Lecture 7: Pipelining Contd.

Kunle Olukotun
Gates 302
kunle@ogun.stanford.edu

http://www-leland.stanford.edu/class/ee282h/

1

More pipelining complications: Interrupts and Exceptions

- Hard to handle in pipelined machines
- Overlapping instrucitons
  - » When can you update the state of the machine
  - » Exceptions cause instructions to be aborted in the middle of execution

2

What kind of interrupts can occur?

» In DLX,
  – IF: page fault, misaligned memory access, memory protection violation
  – ID: Illegal opcode
  – EX: arithmetic overflow
  – MEM: page fault, misaligned memory access, memory protection violation
  – WB: none
» In other processors:
  – Other arithemetic events: fixed/floating under/overflow, divide-by-zero, loss of significance
  – privilege violations (kernel vs. user state)
  – I/O interrupts
  – Debugging interrupts

3

Classifying Interrupts

- Synchronous vs. asynchronous
- User requested vs. coerced
- User maskable vs. nonmaskable
- Within vs.between Instructions
- Resume vs. terminate

4

Classifying Interrupts

Exception	Synchronous?	Coerced?	Maskable?	Within instruction?	Restart?
I/O request	N	Y	N	N	Y
Trap	Y	N	Y	N	Y
FP overfllow	Y	Y	Y	Y	Y
Page fault	Y	Y	N	Y	Y
Undefined instruction	Y	Y	N	Y	N
Power failiure	N	Y	N	Y	Y

5

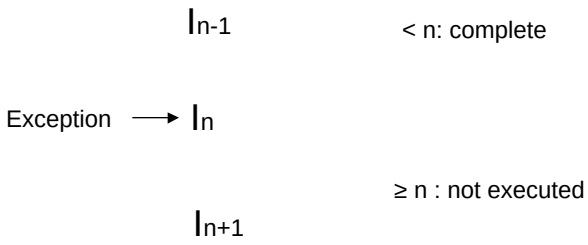
What makes interrupts difficult

Exception	Synchronous?	Coerced?	Maskable?	Within instruction?	Restart?
I/O request	N	Y	N	N	Y
Trap	Y	N	Y	N	Y
FP overfllow	Y	Y	Y	Y	Y
Page fault	Y	Y	N	Y	Y
Undefined instruction	Y	Y	N	Y	N
Power failiure	N	Y	N	Y	Y

6

What makes pipelined interrupts difficult

- Precise Interrupts
  - » Interrupts that occur in the middle of an instruction and must be restartable



7

Precise Interrupts

- Precise interrupts can be hard to implement
  - » Interrupts can occur anywhere during the execution of an instruction
  - » Multiple instructions are being executed, and each may have partially updated state information.
  - » Multiple instructions are being executed, so multiple interrupts can occur at the same time.
- The alternative is “imprecise” interrupts that cannot be restarted, but they can’t be used for:
  - » demand paging memory faults
  - » IEEE 754 compliant floating point interrupts

8

Example of multiple interrupts

Instr	Fault	1	2	3	4	5	6	7
add r1,r2,r3	overflow	IF	ID	***EX***	MEM	WB		
lw r3,0(r5)	MEM alignment errror		IF	ID	EX	***MEM***	WB	
sub r6,r6,r6	IF page fault			***IF***	ID	EX	MEM	WB

- add and sub faults occur at the same time
- sub fault occurs before the lw fault -- out of order

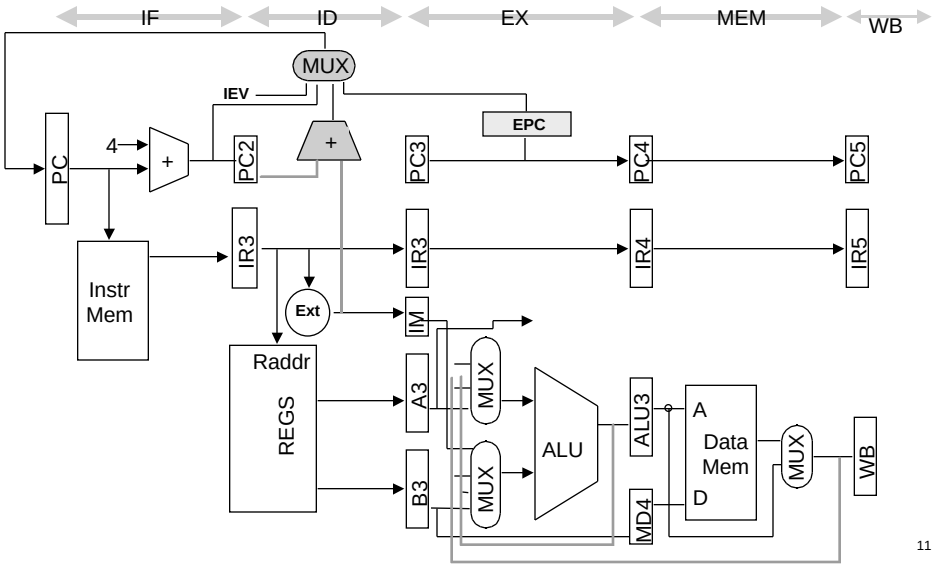
9

Getting to the interrupt routine and back

- In DLX, no state change occurs until MEM or WB. So we can handle interrupts by:
  - » Forcing a “trap” instruction into the pipeline
  - » Turn off all writes until the trap executes. (Turn all instructions into NOPs by clearing pipeline registers other than the PC chain)
  - » Give the faulting PC to the trap routine to allow restart.

10

Getting to the interrupt routine and back



11

Killing Instructions in Pipeline

Exception

	Instruction	1	2	3	4	5	6	7	8	9
Exception Instr.	i	IF	ID	EX	~~MEM~~	~~WB~~				
	i+1		IF	ID	~~EX~~	~~MEM~~	~~WB~~			
	i+2			IF	~~ID~~	~~EX~~	~~MEM~~	~~WB~~		
1st trap instr.	i+3				IF	ID	EX	MEM	WB	
	i+4					IF	ID	EX	MEM	WB

12

Branch Delay Slots

- Complication: Branch delay slots! If the instruction in a delay slot faults, which PC do you save?

```
            beq  r2, loc
            add   r3, r4, r5      <--- overflow here.  Save which PC?
             ...
loc:    sub   r6,r7,r8
```

- Answer: We must save both the address of the "add" and the address of the "sub". In general, save #branch delay slots + 1 PCs. (Or: save the address of the branch!)

13

Simultaneous and out-of-order interrupts

- If two fault conditions occur at the same time, take the interrupt for the earlier instruction. The later one will recur after returning from the interrupt routine.
- A fault for a later instruction may occur before one for an earlier instruction. We have two choices:
  - » Take the interrupts as they occur. Do ***not*** allow previous instructions to complete. Identify the faulting instructions, but restart at the first uncompleted instruction.
  - » Defer all interrupts until the end of an instruction. An instruction's *interrupt status* moves down the pipeline with it. Take the interrupt only at the end of MEM -- the "commit" point at which all interrupt conditions are known, but no effects have occurred.

14

Precise Interrupts in DLX Pipeline

Commit Point

Instr	Fault	1	2	3	4	5	6	7
add r1,r2,r3	overflow	IF	ID	***EX***	MEM	WB		
lw r3,0(r5)	MEM alignment errror		IF	ID	EX	***MEM***	WB	
sub r6,r6,r6	IF page fault			***IF***	ID	EX	MEM	WB

- Interrupt status moves with instruction
- Interrupt is handled when instruction reaches commit point

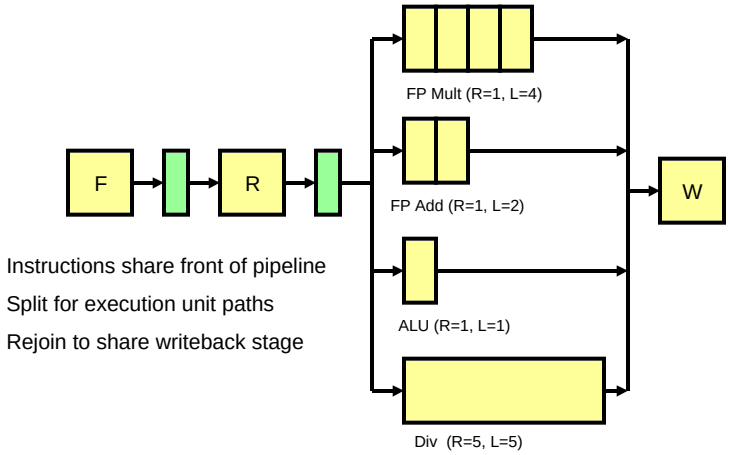
15

More pipelining complications: Multicycle operations

- Some operations are too long to accomodate in any fixed-length pipeline:
  - » integer divide
  - » most floating-point operations
  - » complex CISC addressing modes
- Number of clock cycles varies, perhaps based on execution data
  - » Latency
  - » Repeat rate
    - – 1 cycle for fully pipelined unit
- Cannot use a *static pipeline* with all instructions going through all stages; must use a *dynamic pipeline* with optional stages.
- As long as we're at it, we can do it for simple instructions too!
  - » ALU: IF ID EX WB
  - » STORE: IF ID EX MEM
  - » etc.

16

Pipeline with Multicyle Execution Units



17

The problems

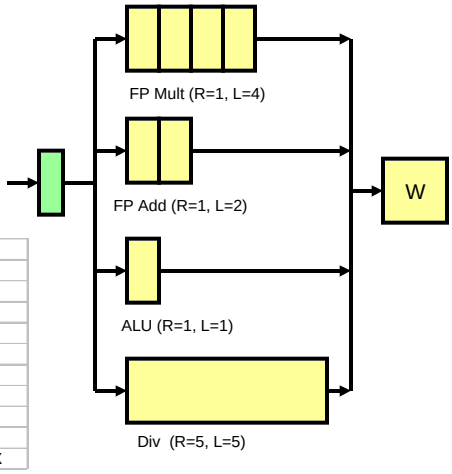
- We now have structural hazards:
  - » Units not fully pipelined may be busy: multiply, divide
  - » Because cycle times vary, multiple instructions may reach WB at the same time
- We now have WAW hazards:
  - » Short instructions that start late may reach WB before long instructions that start early
  - » (WAR hazards are still not possible because we issue in order and register reads always occur in ID)
- We now have interrupt headaches:
  - » Instructions may complete out-of-order, so how can we restart instruction i if instructions i+1 and i+2 are already finished?

18

Reservation Tables

Fetch	X							
Regs		X						
M1								
M2								
M3								
M4								
A1			X					
A2				X				
ALU								
DIV								
W					X			

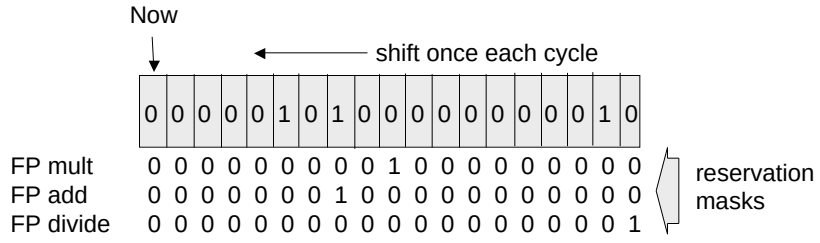
Fetch	X							
Regs		X						
M1								
M2								
M3								
M4								
A1								
A2								
ALU								
DIV			X	X	X	X	X	
W								X



19

Reservation Shift Register

Reservation ***shift register*** indicates what cycles a resource (register write port) will be available:



AND the mask with the reservation; if non-zero, then don't issue the instruction.

20

Competing for the Writeback Port

Consider the sequence:

FDIV
FMULT
NOP
FADD
AND

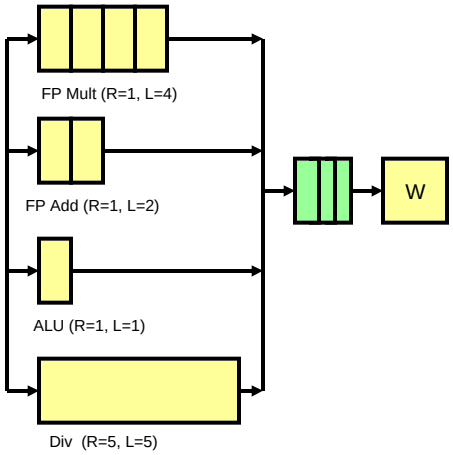
All 4 units request W stage in same cycle

Three approaches

1. Don't issue until reservation table indicates writeback stage will be available

2. Queue requests before writeback stage. Allows subsequent instructions to issue

3. Stall at output of unit until available



21

Dealing with data hazards

- Handle RAW the usual way: Stall the instruction issue until source registers can be forwarded
- WAR hazards can't happen
- WAW hazards:
  - » Example: divf f0, f2, f4 ;finishes second
    subf f0, f8, f10 ;finishes first
  - » Is this a useless sequence that we can produce incorrect results for? No!
  - » Two choices:
    - – Delay issuing subf until divf will enter WBF first
    - – Detect the hazard and turn the divf into a NOP.
  - » Note that an intervening instruction that uses f0 makes this become a standard RAW hazard, and the WAW disappears.

22

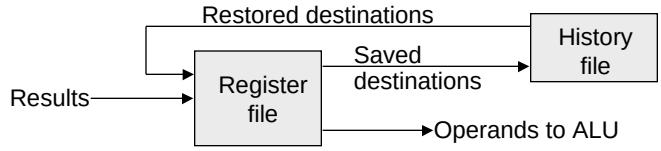
Dealing with out-of-order completion when an exception occurs

- After structural or data hazards have been eliminated, instructions may write results out-of-order:
  - » divf f0, f2, f4
    addf f10, f10, f8 ;this destroys its operand
    subf f12, f12, f14 ;this could interrupt with addf done and divf not
- Several solutions:
  - » Punt: Imprecise interrupts that disallow restart (360/91, CRAY)
  - » Allow the software to optionally determine restart points (DEC Alpha "trapb" barrier causes all prior instructions to commit; can be used in basic blocks without operand destruction to get precise interrupts.)
  - » Delay instruction issue until all prior instruction are known not to cause an interrupt. (Use early rough tests that can have false positives. MIPS 3010, Pentium do this.)
  - » Buffer results to allow the correct register set to be reconstructed ("history file", "reorder buffer", "future file")

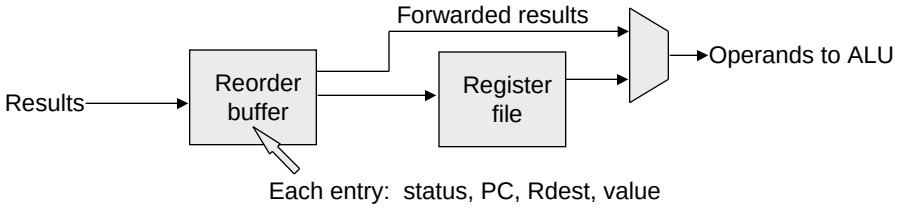
23

Reconstructing the register file for out-of-order completion

#1 History file: *keep original values that can be restored after an exception*



#2 Reorder buffer: *don't write register values out-of-order*

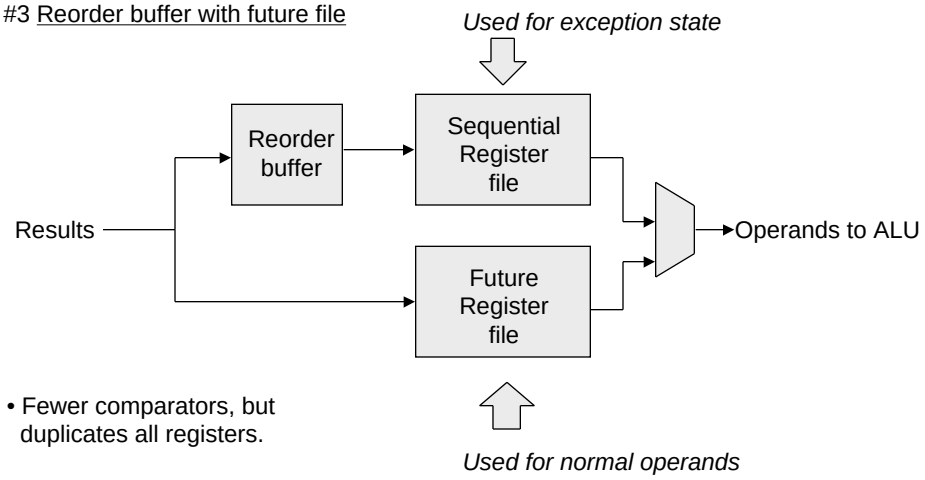


(Requires many ports/compators in reorder buffer, since the same register may appear multiple times.)

24

Reconstructing the register file for out-of-order completion

#3 Reorder buffer with future file



• Fewer comparators, but duplicates all registers.

25

Another pipeline example: high memory latency

- An 8-stage pipeline with multiple stages for memory (the MIPS R4000, approximately):
  - » IF1: Instruction fetch, part 1
  - » IF2: Instruction fetch, part 2
  - » ID Decode, Register fetch, branch target address computation
  - » EX ALU ops, effective address computation branch condition determination
  - » MEM1 memory access, part 1
  - » MEM2 memory access, part 2
  - » MEM3 memory access, part 3
  - » WB Write back to register file
- Assume that the fast cycle time makes register reads and write take a full cycle -- ie, we get no free WB->ID forwarding.

26

The pipeline diagram

Instr	1	2	3	4	5	6	7	8	9	10
i	IF1	IF2	ID	EX	MEM1	MEM2	MEM3	WB		
i+1		IF1	IF2	ID	EX	MEM1	MEM2	MEM3	WB	
I+2			IF1	IF2	ID	EX	MEM1	MEM2	MEM3	WB
i+3				IF1	IF2	ID	EX	MEM1	MEM2	MEM3
i+4					IF1	IF2	ID	EX	MEM1	MEM2
i+5						IF1	IF2	ID	EX	MEM1
i+6							IF1	IF2	ID	EX
i+7								IF1	IF2	ID

What are the forwarding paths?

From stage	To stage	To which subsequent instruction

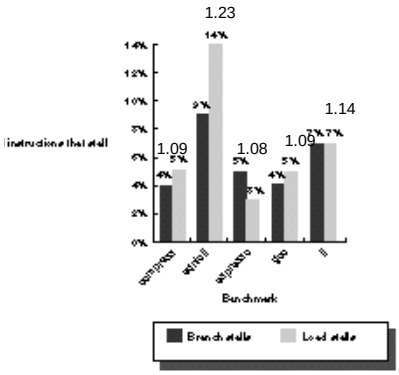
27

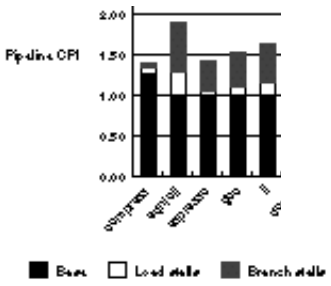
Characteristics of this pipeline

- How many comparators are needed to implement the forwarding decisions?
- What instruction sequences will still cause stalls?
- What is the branch delay?
- What is the load delay?

28

Pipeline Performance





5-stage pipeline

8-stage pipeline

» 8-stage R4400, 200 MHz, 140 SPECint92
» 5-stage R4700, 175 MHz, 130 SPECint92

29

Better performance from more instruction-level parallelism (Chap 4)

- Pipeline CPI = Ideal CPI +
  + structural stalls (resource conflict)
  + control stalls (control dependency)
  + RAW stalls (true data dependency)
  + WAR stalls (anti-dependency, a name dependence)
  + WAW stalls (output dependency, another name dependence)
- More hardware tricks
  - » multiple instruction issue (“superscaler”, VLIW)
  - » dynamic scheduling
  - » advanced branch prediction
  - » speculative execution
- Clever compilers can also help
  - » pipeline scheduling
  - » loop unrolling
  - » register renaming

30