

CMSC 411
Computer Systems Architecture
Lecture 6
Basic Pipelining 3
Long Instructions & MIPS Case Study

Complications with long instructions

- So far, all MIPS instructions take 5 cycles
- But haven't talked yet about the floating point instructions
- Take it on faith that floating point instructions are inherently slower than integer arithmetic instructions
 - doubters may consult Appendix H in H&P online

CMSC 411 - 6 (from Patterson)

2

How slow is slow?

- Some typical times:
 - **latency** is the number of cycles between an instruction that produces a result and one that uses it
 - **initiation interval** is the number of cycles between two instructions of the same kind (for example, two ADD.Fs)

Instruction	Latency	Initiation
ALU uses	0	1
Load/store	1	1
ADD.F,SUB.F	3	1
DIV.F	24	25

CMSC 411 - 6 (from Patterson)

3

Examples

- If we have a sequence of integer instructions
 - ADD
 - SUB
 - AND
 - OR
 - SLLI
- then there are no delays in the pipeline, because
 - initiation=1 means can start one of these instructions every cycle
 - latency=0 means that results from one instruction will be available when the next instruction needs them

CMSC 411 - 6 (from Patterson)

4

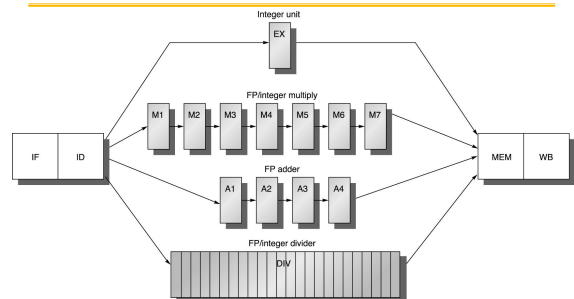
Examples (cont.)

- If we have a sequence of floating point instructions
 - ADD.F
 - SUB.F
- Then initiation=1 means that can start SUB.F one cycle behind ADD.F
- But latency=3 means that this will work right *only if* SUB.F doesn't need ADD.F's results
- If it does need the results, then need 3 instructions in between ADD.F and SUB.F to prevent bubbles in the pipeline

CMSC 411 - 6 (from Patterson)

5

Functional units - Fig. A.31



© 2003 Elsevier Science (USA). All rights reserved.

CMSC 411 - 6 (from Patterson)

6

Examples (cont.) - Fig. A.32

MUL.D	IF	ID	<i>M1</i>	M2	M3	M4	M5	M6	<i>M7</i>	MEM	WB
ADD.D		IF	ID	<i>A1</i>	A2	A3	<i>A4</i>	MEM	WB		
L.D			IF	ID	<i>EX</i>	MEM	WB				
S.D				IF	ID	<i>EX</i>	<i>MEM</i>	WB			

Italics shows where data is needed

blue where a result is available

CMSC 411 - 6 (from Patterson)

7

CMSC 411 - 6 (from Patterson)

8

Hazards caused by long instructions

- The floating point adder and multiplier are pipelined, but the divider is not - that is why the initiation interval for divide is 25
 - A program will run very slowly if it does too many of these!
- It will also run slowly if the results of the divide are needed too soon

FP stalls from RAW hazards – Fig. A.33

Inst.	1	2	3	4	5	6	7	8	9
L.D F4,0(R2)	IF	ID	EX	MEM	WB				
MUL.D F0,F4,F6		IF	ID	stall	M1	M2	M3	M4	M5
ADD.D F2,F0,F8			IF	stall	ID	stall	stall	stall	stall
S.D F2,0(R2)				stall	IF	stall	stall	stall	stall

Inst.	10	11	12	13	14	15	16	17
L.D								
MUL.D	M6	M7	MEM	WB				
ADD.D	stall	stall	A1	A2	A3	A4	MEM	WB
S.D	stall	stall	ID	EX	stall	stall	stall	MEM

CMSC 411 - 6 (from Patterson)

9

CMSC 411 - 6 (from Patterson)

10

Long instructions (cont.)

- It is possible that two instructions enter the WB stage at the same time

ADD.D	IF	ID	A1	A2	A3	A4	MEM	<i>WB</i>
LD		IF	ID	ALU	MEM	WB		
DADD			IF	ID	ALU	MEM	WB	
DADD				IF	ID	ALU	MEM	<i>WB</i>

- A structural hazard

Long instructions (cont.)

- Instructions can finish in the wrong order
- This can cause WAW hazards
- This violation of WB ordering **defeats** the previous strategy for **precise exception handling**

– problem is **out-of-order completion**

- stop fetching
- turn off writes
- let pipeline drain
- handle

DIV.D F0, F2, F4

What happens if sub faults?

ADD R1, R1, R2

And then div?

SUB.D F10, F12, F14

What about R1?

CMSC 411 - 6 (from Patterson)

11

CMSC 411 - 6 (from Patterson)

12

WAW structural hazard

	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	<i>WB</i>
...		IF	ID	EX	MEM	WB					
...			IF	ID	EX	MEM	WB				
ADD.D F2,F4,F6				IF	ID	A1	A2	A3	A4	MEM	<i>WB</i>
...					IF	ID	EX	MEM	WB		
...						IF	ID	EX	MEM	WB	
L.D F2,0(R2)							IF	ID	EX	MEM	<i>WB</i>

CMSC 411 - 6 (from Patterson)

12

Possible fixes

- Give up and just do **imprecise exception handling**
 - tempting, but very annoying to users
- **Delay WB** until all previous instructions complete
 - since so many instructions can be active, this is expensive - requires a lot of supporting hardware
- Write, to memory, a **history file** of register and memory changes so can undo instructions if necessary
 - or keep a **future file** of computed results that are waiting for MEM or WB

CMSC 411 - 6 (from Patterson)

13

Possible fixes (cont.)

- Let the exception handler finish the instructions in the pipeline and then restart the pipe at the next instruction
- Have the floating point units diagnose exceptions in their first or second stages, so can handle them by methods that work well for handling integer exceptions

CMSC 411 - 6 (from Patterson)

14

How to detect hazards in ID

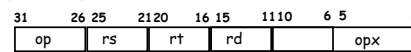
- Early detection would prevent trouble
- Check for structural hazards:
 - will the divide unit clear in time?
 - will WB be possible when we need it?
- Check for RAW data hazards:
 - will all source registers be available when needed?
- Check for WAW data hazards:
 - Is the destination register for any ADD.D, multiply or divide instruction the same register as the destination for this instruction?
- If anything dangerous could happen, delay the execute cycle so no conflict occurs

CMSC 411 - 6 (from Patterson)

15

Review – MIPS Instruction Format

Register-Register

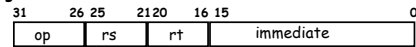


$rd \leftarrow rs \text{ OP } rt$

Examples

ADD R1,R2,R3 // $R1 \leftarrow R2+R3$
 MUL R1,R2,R3 // $R1 \leftarrow R2 \cdot R3$

Register-Immediate



$rt \leftarrow rs \text{ OP } \text{immed}$

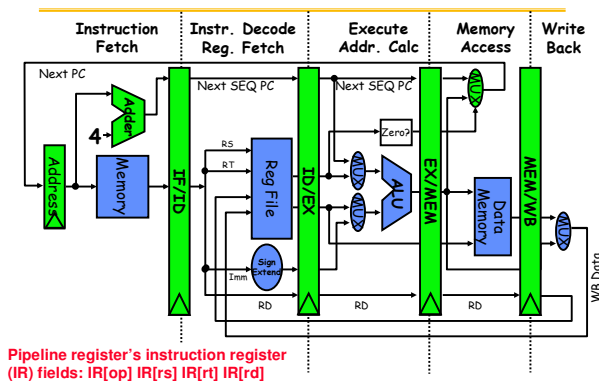
Examples

ADDI R1,R2,8 // $R1 \leftarrow R2+8$
 LW R1,4(R2) // $R1 \leftarrow \text{MEM}(R2+4)$
 SW R1,4(R2) // $\text{MEM}(R2+4) \leftarrow R1$... $\leftarrow rs \text{ OP } rt$

CMSC 411 - 3 (from Patterson)

16

Review– Pipeline Registers

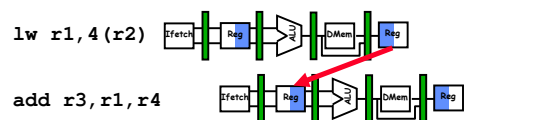


CMSC 411 - 4 (from Patterson)

17

Data Hazard Without Forwarding

Time (clock cycles)

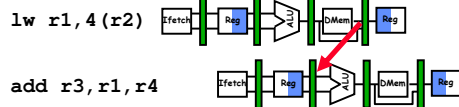


CMSC 411 - 5 (from Patterson)

18

Data Hazard With Forwarding

Time (clock cycles)

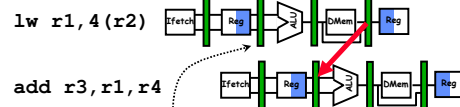


CMSC 411 - 5 (from Patterson)

19

RAW Data Hazard Detection (LW/ADD)

Time (clock cycles)



At time = cycle 2

IF/ID.IR[op] = ADD
IF/ID.IR[rs] = r1
IF/ID.IR[rt] = r4
IF/ID.IR[rd] = r3

ID/EX.IR[op] = LW
ID/EX.IR[rs] = r2
ID/EX.IR[rt] = r1

So insert pipeline stall

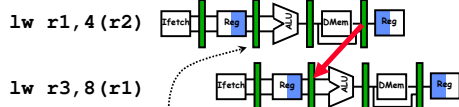
due to RAW hazard if
ID/EX.IR[op] = LW
IF/ID.IR[op] = ADD
ID/EX.IR[rt] = IF/ID.IR[rs]
OR
ID/EX.IR[rt] = IF/ID.IR[rd]

CMSC 411 - 5 (from Patterson)

20

RAW Data Hazard Detection (LW/LW)

Time (clock cycles)



At time = cycle 2

IF/ID.IR[op] = LW
IF/ID.IR[rs] = r1
IF/ID.IR[rt] = r3

ID/EX.IR[op] = LW
ID/EX.IR[rs] = r2
ID/EX.IR[rt] = r1

So insert pipeline stall

due to RAW hazard if
ID/EX.IR[op] = LW
IF/ID.IR[op] = LW
ID/EX.IR[rt] = IF/ID.IR[rs]

CMSC 411 - 5 (from Patterson)

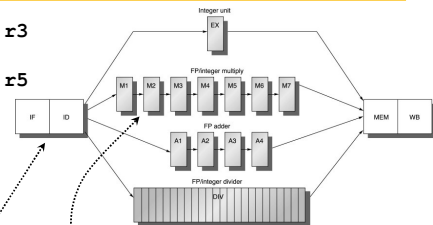
21

WAW Data Hazard Detection (MUL.D/ADD.D)

MUL.D r1, r2, r3

...

ADD.D r1, r4, r5



If at time x

IF/ID.IR[op] = ADD.D
IF/ID.IR[rd] = r1

M1/M2.IR[op] = MUL.D
M1/M2.IR[rd] = r1

So insert pipeline stall due to WAW hazard if

Then ADD.D would write to r1 at time x+6, while MUL.D would write r1 at time x=7, reversing order of writes

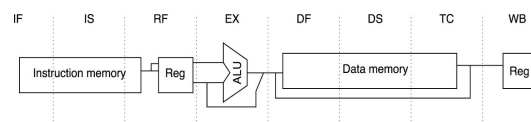
M1/M2.IR[op] = MUL.D
IF/ID.IR[op] = ADD.D
M1/M2.IR[rd] = IF/ID.IR[rd]

CMSC 411 - 5 (from Patterson)

22

A case study: MIPS R4000

- MIPS64 architecture, with deeper 8 stage pipeline
 - to get higher clock rates
 - extra stages come from memory accesses
 - techniques called superpipelining



© 2003 Elsevier Science (USA). All rights reserved.

CMSC 411 - 7 (from Patterson)

23

MIPS R4000 pipeline stages

- IF – 1st half instruction fetch
 - PC selection and start instruction cache access
- IS – 2nd half instruction fetch
 - complete instruction cache access
- RF – instruction decode, register fetch, hazard checking, instruction cache hit detection
- EX – execution
 - includes effective address computation, ALU operation, branch target computation and condition evaluation

CMSC 411 - 7 (from Patterson)

24

MIPS R4000 pipeline (cont.)

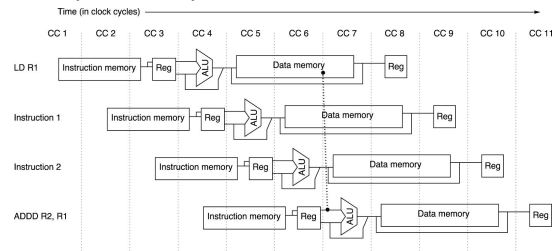
- **DF** – 1st half data fetch
 - 1st half of data cache access
- **DS** – 2nd half data fetch
 - complete data cache access
- **TC** – tag check
 - determine whether data cache access hit
- **WB** – write back for loads and ALU operations

CMSC 411 - 7 (from Patterson)

25

MIPS R4000 pipeline (cont.)

A 2 cycle load delay



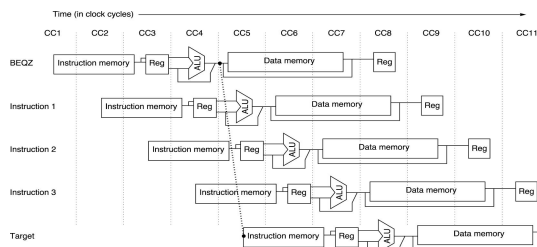
Might need to restart ADD's ALU

CMSC 411 - 7 (from Patterson)

26

MIPS R4000 pipeline (cont.)

A 3 cycle branch delay – 1 delay slot + 2 cycle stall for taken branch (untaken just delay slot)



© 2003 Elsevier Science (USA). All rights reserved.

CMSC 411 - 7 (from Patterson)

27

Forwarding

- Deeper pipeline increases number of levels of forwarding for ALU operations
 - 4 possible sources for an ALU bypass
 - » EX/DF
 - » DF/DS
 - » DS/TC
 - » TC/WB

CMSC 411 - 7 (from Patterson)

28

Floating point pipeline

- 3 functional units
 - divider, multiplier, adder
- Double precision FP ops take
 - from 2 (negate) up to 112 cycles (square root)
- Effectively 8 stages, combined in different orders for various FP operations
 - one copy of each stage, and some instructions use a stage zero or more times, and in different orders
- Overall, rather complicated ...
 - see H&P for more details

CMSC 411 - 7 (from Patterson)

29

R4000 pipeline performance

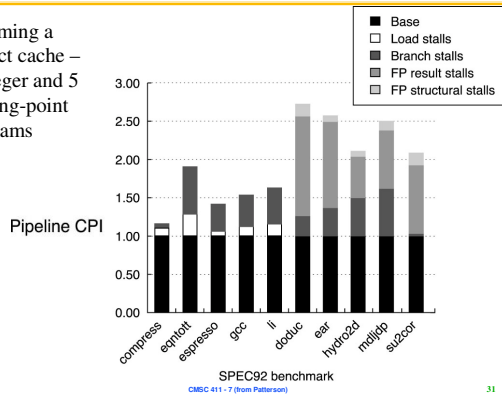
- 4 major causes of pipeline stalls
 - load stalls – from using load result 1 or 2 cycles after load
 - branch stalls – 2 cycles on every taken branch, or empty branch delay slot
 - FP result stalls – RAW hazards for an FP operand
 - FP structural stalls – from conflicts for functional units in FP pipeline

CMSC 411 - 7 (from Patterson)

30

SPEC92 benchmarks

Assuming a perfect cache – 5 integer and 5 floating-point programs



31

Pitfalls

- Unexpected hazards do occur ...
 - for example, when a branch is taken before a previous instruction finishes
- Extensive pipelining can slow a machine down, or lead to worse cost-performance
 - more complex hardware can cause a longer clock cycle, killing the benefits of more pipelining

CMSC 411 - 7 (from Patterson)

32

Pitfalls (cont.)

- A poor compiler can make a good machine look bad
 - compiler writers need to understand the architecture in order to
 - » optimize efficiently and
 - » avoid hazards
 - better to eliminate useless instructions, than make them run faster

CMSC 411 - 7 (from Patterson)

33