CS 152
Computer Architecture and Engineering

Lecture 8 – Pipelining II

2006-9-21

John Lazzaro
(www.cs.berkeley.edu/~lazzaro)

TAs: Udam Saini and Jue Sun

www-inst.eecs.berkeley.edu/~cs152/
Welcome to Lab 3!
From First Class: The Architect’s Contract

To the program, it appears that instructions execute in the correct order defined by the ISA.

As each instruction completes, the machine state (regs, mem) appears to the program to obey the ISA.

What the machine actually does is up to the hardware designers, as long as the contract is kept.

The primary challenge of 152 CPU projects!
Last Time: Performance and Hazards

Seconds Program = Instructions Program

Cycles Instruction / Seconds Cycle

Instr Fetch

Decode & Reg Fetch

Stage #3

Some ways to cope with hazards makes CPI > 1 “stalling pipeline”

Added logic to detect and resolve hazards increases clock period

“Software slows the machine down” Seymour Cray
Today: Hazards

- Visualizing pipelines to evaluate hazard detection and resolution.
- A taxonomy of pipeline hazards.
- A tool kit for hazard resolution.

Tuesday: We apply this knowledge to design a pipelined MIPS CPU that obeys the contract with the programmer.
Reminder: Do the Reading!

The lectures are a gentle introduction, to prepare you to read the book ...

The book presentation of pipelined processors is sufficient to do Lab 3.

These lectures are not.
Visualizing Pipelines
Pipeline Representation #1: Timeline

**Sample Program**

<table>
<thead>
<tr>
<th>Inst</th>
<th>I1: ADD R4,R3,R2</th>
<th>I2: AND R6,R5,R4</th>
<th>I3: SUB R1,R9,R8</th>
<th>I4: XOR R3,R2,R1</th>
<th>I5: OR R7,R6,R5</th>
</tr>
</thead>
</table>

**Time:**

| I1: | IF | ID | EX | MEM | WB |
| I2: | IF | ID | EX | MEM | WB |
| I3: | IF | ID | EX | MEM | WB |
| I4: | IF | ID | EX | MEM | WB |
| I5: | IF | ID | EX | MEM | WB |

**Pipeline is “full”**
Good for visualizing pipeline stalls.

Sample Program

I1: ADD R4, R3, R2
I2: AND R6, R5, R4
I3: SUB R1, R9, R8
I4: XOR R3, R2, R1
I5: OR R7, R6, R5

Time: t1  t2  t3  t4  t5  t6  t7  t8
Stage
IF:  I1  I2  I3  I4  I5  I6  I7  I8
ID:  I1  I2  I3  I4  I5  I6  I7  I8
EX:  I1  I2  I3  I4  I5  I6  I7  I8
MEM: I1  I2  I3  I4  I5  I6  I7  I8
WB:  I1  I2  I3  I4  I5  I6  I7  I8

Pipeline is “full”
Hazard Taxonomy
Structural Hazards

Several pipeline stages need to use the same hardware resource at the same time.

Solution #1: Add extra copies of the resource (only works sometime).

Solution #2: Change resource so that it can handle concurrent use.

Solution #3: Stages “take turns” by stalling parts of the pipeline.
HW 1: Structural Hazard (Single Memory)

HW uses “Solution 3” (stalling pipeline)
Lab 2/3 solution: “Extra copies” of memory

1. **“IF” Stage**
   - Instr Fetch
   - IR
   - Mux, Logic
   - 0x4
   - +

2. **“ID/RF” Stage**
   - Decode & Reg Fetch
   - IR
   - RegFile
   - rs1
   - rs2
   - rd1
   - rd2
   - ws
   - wd
   - WE
   - Ext

3. **“EX” Stage**
   - Execution
   - A
   - ALU
   - op
   - 32
   - 32

4. **“MEM” Stage**
   - Memory
   - Data Memory
   - Addr
   - Dout
   - MemToReg

5. **WB**
   - Write Back
   - R

I and D caches (Final Project) are a hybrid solution
“Solution #2”: Concurrent use ...

1. "IF" Stage
   Instr Fetch

2. "ID/RF" Stage
   Decode & Reg Fetch

3. "EX" Stage
   Execution

4. "MEM" Stage
   Memory

5. WB
   Write Back

ID and WB stages use register file in same clock cycle
Data Hazards: 3 Types (RAW, WAR, WAW)

Several pipeline stages read or write the same data location in an incompatible way.

Read After Write (RAW) hazards.
Instruction I2 expects to read a data value written by an earlier instruction, but I2 executes “too early” and reads the wrong copy of the data.

Note “data value”, not “register”. Data hazards are possible for any architected state (such as main memory). In practice, main memory hazard avoidance is the job of the memory system.
Recall from last lecture: RAW example

Sample program

ADD R4, R3, R2
OR R5, R4, R2

... wrong value of R4 fetched from RegFile, contract with programmer broken! Oops!

This is what we mean when we say Read After Write (RAW) Hazard
Data Hazards: 3 Types (RAW, WAR, WAW)

Write After Read (WAR) hazards. Instruction I2 expects to write over a data value after an earlier instruction I1 reads it. But instead, I2 writes too early, and I1 sees the new value.

Write After Write (WAW) hazards. Instruction I2 writes over data an earlier instruction I1 also writes. But instead, I1 writes after I2, and the final data value is incorrect.

WAR and WAW not possible in our 5-stage pipeline. But are possible in other pipeline designs.
Control Hazards: A taken branch/jump

Note: with branch delay slot, I2 MUST complete, I3 MUST NOT complete.

Sample Program (ISA w/o branch delay slot)

I1: BEQ R4,R3,25
I2: AND R6,R5,R4
I3: SUB R1,R9,R8

Time: t1 t2 t3 t4 t5 t6 t7 t8
Inst
I1: IF ID EX MEM WB
I2: IF ID
I3: IF
I4: 
I5: 
I6: 

EX stage computes if branch is taken

If branch is taken, these instructions MUST NOT complete!
Hazards Recap

- **Structural** Hazards
- **Data** Hazards (RAW, WAR, WAW)
- **Control** Hazards (taken branches and jumps)

On each clock cycle, we must detect the presence of all of these hazards, and resolve them before they break the “contract with the programmer”.
Administtrivia: Upcoming deadlines ...

Friday 9/22: “Xilinx Checkoff”, in section.

Monday 9/25: Lab 2 final report due via the submit program, 11:59 PM.
Lab 3 now available on the web site

Thursday 9/28: At 11:59 PM via email: Lab 2 peer evaluations, and Lab 3 preliminary design document due.
Updated Office Hours

Udam: **MW 2-3 PM, 125 Cory**
Jue: **T4-5 PM, Th 3-4 PM, 125 Cory**
John: **TTh 10-11AM, 315 Soda**
<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Th 9/28</td>
<td>Midterm Review Session in Class</td>
<td></td>
</tr>
<tr>
<td>F 9/29</td>
<td>HW 1 due in class, Lab 3: Preliminary Design Document and Team Evaluations due to TAs, 11:59PM</td>
<td></td>
</tr>
<tr>
<td>Sa 9/30</td>
<td>Lab 3: Preliminary Design Document Review, 12-2PM or 3-5PM, 125 Cory</td>
<td></td>
</tr>
<tr>
<td>Su 10/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M 10/2</td>
<td>Midterm I: 6:00PM to 9:00PM, 306 Soda (HP Auditorium) (note: no class 11-12:30)</td>
<td></td>
</tr>
<tr>
<td>T 10/3</td>
<td>Lab 3: Final Design Document due to TAs, 11:59PM</td>
<td></td>
</tr>
<tr>
<td>W 10/4</td>
<td>HW 2 available (due at Midterm II review session)</td>
<td></td>
</tr>
<tr>
<td>Th 10/5</td>
<td>VLSI</td>
<td></td>
</tr>
<tr>
<td>F 10/6</td>
<td>Lab 3: Initial Xilinx Checkoff, 12-2PM or 3-5PM, 125 Cory</td>
<td></td>
</tr>
</tbody>
</table>

Thursday review session. Will cover format, material, and ground rules for test.

Midterm two weeks from today, in evening, no class that day.

Crunch Week: Homework, Midterm, Lab

Lab 3 final design doc, checkoffs, later in week...
Hazard Resolution Tools
The Hazard Resolution Toolkit

- **Stall** earlier instructions in pipeline.
- **Forward** results computed in later pipeline stages to earlier stages.
- **Add** new hardware or **rearrange** hardware design to eliminate hazard.
- **Change ISA** to eliminate hazard.
- **Kill** earlier instructions in pipeline.
- Make hardware handle **concurrent requests** to eliminate hazard.
Resolving a RAW hazard by stalling

Sample program

ADD R4, R3, R2
OR R5, R4, R2

Keep executing OR instruction until R4 is ready. Until then, send NOPS to IR 2/3.

Let ADD proceed to WB stage, so that R4 is written to regfile.

Freeze PC and IR until stall is over.

New datapath hardware

(1) Mux into IR 2/3 to feed in NOP.

(2) Write enable on PC and IR 1/2
The Hazard Resolution Toolkit

- **Stall** earlier instructions in pipeline.

- **Forward** results computed in later pipeline stages to earlier stages.

- **Add** new hardware or **rearrange** hardware design to eliminate hazard.

- **Change ISA** to eliminate hazard.

- **Kill** earlier instructions in pipeline.

- **Make hardware handle concurrent requests** to eliminate hazard.
Resolving a RAW hazard by forwarding

Sample program

ADD R4, R3, R2
OR R5, R4, R2

Just forward it back!

ALU computes R4 in the EX stage, so ...

Unlike stalling, does not change CPI. May hurt cycle time.
The Hazard Resolution Toolkit

- **Stall** earlier instructions in pipeline.
- **Forward** results computed in later pipeline stages to earlier stages.
- **Add** new hardware or **rearrange** hardware design to eliminate hazard.
- **Change ISA** to eliminate hazard.
- **Kill** earlier instructions in pipeline.
- **Make hardware handle concurrent requests** to eliminate hazard.
Control Hazards: Fix with more hardware

If we add hardware, can we move it here?

Sample Program (ISA w/o branch delay slot)

I1: BEQ R4,R3,25
I2: AND R6,R5,R4
I3: SUB R1,R9,R8
I4:
I5:
I6:

If branch is taken, these instructions MUST NOT complete!

EX stage computes if branch is taken

Time: t1 t2 t3 t4 t5 t6 t7 t8

Inst
I1: IF ID EX MEM WB
I2: IF ID
I3: IF
I4:
I5:
I6:
Resolving control hazard with hardware

Stage #1
Instr Fetch

Stage #2
Decode & Reg Fetch

Stage #3

To branch control logic

Instr Fetch
Stage #1

Decode & Reg Fetch
Stage #2

RegFile

IR

PC

D

Q

Instr Mem

Addr Mem

Data

Ext

A

M

B

RegFile
rs1
rs2
rd1
rd2
ws
wd
WE

Ext

0x4
Control Hazards: After more hardware

Sample Program (ISA w/o branch delay slot)

I1: BEQ R4, R3, 25
I2: AND R6, R5, R4
I3: SUB R1, R9, R8

If we change ISA, can we always let I2 complete ("branch delay slot") and eliminate the control hazard.

If branch is taken, this instruction MUST NOT complete!
From Lecture 1: \textbf{BEQ $1, $2, 25}

Fetch branch inst from memory

```
opcode | rs | rt | offset
```

“\textit{I-Format}"

Decode fields to get: \texttt{BEQ $1, $2, 25}

“Retrieve” register values: $1, $2

Compute if we take branch: $1 == $2 ?

\textbf{ALWAYS prepare to fetch instr that follows the BEQ in the program ("delayed branch"). IF we take branch, the instr we fetch AFTER that instruction is PC + 4 + 100.}

\textit{PC} == “Program Counter”
The Hazard Resolution Toolkit

- **Stall** earlier instructions in pipeline.
- **Forward** results computed in later pipeline stages to earlier stages.
- **Add** new hardware or **rearrange** hardware design to eliminate hazard.
- **Change ISA** to eliminate hazard.
- **Kill** earlier instructions in pipeline.
- **Make hardware handle concurrent requests** to eliminate hazard.
Resolve control hazard by killing instr

Sample program (no delay slot)

### J 200

OR R5, R4, R2

Detect J instruction, mux a NOP into IR 1/2

This hurts CPI.

Can we do better?
The Hazard Resolution Toolkit

- **Stall** earlier instructions in pipeline.
- **Forward** results computed in later pipeline stages to earlier stages.
- **Add** new hardware or **rearrange** hardware design to eliminate hazard.
- **Change ISA** to eliminate hazard.
- **Kill** earlier instructions in pipeline.
- Make hardware handle **concurrent requests** to eliminate hazard.
Structural hazard solution: concurrent use

“IF” Stage
- Instr Fetch

“ID/RF” Stage
- Decode & Reg Fetch

“EX” Stage
- Execution

“MEM” Stage
- Memory

Write Back

Does not come for free ...

ID and WB stages use register file in same clock cycle
Summary: Hazards

- **Visualizing** pipelines to evaluate hazard detection and resolution.
- A **taxonomy** of pipeline hazards.
- A tool kit for hazard **resolution**.

Interesting question raised last term ...
"Write contract to match the hardware?"

What if we left hazards to the compiler?
Register RAW hazards: No stalls or fwds.

How would the contract read?
Can you do forwarding “in software”? 

Sample program:

ADD R4, R3, R2
OR R5, R4, R2

Just forward it back!

ALU computes R4 in the EX stage, so ...

Hint: Expose forwarding to the ISA ...
Yes! By exposing forwarding registers ...

Sample Program

ADD F, R3, R2
NOP
OR R5, F, R2

ADD F, R3, R2

One cycle later ...
One clock cycle later ...

Sample Program
ADD F, R3, R2
NOP
OR R5, F, R2

F value left by ADD ...

Trick used by “RAW” project at MIT ...
Coming up next week ...

Tuesday: Applying hazard tools to a pipelined CPU design.

Thursday: Mid-term review, HW 1 due in class.

<table>
<thead>
<tr>
<th>Date</th>
<th>Activity</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/26</td>
<td>Pipelining III</td>
<td>6.8-9</td>
</tr>
<tr>
<td>9/28</td>
<td>Midterm Review Session in Class</td>
<td></td>
</tr>
<tr>
<td>9/29</td>
<td>Lab 3: Preliminary Design Document Review, 12-2PM or 3-5PM, 125 Cory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HW 1 due in class</td>
<td></td>
</tr>
</tbody>
</table>

CS 152 L8: Pipelining II

UC Regents Fall 2006 © UCB