EE557--FALL 1999
MAKE-UP MIDTERM 1
Closed books, closed notes

Q1: /10
Q2: /10
Q3: /10
Q4: /10
Q5: /15
Q6: /10

TOTAL: /65

Grade: /25
QUESTION 1 (Performance evaluation) 10 points

We are considering adding a new “displaced index” address mode to the DLX instruction set. This will allow code sequences of the form:

\[
\begin{align*}
&\text{ADD R1, R1, R2} \\
&\text{LW R4, 0(R1)}
\end{align*}
\]

to be replaced by one single instruction using the new address mode and a shorter 11 bit offset:

\[
\text{LW R4, 0(R1,R2)}
\]

Among all executed instructions we know that 26% are loads, 9% are stores and 14% are adds.

1. Assuming the new addressing mode can be used for 10% of the loads and stores (accounting for the frequency of this type of address calculation with the shorter offset), what is the ratio of the instruction count (IC) on the enhanced DLX compared to the old DLX?

\[
\text{IC}_{\text{enhanced}} = \frac{0.10 \times 0.26}{\text{IC}_{\text{old}}}.
\]

2. Unfortunately the extra 2 stage addition process that is required to support the displaced indexed mode lengthens the cycle time by 5%. Which machine is faster and by how much?

\[
\text{IC}_{\text{enhanced}} = \frac{0.10 \times 0.26}{\text{IC}_{\text{old}}}.
\]
QUESTION 2 (Performance evaluation) 10 points

You are given the following distributions and cycle times for a perfect memory system machine:

A perfect memory system implies that every data and instruction access hits in the caches. However measurements show that instructions accesses miss in the cache 5% of the time and that data references miss 10% of the time. When either type of miss is encountered an extra 40 cycles is required to complete the instruction. Of course when both types of misses occur in the same instruction a total of 80 extra cycles are required. How much faster does the machine run with the ideal memory than with the real memory system?

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Frequency</th>
<th>Clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALUOps</td>
<td>43%</td>
<td>1</td>
</tr>
<tr>
<td>Loads</td>
<td>21%</td>
<td>2</td>
</tr>
<tr>
<td>Stores</td>
<td>12%</td>
<td>2</td>
</tr>
<tr>
<td>Branches</td>
<td>24%</td>
<td>2</td>
</tr>
</tbody>
</table>

A perfect memory system implies that every data and instruction access hits in the caches. However measurements show that instructions accesses miss in the cache 5% of the time and that data references miss 10% of the time. When either type of miss is encountered an extra 40 cycles is required to complete the instruction. Of course when both types of misses occur in the same instruction a total of 80 extra cycles are required. How much faster does the machine run with the ideal memory than with the real memory system?
QUESTION 3. Single Cycle CPU (10points)

The data path in Figure 1 supports LW, SW, BEQ, and R-format instructions. We wish to add instruction “JUMP_AND_LINK” (JAL), which saves PC+4 in register $31 and jumps to the target address.

ADDM rs, rt, rd

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>25</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>111001</td>
<td>Target</td>
<td></td>
</tr>
</tbody>
</table>

Description:

\[
\begin{align*}
($31) & \leftarrow (PC)+4 \\
(PC) & \leftarrow \text{upper 4 bits of } (PC)+4 \parallel \text{target} \parallel 00
\end{align*}
\]

1. Draw the datapath modifications on Figure 1. Explain below. Make sure that all instructions still execute in one cycle.

2. Specify the new control points (show them on Figure 1 and explain their settings here):

3. Give the values of all control points (0,1 or X) for the new instruction (except ALUOp) (including the new control points):
QUESTION 4 (10pts) 5-stage pipeline

Explain how to implement the JAL instruction of question 3 in the 5-stage pipeline of Figure 2.

1. Draw the datapath modifications on Figure 2. Explain below.

2. Specify the new control points (show them on Figure 2 and explain their settings here):
QUESTION 5 (15pts)

Figure 3 shows a DLX floating point pipeline with multiple execution units. The address (integer) unit takes one clock. The FP adder and FP multiplier are fully pipelined and take 2 and 5 clocks respectively. If structural hazards are due to write-back contention, assume that the earliest instruction gets priority and other instructions are stalled.

Consider the following program.

```
LOOP  LD F0,0(R2)
     LD F4,0(R3)
     MULTD F0,F0,F4
     ADDD F2,F0,F2
     ADDI R2,R2,#8
     ADDI R3,R3,#8
     SUB R5,R4,R2
     BNE R5,R0,LOOP
```

1. In the following table please show the timing diagram for this loop by indicating the stage in each clock for each instruction in the table, under the following assumptions. There is no forwarding or bypassing hardware except that the register file delivers the value written in a register if it is read in the same clock. The branch is handled by flushing the pipeline after the branch is detected, its target address is known and the condition is resolved in the ID stage.

1. Assuming that all memory references hit in the cache, how many cycles does each iteration of this loop take to execute.

   \[ \text{TIME} = \quad \] Clocks per iteration.

2. In the following table show the timing for the same instruction sequence under the following assumptions. There is normal forwarding and bypassing hardware. The branch is handled by predicting it not taken and then flushing in the ID stage if the prediction was
wrong.

TABLE 3.

|                     | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | T14 | T15 | T16 | T17 | T18 |
|---------------------|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|
| LD F0,0(R2)        | IF | ID | EX | ME | WB |     |    |    |    |     |     |     |     |     |     |     |     |
| LD F4,0(R3)        |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |
| MULTD F0,F0,F4     |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |
| ADDD F2,F0,F2      |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |
| ADDI R2,R2,#8      |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |
| ADDI R3,R3,#8      |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |
| SUB R5,R4,R2       |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |
| BNE R5,R0,LOOP     |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |
| LD F0,0(R2)        |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |

Assuming that all memory references hit in the cache, how many cycles does each iteration of this loop take to execute.

TIME= __________________Clocks per iteration.

3. In the following table show the timing for the same instruction sequence under the following assumptions. There is normal forwarding and bypassing hardware and the branch is a one-cycle delayed conditional branch. Schedule the code in the loop including the branch delay slot to minimize execution time. You may reorder instructions and modify the individual instruction operands but do not undertake other loop transformations that change the number or opcode of the instructions in the loop.

TABLE 4.

|                     | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | T14 | T15 | T16 | T17 | T18 |
|---------------------|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|
| LD F0,0(R2)        | IF | ID | EX | ME | WB |     |    |    |    |     |     |     |     |     |     |     |     |     |
| LD F4,0(R3)        |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |
| MULTD F0,F0,F4     |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |
| ADDD F2,F0,F2      |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |
| ADDI R2,R2,#8      |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |
| ADDI R3,R3,#8      |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |
| SUB R5,R4,R2       |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |
| BNE R5,R0,LOOP     |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |
| LD F0,0(R2)        |    |    |    |    |    |   |    |    |    |     |     |     |     |     |     |     |

Assuming that all memory references hit in the cache, how many cycles does each iteration of this loop take to execute.

TIME= __________________Clocks per iteration.
**QUESTION 6 (10pts)**

The MIPS R4000 pipeline is made of an integer/load/store pipeline, a floating point multiply pipeline and a floating point add pipeline similar to Figure 3, but with different latencies and initiation intervals. Table 5 shows the latencies and initiation intervals. Remember that initiation intervals are due to structural hazard on a given pipeline. For example, if a pipeline has an initiation interval of 3, that means that if two consecutive instructions need to use that pipeline the second one will be stalled for two cycles, whether or not there is a dependency. This is important for this problem.

**TABLE 5.**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Latencies</th>
<th>Initiation Intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDD</td>
<td>4 (3 if followed by a store)</td>
<td>3</td>
</tr>
<tr>
<td>MULTD</td>
<td>8 (7 if followed by a store)</td>
<td>4</td>
</tr>
<tr>
<td>LD</td>
<td>2 (1 if followed by a store)</td>
<td>1</td>
</tr>
<tr>
<td>SUBI, ADDI, SUB</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The branch delay (i.e. the number of clocks before the outcome of the branch condition is known) is 3. The MIPS architecture has a single-cycle delayed branch. The R4000 uses a predict-not-taken strategy for the remaining two cycles of the branch delay.

Consider the following program (This is a loop that has been unrolled twice).

```
LOOP
  LD F2,0(R1)
  MULTD F4,F0,F2
  LD F6,0(R2)
  ADDD F6,F4,F6
  SD 0(R2), F6
  ADDI R1,R1,#8
  ADDIR2,R2,#8
  LD F8,0(R1)
  MULTD F10,F0,F8
  LD F12,0(R2)
  ADDD F12,F10,F12
  SD 0(R2), F10
  ADDI R1,R1,#8
  ADDI R2,R2,#8
  SUBI R5,R5,#2
  BNE R5,R0,LOOP
```

Assuming that all memory references hit in the cache, how many cycles does each iteration of this loop takes after scheduling the code in the loop (including the branch delay slot) to minimize execution time. You may reorder instructions and modify the individual instruction operands **but do not undertake other loop transformations that change the number or opcode of the instructions in the loop**. Ignore contention for register ports.
Figure 1. Simplified single cycle datapath
Figure 2: Simplified 5-stage pipeline
Figure 3. DLX pipeline with multiple execution units.

IF → ID → A1 → A2 → ME → WB

IF → ID

EX with FP Adder

ME with FP Multiplier

M1 → M2 → M3 → M4 → M5

Integer+LOAD/STORE