CMSC 411
Computer Systems Architecture
Lecture 6
Basic Pipelining (cont.)

Alan Sussman
als@cs.umd.edu

Administrivia

- **Answers to HW #1 posted**
  - **password protected, with instructions sent via email**
- **Homework problems for Unit 3 posted, due Thursday, Feb. 26**

Review: Exceptions

- **Ideally, pipeline can be interrupted so that instructions before the fault complete. Then want to restart execution just after the faulting instruction -** precise exception handling

When do MIPS exceptions occur?

- **IF**
  - **page fault on instruction fetch**
  - **misaligned memory access**
  - **memory protection violation**
- **ID**
  - **undefined or illegal opcode**
- **EX**
  - **arithmetic exception**
- **MEM**
  - **page fault on data fetch/store**
  - **misaligned memory access**
  - **memory protection violation**
- **WB: None!**

Examples of exception handling

LD	IF	ID	EX	MEM	WB	
ADD		IF	ID	EX	MEM	WB

- Handle the MEM fault first, then restart

LD	IF	ID	EX	MEM	WB	
ADD		IF	ID	EX	MEM	WB

- IF fault occurs first, even though LD will fault later
- But for precise exceptions, must handle LD fault first

How is this done?

- Answer: **Don't handle exceptions until the WB stage**
  - **each instruction has an associated status vector that keeps track of faults**
  - **any bit set in the status vector turns off register writes and memory writes**
  - **in WB stage, the status vector is checked and any fault is handled**
  - **So, since instructions reach WB in proper order, faults for earlier instructions are handled *before* faults for later instructions**
    - » **Unfortunately, will need to violate this later (for instructions that don't reach WB in proper order)**

Commitment

- **When an instruction is guaranteed to complete, it is committed**
- **Life is easier if no instruction changes the permanent machine state before it is committed**
- **In MIPS, commitment occurs at the end of the MEM stage - that's why register update occurs in the stage after that**
- **Some machines muddy the state before commitment, and the exception handler must do its best to restore the state that existed before the instruction started**

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**

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

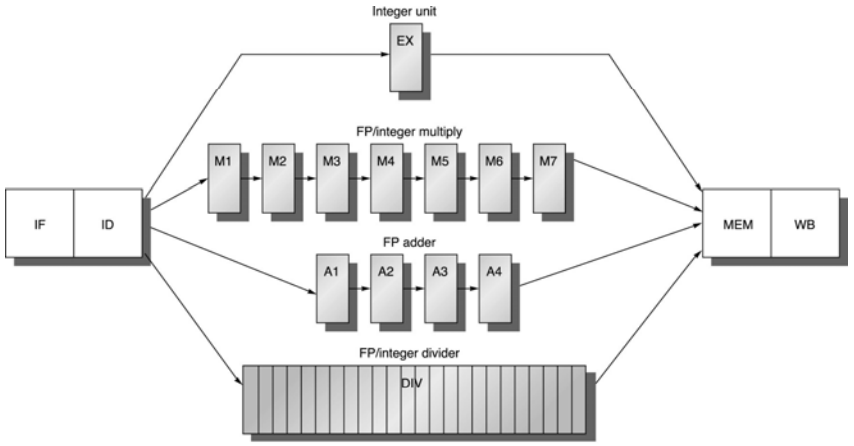
Examples

- **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 - Fig. A.31



Examples (cont.) - Fig. A.32

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 – 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

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 t previous strategy for precise exceptior**
  - **problem is *out-of-order completion***

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

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?

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**