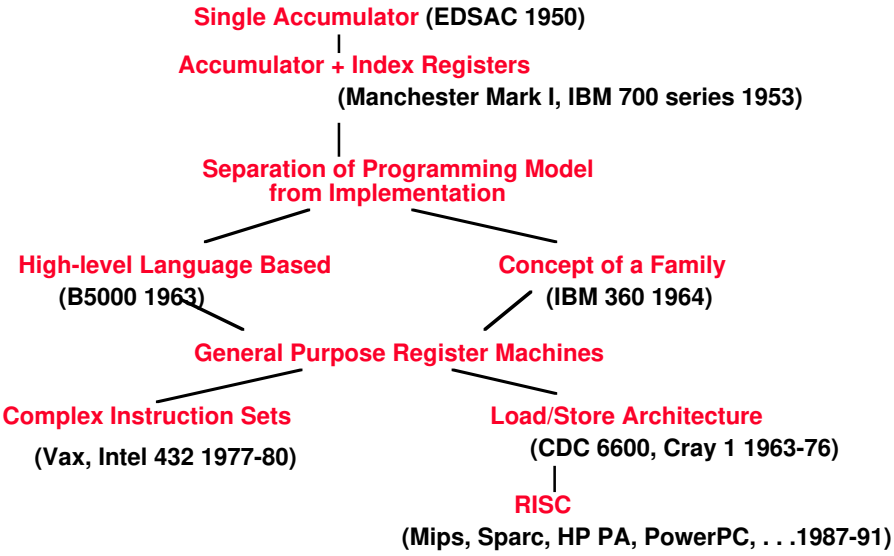
Lecture 7: Introduction to Pipelining, Structural Hazards, and Forwarding

Professor Randy H. Katz
Computer Science 252
Spring 1996

Latest NRC Rankings!!

	Quality of Faculty			Ranking		
	1982	1995	Change	1982	1995	Change
Stanford	5.0	4.97	−0.03	1	1	0
MIT	4.9	4.91	0.01	2	2	0
CMU	4.8	4.76	−0.04	3	4	−1
Berkeley	4.5	4.88	0.38	4	3	1
Cornell	4.3	4.64	0.34	5	5	0
Illinois	3.8	4.09	0.29	6	8	−2
UCLA	3.8	na		6	na	
Yale	3.5	na		8	na	
U.Wash.	3.4	4.04	0.64	9	9	0
USC	3.2	na		10	na	
UT/Austin	3.2	4.18	0.98	10	7	3
Wisconsin	3.2	4.00	0.80	10	10	0
Maryland	3.1	na		13	na	
Princeton	3.0	4.31	1.31	14	6	8

Review: Instruction Set Evolution



Review: 80x86 v. DLX Instruction Counts

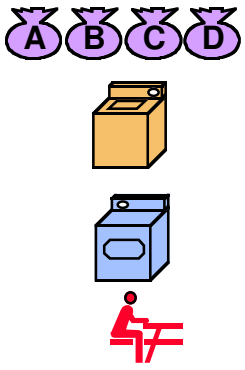
SPEC pgm	*x86*	*DLX*	*DLX÷86*
gcc	3,771,327,742	3,892,063,460	1.03
espresso	2,216,423,413	2,801,294,286	1.26
spice	15,257,026,309	16,965,928,788	1.11
nasa7	15,603,040,963	6,118,740,321	0.39

Review From Last Time
Intel Summary

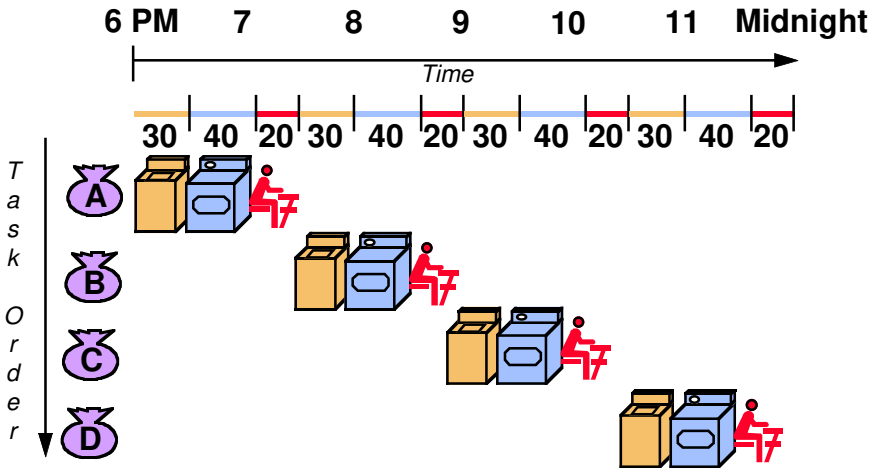
- Archeology: history of instruction design in a single product
  - Address size: 16 bit vs. 32-bit
  - Protection: Segmentation vs. paged
  - Temp. storage: accumulator vs. stack vs. registers
- "Golden Handcuffs" of binary compatibility affect design 20 years later, as Moore predicted
- Not too difficult to make faster, as Intel has shown
- HP/Intel announcement of common future instruction set by 2000 means end of 80x86???
- "Beauty is in the eye of the beholder"
  - At 50M/year sold, it is a beautiful business

Pipelining: Its Natural!

- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- "Folder" takes 20 minutes



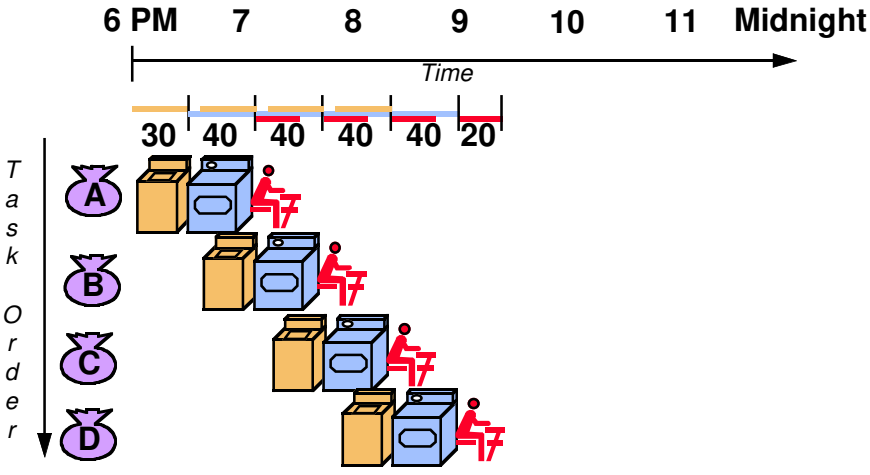
Sequential Laundry



- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

RHK.SP96 7

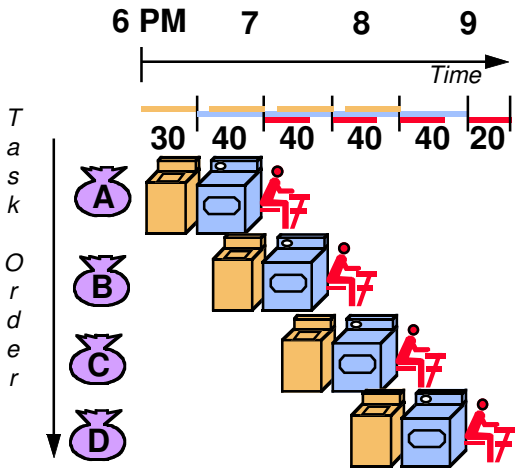
Pipelined Laundry
Start work ASAP



- Pipelined laundry takes 3.5 hours for 4 loads

RHK.SP96 8

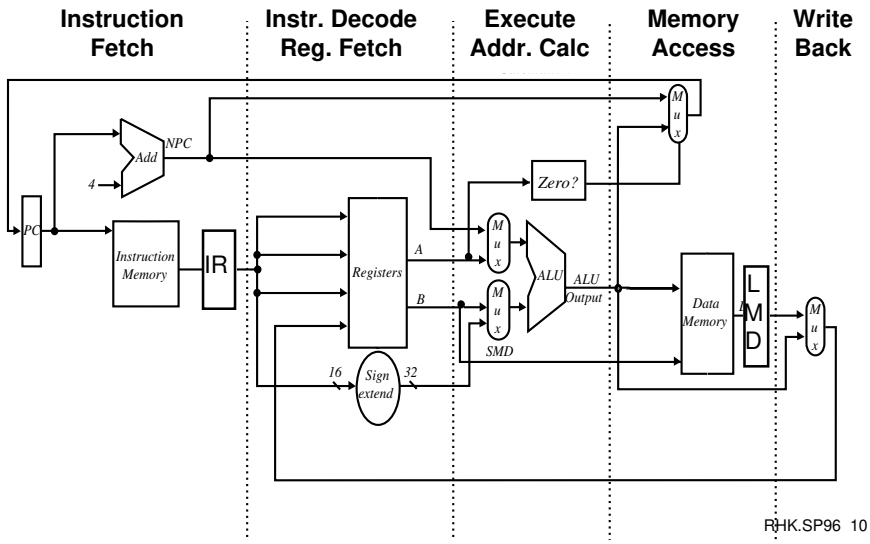
Pipelining Lessons



- 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

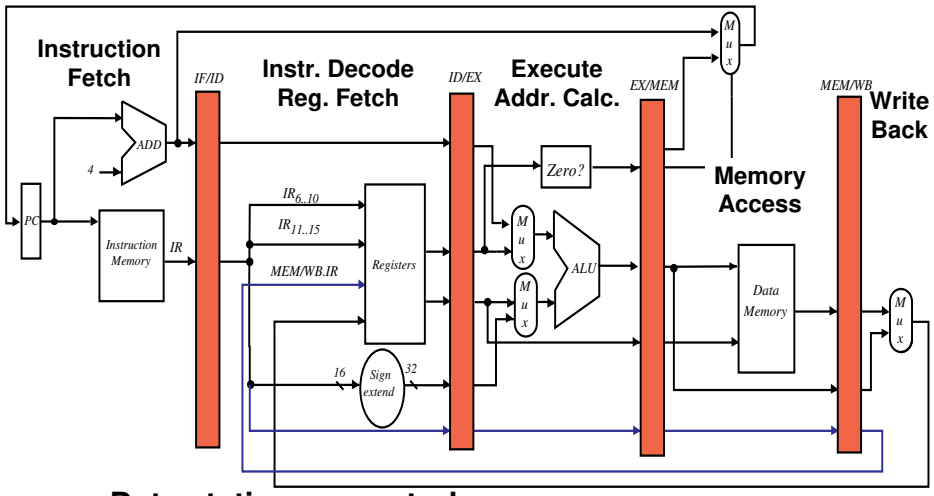
5 Steps of DLX Datapath

Figure 3.1, Page 130



Pipelined DLX Datapath

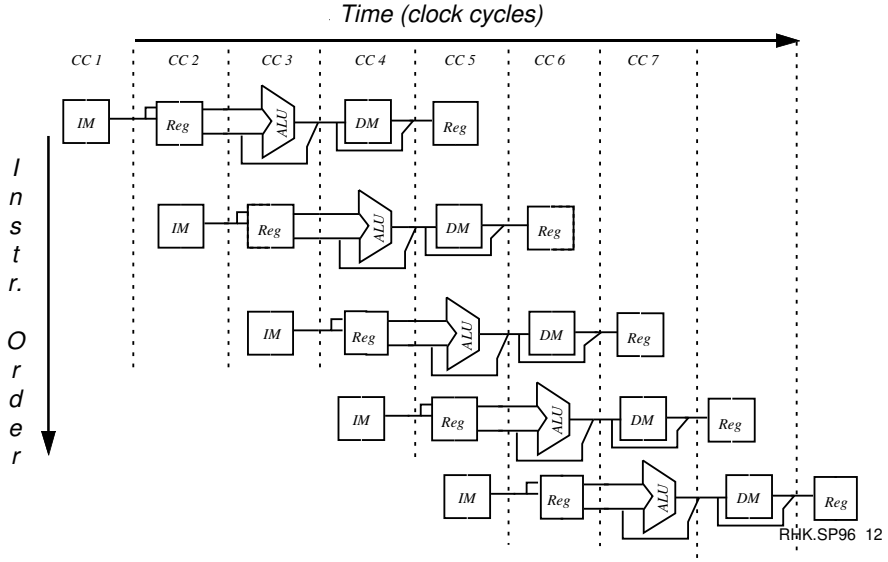
Figure 3.4, page 137



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

Visualizing Pipelining

Figure 3.3, Page 133

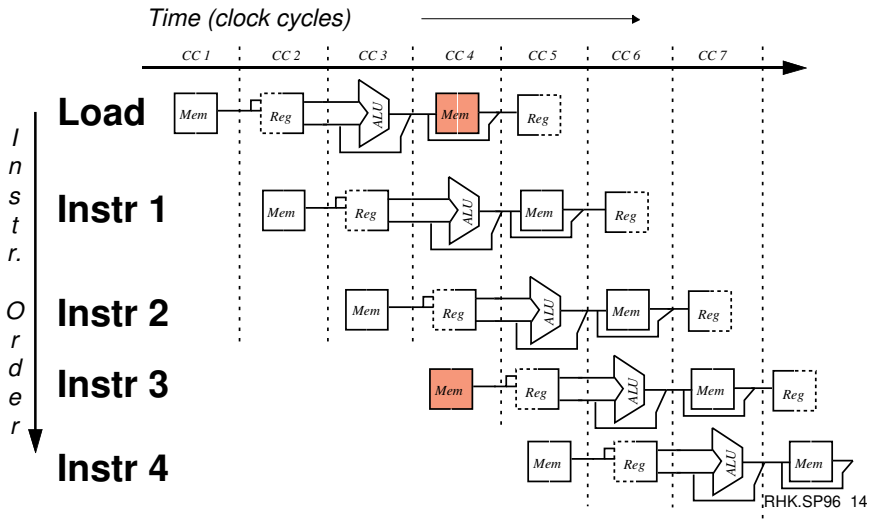


Its Not That Easy for Computers

- Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: HW cannot support this combination of instructions
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline
  - Control hazards: Pipelining of branches & other instructions that change the PC
- Common solution is to stall the pipeline until the hazard is resolved, inserting one or more "bubbles" in the pipeline

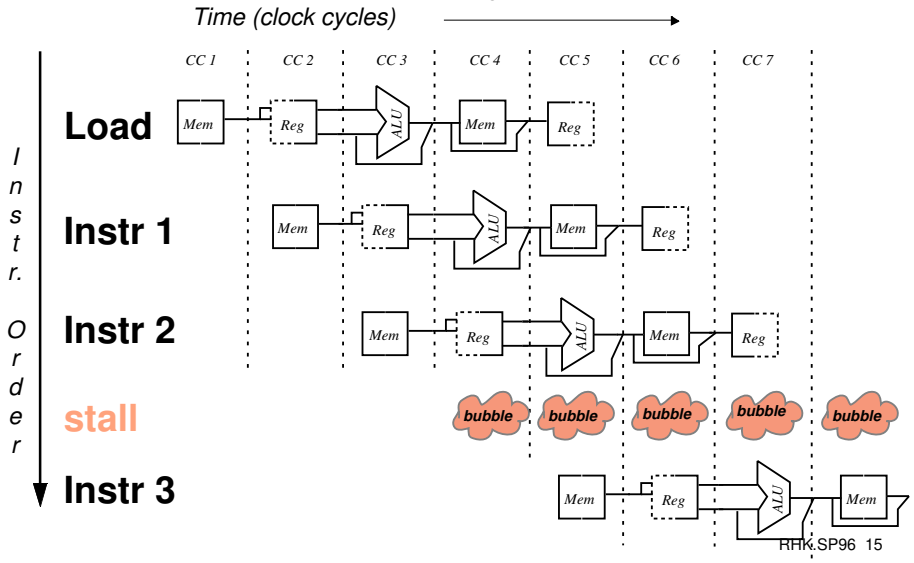
One Memory Port/Structural Hazards

Figure 3.6, Page 142



One Memory Port/Structural Hazards

Figure 3.7, Page 143



Speed Up Equation for Pipelining

$$\text{Speedup from pipelining} = \frac{\text{Ave Instr Time unpipelined}}{\text{Ave Instr Time pipelined}}$$

$$= \frac{\text{CPI}_{\text{unpipelined}} \times \text{Clock Cycle}_{\text{unpipelined}}}{\text{CPI}_{\text{pipelined}} \times \text{Clock Cycle}_{\text{pipelined}}}$$

$$= \frac{\text{CPI}_{\text{unpipelined}}}{\text{CPI}_{\text{pipelined}}} \times \frac{\text{Clock Cycle}_{\text{unpipelined}}}{\text{Clock Cycle}_{\text{pipelined}}}$$

$$\text{Ideal CPI} = \text{CPI}_{\text{unpipelined}} / \text{Pipeline depth}$$

$$\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{CPI}_{\text{pipelined}}} \times \frac{\text{Clock Cycle}_{\text{unpipelined}}}{\text{Clock Cycle}_{\text{pipelined}}}$$

Speed Up Equation for Pipelining

$$CPI_{pipelined} = \text{Ideal CPI} + \text{Pipeline stall clock cycles per instr}$$

$$\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \times \frac{\text{Clock Cycle}_{unpipelined}}{\text{Clock Cycle}_{pipelined}}$$

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Clock Cycle}_{unpipelined}}{\text{Clock Cycle}_{pipelined}}$$

Example: Dual-port vs. Single-port

- Machine A: Dual ported memory
- Machine B: Single ported memory, but its pipelined implementation has a 1.05 times faster clock rate
- Ideal CPI = 1 for both
- Loads are 40% of instructions executed

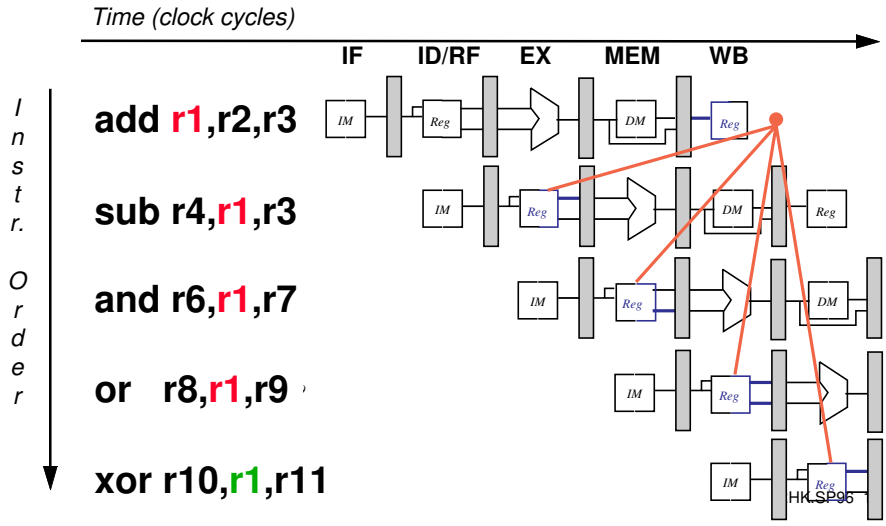
$$\text{SpeedUp}_A = \text{Pipeline Depth}/(1 + 0) \times (\text{clock}_{unpipe}/\text{clock}_{pipe})$$
$$= \text{Pipeline Depth}$$
$$\text{SpeedUp}_B = \text{Pipeline Depth}/(1 + 0.4 \times 1) \times (\text{clock}_{unpipe}/(\text{clock}_{unpipe} / 1.05)$$
$$= (\text{Pipeline Depth}/1.4) \times 1.05$$
$$= 0.75 \times \text{Pipeline Depth}$$

SpeedUp_A / SpeedUp_B = Pipeline Depth/(0.75 x Pipeline Depth) = 1.33

- Machine A is 1.33 times faster

Data Hazard on R1

Figure 3.9, page 147



Three Generic Data Hazards

$Instr_I$ followed be $Instr_J$

- **Read After Write (RAW)**
  $Instr_J$ tries to read operand before $Instr_I$ writes it

Three Generic Data Hazards

Instr$_I$ followed be Instr$_J$

- Write After Read (WAR)
  Instr$_J$ tries to write operand before Instr$_I$ reads it

- Can't happen in DLX 5 stage pipeline because:
  - All instructions take 5 stages,
  - Reads are always in stage 2, and
  - Writes are always in stage 5

Three Generic Data Hazards

Instr$_I$ followed be Instr$_J$

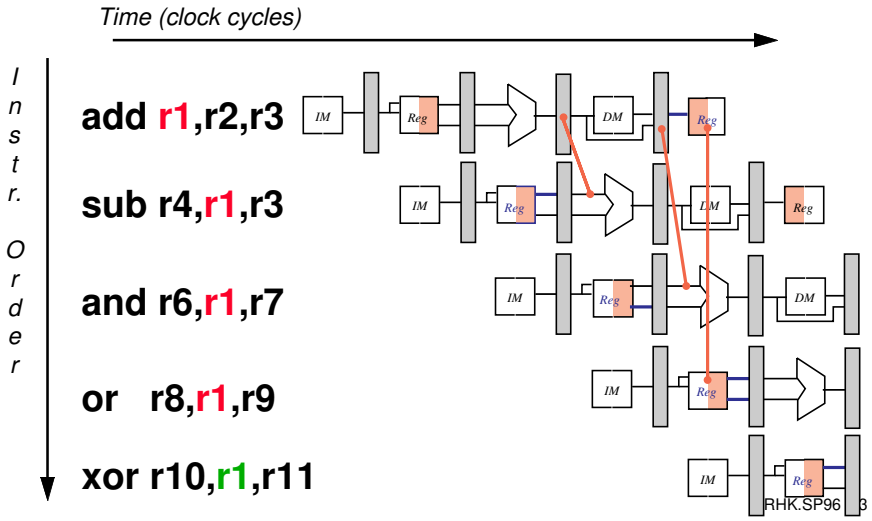
- Write After Write (WAW)
  Instr$_J$ tries to write operand before Instr$_I$ writes it
  - Leaves wrong result (Instr$_I$ not Instr$_J$)

- Can't happen in DLX 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Writes are always in stage 5

- Will see WAR and WAW in later more complicated pipes

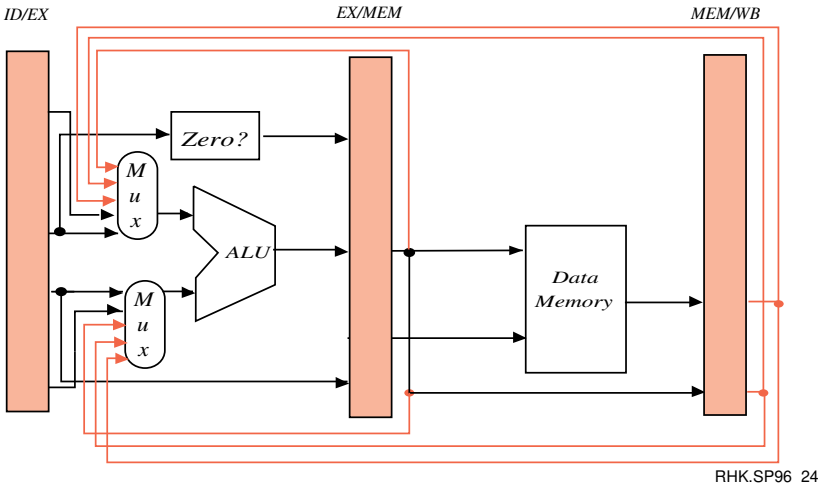
Forwarding to Avoid Data Hazard

Figure 3.10, Page 149



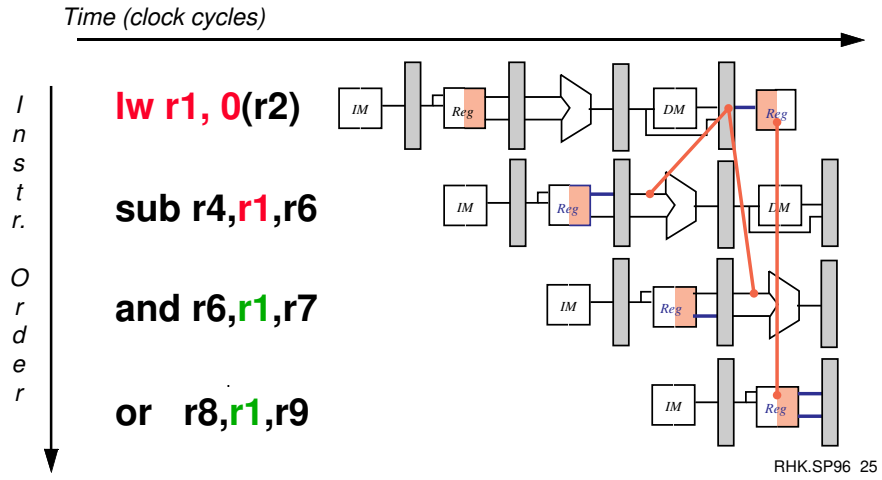
HW Change for Forwarding

Figure 3.20, Page 161



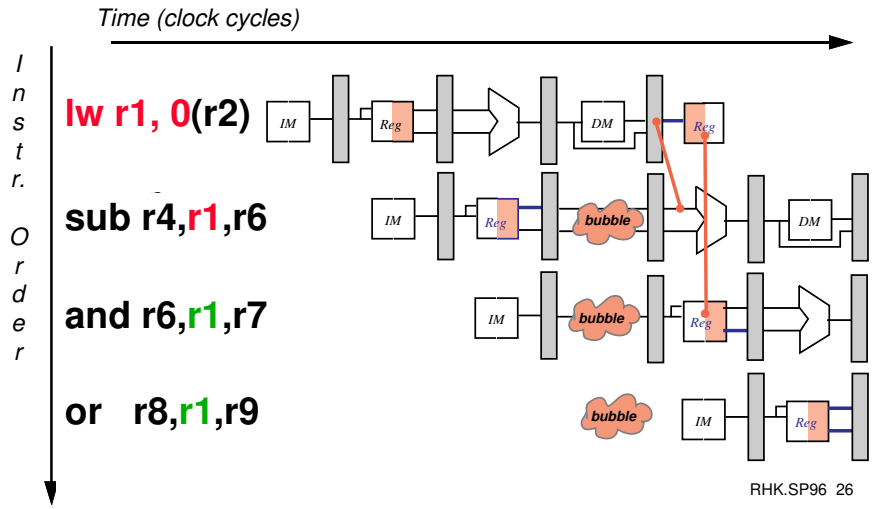
Data Hazard Even with Forwarding

Figure 3.12, Page 153



Data Hazard Even with Forwarding

Figure 3.13, Page 154



Software Scheduling to Avoid Load Hazards

Try producing fast code for

a = b + c;

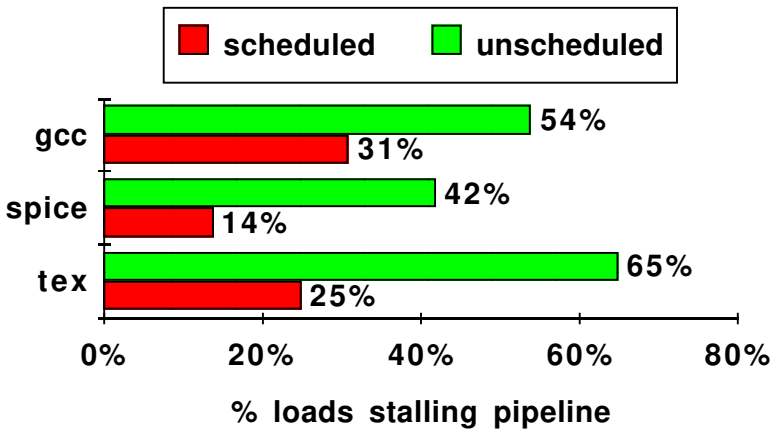
d = e – f;

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

RHK.SP96 27

Compiler Avoiding Load Stalls



RHK.SP96 28

Pipelining Summary

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

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

- **Hazards limit performance on computers:**
  - **Structural: need more HW resources**
  - **Data: need forwarding, compiler scheduling**
  - **Control: discuss next time**