Last time: A Hazard Taxonomy

- **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”.
Last Time: 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.
Today: Putting it All Together

- Specifications for Lab 3
- At-risk hazards for Lab 3
- Preferred hazard resolution tools.
- Tips for control design
Lab 3: ISA Specifications

Also: RESET signal, BREAK release signal, etc ...

<table>
<thead>
<tr>
<th>Type</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>arithmetic</td>
<td>addu, subu, addiu</td>
</tr>
<tr>
<td>logical</td>
<td>and, andi, or, ori, xor, xori, lui</td>
</tr>
<tr>
<td>shift</td>
<td>sll, sra, srl</td>
</tr>
<tr>
<td>compare</td>
<td>slt, slti, sltu, sltui</td>
</tr>
<tr>
<td>control</td>
<td>beq, bne, bgez, bltz, j, jr, jal</td>
</tr>
<tr>
<td>data transfer</td>
<td>lw, sw</td>
</tr>
<tr>
<td>Other:</td>
<td>break</td>
</tr>
</tbody>
</table>

The break instruction is special. See COD for its bitfield. Although this is normally an exception-causing instruction, you should treat it more like a halt instruction. After being decoded, the break instruction should freeze the pipeline from advancing further. This means that the PC will not advance further, the break instruction will stay in the decode stage, and later instructions will drain from the pipeline as they complete. The proper terminology for this is that the break instruction will "stall" in the decode stage. Assume that there will be a single input signal called "release" that comes from outside. When it is high, you should release a blocked break instruction exactly once (you need to
And

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>6</th>
<th>5</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECIAL</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>0</td>
<td>00000</td>
<td>AND</td>
<td>100100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** `AND rd, rs, rt`

**Purpose:**
To do a bitwise logical AND

**Description:** `rd ← rs AND rt`

The contents of GPR `rs` are combined with the contents of GPR `rt` in a bitwise logical AND operation. The result is placed into GPR `rd`.

**Restrictions:**
None

**Operation:**

\[
\text{GPR}[rd] \leftarrow \text{GPR}[rs] \text{ and } \text{GPR}[rt]
\]

**Exceptions:**
None
Hazard Diagnosis
Data Hazards: Read After Write

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.

Lab 3 solution: use forwarding heavily, fall back on stalling when forwarding won’t work or slows down the critical path too much.
Full bypass network ...

Mux, Logic

RegFile

rs1
rs2
ws
wd
WE
rd1
rd2

Ext

IR

Data Memory

WE

Din
Addr

MemToReg

Dout

WC

op

A
L
U

B

M

M

Y

R

ID (Decode)

EX

MEM

WB
Common bug: Multiple forwards...

Which do we forward from?

ADD R4,R3,R2
OR R2,R3,R1 AND R2,R2,R1

Which do we forward from?
Data Hazards: WAR and 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. However, TA test code checks for these, and every semester a few WAR/WAWs are found. Why?
Questions about LW and forwarding

Will this work as shown?

```
ADDIU R1 R1 24
OR R3,R3,R2 LW R1 128(R29)
```

```
IR
ID (Decode)
EX
MEM
WB
IR
Mux, Logic
RegFile
rs1
rs2
rd1
rd2
ws
wd
WE

Data Memory
Addr
Din
Dout
MemToReg

B
M
IR

A
ALU
32
32
op
Y

WE, MemToReg

Ext

From WB

R
From WB
```
Questions about LW and forwarding

Will this work as shown?

```
ADDIU R1 R1 24
LW R1 128(R29) OR R1,R3,R1
```

```
From WB
Mux,Logic
RegFile
rs1
rs2
ws
wd
WE
rd1
rd2
```

```
IR
WE, MemToReg
MEM
IR
EX
IR
ID (Decode)
IR
```

```
R
```

```
Ext
```

```
Data Memory
```

```
MemToReg
```

```
Ext
```

```
```

ADDIU R1 R1 24
LW R1 128(R29) OR R1,R3,R1
```
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.

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.
Branches and Hazards
Recall: Control hazard and hardware

Stage #1
Instr Fetch

Stage #2
Decode & Reg Fetch

Stage #3

To branch control logic

0x4

IR

RegFile
rs1 rs2 rd1
ws rd2
wd WE

Ext

A

M

B

PC

Instr Mem

Addr Data

D Q

Q

IR

IR

IR

+
Recall: After more hardware, change ISA

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
If branch is taken, this instruction MUST NOT complete!

ID stage computes if branch is taken
Questions about branch and forwards

**BEQ R1 R3 label**

**Will this work as shown?**

**OR R3, R3, R1**

**To branch control logic**

**Mux, Logic**

**RegFile**

rs1
ds2
ws
wd

**WE**

**wb**

**rd1**

**rd2**
Lessons learned

- Pipelining is **hard**
- Study **every** instruction
- Write test code **in advance**
- Think about interactions ...
Control Implementation
Recall: What is single cycle control?

Combinational Logic
(Only Gates, No Flip Flops)
Just specify logic functions!
In pipelines, all IR registers are used

Combinational Logic
(Only Gates, No Flip Flops)
(add extra state outside!)

A “conceptual” design -- for shortest critical path, IR registers may hold decoded info, not the complete 32-bit instruction
Two goals when specifying control logic

**Bug-free:** One “0” that should be a “1” in the control logic function breaks contract with the programmer.

Should be easy for humans to read and understand: sensible signal names, symbolic constants ...

**Efficient:** Logic function specification should map to hardware with good performance properties: fast, small, low power, etc.
Midterm week begins on Thursday ...

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

Midterm (6-9, 310 Soda), no class that day.
And concurrently, Lab 3 deadlines ...

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Th 9/29</td>
<td>Midterm Review Session in Class</td>
<td>PDF</td>
</tr>
<tr>
<td>F 9/30</td>
<td>HW 1 due in class</td>
<td>Lab 3: Preliminary Design Document and Team Evaluations due to TAs, 11:59PM</td>
</tr>
<tr>
<td>Sa 10/1</td>
<td>Lab 3: Preliminary Design Document Review, 12-2PM or 3-5PM, 125 Cory</td>
<td></td>
</tr>
<tr>
<td>Su 10/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M 10/3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T 10/4</td>
<td>Midterm I: 6:00PM to 9:00PM, 310 Soda Hall (note: no class 11-12:30)</td>
<td></td>
</tr>
<tr>
<td>W 10/5</td>
<td>Lab 3: Final Design Document due to TAs, 11:59PM</td>
<td></td>
</tr>
<tr>
<td>Th 10/6</td>
<td>VLSI</td>
<td>HW 2 available (due at Midterm II review session)</td>
</tr>
<tr>
<td>F 10/7</td>
<td>Lab 3: Initial Xilinx Checkoff, 12-2PM or 3-5PM, 125 Cory</td>
<td></td>
</tr>
</tbody>
</table>

Lab 2 team evals, Lab 3 design document, Weekend: start design work
Admin: Team Evaluations due Thursday

<table>
<thead>
<tr>
<th>Name</th>
<th>Score</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue Superstar</td>
<td>30</td>
<td>Sue really helped along the group. Sue figured out how to handle interlocks in the pipeline, and stayed extra long last week to fix out last bug.</td>
</tr>
<tr>
<td>Teddy Tryhard</td>
<td>13</td>
<td>Teddy was always at our meetings, had a very positive attitude, and did everything the group asked him to. However, he often made mistakes and needed help.</td>
</tr>
<tr>
<td>Annie Average</td>
<td>20</td>
<td>Annie did a good job.</td>
</tr>
<tr>
<td>Ned Neverthere</td>
<td>5</td>
<td>Ned never showed up to group meetings. We ended up reimplementing the one piece that he did give us.</td>
</tr>
</tbody>
</table>
### Admin: Design Document Deadlines

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/29</td>
<td>Th: Midterm Review Session in Class</td>
<td>PDF</td>
</tr>
<tr>
<td>9/30</td>
<td>F: Preliminary Design Document Review</td>
<td>Lab 3: Preliminary Design Document Review, 12-2PM or 3-5PM, 125 Cory</td>
</tr>
</tbody>
</table>

1. The identity of the **spokesperson** for the lab, and a roster of group members. The responsibility of the spokesperson is to communicate questions to the TA and reply to questions from the TA. Choose a different spokesperson than the one you had for Lab 2.
2. A short description of the structure of the design. The description will be accompanied by preliminary high-level schematics of your datapath, and a preliminary discussion of the controller.
3. A description of the unit test benches and multi-unit test benches you intend to create for your processor, and a description of machine language programs you intend to write to use in complete processor testing. Also, a **test plan**, using the epoch charting method shown in the 9/8 lecture, that shows when you plan to run each type of test.
4. A tentative **division of labor**, showing the tasks each group member intends to do.
5. The "**paranoia**" section: discuss potential areas of difficulty in the lab. An early guess of critical timing paths for the design should be a part of this section.
## Lab 3 deadlines after the mid-term ...

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Th 9/29</td>
<td>Midterm Review Session in Class</td>
<td>PDF</td>
</tr>
<tr>
<td>F 9/30</td>
<td>Lab 3: Preliminary Design Document and Team Evaluations due to TAs, 11:59PM</td>
<td></td>
</tr>
<tr>
<td>Sa 10/1</td>
<td>Lab 3: Preliminary Design Document Review, 12-2PM or 3-5PM, 125 Cory</td>
<td></td>
</tr>
<tr>
<td>Su 10/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M 10/3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T 10/4</td>
<td>Midterm I: 6:00PM to 9:00PM, 310 Soda Hall (note: no class 11-12:30)</td>
<td></td>
</tr>
<tr>
<td>W 10/5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Th 10/6</td>
<td>VLSI</td>
<td>PDF, PDF</td>
</tr>
<tr>
<td>F 10/7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lab 3: Final Design Document due to TAs, 11:59PM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HW 2 available (due at Midterm II review session)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lab 3: Initial Xilinx Checkoff, 12-2PM or 3-5PM, 125 Cory</td>
<td></td>
</tr>
</tbody>
</table>

Lab 3 design doc + Xilinx checkoff later in week...
Are pipelined CPUs a “solved problem”? 

Embedded CPU -- “embedded into products”. Unlike desktop/laptop/server, where customer knows he is buying a “computer”.

Embedded CPU -- runs one program its entire life.
Theme: don’t do hardware in a vacuum

Optimize architecture, compilers for code that runs on the hardware, and the CAD tools that create the hardware **together**.
Example: Customize “ALU” for the app

Designing a new CPU just for one product is hard. Why?

\[
\text{Seconds} \quad = \quad \frac{\text{Instructions}}{\text{Program}} \quad \frac{\text{Cycles}}{\text{Instruction}} \quad \frac{\text{Seconds}}{\text{Cycle}}
\]
Teaching compilers and operating systems about unusual memory systems is hard.
Example: Energy & operating systems

Teaching compilers and operating systems about $E = \frac{1}{2} C V^2$ is hard.