EE557--FALL 2000

MIDTERM 2

Open books and notes
Time limit: 1 hour and 20 minutes MAX. No extension.

Q1: /12
Q2: /8
Q3: /9
Q4: /8
Q5: /8
Q6: /5

TOTAL: /50

Grade: /25
QUESTION 1(Tomasulo) 12 points

Consider the following loop.

```
LOOP  LD F2,0(R1)
      MULTD F4,F2,F0
      LD F6,0(R2)
      ADDD F6,F4,F6
      SD 0(R2), F6
      ADDI R1,R1,#8
      ADDI R2,R2,#8
      SGTI R3,R1,done
      BEQZ R3, LOOP
```

This is the same loop as the one in problem 5 of homework 3. As you did in this problem, fill the following table for Tomasulo algorithm, assuming now that up to two instructions can be issued in any cycle.

Show the schedule by filling the instruction status table. Each entry in the table shows the clock cycle when a given instruction went through a specific phase of execution. Show the content of the table up to the point when the sgti instruction write its results.

Assume that functional units are not pipelined, but that there are enough functional units to avoid any stalls due to structural hazards on functional units or reservation stations.

Use the latencies shown in Figure 4.63. These latencies mean that the execution time of a FPMPY is 7 clocks, that the execution time of FPADD is 5 clocks and that any integer instruction executes in one clock.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Issue</th>
<th>Exec. complete</th>
<th>Write Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F2,0(R1)</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>MULTD F4,F2,F0</td>
<td>1</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>LD F6,0(R2)</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>ADDD F6,F4,F6</td>
<td>2</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>SD 0(R2),F6</td>
<td>3</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>ADDI R1,R2,#8</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>ADDI R2,R2,#8</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>SGTI R3,R1,done</td>
<td>4</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
QUESTION 2 (software pipeline) 8 points

Apply software pipelining to the loop in question 1 so that no true data dependencies on floating point operands remain in the body of the loop. Show the new loop below (you may ignore the startup and cleanup code).

```
LOOP    LD F2,0(R1)
        MULTD F4,F2,F0
        LD F6,0(R2)
        ADDD F6,F4,F6
        SD 0(R2), F6
        ADDI R1,R1,#8
        ADDI R2,R2,#8
        SGTI R3,R1,done
        BEQZ R3, LOOP
```

The pipeline must be such that the result of instructions in one pipeline stage are used in the next pipeline stage. Thus we must group the MULTD F4 and the LD F6 instructions in the same pipeline stage. Here is a possible schedule.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LD F2,0(R1)</td>
<td>MULTD F4</td>
<td>ADDD F6</td>
<td>SD F6, 0(R2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LD F6, 0(R2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>MULTD F4</td>
<td>ADDD F6</td>
<td>SD F6,8(R2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LD F6,8(R2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>MULTD F4</td>
<td>ADDD F6</td>
<td>SD F6,16(R2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LD F6,16(R2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MULTD F4</td>
<td>ADDD F6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LD F6,24(R2)</td>
<td></td>
</tr>
</tbody>
</table>

In this table each row contains the execution of the original loop. Each column show the execution of the pipelined loop. Column 3 shows the first complete iteration of the pipelined loop. Unfortunately this new loop will not give the same result as the initial loop because the value added in F6 is overwritten by the following load in F6 and thus the value of the store is lost. We need to rename register F6. Let’s rename F6 to F8.

The new code is:

```
LOOP    SD F6, 0(R2)
        ADDD F6,F4,F8
        MULTD F4,F2,F0
        LD F8,16(R2)
        LD F2,24(R1)
        ADDI R1,R1,#8
        ADDI R2,R2,#8
        SGTI R3,R1,done
        BEQZ R3, LOOP
```
QUESTION 3. 9 points

1. Prove that if for two array elements $A[a \times i + b]$ and $A[c \times i + d]$ there is a true dependence then $\text{GCD}(c,a)$ divides $(d-b)$

Answer:

For a true dependency, we must have $k$ and $j$ such that:

$a \times k + b = c \times j + d$ or $a \times k - c \times j = d - b$

Let $a = a' \times \text{GCD}(c,a)$ and let $c = c' \times \text{GCD}(c,a)$

then we must have: $(a' \times k - c' \times j) \times \text{GCD}(c,a) = d - b$

and since $a' \times k - c' \times j$ is an integer we must have that $\text{GCD}(c,a)$ divides $(d-b)$.

2. In the following loop

```plaintext
for (i=1, i<=100, i++)
```

Apply the GCD test to detect loop-carried true dependencies (if any).

Answer:

There is no true dependency in this loop, because the write happens after the read.

3. In the following loop

```plaintext
for (i=1, i<=100, i++)
```

Apply the GCD test to detect loop-carried true dependencies (if any).

Answer:

In this case, $a=5$, $b=1$, $c=1$ and $d=0$. So $\text{GCD}(a,c) = 1$, which divides any integer. Thus there may be a true dependency.

QUESTION 4. (8points)

Consider the following loop:

\[
\text{for } (i=1, i\leq 100, i++) \\
\]

The compiled code looks as follows:

```
LOOP:   LD F2, 0(R1)    /load A[i-3]
        LD F4, 24(R1)  /load A[i]
        ADDD F4,F2,F4
        SD 24(R1), F4  /store A[i]
        ADDI R1,R1,8
        STGI R3,R1,R2  /R2 contains the maximum value for R1;
        BEQZ R3, LOOP   /R3 is set to 1 if R1>R2
```

Please unroll the loop as much as possible, remove all useless instructions, rename registers, and then schedule the code to maximize ILP.

Answer:

Step 1. Unroll and remove useless instructions.

```
LOOP:   LD F2, 0(R1)
        LD F4, 24(R1)
        ADDD F4,F2,F4
        SD 24(R1), F4
        LD F2, 8(R1)
        LD F4, 32(R1)
        ADDD F4,F2,F4
        SD 32(R1), F4
        LD F2, 16(R1)
        LD F4, 40(R1)
        ADDD F4,F2,F4
        SD 40(R1), F4
        ADDI R1,R1,24
        STGI R3,R1,R2
        BEQZ R3, LOOP
```

Unrolling more is not useful because the first load of the fourth iteration reads the value produced by the store of the first iteration, which would prevent that load from executing before the store.
Step 2: Rename:

```
LOOP:   LD F2, 0(R1)  
      LD F4, 24(R1) 
      ADDD F4,F2,F4  
      SD 24(R1), F4  
      LD F6, 8(R1) 
      LD F8, 32(R1) 
      ADDD F8,F6,F8  
      SD 32(R1), F8  
      LD F10, 16(R1) 
      LD F12, 40(R1) 
      ADDD F12,F10,F12  
      SD 40(R1), F12  
      ADDI R1,R1,24  
      STGI R3,R1,R2 
      BEQZ R3, LOOP
```

Step 3: schedule:

```
LOOP:   LD F2, 0(R1)  
      LD F4, 24(R1) 
      LD F6, 8(R1) 
      LD F8, 32(R1) 
      LD F10, 16(R1) 
      LD F12, 40(R1) 
      ADDD F4,F2,F4  
      ADDD F8,F6,F8  
      ADDD F12,F10,F12  
      SD 24(R1), F4  
      SD 32(R1), F8  
      SD 40(R1), F12  
      ADDI R1,R1,24  
      STGI R3,R1,R2 
      BEQZ R3, LOOP
```
QUESTION 5. 8 points

A. An \((m,n)\) branch prediction scheme uses \(m\) bits of global history and \(n\) bit history counters. Consider the following nested loop:

Consider the following loop.

```
LOOP1 I1
   I2
LOOP2 I3
   I4
   I5
   BEQZ R3, LOOP2
   BEQZ R4, LOOP1
```

When this loop is executed for the first time, all history counters are zero. The values of \(R3\) and \(R4\) are such that \(LOOP1\) is executed 10 times and, in each iteration of \(LOOP1\), \(LOOP2\) is executed 100 times. Give the success rate for the following prediction scheme (the success rate is the number of good predictions divided by the number of branch instructions executed)

1. \((0,1)\)

Answer:

In this case the two branch instructions are independent. The second branch is executed 10 times and it is misspredicted on the first and last iteration (total misspredicts = 2). The first branch is executed 1000 times and is also misspredicted at its first and last execution in each iteration of \(Loop1\) (total misspredict = 20). So the total number of branches executed is 1010 and the total number of misspredict is 22. The success rate is

\[
\frac{1010-22}{1010} = .978
\]

2. \((2,2)\)

In the following table, we show the prediction in every execution of \(loop1\). Each row corresponds to one branch execution. There are 101 branch executions in every iteration of \(loop1\). G.H. is the global history before the branch, G.H+ is the next global history. HC1(mn) (HC2(mn)) is the value of the history counter for the branch of \(loop\ 1\) with \(G.H\) of \(mn\).

<table>
<thead>
<tr>
<th>Iteration (loop1,branch#)</th>
<th>G.H.</th>
<th>H.C.</th>
<th>H.C.+</th>
<th>Pred.</th>
<th>Out</th>
<th>G.H.+</th>
<th>Misspredict</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1</td>
<td>00</td>
<td>HC2(00)=00</td>
<td>HC2+(00)=01</td>
<td>NT</td>
<td>T</td>
<td>01</td>
<td>Y</td>
</tr>
<tr>
<td>1,2</td>
<td>01</td>
<td>HC2(01)=00</td>
<td>HC2+(01)=01</td>
<td>NT</td>
<td>T</td>
<td>11</td>
<td>Y</td>
</tr>
<tr>
<td>1,3</td>
<td>11</td>
<td>HC2(11)=00</td>
<td>HC2+(11)=01</td>
<td>NT</td>
<td>T</td>
<td>11</td>
<td>Y</td>
</tr>
</tbody>
</table>
There are 6 mispredicts in the first iteration of loop1, and 3 mispredicts in the second iteration of loop1. After that, the pattern shows that the only mispredicted branches are the branch of loop2 for iterations 3 thru 10 and the branch of loop1 in iteration 10, for a total of 9 mispredicts. The total number of mispredict is 18 and the success rate is:

\[(1010-18)/1010=.982\]

B. Explain what is meant by “branch folding”, how it is implemented in practice and what its effects are on performance.

Answer:

If in the BTB we can store the target instruction and the target address, then both the branch and its target instruction are fetched at the same time. If the branch is taken, the target instruction is executed and the execution of the branch took zero time. It’s as if the branch was not present in the code.
**QUESTION 6. 5 points**

1. Consider the following loop.

   ```
   LOOP  SUBI R1,R1,#8
         BEQZ R1, L1
         LD F2, 0(R1)
         J EXIT
   L1:   LD F2, 1000(R1)
   EXIT  ADDD F2, F2,F4
         SD 1000(R1),F2
         BNEZ R1, LOOP
   ```

   Eliminate the BEQZ instruction and the J instruction by using the predicated load instruction, LDC Fi, 0(Rj), Rk (the load occurs unless Rk=0) and then write the new code below.

   **Answer:**

   ```
   LOOP  SUBI R1,R1,#8
         LD F2, 1000(R1)
         LDC F2, 0(R1),R1
         ADDD F2, F2,F4
         SD 1000(R1),F2
         BNEZ R1, LOOP
   ```