CMSC 411
Computer Systems Architecture
Lecture 7
Basic Pipelining (cont.)

Alan Sussman
als@cs.umd.edu

Administrivia

- HW #2, on pipelining, out by tomorrow
  - due a week from Tuesday
- First project, also on pipelining, out early next week.
- We'll start unit on memory hierarchy next week, probably on Tuesday
  - start reading Appendix B

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

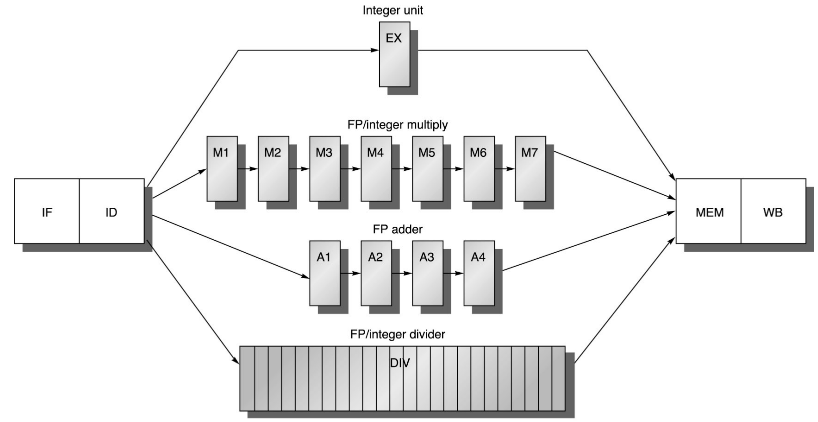
Examples

- If have a string of 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, and latency=0 means that results from one instruction will be available when the next instruction needs them.

Examples (cont.)

- **Suppose have a string of 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 two instructions in between** ADD.F **and** SUB.F **to prevent bubbles in the pipeline**

Functional units



Examples (cont.)

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

Italics shows where data is needed, blue where a result is available

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

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

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	WB
LD		IF	ID	ALU	MEM	WB		
DADD			IF	ID	ALU	MEM	WB	
DADD				IF	ID	ALU	MEM	WB

- 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*

DIV.D	F0, F2, F4
ADD	R1, R1, R2
SUB.D	F10, F12, F14

What happens if sub faults?

And then div?

What about R1?

1) *stop fetching*
2) *turn off writes*
3) *let pipeline drain*
4) *handle*

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	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

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**

Possible fixes (cont.)

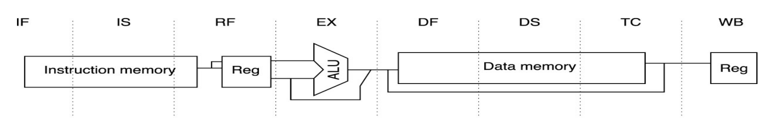
- **Let the** exception handler **finish the instructions in the pipeline and then restart the pipe at the next instruction**
- **Have the floating point units** diagnose exceptions in their first or second stages**, so can handle them by methods that work well for handling integer exceptions**

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**

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***



MIPS R4000 pipeline stages

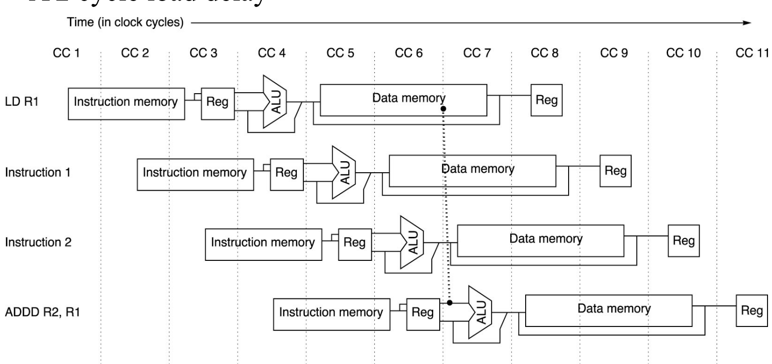
- IF – **1st half instruction fetch**
  - **PC selection and start instruction cache access**
- IS – **2nd half instruction fetch**
  - **complete instruction cache access**
- RF – **instruction decode, register fetch, hazard checking, instruction cache hit detection**
- EX – **execution**
  - **includes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

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**

MIPS R4000 pipeline (cont.)

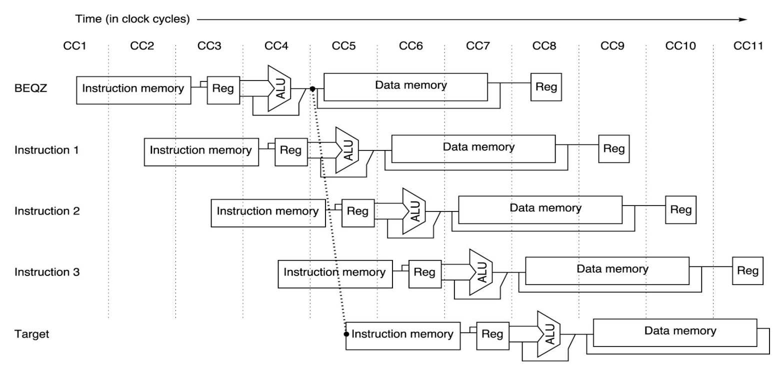
A 2 cycle load delay



Might need to restart ADDD's ALU

MIPS R4000 pipeline (cont.)

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



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**

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**

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**