ECE 2300
Digital Logic & Computer Organization
Spring 2024

More Pipelined Microprocessor
Announcements

• Prelim 2 tonight in PHL 101, 7:30-9:00pm
  – Arrive early by 7:25pm

• Prelab 4a due Monday
Review: Pipelined Processor w/ Forwarding
Review: Data Hazards with Load

• 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
Review: HW Forwarding for Load

LW R1, 0(R2)

OR R4, R1, R3

SUB R5, R2, R1

AND R6, R1, R2

ADDI R7, R7, 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

  IF | ID | ALU | MEM | WB
  IF | ID | ALU | MEM | WB
  IF | ID | ALU | MEM | WB

• 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

- **CU** (Control Unit)
- **ADD** (Adder)
- **RF** (Register File)
- **Decoder**
- **IF/ID** (Instruction Fetch/Instruction Decode)
- **ID/EX** (Instruction Execution)
- **EX/MEM** (Execution to Memory)
- **MEM/WB** (Memory to Write Back)

#### Instruction Format and Operation

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Format</th>
<th>OP</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BEQ rt,rs,target</strong></td>
<td>I</td>
<td>1000</td>
<td>(\text{if}(R[rs] == R[rt]) \text{PC} = \text{PC} + \text{sext}({\text{imm},1'b0}))</td>
</tr>
<tr>
<td><strong>BNE rt,rs,target</strong></td>
<td>I</td>
<td>1001</td>
<td>(\text{if}(R[rs] \neq R[rt]) \text{PC} = \text{PC} + \text{sext}({\text{imm},1'b0}))</td>
</tr>
<tr>
<td><strong>BGEZ rs,target</strong></td>
<td>I</td>
<td>1010</td>
<td>(\text{if}(R[rs] \geq 0) \text{PC} = \text{PC} + \text{sext}({\text{imm},1'b0}))</td>
</tr>
<tr>
<td><strong>BLTZ rs,target</strong></td>
<td>I</td>
<td>1011</td>
<td>(\text{if}(R[rs] &lt; 0) \text{PC} = \text{PC} + \text{sext}({\text{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
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
ADDI R7, R7, 3

The ADDI is always executed after the BEQ.
If the BEQ is taken, executing the ADDI must not cause incorrect behavior.
Exercise: Filling Branch Delay Slot

BEQ R2,R3,X
NOP
OR R4,R1,R3

X: AND R6,R1,R2
ADD R5,R6,R4
ADDI R7,R2,3
SUB R6,R2,R2
...}

• Can you identify one instruction to fill the branch delay slot?
Pipeline Control Requirements

• Generate control signals for each stage
  – IF: PCJ
  – EX: MB, F
  – MEM: MW, MD
  – WB: LD

• Detect forwarding conditions and generate MUX control signals

• Detect data hazards and insert pipeline bubbles
  – Assume **no load delay slot** defined in ISA

• Assume **one branch delay slot** defined in ISA
Generating Control for Each Stage
Forwarding Unit (Partial)

MUX controls
- ID.SA
- ID.SB
- ID.R-type
- EX.SA
- EX.SB
- EX.R-type

Forwarding Unit
- MEM.R-type
- MEM.DR
- WB.R-type
- WB.DR

CU
- =?

Adder
- sign bit

IF/ID
- PCJ
- PCL

Inst RAM
- +2

Decoder
- PC
- RAM
- Inst

SE
- D_in

RF
- LD
- SA
- SB
- DR

ID/EX
- ID.R
- ID.SA
- ID.SB

EX/MEM
- EX.R
- EX.SA
- EX.SB

MEM/WB
- MEM
- F

Data RAM
- D_IN

ALU
- MB
- F_m ... F_0

LD
- MW
- MD

MUX controls for ALU and Memory

Lecture 19: 18
R-type to R-type Forwarding

**Forwarding path** Conditions to enable the forwarding

1. **MEM→EX**: MEM.DR == (EX.SA or EX.SB)
2. **WB→EX**: (1) WB.DR == (EX.SA or EX.SB) and (2) there is no forwarding from MEM→EX in this cycle
3. **WB→ID**: WB.DR == (ID.SA or ID.SB)
Another Example: R-type to R-type Forwarding

Forwarding path from ADD to SUB (via WB→EX) must be disabled so the updated R1 produced by OR is forwarded to SUB (via MEM→EX)
Summary: R-type to R-type Forwarding Conditions

1. **MEM → EX**
   - MEM.DR == (EX.SA or EX.SB)

2. **WB → EX**
   - WB.DR == (EX.SA or EX.SB) \textit{and}
   - MEM.DR != (EX.SA or EX.SB)

3. **WB → ID**
   - WB.DR == (ID.SA or ID.SB)
R-type to Branch Forwarding

ADD R1, R2, R3
OR R4, R6, R3
BGEZ R1, X
AND R6, R5, R2
X: ADDI R7, R7, 3

- There are other forwarding conditions to handle as well (such as WB → ID)
Pipeline with Fwding + Branch HW in ID

AND R6, R5, R2
BGEZ R1, X
OR R4, R6, R3
ADD R1, R2, R3
Data Hazards Requiring Bubbles

- Occur when instructions are too close together for forwarding to work
- Requires adding bubbles in the pipeline
- Data hazard conditions to detect and handle
  - Load followed by R-type
  - Load followed by I-type ALU instruction
  - Load followed by Load
  - Load followed by Store (two cases)
  - Load followed by Branch
  - ALU instruction followed by Branch
Load Followed by R-type Instruction

if (EX.Load && ID.R-type) {
  if (EX.DR == (ID.SA || ID.SB)) {
    Insert NOP into EX in next cycle // Insert bubble
    Don’t Load IF/ID in next cycle // Hold instruction in ID for a cycle
    Don’t Load PC in next cycle // Hold instruction in IF for a cycle
  }
}

LW R1, 0(R2)

OR R4, R1, R3
Load Followed by R-type Instruction

Lecture 19: 26
Load Followed by R-type Instruction

OR R4, R1, R3  LW  R1, 0(R2)
Load Followed by R-type Instruction

IM
LW R1,0(R2)
OR R4,R1,R3
SUB R5,R2,R1
AND R6,R1,R2

Reg

ALU

DM

Reg

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

Detect hazard
Load Followed by R-type Instruction

IM
LW R1, 0(R2)
OR R4, R1, R3
SUB R5, R2, R1
AND R6, R1, R2

Reg

ALU

DM

Reg

LW R1, 0(R2)

OR R4, R1, R3

Hold these instructions

SUB R5, R2, R1

NOP

Insert bubble
Load Followed by R-type Instruction

Pipeline continues normally
R1 value forwarded from WB to EX and ID
'Clear' puts 0 into ID/EX to create a bubble (or NOP)

'PCL' and 'IF/IDL': write enables for the PC and IF/ID registers
Next Time

- H&H 8-8.3