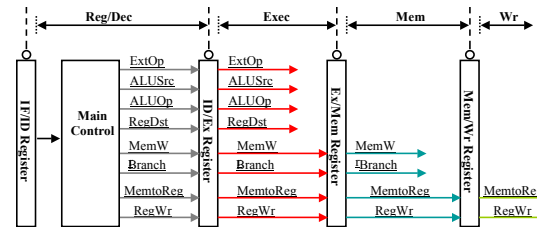


Recall: Single cycle control!

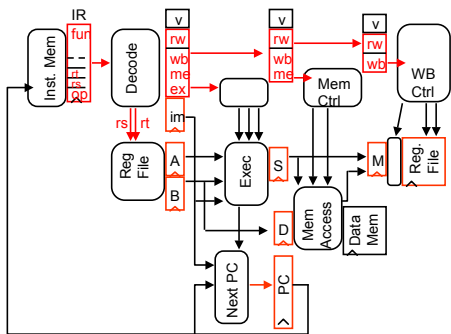


Data Stationary Control

- The Main Control generates the control signals during Reg/Dec
 - Control signals for Exec (ExtOp, ALUSrc, ...) are used 1 cycle later
 - Control signals for Mem (MemWr Branch) are used 2 cycles later
 - Control signals for Wr (MementoReg MemWr) are used 3 cycles later



Datpath + Data Stationary Control



Administrivia

- Lab 4 Project document Thursday 9 PM paper or email
- Reading Chapter 6, sections 6.1 to 6.5
- Midterm Wed Oct 8 5:30 - 8:30 in 1 LeConte
 - Midterm review Sunday Oct 4, 5 PM in 306 Soda
 - Bring 1 page, handwritten notes, both sides
 - Meet at LaVal's Northside afterwards for Pizza
- Office hours
 - Mon 4 – 5:30 Jack, Tue 3:30-5 Kurt,
 - Wed 3 – 4:30 John, Thu 3:30-5 Ben
 - Dave's office hours Tue 3:30 – 5



Let's Try it Out

```

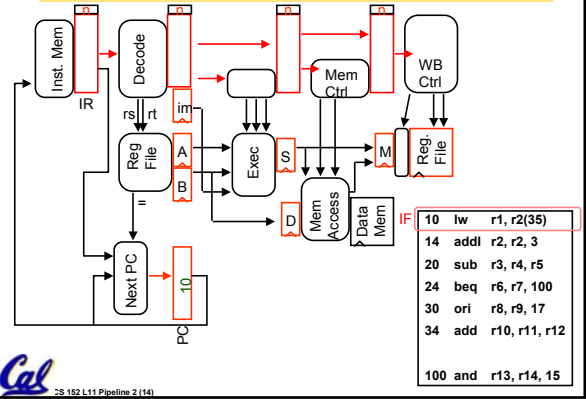
10 lw r1, r2(35)
14 addl r2, r2, 3
20 sub r3, r4, r5
24 beq r6, r7, 100
30 ori r8, r9, 17
34 add r10, r11, r12

100 and r13, r14, 15
    
```

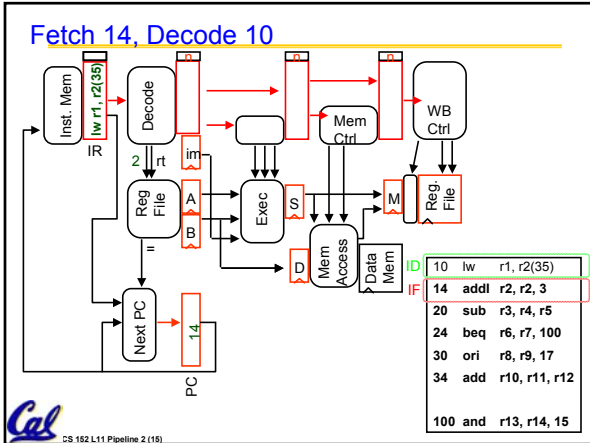
these addresses are octal



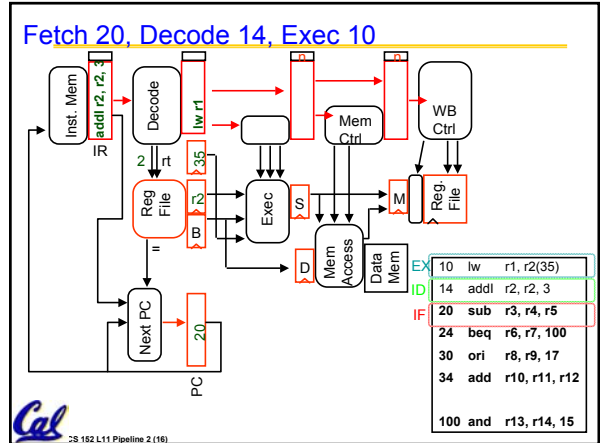
Start: Fetch 10



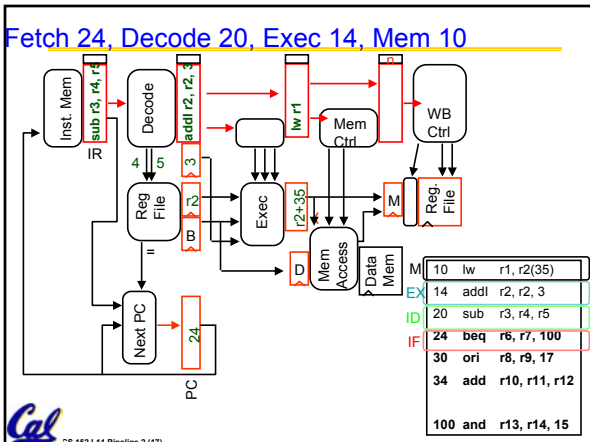
Fetch 14, Decode 10



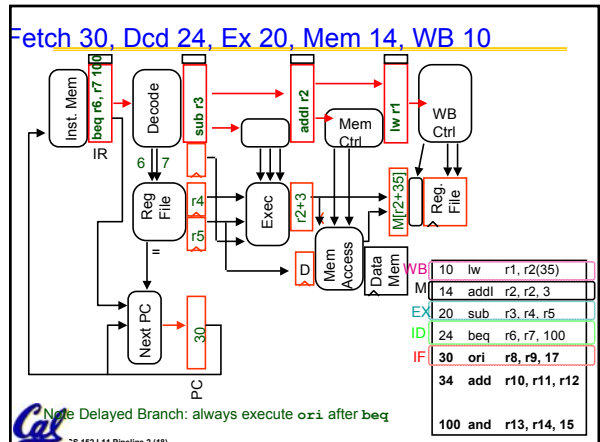
Fetch 20, Decode 14, Exec 10



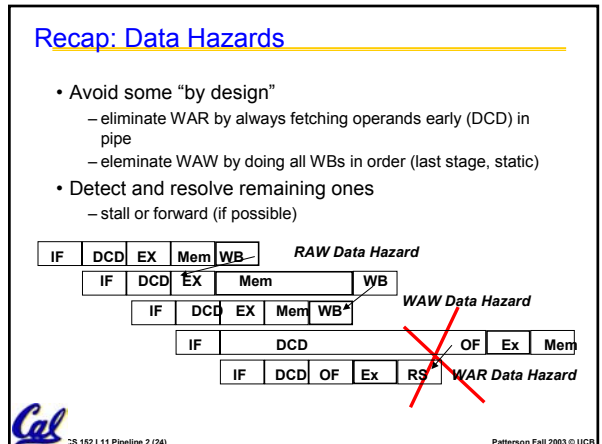
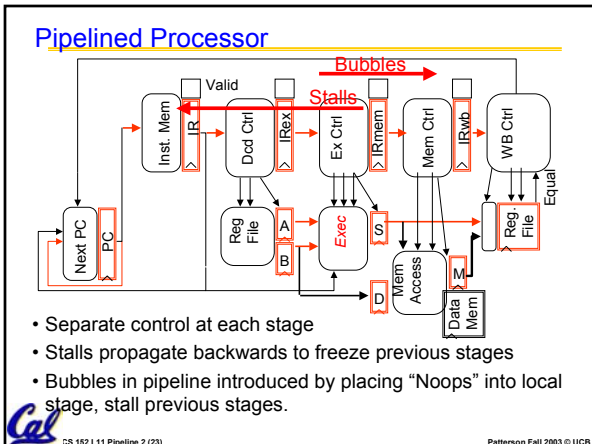
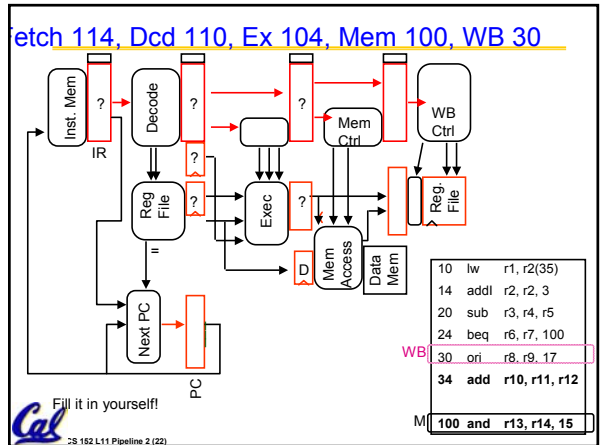
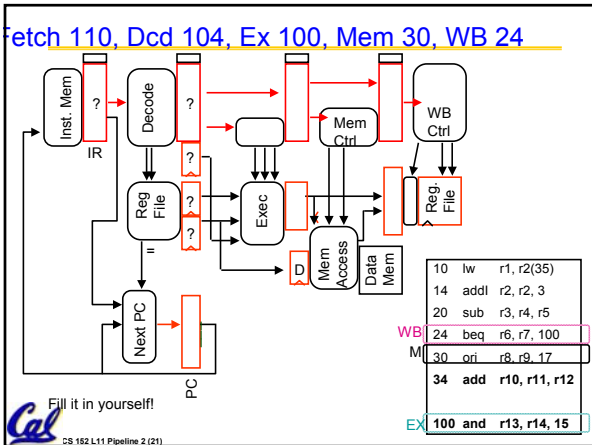
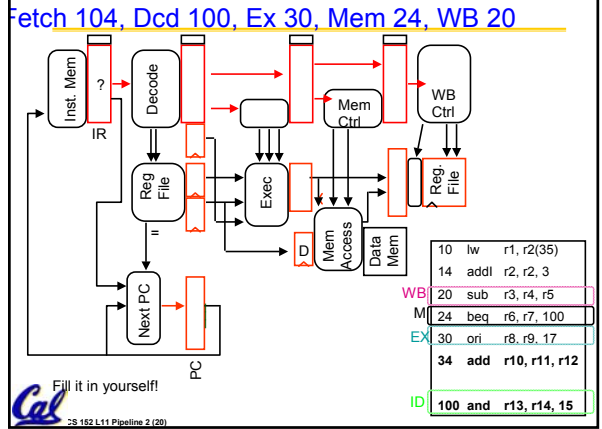
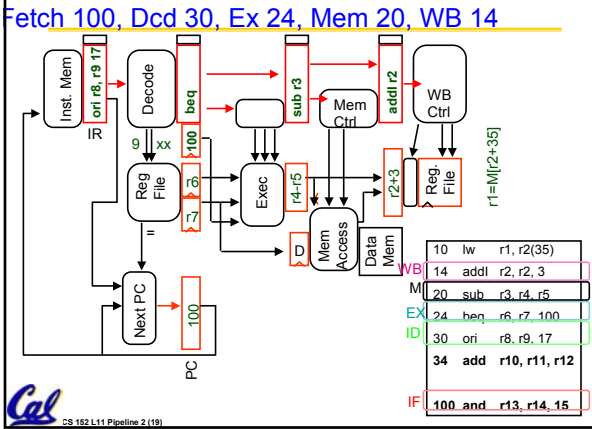
Fetch 24, Decode 20, Exec 14, Mem 10



Fetch 30, Dcd 24, Ex 20, Mem 14, WB 10

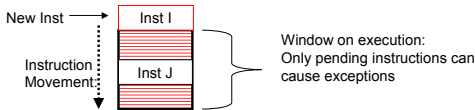


Note Delayed Branch: always execute ori after beq



Hazard Detection

- Suppose instruction i is about to be issued and a predecessor instruction j is in the instruction pipeline.



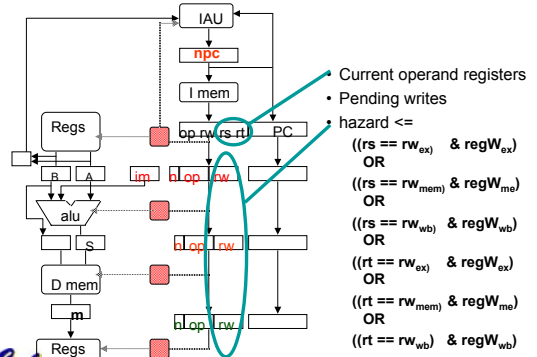
- A RAW hazard exists on register ρ if $\rho \in \text{Rregs}(i) \cap \text{Wregs}(j)$
 - Keep a record of pending writes (for inst's in the pipe) and compare with operand regs of current instruction.
 - When instruction issues, reserve its result register.
 - When on operation completes, remove its write reservation.
- A WAW hazard exists on register ρ if $\rho \in \text{Wregs}(i) \cap \text{Wregs}(j)$
- A WAR hazard exists on register ρ if $\rho \in \text{Wregs}(i) \cap \text{Rregs}(j)$

Cal

CS 152 L11 Pipeline 2 (26)

Patterson Fall 2003 © UCB

Record of Pending Writes In Pipeline Registers

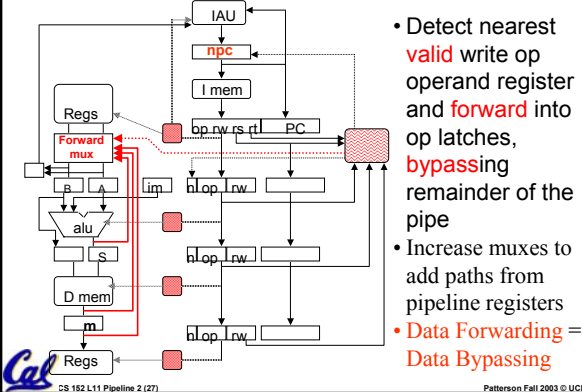


Cal

CS 152 L11 Pipeline 2 (26)

Patterson Fall 2003 © UCB

Resolve RAW by "forwarding" (or bypassing)

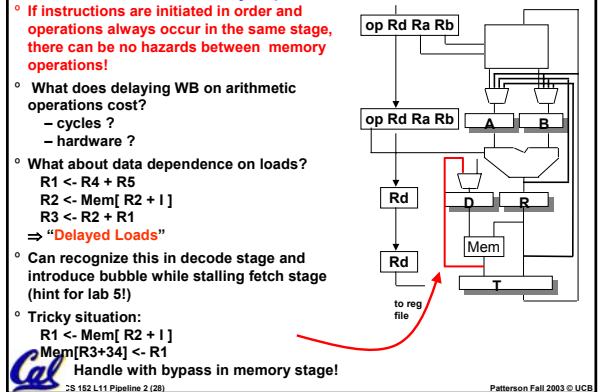


Cal

CS 152 L11 Pipeline 2 (27)

Patterson Fall 2003 © UCB

What about memory operations?

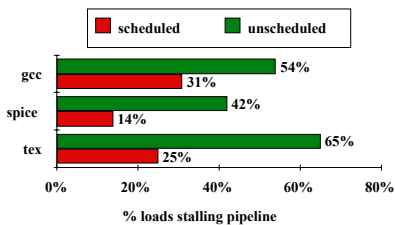


Cal

CS 152 L11 Pipeline 2 (28)

Patterson Fall 2003 © UCB

Compiler Avoiding Load Stalls:

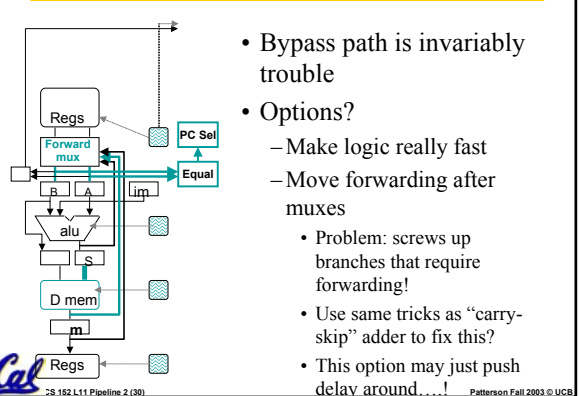


Cal

CS 152 L11 Pipeline 2 (29)

Patterson Fall 2003 © UCB

Question: Critical Path???



Cal

CS 152 L11 Pipeline 2 (30)

Patterson Fall 2003 © UCB

Is CPI = 1 for our pipeline?

- Remember that CPI is an "Average # cycles/inst"



- CPI here is 1, since the average throughput is 1 instruction every cycle.
- What if there are stalls or multi-cycle execution?
- Usually CPI > 1. **How close can we get to 1??**

Cal

Recall: Compute CPI?

- Start with Base CPI
- Add stalls

$$CPI = CPI_{base} + CPI_{stall}$$

$$CPI_{stall} = STALL_{type-1} \times freq_{type-1} + STALL_{type-2} \times freq_{type-2}$$

- Suppose:
 - $CPI_{base} = 1$
 - $freq_{branch} = 20\%$, $freq_{load} = 30\%$
 - Suppose branches always cause 1 cycle stall
 - Loads cause a 100 cycle stall 1% of time
- Then: $CPI = 1 + (1 \times 0.20) + (100 \times 0.30 \times 0.01) = 1.5$
- Multicycle? Could treat as:

$$CPI_{stall} = (CYCLES - CPI_{base}) \times freq_{inst}$$

Cal

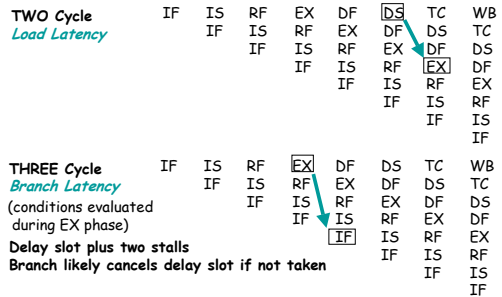
Case Study: MIPS R4000 (200 MHz)

- 8 Stage Pipeline:
 - IF—first half of fetching of instruction; PC selection happens here as well as initiation of instruction cache access.
 - IS—second half of access to instruction cache.
 - 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 access to data cache.
 - DS—second half of access to data cache.
 - TC—tag check, determine whether the data cache access hit.
 - WB—write back for loads and register-register operations.
- 8 Stages:

What is impact on Load delay? Branch delay? Why?

Cal

Case Study: MIPS R4000



Cal

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

Cal

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						

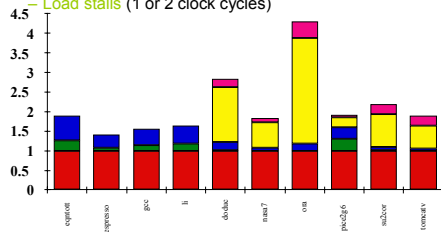
Stages:

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		

Cal

R4000 Performance

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



Cal

Summary

- What makes it easy
 - all instructions are the same length
 - just a few instruction formats
 - memory operands appear only in loads and stores
- Hazards limit performance
 - Structural: need more HW resources
 - Data: need forwarding, compiler scheduling
 - Control: early evaluation & PC, delayed branch, prediction
- Data hazards must be handled carefully:
 - RAW data hazards handled by forwarding
 - WAW and WAR hazards don't exist in 5-stage pipeline
- MIPS I instruction set architecture made pipeline visible (delayed branch, delayed load)
- More performance from deeper pipelines, parallelism

Cal