DUE: November 9, 2005

READ:
- Portions of Chapters 5, 6, 7, 8, 9 and 14 of the Sima book and
- Portions of Chapters 3, 4, Appendix A and Appendix G of the Hennessy book

ASSIGNMENT: There are four problems developed from the Hennessy book.

Solve all homework and exam problems as shown in class and past exam solutions.

1) Consider the piece of code, the DAXPY code given in Problem 4.8 of the Hennessy book. Do not solve the problem. Just consider the code. Note about the “DSUBUI” instruction mentioned in previous homework assignments.

Assume that the MIPS is implemented as the 2-way superscalar hardware-speculative Tomasulo algorithm machine as discussed in class. That is, this is machine model number 4. Instructions are issued in order. A pair of instructions is issued per cycle, provided that static issuing is preserved. There are enough number of CDB buses to eliminate bottlenecks. Two instructions can be committed per cycle provided that they are at the head of the ROB. This means two data cache accesses can be made for two store instructions or one load and one store or two loads at the same time, if there are no address conflicts.

Also, assume that there is a perfect memory with no stalls and the functional unit timings are as listed on page A-74 of the Hennessy book: double-precision FP operations ADD.D, MUL.D and DIV.D take 3, 11 and 41 clock periods, respectively. Another assumption is that there are enough functional units for integer instructions not to cause stalls. There is a separate branch unit. A branch instruction takes two clock periods (IF and ID stages) and then stays in the ROB until it determines the condition and target address and is the head of the ROB.

In which clock period, will the first iteration of the Problem 4.8 loop be completed? That is, what is the last clock period in which the Commit stage of an instruction from the first iteration be done last? Also, show which instructions are flushed out of the pipeline. To answer it, continue the following table, without showing the hardware tables:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>WR</th>
<th>CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>F0, 0(R1)</td>
<td>1</td>
<td>2</td>
<td>3-4</td>
<td>5</td>
</tr>
<tr>
<td>MUL.D</td>
<td>F0, F0, F2</td>
<td>1</td>
<td>2</td>
<td>3/5 - 15</td>
<td>16</td>
</tr>
<tr>
<td>...Continue...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Indicate any assumptions made during the execution of the loop, if a situation not discussed in class is encountered.
2) Consider the scalar DAXPY code with 11 instructions on page G-9 of the Hennessy book. It is slightly more complex than the scalar DAXPY code shown in Problem 4.8 of the Hennessy book.

Assume that we have the VLIW MIPS processor as discussed in class. That is, this is machine model number 5. Assume that the latencies are as specified on page 304 of the Hennessy book. Finally, assume that Rx is R1 and Ry is R2. Assume that there are as many vector elements as needed for your unrolling.

Show how you would unroll and issue instructions of this program similar to the VLIW execution sequence given in Figure 4.5 of the Hennessy book.

Note that you will not convert the code on G-9 to a high-level code to optimize it to get more parallelism. Just convert the page G-9 code with 11 instructions to a VLIW code.

3) Solve Problem G.2 (b and c) of the Hennessy book. The processor in this question is VMIPS. That is, this is machine model number 6. Show the execution timings as discussed in class.

In part (b), there is chaining but a single memory pipeline connects the CPU to the memory.

In part (c), there is again chaining, but three memory pipelines are provided. Three separate memory controllers allow up to three simultaneous pipelined memory accesses if the memory addresses do not conflict.

4) Solve Problem G.5 (b) of the Hennessy book. The processor in this question is VMIPS. That is, this is machine model number 6.

To solve G.5(b), first part (a) has to be solved, resulting in two loops. One of the loops would have a dependence and the other one would have no dependence. In part (b), you will convert the loop with no dependence to a VMIPS code! You can use register Ra to point at A[i], and Rb to point at B[i]. Note that the code in the book is in FORTRAN. Its C-like version is as follows:

```c
C = 0.0;
for (i = 1; i <= 64; i++)
    { A[i] = A[i] + B[i];
      C = C + A[i]; }
```

RELEVANT QUESTIONS AND ANSWERS

Q1) Consider the following CS613 MIPS code (machine model # 1) where R1 is with the initial value of \((16)_{10}\).

```
loop : L.D F2, 0(R1) ; Statement S1
    ADD.D F0, F0, F2 ; S2. F0 is initially 0
    DADDI R1, R1, #(-8)_{10} ; S3
    BNEZ R1, loop ; S4
    S.D 0(R2), F0 ; S5
```

This code is now run on the 2-way superscalar, statically issued, dynamically scheduled, speculative MIPS. That is,
this is *machine model number 4*. Instructions are issued in order. A pair of instructions is issued per cycle, provided that static issuing is preserved. There are enough number of CDB buses to eliminate bottlenecks. **Two** instructions can be committed per cycle if they are at the head of the ROB. This means two data cache accesses can be made for two store instructions or one load and one store or two loads at the same time, if there are no address conflicts.

Assume that, the latencies of the functional units are as listed on page 304 of the Hennessy book. In addition, assume that there is a perfect memory with no stalls. Another assumption is that there are enough functional units for integer instructions not to cause stalls. There is a separate branch unit. A branch instruction takes two clock periods (IF and ID stages) and then stays in the ROB until it determines the condition and target address and is the head of the ROB.

Show how the instructions are executed until the loop is completed. When would the execution be complete? Also, show which instructions are flushed out of the pipeline. Indicate any assumptions made during the execution of the loop, if a situation not discussed in class is encountered. To answer the question, continue the table shown below:

<table>
<thead>
<tr>
<th>iter #</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>WR</th>
<th>CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>3-4</td>
</tr>
<tr>
<td>ADD.D</td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>3/5-8</td>
</tr>
</tbody>
</table>

**A1)** The execution of instructions is shown below. The execution timings are obtained based on the following hardware and software properties of the machine model number 4:

a) The BNEZ takes two clock periods to execute so that the condition and the target address are determined as the CS613 MIPS CPU does, but no delay slot is used. For the BNEZ, the clock periods in EX are not needed and we assume the branch condition and target address are determined while the branch is in the ROB.

b) There is a separate branch unit sophisticated enough to make predictions whether a branch has been executed before or not. The branch unit inspects instructions for branch possibilities and starts processing them before they are even in the IF stage. It has hardware tables with prediction bits and its own ALU to make address calculations. If a branch is not executed before, it can make a prediction based on the opcode, the displacement and any input from the compiler. Thus, when the superscalar MIPS encounters the BNEZ instruction in the above loop for the first time, the branch unit decides to take the branch since it is a conditional branch with a negative displacement signaling that it could be a loop-ending branch which is almost always taken, except, for example, the last iteration of the loop.

c) The loop requires two iterations, but the branch unit speculates and decides to have the 3rd iteration and more. But, in the 15th clock period, it is realized that the speculation is wrong and so the ROB is flushed and the instruction that follows the BNEZ in the code is fetched (the S.D) in the 16th clock period.
Q2) Assume that the MIPS is implemented as the 2-way superscalar hardware-speculative Tomasulo algorithm machine as discussed in class. That is, this is machine model number 4.

Instructions are issued in order. Any pair of instructions is issued per cycle, provided that static issuing is preserved. There are enough number of CDB buses to eliminate bottlenecks. Only one instruction can be committed per cycle if it is at the head of the ROB.
Assume that, the latencies of the functional units are as listed on page 304 of the Hennessy book. In addition, assume that there is a perfect memory with no stalls. Another assumption is that there are enough functional units for integer instructions not to cause stalls. There is a separate branch unit. A branch instruction takes two clock periods (IF and ID stages) and then stays in the ROB until it determines the condition and target address and is the head of the ROB.

Given the old MIPS code below (machine model number 0) and that the loop has two (2) iterations, in which clock period will the second iteration of the loop be completed? That is, what is the last clock period in which the Commit stage of an instruction from the second iteration is done? Show which instructions are flushed out of the pipeline. Indicate any assumptions made during the execution of the loop, if a situation not discussed in class is encountered.

A2) The execution of the loop for two iterations for the superscalar case is as follows:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Instruction</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>WR</th>
<th>CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L.D F0, 0(R2)</td>
<td>1</td>
<td>2</td>
<td>3-4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>MUL.D F0, F0, F0</td>
<td>1</td>
<td>2</td>
<td>3/5-8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>L.D F1, 0(R3)</td>
<td>2</td>
<td>3</td>
<td>4-5</td>
<td>6</td>
<td>7/11</td>
</tr>
<tr>
<td>1</td>
<td>L.D F2, 0(R4)</td>
<td>2</td>
<td>3</td>
<td>4-5</td>
<td>6</td>
<td>7/12</td>
</tr>
<tr>
<td>1</td>
<td>MUL.D F3, F1, F2</td>
<td>3</td>
<td>4</td>
<td>5/6-9</td>
<td>10</td>
<td>11/13</td>
</tr>
<tr>
<td>1</td>
<td>ADD.D F4, F0, F3</td>
<td>3</td>
<td>4</td>
<td>5/10-13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>S.D 0(R1), F4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8/16</td>
</tr>
<tr>
<td>1</td>
<td>DADDI R1, R1, #8</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8/17</td>
</tr>
<tr>
<td>1</td>
<td>DADDI R2, R2, #8</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9/18</td>
</tr>
<tr>
<td>1</td>
<td>DADDI R3, R3, #8</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9/19</td>
</tr>
<tr>
<td>1</td>
<td>DADDI R4, R4, #8</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10/20</td>
</tr>
<tr>
<td>1</td>
<td>DADDI R5, R5, #(-1)_{10}</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10/21</td>
</tr>
<tr>
<td>1</td>
<td>BNEZ R5, loop</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
<td>9/22</td>
</tr>
<tr>
<td>Iteration</td>
<td>Instruction</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WR</td>
<td>CM</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------</td>
<td>----</td>
<td>----</td>
<td>------</td>
<td>----</td>
<td>-------</td>
</tr>
<tr>
<td>2</td>
<td>L.D F0, 0(R2)</td>
<td>7</td>
<td>8</td>
<td>9-10</td>
<td>11</td>
<td>12/23</td>
</tr>
<tr>
<td>2</td>
<td>MUL.D F0, F0, F0</td>
<td>8</td>
<td>9</td>
<td>10/11-14</td>
<td>15</td>
<td>16/24</td>
</tr>
<tr>
<td>2</td>
<td>L.D F1, 0(R3)</td>
<td>8</td>
<td>9</td>
<td>10-11</td>
<td>12</td>
<td>13/25</td>
</tr>
<tr>
<td>2</td>
<td>L.D F2, 0(R4)</td>
<td>9</td>
<td>10</td>
<td>11-12</td>
<td>13</td>
<td>14/26</td>
</tr>
<tr>
<td>2</td>
<td>MUL.D F3, F1, F2</td>
<td>9</td>
<td>10</td>
<td>11/13-16</td>
<td>17</td>
<td>18/27</td>
</tr>
<tr>
<td>2</td>
<td>ADD.D F4, F0, F3</td>
<td>10</td>
<td>11</td>
<td>12/17-20</td>
<td>21</td>
<td>22/28</td>
</tr>
<tr>
<td>2</td>
<td>S.D 0(R1), F4</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14/29</td>
</tr>
<tr>
<td>2</td>
<td>DADDI R1, R1, #8</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15/30</td>
</tr>
<tr>
<td>2</td>
<td>DADDI R2, R2, #8</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15/31</td>
</tr>
<tr>
<td>2</td>
<td>DADDI R3, R3, #8</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16/32</td>
</tr>
<tr>
<td>2</td>
<td>DADDI R4, R4, #8</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16/33</td>
</tr>
<tr>
<td>2</td>
<td>DADDI R5, R5, #(-1)10</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17/34</td>
</tr>
<tr>
<td>2</td>
<td>BNEZ R5, loop</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17/35</td>
</tr>
<tr>
<td>3</td>
<td>L.D F0, 0(R2)</td>
<td>14</td>
<td>15</td>
<td>16-17</td>
<td>18</td>
<td>19/35</td>
</tr>
<tr>
<td>3</td>
<td>MUL.D F0, F0, F0</td>
<td>14</td>
<td>15</td>
<td>16/18-21</td>
<td>22</td>
<td>23/35</td>
</tr>
<tr>
<td>3</td>
<td>L.D F1, 0(R3)</td>
<td>15</td>
<td>16</td>
<td>17-18</td>
<td>19</td>
<td>20/35</td>
</tr>
<tr>
<td>3</td>
<td>L.D F2, 0(R4)</td>
<td>15</td>
<td>16</td>
<td>17-18</td>
<td>19</td>
<td>20/35</td>
</tr>
<tr>
<td>3</td>
<td>MUL.D F3, F1, F2</td>
<td>16</td>
<td>17</td>
<td>18/19-22</td>
<td>23</td>
<td>24/35</td>
</tr>
<tr>
<td>3</td>
<td>ADD.D F4, F0, F3</td>
<td>16</td>
<td>17</td>
<td>18/23-26</td>
<td>27</td>
<td>28/35</td>
</tr>
<tr>
<td>3</td>
<td>S.D 0(R1), F4</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21/35</td>
</tr>
<tr>
<td>3</td>
<td>DADDI R1, R1, #8</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21/35</td>
</tr>
<tr>
<td>3</td>
<td>DADDI R2, R2, #8</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22/35</td>
</tr>
<tr>
<td>3</td>
<td>DADDI R3, R3, #8</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22/35</td>
</tr>
<tr>
<td>3</td>
<td>DADDI R4, R4, #8</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23/35</td>
</tr>
<tr>
<td>3</td>
<td>DADDI R5, R5, #(-1)10</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23/35</td>
</tr>
<tr>
<td>3</td>
<td>BNEZ R5, loop</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24/35</td>
</tr>
<tr>
<td>4</td>
<td>L.D F0, 0(R2)</td>
<td>20</td>
<td>21</td>
<td>22-23</td>
<td>24</td>
<td>25/35</td>
</tr>
<tr>
<td>4</td>
<td>MUL.D F0, F0, F0</td>
<td>21</td>
<td>22</td>
<td>23/24-27</td>
<td>28</td>
<td>29/35</td>
</tr>
<tr>
<td>4</td>
<td>L.D F1, 0(R3)</td>
<td>21</td>
<td>22</td>
<td>23-24</td>
<td>25</td>
<td>26/35</td>
</tr>
<tr>
<td>4</td>
<td>L.D F2, 0(R4)</td>
<td>22</td>
<td>23</td>
<td>24-25</td>
<td>26</td>
<td>27/35</td>
</tr>
</tbody>
</table>

These 44 instructions are flushed out at the end of the 35th clock period.
The two iterations of the loop take 35 clock periods to complete! 44 instructions are flushed out of the pipeline, as opposed to 11 in the scalar case.

The superscalar case shows that considerable amount of unfinished work is wasted when the loop completes. This is

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Instruction</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>WR</th>
<th>CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>MUL.D F3, F1, F2</td>
<td>22</td>
<td>23</td>
<td>24/26-29</td>
<td>30</td>
<td>31/35</td>
</tr>
<tr>
<td>4</td>
<td>ADD.D F4, F0, F3</td>
<td>23</td>
<td>24</td>
<td>25/30-33</td>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>S.D 0(R1), F4</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27/35</td>
</tr>
<tr>
<td>4</td>
<td>DADDI R1, R1, #8</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28/35</td>
</tr>
<tr>
<td>4</td>
<td>DADDI R2, R2, #8</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28/35</td>
</tr>
<tr>
<td>4</td>
<td>DADDI R3, R3, #8</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29/35</td>
</tr>
<tr>
<td>4</td>
<td>DADDI R4, R4, #8</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29/35</td>
</tr>
<tr>
<td>4</td>
<td>DADDI R5, R5, #(-1)_10</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30/35</td>
</tr>
<tr>
<td>4</td>
<td>BNEZ R5, loop</td>
<td>26</td>
<td>27</td>
<td></td>
<td></td>
<td>28/35</td>
</tr>
<tr>
<td>5</td>
<td>L.D F0, 0(R2)</td>
<td>27</td>
<td>28</td>
<td>29-30</td>
<td>31</td>
<td>32/35</td>
</tr>
<tr>
<td>5</td>
<td>MUL.D F0, F0, F0</td>
<td>27</td>
<td>28</td>
<td>29/31-34</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>L.D F1, 0(R3)</td>
<td>28</td>
<td>29</td>
<td>30-31</td>
<td>32</td>
<td>33/35</td>
</tr>
<tr>
<td>5</td>
<td>L.D F2, 0(R4)</td>
<td>28</td>
<td>29</td>
<td>30-31</td>
<td>32</td>
<td>33/35</td>
</tr>
<tr>
<td>5</td>
<td>MUL.D F3, F1, F2</td>
<td>29</td>
<td>30</td>
<td>31/32-35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ADD.D F4, F0, F3</td>
<td>29</td>
<td>30</td>
<td>31/35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>S.D 0(R1), F4</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34/35</td>
</tr>
<tr>
<td>5</td>
<td>DADDI R1, R1, #8</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34/35</td>
</tr>
<tr>
<td>5</td>
<td>DADDI R2, R2, #8</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>DADDI R3, R3, #8</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>DADDI R4, R4, #8</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>DADDI R5, R5, #(-1)_10</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>BNEZ R5, loop</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>L.D F0, 0(R2)</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>MUL.D F0, F0, F0</td>
<td>34</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>L.D F1, 0(R3)</td>
<td>34</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>L.D F2, 0(R4)</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>MUL.D F3, F1, F2</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
because multiple issue is used to increase the ILP and it takes considerable time to determine the misspeculation due to committing one instruction per cycle. Thus, committing multiple instructions to match the issue rate is needed. Wasted instructions also means that the memory hierarchy is accessed unnecessarily. This situation would be worse if the FP latencies were longer than 4 clock periods. Thus, an important aspect of superscalar design is to restrain ILP in order not to pressure the memory hierarchy which can become a bottleneck eventually. However, in the current climate of higher and higher ILP rates, such thinking is not a high priority on the list.

Note that the superscalar execution assumes that there can be two data cache accesses at the same time. For example in the first iteration of the loop, the second and third L.D instructions access the data cache in clock period 5. Since we decided to issue any two instructions at the same, it is plausible to assume that the communication with the data cache has been correspondingly improved to allow two simultaneous accesses.

Overall, we see that speeds of various sections of the pipeline must match to take the full advantage of the new hardware.

Finally, running the old code on the new processor is slow. A contemporary compiler would have to move the DADDI instructions up between FP instructions.

Q3) A 2-way superscalar MIPS with static issuing and hardware based speculative execution has been designed in class. This is machine model number 4. You are asked to modify it to have dynamic issuing. Assume that the latencies are as mentioned on page 304 of the Hennessy book.

How does your design work? Does dynamic issuing really help the superscalar MIPS? Elaborate on this by showing cases where sequences of instructions clearly make use of the new issue policy of the MIPS. Be very specific. Is the pressure on the compiler decreased or increased?

A3) In dynamic issuing if an instruction is stalled in the instruction queue of the ID stage, the instruction(s) behind it can be issued, bypassing the stalled instruction.

For the statically issued superscalar MIPS, two instructions are analyzed per clock period in the ID stage: I1 and I2. I1 is stalled if there is no entry in its reservation station and/or no slot in the ROB. I2 is stalled if there is no entry in its reservation station and/or no slot in the ROB. I2 is stalled if I1 is stalled.

We will keep the basic structure used for the superscalar MIPS for easier understanding of the discussion below. The new MIPS still issues two instructions from the instruction queue to the reservation stations and to the reorder buffer per clock period, by keeping in mind that

i) we should be able to recover from wrong speculations,
ii) keep precise interrupts and
iii) detect all potential hazards between an instruction that is issued out-of-order and the instruction(s) it bypasses.

There are four questions that we have to deal with in dynamic issuing:

a) how do we pick instructions to bypass stalled instructions in the ID stage?
b) where do we keep bypassed instructions?
c) how do we keep track of bypassed instructions?
d) how do we ensure in-order completion?

Question (a): If static issuing is used, the first instruction of the two instructions, I1, is stalled, I2 is also stalled. In dynamic issuing, I2 and I3 can be issued as long as there is no structural hazard associated with them:
Note that if say, I3, cannot be issued with I2, I4 can be issued or I5 or I6. Thus, the ID stage has to have a large “window” of instructions to analyze per clock period, making the ID stage very complex. For the rest of the discussion of this problem, assume that I1 cannot be issued, but I2 and I3 can be issued.

Question (b): one solution is that we move stalled instructions out of the ID stage (out of the instruction queue) and to the ROB, but treat them as unissued!!! Since I2 and I3 are issued we move them to both the reorder buffer and their reservation stations. Keeping stalled instructions in the reorder buffer requires that the reorder buffer be larger now to allow more instructions:

Question (c): to “remember” that there are stalled instructions in the ROB a linked list in hardware can be used. What happens if there are a number of stalled instructions in the ROB “that can be issued” in addition to the “issuable” instructions in the instruction queue? We have to give priority to the stalled instructions in the ROB and so we try to move (issue) two instructions from the ROB to the reservation stations. If we cannot, we try to move one instruction from the instruction queue and one from the ROB to the reservation stations. In another scenario, two instructions are issued from the instruction queue. When we move an instruction from the ROB to a reservation station, we “delink” it from the linked list and treat it as issued. The ROB this way satisfies correct speculative execution and precise interrupts. But, the CM stage is much more complex now: we must be able to write from the ROB to the reservation stations after checking that there is no resource dependency, i.e., there is a free reservation station.

Question (d): In-order completion is ensured by retiring only those instructions that are at the head of the ROB which are in the order they were stored. Recovery from wrong speculations is done when an incorrectly speculated branch reaches the head of the queue. The instructions behind the branch are flushed out of the ROB. It is possible that some of the flushed instructions are unissued instructions.... Handling precise interrupts is as before: when an instruction reaches the head of the queue its exception is handled. Our solution implies that the CPU will issue two instructions every clock period as long as the ROB is not full. It is guaranteed! One would try to keep the ROB not filled up, by trying to commit more than two instructions per clock period.

Dynamic issuing helps if there are long-latency instructions with true dependencies among them following each other in the code. In static issuing, after early instructions are issued to their reservation stations, they wait there a long time since they wait for long latency instructions. This quickly results in full reservation station. Structural hazards gradually develop, forcing instructions in the ID stage to stall. This means independent instructions behind the stalled instructions in the ID stage stall. Note that this situation can happen if the CPU is running an old code. The pressure on the compiler is reduced now since the hardware looks for independent instructions and arbitrarily searches ahead as long as there is space in the reorder buffer. The compiler does not have to do a complete global analysis of the program for best possible sequence of independent instructions. The compiler can be old.
**Q4)** Write the scalar MIPS code of CS613 for the following piece of high-level code:

\[
\begin{align*}
k &= 0.0 \\
\text{for (i = 1 ; i<= 64 ; i++)} \\
k &= k + A[i] * B[i]
\end{align*}
\]

Then, assuming that the MIPS is a VLIW machine as described in class. That is, this is machine model number 5. Show how the instructions are issued for each clock period. Your code will be based on the latencies as specified on page 304 of the Hennessy book.

**A4)** This code implements the dot product. The scalar code for this piece of high-level language code is as follows:

```
L.D       F0, #0

loop :   L.D       F2, 0(R8) ; R8 points at the end of vector A
         L.D       F4, 0(R9) ; R9 points at the end of vector B
         DADDI     R9, R9, #(-8)_{10}
         MUL.D     F6, F2, F4
         NOP
         DADDI     R8, R8, #(-8)_{10}
         BNEZ      R8, loop
         ADD.D     F0, F0, F6
         NOP
         NOP

S.D       0(R10), F0 ; R10 points at scalar “k”
```

We unroll the loop eight times in order to have as many operations as possible during the VLIW execution. After the loop is exited we have to do more computations: four more instructions (three adds and a store) as shown on the next page. Though, it seems there are a lot of NOPs after we exit the loop, it is **not** the case: instructions out of the original loop are placed around these four instructions.

Note also the importance of having many registers in the architecture for effective VLIW execution. In this case, having many FP registers helps unroll the loop eight times:

<table>
<thead>
<tr>
<th>Mem 1</th>
<th>Mem 2</th>
<th>FP 1</th>
<th>FP 2</th>
<th>Int/Br</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>L.D F2, #0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D F4, #0</td>
<td>L.D F6, #0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D F8, 0(R8)</td>
<td>L.D F10, 0(R9)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D F12, (-8)_{10}(R8)</td>
<td>L.D F14, (-8)_{10}(R9)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D F16, (-16)_{10}(R8)</td>
<td>L.D F18, (-16)_{10}(R9)</td>
<td>MUL.D F8, F8,F10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D F20, (-24)_{10}(R8)</td>
<td>L.D F22, (-24)_{10}(R9)</td>
<td>MUL.D F12, F12, F14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D F24, (-32)_{10}(R8)</td>
<td>L.D F26, (-32)_{10}(R9)</td>
<td>MUL.D F16, F16, F18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D F28, (-40)_{10}(R8)</td>
<td>L.D F30, (-40)_{10}(R9)</td>
<td>MUL.D F20, F20, F22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D F8, (-48)_{10}(R8)</td>
<td>L.D F10, (-48)_{10}(R9)</td>
<td>MUL.D F24, F24, F26</td>
<td>ADD.D F0, F0, F8</td>
<td></td>
</tr>
<tr>
<td>L.D F12, (-54)_{10}(R8)</td>
<td>L.D F14, (-54)_{10}(R9)</td>
<td>MUL.D F28, F28, F30</td>
<td>ADD.D F2, F2, F12</td>
<td></td>
</tr>
</tbody>
</table>
Q5) Consider the following code:

\[ \text{k} = 0.0 ; \]
\[ \text{for } (i = 1 ; i \leq 64 ; i++) \]
\[ \text{k} = \text{k} + A[i] \times B[i] \]

a) Write the above code in terms of VMIPS instructions. That is, this is machine model number 6. Indicate the approximate execution time for your code, assuming that there is no chaining, the MIPS is 2-way superscalar with scalar FP and scalar integer pipeline latencies are as given on page 304 of the Hennessy book and the branch latency is 1, except that the L.D and S.D take 12 clock periods each. Indicate any assumptions made during the execution of the loop, if a situation not discussed in class is encountered.

b) Then, invent and describe new vector instructions for the VMIPS to speed up the processor and mention the new approximate execution time. Is there any major hardware change in the VMIPS now? Indicate any assumptions made during the execution of the loop, if a situation not discussed in class is encountered.

A5) a) This piece of code implements the “dot product. We will use the loop fission technique mentioned in question G.5 of the Hennessy book to obtain the machine code:
```c
k = 0.0;
for (i = 1 ; i <= 64 ; i++)
    dot[i] = A[i] * B[i];
for (i = 1 ; i <= 64 ; i++)
k = k + dot[i];
```

The VMIPS code is as follows:

```
LV    V1, Ra ; time : 12 + 63
LV    V2, Rb ; 12 + 63
MULV.D V3, V2, V1 ; 7 + 63
SV    Rdot, V3 ; 12 + 63
L.D    F0, #0 ; 12

loop :
L.D    F2, 0(Rd) ; instructions
DADDI  R8, R8, #(-8) ; in the loop
ADD.D  F0, F0, F2 ; take
BNEZ   Rd, loop ; 3 * 64
S.D    Rk, F0 ; 12 + 1 (ADD.D to S.D latency delay)
```

“Rd” is for the dot vector just like the Rdot register is. But, Rd initially points at the last element of the dot vector, not at the first element of the dot vector. The “L.D F0, #0” instruction takes 12 clock periods then the instructions in the loop below it are completed every clock period after. Note that in this second loop, there is no vector instruction !!! The timing assumes there is no chaining. The approximate number of clock periods is 512.

Note that the same application (dot-product) is studied in the previous Past Exam question (Q3 and A3 above) for the VLIW MIPS. There, the approximate execution time is 132 clock periods. Why such a large difference? First, the latencies of VLIW instructions are shorter than those of VMIPS instructions. Second, for the VMIPS, the lack of chaining increases the execution time by limiting the overlapping of vector instructions. Third, the lack of multiple memory pipelines limits simultaneous independent vector loads and stores to proceed. Finally, the lack of overlapping of vector instruction executions with the scalar instruction executions in the loop forces the loop to start after all vector instructions are completed. If these limitations are removed, which would be the case for a supercomputer, the approximate execution time for the VMIPS would be approximately 245 clock periods.

Still the time is not better than the VLIW case !! However, the VLIW case suffers from two problems not observed in the VMIPS case and can make the VLIW slower: the VLIW code is large with 3240 bits (which also includes instructions not related to the dot product operation). The VMIPS code is 384 bits. The second and more important problem is that the memory interface unit of VLIW machines is not as sophisticated as that of vector machines. Thus, there will be a lot of idle clock periods for fetching so many VLIW instruction bits and also performing two parallel data accesses per clock period. The VLIW execution time would be worse than 132 clock periods...

**b)** The dot-product is a “reduction” operation where two vectors (A and B) are reduced to a scalar number “k.” So, we devise a new VMIPS vector instruction called DPV (Dot Product of Vectors):

```
DPV    F0, V1, V2 ; F0 <--- V1 * V2
```

Assume that the latency for the DPV is 10 clock periods, since after the multiplication an addition is needed. Then the dot product program becomes:

```
LV    V1, Ra ; time : 12 + 63
LV    V2, Rb ; 12 + 63
DPV   F0, V1, V2 ; 10 + 63
S.D   Rk, F0 ; 12
```

The approximate time is 235 clock periods.

The speedup is 2.18
The hardware is changed such that the vector FP multiplier unit is chained to the scalar FP add unit (which stores in a
calar FP register). The chaining requires a new bus, a Vector-Scalar, VS bus, from the vector multiplier to the scalar
FP Add functional unit. In the context of speculative superscalar MIPS execution, this means a reservation station
entry for the DPV in the integer section. That reservation station waits for the 64 results coming on the VS bus from
the vector unit. The VS bus is parallel to the integer and FP CDB bus(es).

If full chaining and multiple memory pipes are employed, the execution time would be 109 clock periods.

Q6) Superscalar processors are closing the speed gap they have with vector processors when they run vector-orien-
ted applications with loop-level parallelism. One main advantage of vector processors that still remains, is their
handling of pipelined non-unit stride and sparse matrix memory accesses.

Suggest hardware and software techniques that can help superscalar processor close this gap too. You may invent
your own reasonable solutions.

A6) A number of software techniques can be tried on the scalar code so that
i) the access pattern of the code is changed from rows/columns to blocks (blocking). That is, the code accesses a
block containing a portion of a number of rows and columns. In order to do that arrays are stored blockwise, not row-
major nor column-major (see Blocking on page 433 of the Hennessy book).
ii) back-to-back nonsequential memory accesses are converted to sequential memory accesses : loop interchange (see
Loop Interchange on page 432 of the Hennessy book).
iii) data brought into the cache is used again and again without misses then removed from the cache : loop fusion (not
loop fission).
iv) The compiler unrolls loops to access elements in advance. Also, for non-loop situations, the compiler rearranges
the code so that memory reference instructions are moved up and down.

A number of hardware+software techniques can be used :

i) Special buffers between the memory and the cache can prefetch data automatically. If there are special data
prefetch instructions for caches, the compiler inserts them appropriately.
ii) The data cache can deliver in a pipelined fashion where the first memory request to a cache block is checked for
TLB and cache hits, taking more than one clock period. After that, in every clock period an element from the same
block is delivered : TLB translation and cache hit checking are done only once, for, say, 64 data elements. Note that
this is an active research area now.
iii) Second and third level cache memories can perform similarly.
iv) There can be indexed memory access instructions for cache memories

Q7) Consider the following VMIPS code :

L.D F0, Ra
LV V1, Rx
MULVS D V2, V1, F0
LV V3, Ry
ADDVD V4, V2, V3
SV Ry, V4

a) Write down the corresponding algorithm in the style used in class. Assume that the VLR and VM registers are ini-
tialized to allow 64-element vector operations. What is the time complexity of the algorithm ?
b) Determine how many clock periods it would take to run the above VMIPS vector code. Assume that the hardware allows chaining but there is only one memory pipeline. Indicate any assumptions made during the execution of the loop, if a situation not discussed in class is encountered.

A7) a) 
\[
\text{for (i = 1 ; i <= 64 ; i++)} \\
\quad Y[i] = (a \times X[i]) + Y[i]
\]

There is a single loop with one statement. The loop iterates n times, the length of the vectors. The time complexity is then linear: \( O(n) \).

b) We observe that due to one memory pipeline there are three convoys:
- \( \text{LV} + \text{MULVS.D} \)
- \( \text{LV} + \text{ADVD.D} \)
- \( \text{SV} \)

Then the execution time is:

Convoy 1:
- \( \text{L.D} \)
- \( \text{LV} \)
- \( \text{MULVS.D} \)

Convoy 2:
- \( \text{LV} \)
- \( \text{ADVD.D} \)

Convoy 3 (SV)

The approximate execution time = 237 clock periods

Q8) Consider the following piece of old MIPS code written for its unpipelined version:

```
DADDI R1, R0, #(64)_{10} ; Memory accesses are commented below :
L.D F0, 0(Rk) ; Rk points at constant k
loop: LD Ra, 0(Rindexa) ; Rindexa points at index vector for vector A
L.D F2, 0(Rb) ; Rb points at vector B
L.D F4, 0(Rd) ; Rd points at vector D
ADD.D F6, F2, F0
MUL.D F8, F6, F4
S.D 0(Ra), F6 ; Stores to vector A
S.D 0(Rc), F8 ; Stores to vector C
DADDI R1, R1, #(-1)_{10}
DADDI Rindexa, Rindexa, #4 ; Rindexa is advanced
DADDI Rb, Rb, #8 ; Rb is advanced
DADDI Rc, Rc, #8 ; Rc is advanced
DADDI Rd, Rd, #8 ; Rd is advanced
BNEZ R1, loop
```

Write that code in terms of VMIPS instructions (vectorize it). Indicate the approximate execution time for your code, assuming that there is no chaining, a single memory pipeline and the MIPS scalar FP and scalar integer pipeline latencies are as given on page 304 of the Hennessy book. Indicate any assumptions made during the execution of the loop, if a situation not discussed in class is encountered.
A8)  

LD  F0, 0(Rk) ; 12  
LV  Vb, Rindexa ; Convoy 1  
LV  Vb, Rb ; Convoy 2  
LV  Vd, Rd ; Convoy 3  
ADDVS.D  V_a, V_b, F0 ; Convoy 3  
MULV.D  V_c, V_a, V_d ; Convoy 4  
SVI  (R0 + V_p), V_a ; Convoy 4  
SV  Rc, V_c ; Convoy 5  

LD  F0 ; Convoy 1  
LV  Vb, Rindexa ; Convoy 1  
LV  Vb, Rb ; Convoy 2  
LV  Vd, Rd ; Convoy 3  
ADDVS.D  V_a, V_b, F0 ; Convoy 3  
MULV.D  V_c, V_a, V_d ; Convoy 4  
SVI  (R0 + V_p), V_a ; Convoy 4  
SV  Rc, V_c ; Convoy 5  

The approximate execution time = 387 clock periods

Q9) Consider the old MIPS code obtained in Past Exam Question 2 above again. The processor in this question is VMIPS. That is, this is machine model number 6.

The VMIPS vector and scalar pipeline timings are as given on page G-13 and 304 of the Hennessy book, respectively. Assume that there is no chaining. Also assume that there is one memory pipeline. Finally, assume that the VM and VLR registers are already initialized.

Rewrite the old MIPS code in terms of VMIPS instructions. Show the execution timings as discussed in class. Assume that the loop has 64 iterations. Indicate any assumptions made during the execution of the loop, if a situation not discussed in class is encountered.

A9) The VMIPS code is as follows:

LV  V1, R2 ; Load from M  
MULV.D  V1, V1, V1 ; M[i] * M[i]  
LV  V2, R3 ; Load N  
LV  V3, R4 ; Load Q  
MULV.D  V4, V2, V3 ; N[i] * Q[i]  
ADDV.D  V5, V4, V1 ; M[i] * M[i] + N[i] * Q[i]  
SV  R1, V5 ; Store to K

Since there are only 64 iterations and the associated registers are already initialized, the vectorized code does not have any scalar instruction above.
The timing of the vectorized code execution is as follows:

The approximate execution time = 439 clock periods