Computer Architecture

EE 4720

Final Examination

4 May 2015, 10:00–12:00 CDT

Problem 1  ________  (20 pts)
Problem 2  ________  (20 pts)
Problem 3  ________  (20 pts)
Problem 4  ________  (20 pts)
Problem 5  ________  (20 pts)

Exam Total  ________  (100 pts)

Good Luck!
Problem 1: (20 pts) The instruction sequence below performs the calculation \( r2+3+r5 \) and requires two MIPS instructions to do so. If these two instructions were in the same fetch group in a two-way superscalar processor then the second one would stall, this is shown in the execution below.

\[
\begin{align*}
\text{addi } r1, r2, 3 & \quad \text{IF ID EX ME WB} \\
\text{add } r4, r1, r5 & \quad \text{IF ID -> EX ME WB}
\end{align*}
\]

An FA-ALU has two regular inputs, and a third input reserved for small quantities, here for values < 16, and it can produce a result in the same time as an ordinary two-input ALU. An FA-ALU can be used to execute instruction pairs like the one above without a stall, call such an execution a fused add.

Illustrated below is a 2-way superscalar implementation with an FA-ALU. Most bypass paths have been removed to provide space for the solution to this problem. In addition to a 4-bit input for the third operand, the FA-ALU has a control input, \( FA \), which should be set to 1 when the third input is to be used.

\( \text{(a)} \) Add datapath connections to this implementation so that the FA-ALU can be used for fused adds. \textit{Hint: The datapath changes are not just for that new FA-ALU input.}

\( \square \) Add datapath so that the correct operands are delivered to the FA-ALU for fused adds.

\( \square \) Non-fusable sequences must still execute correctly. (Don’t break existing functionality.)

\( \square \) Make reasonable cost and clock frequency tradeoffs.

\( \text{(b)} \) Add control logic.

\( \square \) Design logic to generate the \( FA \) signal for the FA-ALU and deliver it in the correct cycle.

\( \square \) Consider \( \square \) instruction type, \( \square \) dependencies, and \( \square \) operand size.

\( \square \) Add control signals for the datapath added in the previous part.

\( \square \) Make reasonable cost and clock frequency tradeoffs.
Problem 1, continued: You can use logic blocks such as \texttt{isADD} and \texttt{isADDI} in your solution.

\begin{verbatim}
addi r1, r2, 3  # Sample fusable sequence.
add   r4, r1, r5
\end{verbatim}

Solution appears below. The datapath is in \textcolor{blue}{blue} and control logic is in \textcolor{green}{green}, and an alternative solution appears in \textcolor{purple}{purple}.

The datapath needs to be changed so that the immediate and \texttt{rs} register value from the slot 0 (\texttt{addi}) instruction are fed to the slot 1 FA-ALU. Connecting the immediate is straightforward. Two solutions are shown for the \texttt{rs} register value. In the \textbf{Plan A} solution the slot 1 \texttt{rs} register number is switched to the slot 0 \texttt{rs} register number. That requires just a 5-bit mux. In the not-as-good \textbf{Plan B} solution the value is moved, require an additional 32-bit mux input. Notice that the value is taken at the output of the upper ALU mux, this way a bypassed value can be used. That's possible in the \textbf{Plan A} solution is the output of the ID-stage mux is used for the bypass control logic (which isn't shown).

The control logic, in \textcolor{green}{green}, needs to check for three things: the presence of the two instructions, whether there is a dependence, and whether the immediate will fit in four bits. The solution only works for the arrangement of instructions shown above, and that's all that was needed for full credit. In real life one might want this trick to work for similar sequences, for example if the dependence were carried into the \texttt{rt} rather than the \texttt{rs} register.

\begin{itemize}
\item \textbf{Plan A:} Inexpensive solution: For fused pair replace slot 1's \texttt{rs} register with slot 0's \texttt{rs} register. (For pairs in which the dependence is with slot 1's \texttt{rs} register.)
\item \textbf{Plan B:} Expensive Solution: Add connection for value in \texttt{EX} (rather than register number in \texttt{ID} as is done for Plan A).
\end{itemize}
Problem 2: (20 pts) Show the execution of the code fragments below on the illustrated MIPS implementations. All branches are taken.

(a) Show the execution of the code below on the following implementation. The branch is taken.

Solution appears below.

Common mistakes: squashing the delay slot instruction (the \texttt{and}); not noticing that the branch depends on the \texttt{sub}.

\begin{verbatim}
# SOLUTION
# Cycle  0 1 2 3 4 5 6 7 8 9 10 11 12
add r4, r2, r3 IF ID EX ME WB
lw r6, 8(r4) IF ID EX ME WB
sub r1, r6, r5 IF ID \rightarrow\ EX ME WB
beq r7, r1 TARG IF \rightarrow\ ID ----> EX ME WB
and r8, r7, r10 IF \rightarrow\ ID \rightarrow\ ID EX ME WB
or r11, r12, r13
xor r14, r11, r8
TARG:
sw r1, 0(r2) IF ID EX ME WB
# Cycle  0 1 2 3 4 5 6 7 8 9 10 11 12
\end{verbatim}
(b) Show the execution of the MIPS FP code below on the following implementation. Assume that there is a bypass from WF to M1 and A1.

Solution appears below. Common mistakes: Not realizing that the `lwc1` uses WF (not WB) and therefore it must stall to avoid the structural hazard. Another common mistake is stalling in a stage other than ID.

```plaintext
# SOLUTION
# Cycle       0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
add.s  f6, f2, f3     IF ID A1 A2 A3 A4 WF
add.s  f1, f7, f8     IF ID A1 A2 A3 A4 WF
lwc1   f9, 0(r10)     IF -----> EX ME WF
mul.s  f4, f1, f9     IF -----> ID -> M1 M2 M3 M4 M5 M6 WF
```
(c) Show the execution of the MIPS code below on the following 2-way superscalar implementation. The branch is taken. Fetch groups in this implementation must be 8-byte-aligned.

START: Instruction address of add is 0x1000

<table>
<thead>
<tr>
<th># Cycle</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>add r4, r2, r3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>ME</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw r6, 8(r4)</td>
<td>IF -&gt; ID</td>
<td>-&gt; EX</td>
<td>ME</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub r1, r6, r5</td>
<td>IF -&gt; ID</td>
<td>-&gt; EX</td>
<td>ME</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beq r7, r15 TARG</td>
<td>IF -&gt; ID</td>
<td>-&gt; EX</td>
<td>ME</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>and r8, r7, r10</td>
<td>IF -&gt; ID</td>
<td>EX</td>
<td>ME</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or r11, r12, r13</td>
<td>IFx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xor r14, r11, r8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TAG: Instruction address of sw is 0x2004

<table>
<thead>
<tr>
<th># Cycle</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>sll r16, r17, 8</td>
<td>IFx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw r1, 0(r2)</td>
<td>IF -&gt; ID</td>
<td>-&gt; EX</td>
<td>ME</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lui r15, 0x1234</td>
<td>IF -&gt; ID</td>
<td>EX</td>
<td>ME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Solution appears below. Note that the sll is fetched only because the fetch unit can only fetch an address that's a multiple of 8, but it is squashed as soon as it arrives.