

UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Advanced Computer Architecture
ECE 568

Part 3

Pipelining - Data Hazards

Israel Koren
Fall 2011

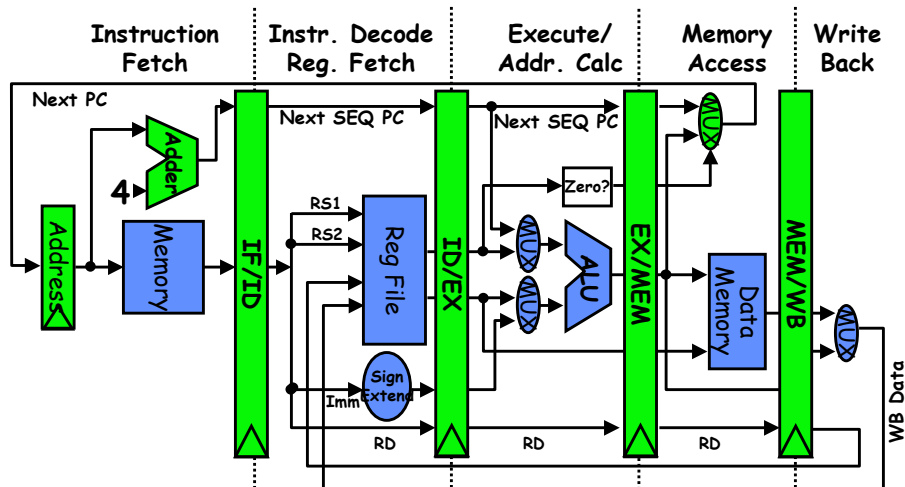
ECE568/Koren Part.3.1

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

MIPS Instruction pipeline

- ◆ Five-stage pipeline: IF, ID, EX, MEM, WB
- ◆ Focus on Data hazards



ECE568/Koren Part.3.2

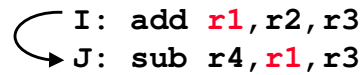
Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Three Generic Data Hazards - RAW

- ◆ **Read After Write (RAW)**

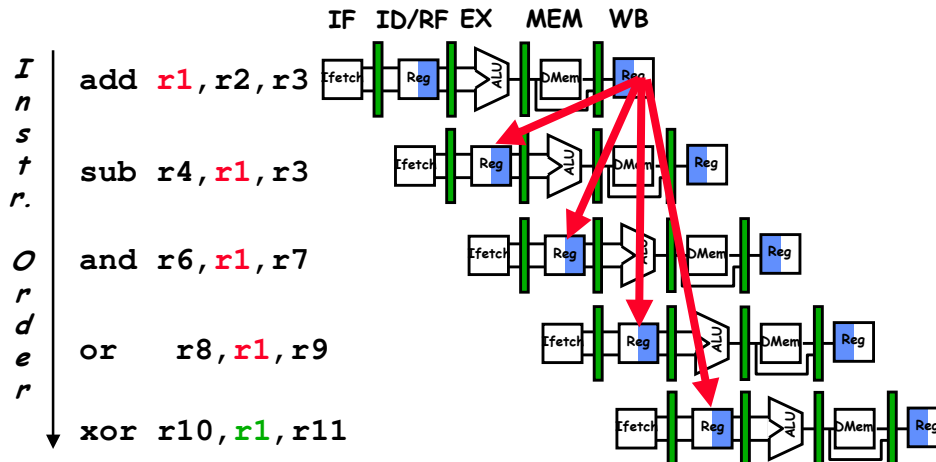
Instr_J tries to read operand before Instr_I writes it



- ◆ Caused by a **"Dependence"** (in compiler nomenclature). This hazard results from an actual need for communication

Data Hazard on R1

Time (clock cycles)



Three Generic Data Hazards - WAR

◆ Write After Read (WAR)

Instr_J writes operand before Instr_I reads it

↪ I: **div** r4, **r1**, r3
J: add **r1**, r2, r3
K: mul r6, **r1**, r7

◆ Called an “**anti-dependence**” by compiler writers.
This results from reuse of the name “**r1**”.

◆ Can not happen in MIPS 5-stage pipeline because:

- All instructions take 5 stages, and
- Reads are always in stage 2 (ID/RF), and
- Writes are always in stage 5 (WB)

Three Generic Data Hazards - WAW

◆ Write After Write (WAW)

Instr_J writes operand before Instr_I writes it.

↪ I: **div** **r1**, r4, r3
J: add **r1**, r2, r3
K: mul r6, **r1**, r7

◆ Called an “**output dependence**” by compiler writers
This also results from the reuse of name “**r1**”.

◆ Can not happen in MIPS 5-stage pipeline because:

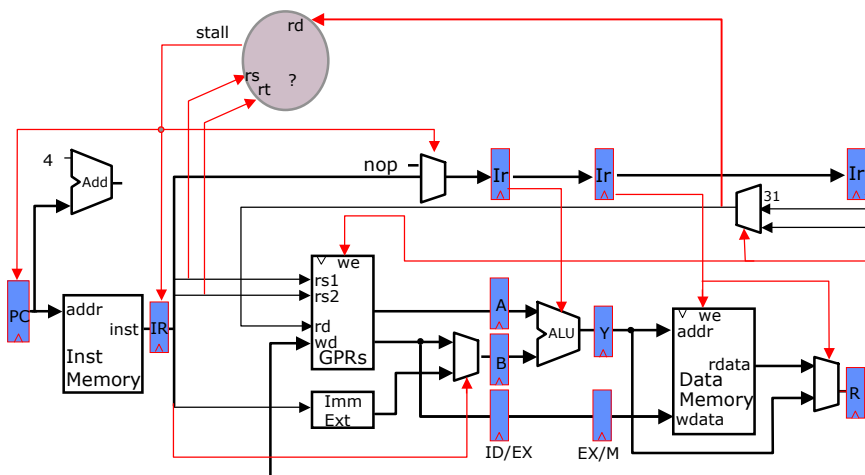
- All instructions take 5 stages, and
- Writes are always in stage 5

◆ Will see WAR and WAW in later more complicated pipes (out-of-order execution)

Dealing with data hazards

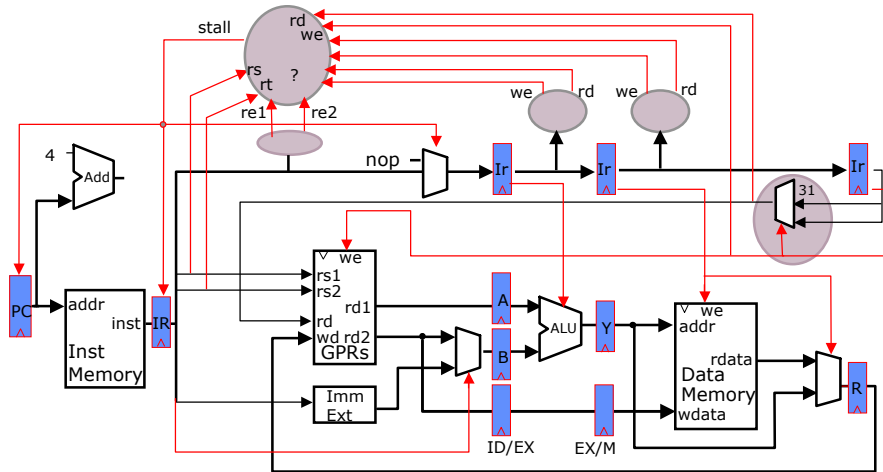
- ◆ 1. Introduce **stalls** (bubbles)
 - Capability to stall instructions and then release when it becomes safe to do so
- ◆ 2. Internal data forwarding
- ◆ 3. Reschedule instructions
 - Static rescheduling - by the compiler
 - Dynamic rescheduling - by the hardware
- ◆ A combination of the above

Interlock Control Logic



Compare the *source registers* of the instruction in the decode stage with the *destination register* of the previous instructions.

Interlock Control Logic



Should we always stall if the rs field matches some rd?
 not every instruction writes a register \Rightarrow we
 not every instruction reads a register \Rightarrow re

ignoring jumps & branches

ECE568/Koren Part.3.9

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Source & Destination Registers

R-type:

op	rs	rt	rd		func
----	----	----	----	--	------

I-type:

op	rs	rt	immediate 16
----	----	----	--------------

J-type:

op	immediate 26
----	--------------

source(s) destination

ALU	$rd \leftarrow (rs) \text{ func } (rt)$	rs, rt	rd
ALUi	$rt \leftarrow (rs) \text{ op } \text{imm}$	rs	rt
LW	$rt \leftarrow M[(rs) + \text{imm}]$	rs	rt
SW	$M[(rs) + \text{imm}] \leftarrow (rt)$	rs, rt	
BZ	$\text{cond}(rs)$		
	<i>true:</i> $PC \leftarrow (PC) + \text{imm}$	rs	
	<i>false:</i> $PC \leftarrow (PC) + 4$	rs	
J	$PC \leftarrow (PC) + \text{imm}$		
JAL	$r31 \leftarrow (PC), PC \leftarrow (PC) + \text{imm}$		31
JR	$PC \leftarrow (rs)$	rs	
JALR	$r31 \leftarrow (PC), PC \leftarrow (rs)$	rs	31

ECE568/Koren Part.3.10

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Load & Store Hazards

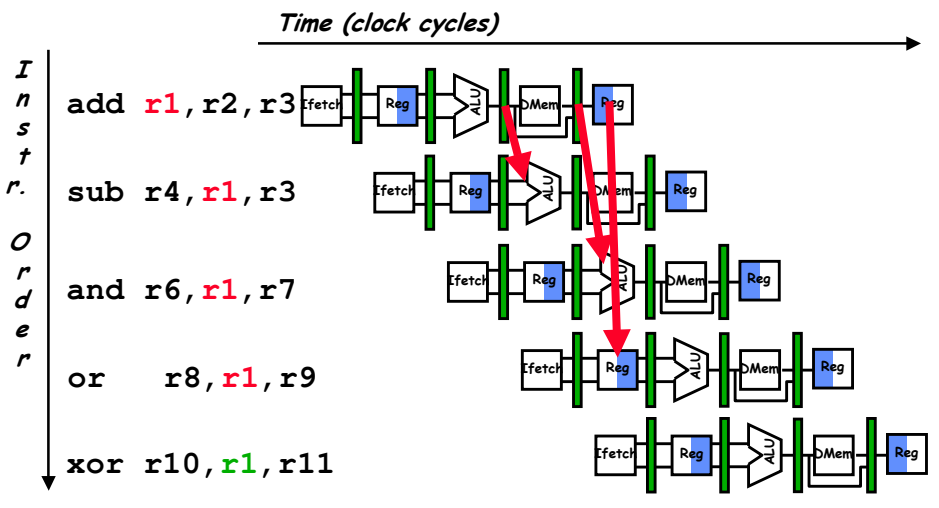
```

...
M[(r1)+7] ← (r2)
r4 ← M[(r3)+5]
...
    
```

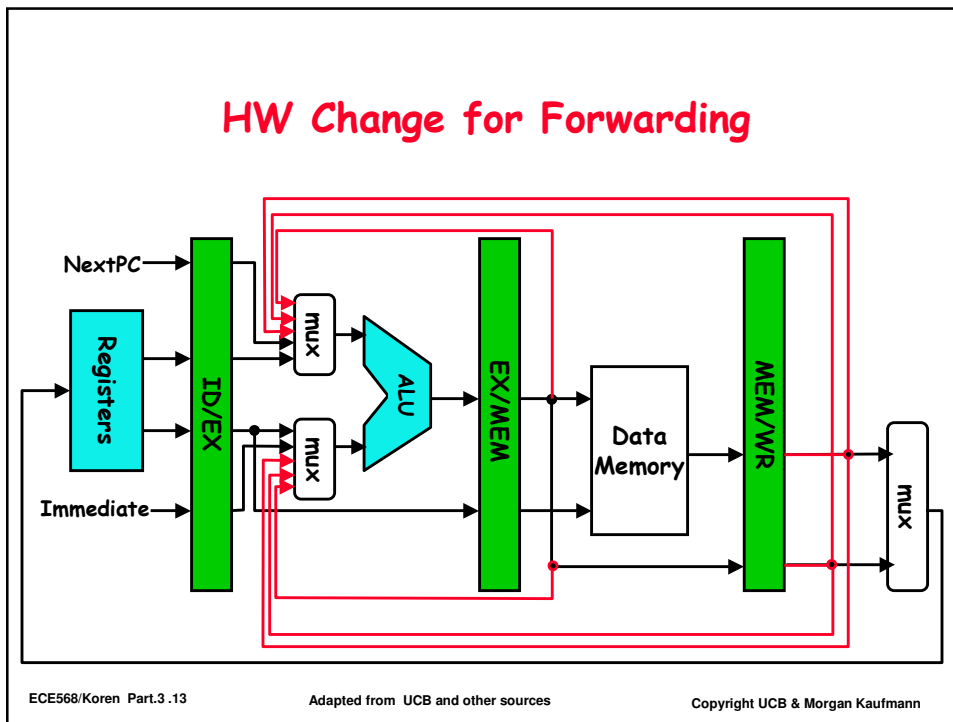
$(r1)+7 = (r3)+5 \Rightarrow \text{data hazard}$

However, the hazard is avoided because *our memory system completes writes in one cycle*

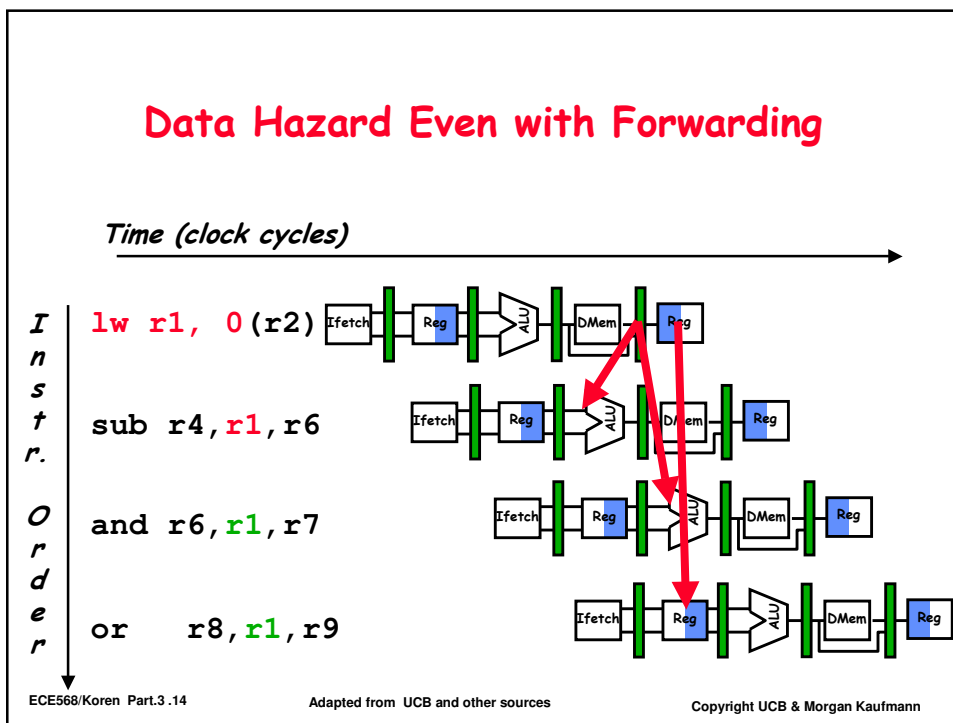
Forwarding to Avoid RAW Data Hazard

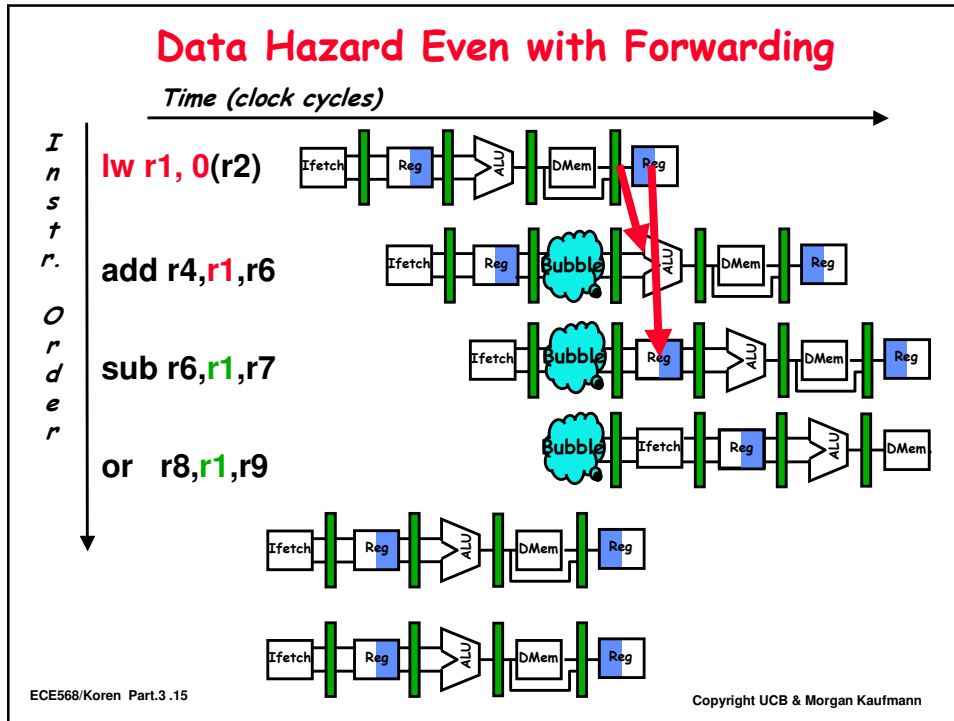


HW Change for Forwarding



Data Hazard Even with Forwarding

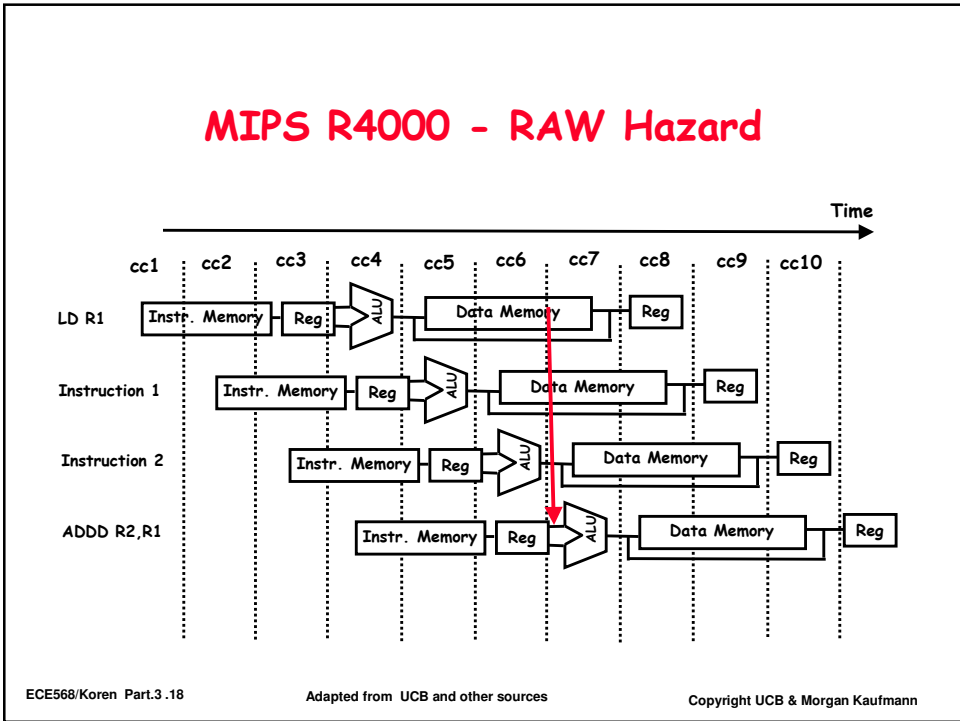
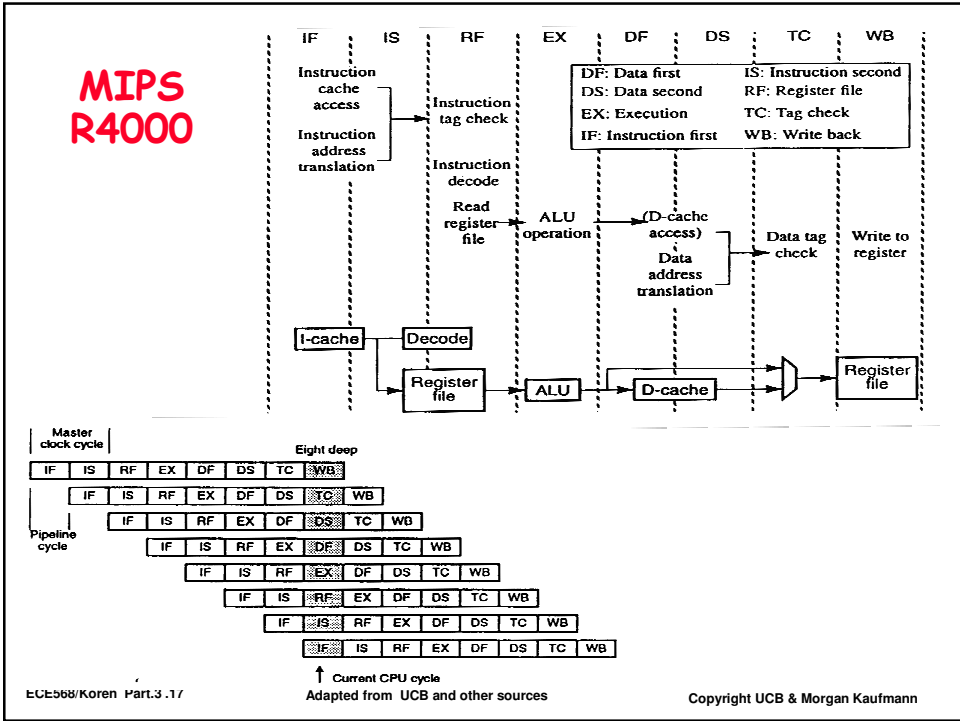




Detecting pipeline data hazards

Situation	Example code sequence	Action
No dependence	LW R1, 45 (R2) ADD R5, R6, R7 SUB R8, R6, R7 OR R9, R6, R7	No hazard possible because no dependence exists on R1 in the immediately following three instructions.
Dependence requiring stall	LW R1, 45 (R2) ADD R5, R1, R7 SUB R8, R6, R7 OR R9, R6, R7	Comparators detect the use of R1 in the ADD and stall the ADD (and SUB and OR) before the ADD begins EX.
Dependence overcome by forwarding	LW R1, 45 (R2) ADD R5, R6, R7 SUB R8, R1, R7 OR R9, R6, R7	Comparators detect use of R1 in SUB and forward result of load to ALU in time for SUB to begin EX.
Dependence with accesses in order	LW R1, 45 (R2) ADD R5, R6, R7 SUB R8, R6, R7 OR R9, R1, R7	No action required because the read of R1 by OR occurs in the second half of the ID phase, while the write of the loaded data occurred in the first half.

ECE568/Koren Part.3.16 Adapted from UCB and other sources Copyright UCB & Morgan Kaufmann



Pipelining with stalls

$$CPI_{\text{pipelined}} = CPI_{\text{ideal}} + \text{Average Stall cycles per Inst}$$

$$\text{Speedup} = \text{Pipeline depth} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}} \times \frac{CPI_{\text{ideal}}}{CPI_{\text{ideal}} + \text{Avg.stall CPI}}$$

For simple RISC pipeline, $CPI_{\text{ideal}} = 1$:

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

ECE568/Koren Part.3 .19

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Benefits of forwarding (LOAD only)

$$\text{Speedup} = \frac{\text{Pipeline_depth}}{1 + \text{Stall_CPI}} \times \frac{T_{\text{unpipelined}}}{T_{\text{pipelined}}}$$

Assume: 30% Load instr., 50% of the time the following instr. needs the data, 30% of the time only the 2nd instr. needs the data; w/o forwarding the 1st instr. requires 2 stall cycles, the 2nd requires 1 stall cycle

$$\text{Speedup}_{\text{w/forwarding}} = \frac{5}{1 + 1 \times .3 \times .5} \times \frac{1}{1.05} = 4.14$$

$$\text{Speedup}_{\text{w/o forwarding}} = \frac{5}{1 + .3(2)} \times \frac{1}{1.05} = 3.43$$

$$\text{Speedup due to forwarding} = 4.14/3.43 = 1.2$$

ECE568/Koren Part.3 .20

Copyright 2011 Koren UMass

Benefits of forwarding (All instructions)

Assume: Remaining 70% Not-Load instr., 20% of the time the instr. following Load needs the data, 10% of the time only the 2nd instr. needs the data; w/o forwarding the 1st instr. requires 2 stall cycles, the 2nd requires 1 stall cycle.

$$\text{Speedup w/o forwarding} = \frac{5}{1 + .39} \times \frac{1}{1.05} = 2.63$$

$$\text{Speedup due to forwarding} = 4.14 / 2.63 = 1.57$$

Software Scheduling to Avoid Load Hazards

Try producing fast code for

$a = b + c;$

$d = e - f;$

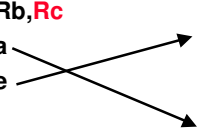
assuming $a, b, c, d, e,$ and f in memory.

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:

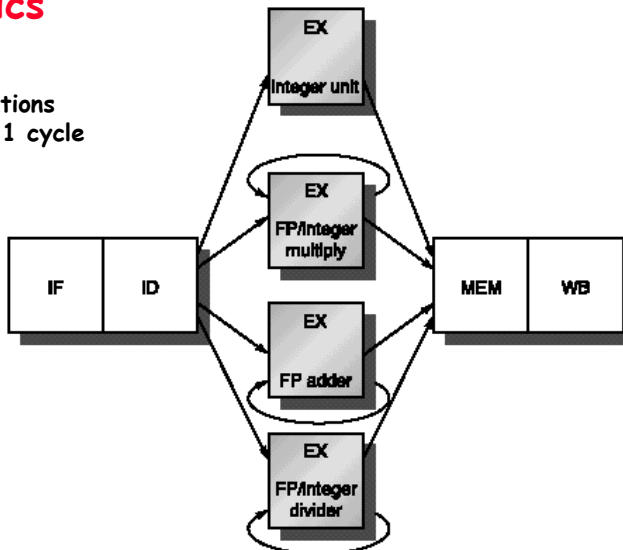
```
LW   Rb,b
LW   Rc,c
LW   Re,e
ADD  Ra,Rb,Rc
LW   Rf,f
SW   a,Ra
SUB  Rd,Re,Rf
SW   d,Rd
```



Multi-cycle operation - Basics

Floating-point operations
can not complete in 1 cycle

Assume:
 OP. #cycles
 Int.op 1
 FP.Add 4
 FP.Mult 7
 FP.Div 24

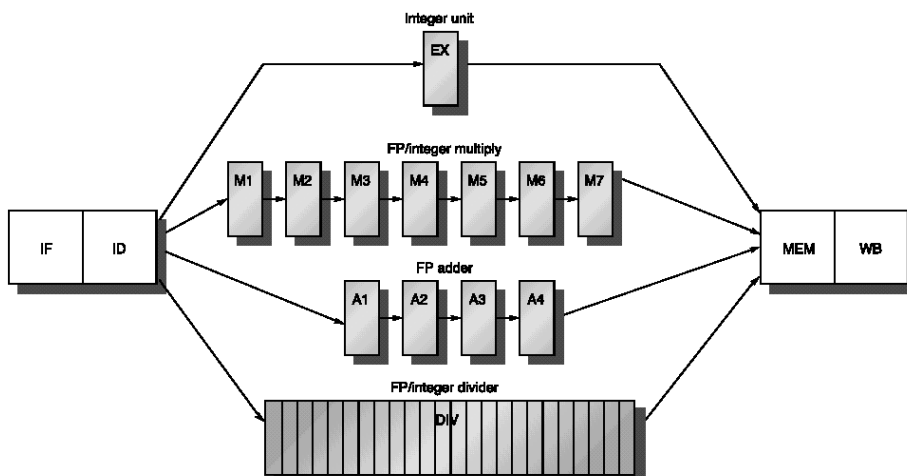


ECE568/Koren Part.3 .23

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Multi-cycle operation - w/o & w/pipelining



ECE568/Koren Part.3 .24

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Multi-cycle - Hazards

Instruction	Clock cycle number																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
L.D F4,0(R2)	IF	ID	EX	MEM	WB												
MUL.D F0,F4,F6		IF	ID	stall	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
ADD.D F2,F0,F8			IF	stall	ID	stall	stall	stall	stall	stall	stall	A1	A2	A3	A4	MEM	
S.D F2,0(R2)					IF	stall	stall	stall	stall	stall	stall	ID	EX	stall	stall	stall	MEM

Instruction	Clock cycle number											
	1	2	3	4	5	6	7	8	9	10	11	
MUL.D F0,F4,F6		IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
...			IF	ID	EX	MEM	WB					
...				IF	ID	EX	MEM	WB				
ADD.D F2,F4,F6					IF	ID	A1	A2	A3	A4	MEM	WB
...						IF	ID	EX	MEM	WB		
...							IF	ID	EX	MEM	WB	
L.D F2,0(R2)								IF	ID	EX	MEM	WB

ECE568/Koren Part.3 .25

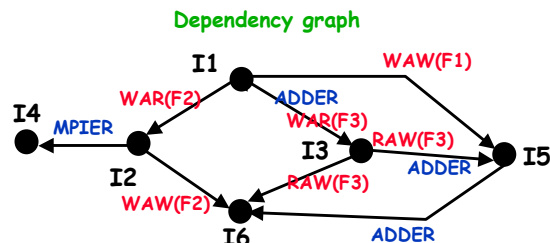
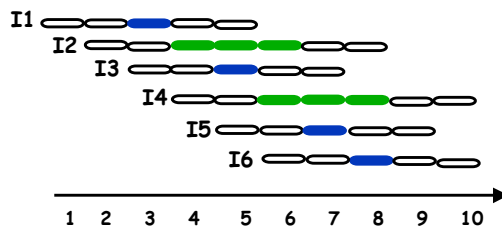
Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Hazards in multi-cycle operation

FP adder & unpipelined multiplier taking 1 & 3 cycles, respectively

- I1: F1 ← F2 + F3
- I2: F2 ← F4 × F5
- I3: F3 ← F3 + F4
- I4: F6 ← F6 × F6
- I5: F1 ← F3 + F5
- I6: F2 ← F3 + F4



$T(seq_ex)=?$

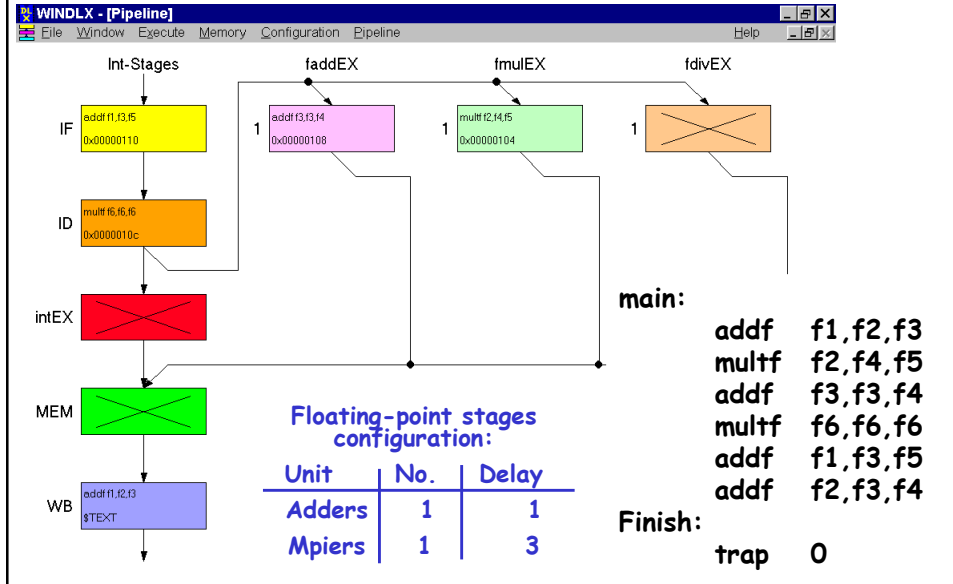
$T(Ideal)= 10$

Use WinDLX (flt1.s)

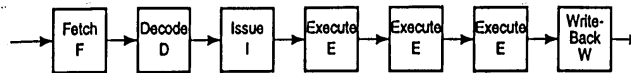
ECE568/Koren Part.3 .26

Copyright 2011 Koren UMass

WINDLX software



Software Static Scheduling - Multi-cycle



$X = Y + Z \ \& \ A = B * C$

In-order instruction issuing	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2			
$R_1 \leftarrow (Y)$	F	D	I	E	E	E	W																		
$R_2 \leftarrow (Z)$		F	D	I	E	E	E	W																	
$R_3 \leftarrow (R_1) + (R_2)$			F	D	*	*	I	E	E	E	W														
$X \leftarrow (R_3)$				F	*	*	D	*	*	I	E	E	E	W											
$R_4 \leftarrow (B)$						F	*	*	D	I	E	E	E	W											
$R_5 \leftarrow (C)$									F	D	I	E	E	E	W										
$R_6 \leftarrow (R_4) \times (R_5)$										F	D	*	*	I	E	E	E	W							
$A \leftarrow (R_6)$											F	*	*	D	*	*	I	E	E	E	W				

Static Scheduling - Optimizing Compiler

Reordered instruction issuing	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
$R_1 \leftarrow (Y)$	F	D	I	E	E	E	W									
$R_2 \leftarrow (Z)$		F	D	I	E	E	E	W								
$R_4 \leftarrow (B)$			F	D	I	E	E	E	W							
$R_5 \leftarrow (C)$				F	D	I	E	E	E	W						
$R_3 \leftarrow (R_1) + (R_2)$					F	D	I	E	E	E	W					
$R_6 \leftarrow (R_4) \times (R_5)$						F	D	*	I	E	E	E	W			
$X \leftarrow (R_3)$							F	*	D	I	E	E	E	W		
$A \leftarrow (R_6)$								F	D	*	I	E	E	E	W	

- 16 vs. 22 cycles
- Cheap to implement
- May need NOPs

Source: J. Smith, IEEE
Computer, July 1989.