ECE 2300
Digital Logic & Computer Organization
Spring 2021

More Pipelined Microprocessor
Announcements

• **Prelim 2 next Tuesday @ 1pm**
  – Enter exam Zoom room 5 mins earlier
  – Exam protocol will be (re)posted soon
  – TA-led Prelim 2 review at 8pm tonight
  – No instructor office hour today
    • Rescheduled to Sunday April 18th @ 4:30pm

• **Lab 3(B) due tomorrow**
  – Use combinational logic for output/next state logic
    (refer to Lecture 9)
  – Lab 3 report deadline will be extended by 10 days
    (stay tuned)
Pipelined Microprocessor

Key idea: Keep all resources fully utilized
The OR, SUB, and AND instructions are *data dependent* on the ADD instruction.
Review: Compiler Inserts NOPs (Solution 1)

ADD R1, R2, R3

NOP

NOP

NOP

OR R4, R1, R3
Example: Data Hazards

- How many NOPs need to be inserted to avoid data hazards in the following instruction sequence

```plaintext
SUB R4, R2, R3
ADD R1, R1, 1
ADDI R2, R4, 1
OR R5, R3, R4
```

```
SUB R4, R2, R3
NOP
NOP
ADD R1, R1, 1
ADDI R2, R4, 1
OR R5, R3, R4
```
Review: HW Stalls the Pipeline (Solution 2)

The pipeline is stalled for three cycles
Solution 3: HW Forwarding (Bypassing)

How to forward to the AND instruction?
Solution 3: HW Forwarding (Bypassing)

ADD \( R1, R2, R3 \)

OR \( R4, R1, R3 \)

SUB \( R5, R2, R1 \)

AND \( R6, R1, R2 \)

ADDI \( R7, R7, 3 \)
Pipeline Modifications for Forwarding?

IF/ID  ID/EX  ALU  EX/MEM  DM  MEM/WB  Reg
Pipelined Microprocessor w/o Forwarding
Pipelined Processor with Forwarding
Forwarding in Action

ADD R1, R2, R3

OR R4, R1, R3
Forwarding in Action

ADD $R1, R2, R3$

OR $R4, R1, R3$

ADD $R1, R2, R3$
HW Forwarding

Trade-off between performance and cost
Example: Data Hazards w/o Forwarding

- Identify all data hazards in the following instruction sequences by circling each source register that is read before the updated value is written back.

  ADD R1, R2, R3
  OR R4, R1, R3
  SUB R5, R2, R1
  AND R6, R1, R2
Example: Data Hazards w/ Forwarding

- Identify all data hazards in the following instruction sequences by circling each source register that is read before the updated value is written back

  ADD R1, R2, R3
  OR R4, R1, R3
  SUB R5, R2, R1
  AND R6, R1, R2

Data hazards resolved by R-type to R-type forwarding
Another Example: Data Hazards w/ Forwarding

- Identify all data hazards in the following instruction sequences by circling each source register that is read before the updated value is written back

  LW R1, 0(R2)
  OR R4, R1, R3
  SUB R5, R2, R1

Data hazard not resolved by R-type to R-type forwarding
Data Hazards Caused by Load

LW $R1, 0(R2)$

OR $R4, R1, R3$

SUB $R5, R2, R1$

AND $R6, R1, R2$

ADDI $R7, R7, 3$
Load Instructions and Forwarding

LW $R1, 0(R2)$

OR $R4, R1, R3$

SUB $R5, R2, R1$

AND $R6, R1, R2$

ADDI $R7, R7, 3$
Solution 1: Compiler Inserts NOP Instruction

LW R1, 0(R2)

NOP

OR R4, R1, R3

SUB R5, R2, R1

AND R6, R1, R2
Solution 2: HW Stalls the Pipeline

LW R1, 0(R2)

OR R4, R1, R3

SUB R5, R2, R1

AND R6, R1, R2

ADDI R7, R7, 3
Solution 3: Delay Slots

• A *delay slot* is a location in the program where the compiler is *required* to insert an instruction between dependent instructions.

• The ISA defines the delay slots.

• The compiler can fill delay slots with NOPs.

• Even better: Move a non-dependent instruction from elsewhere in the program into the delay slot. – Doing so must not change the function of the program.
Example 1: Filling the Load Delay Slot

```plaintext
LW R1, 0(R2)
ADDI R7, R7, 3
OR R4, R1, R3
SUB R5, R2, R1
AND R6, R1, R2
ADDI R7, R7, 3
```
Example 2: Filling the Load Delay Slot?

LW R1, 0(R2)

NOP

OR R4, R1, R3

SUB R5, R2, R1

AND R6, R1, R2

ADDI R7, R6, 3

The ADDI instruction cannot be moved to the delay slot due to data dependence on AND
The Problem with Branches

If the condition is met, the PC is updated at the end of EX, after we’ve already fetched the next two instructions.
The Problem with Branches

BEQ R2, R3, X
OR R4, R1, R3
SUB R5, R2, R1
X: AND R6, R1, R2
ADDI R7, R7, 3
... 

OR and SUB are fetched before the branch condition is evaluated
Control Hazard

- Occurs when instructions following a branch are fetched before the branch outcome is known

BEQ R2, R3, X
OR R4, R1, R3
SUB R5, R2, R1
X: AND R6, R1, R2

- What should happen
  - If branch is not taken, next fetched instruction should be at address PC+2 (OR)
  - If branch is taken, next fetched instruction should be at address X (AND)

- What actually happens
  - Instructions at PC+2 and PC+4 are fetched before branch outcome is known
Reducing the Branch Delay

• We already calculate the branch target in ID

• Put dedicated hardware to also evaluate the condition in ID
Evaluating Branch Condition in ID Stage

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Format</th>
<th>OP</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEQ rt,rs,target</td>
<td>I</td>
<td>1000</td>
<td>if(R[rs] == R[rt]) PC = PC + sext({imm,1'b0})</td>
</tr>
<tr>
<td>BNE rt,rs,target</td>
<td>I</td>
<td>1001</td>
<td>if(R[rs] ≠ R[rt]) PC = PC + sext({imm,1'b0})</td>
</tr>
<tr>
<td>BGEZ rs,target</td>
<td>I</td>
<td>1010</td>
<td>if(R[rs] ≥ 0) PC = PC + sext({imm,1'b0})</td>
</tr>
<tr>
<td>BLTZ rs,target</td>
<td>I</td>
<td>1011</td>
<td>if(R[rs] &lt; 0) PC = PC + sext({imm,1'b0})</td>
</tr>
</tbody>
</table>
Branch Delay Slot

• If the ISA defines a branch delay slot, the instruction immediately following a branch is always executed after the branch

• The compiler finds an instruction to put there, or puts in a NOP

• The hardware must execute the instruction immediately following the branch, regardless of whether the branch is taken or not
Filling the Branch Delay Slot with a NOP

BEQ R2, R3, X
NOP
OR R4, R1, R3
SUB R5, R2, R1
X: AND R6, R1, R2
ADDI R7, R7, 3
...

Lecture 18: 33
Filling the Branch Delay Slot

BEQ R2, R3, X
ADDI R7, R7, 3
OR R4, R1, R3
SUB R5, R2, R1
X: AND R6, R1, R2

The ADDI is *always* executed after the BEQ. If the BEQ is taken, executing the ADDI must not cause incorrect behavior.
Branch Target Address (PC+2+OFF)

Branch delay slot is accounted for in the branch target calculation
Before Next Class

Next Time

More Pipelined Microprocessor