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: A 5-stage pipelined CPU

1. "IF" Stage
   Instr Fetch

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

3. "EX" Stage
   Execution

4. "MEM" Stage
   Memory

5. "WB" Stage
   Write Back
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 book presentation of pipelined processors is sufficient to do Lab 3.

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>PDF</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>T 9/20</td>
<td>Pipelining I</td>
<td>PDF</td>
<td>6.1-4</td>
</tr>
<tr>
<td>Th 9/22</td>
<td>Pipelining II</td>
<td>PDF</td>
<td>6.5-7</td>
</tr>
<tr>
<td>T 9/27</td>
<td>Pipelining III</td>
<td>PDF</td>
<td>6.8-9</td>
</tr>
</tbody>
</table>

These lectures are not.

The lectures are a gentle introduction, to prepare you to read the book ...
Visualizing Pipelines
Pipeline Representation #1: Timeline

IF (Fetch) --> ID (Decode) --> EX (ALU) --> MEM --> WB

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
I6:

Time: t1 t2 t3 t4 t5 t6 t7 t8

Inst

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
I6: IF ID EX MEM WB

Pipeline is “full”
Representaion #2: Resource Usage

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
ID: I1 I2 I3
EX: I1 I2
MEM: I1
WB: I1 I2 I3 I4

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.
Structural hazard solution: 2 memories

What if we merged Data and Instr memories?
Structural hazard solution: 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

Sample Program (ISA w/o branch delay slot)

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

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

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”.
Administrivia: Upcoming deadlines ...

Friday 9/23: “Xilinx Checkoff”, in section. For non-150 students, “150 Lab Lecture 4”, 2-3 PM, 125 Cory.

Monday 9/26: Lab 2 final report due via the submit program, 11:59 PM.

Lab 3 now available on the web site

Thursday 9/29: At 11:59 PM via email: Lab 2 peer evaluations, and Lab 3 preliminary design document due.
## Crunch Week: Homework, Midterm, Lab

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

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

### HW graded on effort

<table>
<thead>
<tr>
<th>Date</th>
<th>Activity</th>
<th>Time/Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/29</td>
<td>Midterm Review Session in Class</td>
<td>PDF</td>
</tr>
<tr>
<td>9/30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/4</td>
<td>Midterm I: 6:00PM to 9:00PM, 310 Soda Hall</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(note: no class 11-12:30)</td>
<td></td>
</tr>
<tr>
<td>10/5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/6</td>
<td>VLSI</td>
<td>PDF</td>
</tr>
<tr>
<td>10/7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- HW 1 due in class
- Lab 3: Preliminary Design Document and Team Evaluations due to TAs, 11:59PM
- Lab 3: Preliminary Design Document Review, 12-2PM or 3-5PM, 125 Cory
- Lab 3: Final Design Document due to TAs, 11:59PM
- HW 2 available
- (due at Midterm II review session)
- Lab 3: Initial Xilinx Checkoff, 12-2PM or 3:5PM, 125 Cory

### Lab 3 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

![Diagram showing the pipeline stages and resolving a RAW hazard]

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.

New datapath hardware

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

(2) Write enable on PC and IR 1/2

Freeze PC and IR until stall is over.

Let ADD proceed to WB stage, so that R4 is written to regfile.
Resolving a RAW hazard by forwarding

Sample program
ADD R4, R3, R2
OR R5, R4, R2

Instr Fetch
Decode & Reg Fetch
Execution

ALU computes R4 in the EX stage, so ...
Just forward it back!

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

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:

If we add hardware, can we move it here?

If branch is taken, these instructions MUST NOT complete!

EX stage computes if branch is taken
Resolving control hazard with hardware

To branch control logic

Stage #1  Instr Fetch
Stage #2  Decode & Reg Fetch
Stage #3

PC  +  0x4

Instr Mem

Addr  Data

IR

RegFile
rs1  rd1
rs2  rd2
ws  WE
wd

Ext

IR

A

M

B

==
Control Hazards: After more hardware

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

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
I3:               ID
I4:               EX
I5:               MEM
I6:               WB

If branch is taken, this instruction MUST NOT complete!
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

Can we do better?

Compute new PC using hardware not shown ...
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

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 from last class ...
“Write contract to match the hardware?”

What if we left hazards to the compiler?
Case 1: RAW Data Hazards ...

How would the contract read?
Can we simulate forwarding in software?

Sample program:

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

Just forward it back!

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

Unlike stalling, does not change CPI. May hurt cycle time.
Case 2: Control Hazards ...

How would the contract read?
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>Th 9/29</th>
<th>Midterm Review Session in Class</th>
<th>PDF</th>
<th>HW 1 due in class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Lab 3: Preliminary Design Document and Team Evaluations due to TAs, 11:59PM</td>
</tr>
<tr>
<td>F 9/30</td>
<td></td>
<td></td>
<td>Lab 3: Preliminary Design Document Review, 12-2PM or 3-5PM, 125 Cory</td>
</tr>
</tbody>
</table>