ECE 2300
Digital Logic & Computer Organization
Spring 2018

More Pipelined Microprocessor
Announcements

• No instructor office hour today
  – Rescheduled to Monday April 16, 4:00-5:30pm

• Prelim 2 review sessions
  – Friday April 13, 4:30-6:00pm, PHL 219
  – Monday April 16, 7:00-8:30pm, PHL 203
Pipelined Microprocessor

Key idea: Keep all resources fully utilized
The OR, SUB, and AND instructions are data dependent on the ADD instruction.
Example: Data Hazards

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

  SUB R4, R2, R3
  ADDI R1, R1, 1
  ADDI R2, R4, 1
  ADDI R3, R4, 1
  SUB R5, R3, R4
Review: Compiler Inserts NOPs (Solution 1)

ADD R1, R2, R3

NOP

NOP

NOP

OR R4, R1, R3
Review: HW Stalls the Pipeline (Solution 2)

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

- **ADD** $R1, R2, R3$
  - Instruction Memory (IM) → Register (Reg) → ALU
  - ALU output to Data Memory (DM) → Register (Reg)

- **OR** $R4, R1, R3$
  - Instruction Memory (IM) → Register (Reg) → ALU
  - ALU output to Data Memory (DM) → Register (Reg)

- **SUB** $R5, R2, R1$
  - Instruction Memory (IM) → Register (Reg) → ALU
  - ALU output to Data Memory (DM) → Register (Reg)

- **AND** $R6, R1, R2$
  - Instruction Memory (IM) → Register (Reg) → ALU
  - ALU output to Data Memory (DM) → Register (Reg)

- **ADDI** $R7, R7, 3$
  - Instruction Memory (IM) → Register (Reg) → ALU
  - ALU output to Data Memory (DM) → Register (Reg)
Pipeline Modifications for Forwarding?
Pipelined Microprocessor w/o Forwarding
Pipelined Processor with Forwarding

Lecture 18: 11
Forwarding in Action

Control Signals

Adder

IF/ID

Decoder

RD
LD
SA
SB
DR
D_in

RF

ID/EX

OR R4, R1, R3

ADD R1, R2, R3

MEM/WB

ALU

Data RAM

EX/MEM

OR R4, R1, R3

ADD R1, R2, R3

OR R4, R1, R3

ADD R1, R2, R3

Lecture 18: 12
Forwarding in Action

SUB R5, R2, R1
OR R4, R1, R3
ADD R1, R2, R3
Forwarding in Action

```
AND R6, R1, R2
SUB R5, R2, R1
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
  SUB 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
  
  \[
  \text{ADD R1, R2, R3} \\
  \text{OR R4, R1, R3} \\
  \text{SUB R5, R2, R1} \\
  \text{SUB R6, R1, R2}
  \]

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
  
  \[
  \begin{align*}
  &\text{LW R1, 0(R2)} \\
  &\text{OR R4, }\text{\( \circ \)}\text{R1, R3} \\
  &\text{SUB R5, R2, R1}
  \end{align*}
  \]

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

Lecture 18: 20
Solution 1: Compiler Inserts NOP Instruction

LW R1, 0(R2)
LW R1, 0(R2)

NOP
NOP

OR R4, R1, R3
OR R4, R1, R3

SUB R5, R2, R1
SUB R5, R2, R1

AND R6, R1, R2
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.
Filling the Load Delay Slot

LW R1, 0(R2)

ADDI R7, R7, 3

OR R4, R1, R3

SUB R5, R2, R1

AND R6, R1, R2

ADDI R7, R7, 3
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 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
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.
Reducing the Branch Delay

• We already calculate the branch target in ID

• Put dedicated hardware to also evaluate the condition in ID
  – Hence only 1 branch delay slot needed
Evaluating Branch Condition in ID Control Signals

<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>

Lecture 19: 20
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

...
Filling the Branch Delay Slot

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