Review Problem 31

- What registers are being read and written in the 5th cycle of a pipelined CPU running this code?

```
ADD x1, x2, x3
ORR x4, x5, x6
SUB x7, x8, x9
EOR x10, x11, x12
AND x13, x14, x15
```

Writes x1

Reads x11, x12
Solution #3: Branch Delay Slot

Redefine branches: Instruction directly after branch always executed
Instruction after branch is the delay slot

Compiler/assembler fills the delay slot

ADD X1, X0, X4
CBZ X2, FOO
ADD X1, X0, X4
SUB X2, X0, X3
ADD X1, X0, X4
CBZ X1, FOO
SUB X2, X0, X3
ADD X1, X0, X4
CBZ X1, FOO
ADD X3, X3, X3
FOO:
ADD X1, X2, X0
ADD X1, X0, X4
CBZ X1, FOO
ADD X3, X3, X3

No wasted cycles
No wasted cycles
Either branch will either branch will be executed
waste 1 cycle if prediction is wrong

Insert a nop.
80% non-branch

80% branch

50% / 50% Br. taken or not

guess is correct → No waste

guess is wrong

\[ CPI = 0.8 \times 1 + 0.2(0.5 \times 1 + 0.5 \times 2) \]

\[ = 0.8 + 0.2 \times 1.5 \]

\[ = 0.8 + 0.3 = 1.1 \]
Data Hazards

Consider the following code:

ADD X0, X1, X2
SUB X3, X0, X4
AND X5, X0, X6
ORR X7, X0, X8
EOR X9, X0, X10
Design Register File Carefully

What if reads see value after write during the same cycle?

- ADD \( X0, X1, X2 \)
- SUB \( X3, X0, X4 \)
- AND \( X5, X0, X6 \)
- ORR \( X7, X0, X8 \)
- EOR \( X9, X0, X10 \)

<table>
<thead>
<tr>
<th>Clock</th>
<th>Cycle 1</th>
<th>Cycle 2</th>
<th>Cycle 3</th>
<th>Cycle 4</th>
<th>Cycle 5</th>
<th>Cycle 6</th>
<th>Cycle 7</th>
<th>Cycle 8</th>
<th>Cycle 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>X</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>X</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORR</td>
<td>√</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td></td>
<td></td>
<td>Wr</td>
<td></td>
</tr>
<tr>
<td>EOR</td>
<td>√</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td></td>
<td></td>
<td></td>
<td>Wr</td>
</tr>
</tbody>
</table>
Forwarding

Add logic to pass last two values from ALU output to ALU input(s) as needed

Forward the ALU output to later instructions

ADD $X0, X1, X2
SUB $X3, $X0, X4
AND $X5, $X0, X6
ORR $X7, $X0, X8
EOR $X9, $X0, X10
Forwarding (cont.)

Requires values from last two ALU operations. Remember destination register for operation. Compare sources of current instruction to destinations of previous 2.

Current instr: ADD xo, X3, X31

Any instr that don't write the register?
Data Hazards on Loads

LDUR \texttt{X0, [X31, 0]}
SUB \texttt{X3, X0, X4}
AND \texttt{X5, X0, X6}
ORR \texttt{X7, X0, X8}
EOR \texttt{X9, X0, X10}

<table>
<thead>
<tr>
<th>Cycle 1</th>
<th>Cycle 2</th>
<th>Cycle 3</th>
<th>Cycle 4</th>
<th>Cycle 5</th>
<th>Cycle 6</th>
<th>Cycle 7</th>
<th>Cycle 8</th>
<th>Cycle 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDUR</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\sqrt{AND}</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORR</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EOR</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Hazards on Loads (cont.)

Solution:

Use same forwarding hardware & register file for hazards 2+ cycles later
Force compiler to not allow register reads within a cycle of load
Fill delay slot, or insert no-op.