CSEE W3827
Fundamentals of Computer Systems
Homework Assignment 6
Solutions

Prof. Martha A. Kim
Columbia University
Due December 2, 2014 at 10:10.

Write your name and UNI on your solutions
Show your work for each problem; we are more interested in how you get the answer than whether you get the right answer.
Many of the problems in this assignment will involve analysis of the execution of this implementation of countlocalminima. It is derived from the HW#4 solution, modified slightly to use only instructions supported by our MIPS CPU implementations.

```
i0:    addi  $a1, $a1, -1
i1:    add   $a1, $a1, $a1
i2:    add   $a1, $a1, $a1
i3:    add   $a1, $a1, $a0
i4:    addi  $a0, $a0, 4
i5:    add   $v0, $0, $0
i6:    addi  $t3, $0, 2
i7_top: beq   $a0, $a1, i19_done
i8:    lw    $t0, -4($a0)
i9:    lw    $t1, 0($a0)
i10:   lw    $t2, 4($a0)
i11:   slt   $t0, $t1, $t0
i12:   slt   $t2, $t1, $t2
i13:   add   $t0, $t0, $t2
i14:   beq   $t0, $t3, i17_inc
i15_adv: addi  $a0, $a0, 4
i16:   beq   $0, $0, i7_top
i17_inc: addi  $v0, $v0, 1
i18:   beq   $0, $0, i15_adv
i19_done:
```
1. (20 pts.) Imagine how countlocalminima would execute on a fully bypassed 5-stage MIPS pipeline (i.e., branches resolved in D, forwarding from W-E, M-E, and M-D). List all pairs of instructions between which one or more bubbles would occur. If a bubble occurs between i3 and i4, then you should write i3 → i4. (HINT: Think systematically through all scenarios that result in an empty slot in the pipeline.)

Bubbles occur when ...
... a lw is immediately followed by its consumer. This does not occur in code above.
... a beq is taken
  • i7_top → i19_done
  • i14 → i17_inc
  • i16 → i7_top
  • i18 → i15_adv
... a beq immediately follows its producer (2 bubbles if that producer is a lw)
  • i7_top, as fall through, no bubbles
  • i7_top, as branch target, no bubbles
  • i14 uses $t0, so i13 → i14
  • i16 uses only $0, so no bubbles
  • i18 uses only $0, so no bubbles
2. (20 pts.) Now list where in the program (still count local minima on the fully bypassed 5-stage pipe) data operands would be forwarded, and which forwarding path would be used. If i3 forwards the future value of $a0 to i4 using the M-E forwarding path, write $a0, i3 \rightarrow i4, M-E, per the example below. (HINT: Think through the values consumed by each instruction systematically.)

- $a1$ for i0, as producer unknown, assuming no forward
- $a1, i0 \rightarrow i1, M-E$
- $a1, i1 \rightarrow i2, M-E$
- $a1, i2 \rightarrow i3, M-E$
- $a0$ for i3, no forward due to distant producer
- $a0$ for i4, no forward due to distant producer
- $0$ for i5, never forward
- $0$ for i6, never forward
- $a0$ for i7_top, when falling through, no forward due to distant producer
- $a1$ for i7_top, when falling through, no forward due to distant producer
- $a0$ for i7_top, when branch target, no forward due to branch taken bubble making producer (i15_adv) distant
- $a1$ for i7_top, when branch target, no forward due to distant producer
- $a0$ for i8, no forward, as never forwarded to i7_top above
- $a0$ for i9, no forward due to distant producer
- $a0$ for i10, no forward due to distant producer
- $t0$ for i11, no forward due to distant producer
- $t1, i9 \rightarrow i11, W-E$
- $t1$ for i12, no forward due to distant producer
- $t2, i10 \rightarrow i12, W-E$
- $t0, i11 \rightarrow i13, W-E$
- $t2, i12 \rightarrow i13, M-E$
- $t0, i13 \rightarrow i14, M-D$
- $t3$ for i14, no forward due to distant producer
- $a0$ for i15_adv, when falling through, no forward due to distant producer
- $a0$ for i15_adv, when branch target, no forward due to distant producer
- $0$ for i16, never forward
- $v0$ for i17_inc, arrive only via branch target, no forward due to distant producer
- $0$ for i18, never forward
3. (15 pts.) Assuming a very large array such that the repeating loop body dominates the execution of the code snippet, what is the CPI of `countlocalminima` code? Assume that the beq in i14 is taken 20% of the time.

The cleanest way to reason about this loop body is in two cases:

- iterations when branch i14 is taken
  - In this case, the dynamic instruction sequence is i7 - i14, i17, i18, i15, i16.
  - i14, i16, and i18 are all taken branches and have a CPI of 2, i13 has a bubble and thus also a CPI of 2, while the remaining 8 have a CPI of 1
  - So the CPI for these iterations is \( \frac{4}{12} \times 2 + \frac{8}{12} \times 1 = 1.33 \)

- and iterations when branch i14 is not taken
  - In this case, the dynamic instruction sequence is i7 - i14, i15, i16.
  - i14 is a correctly predicted branch, and so has a CPI of 1, i16 as a taken branch has a CPI of 2, i13 still has a bubble for a CPI of 2, and the remaining 7 instructions have a CPI of 1.
  - So the CPI for these iterations is \( \frac{2}{10} \times 2 + \frac{8}{10} \times 1 = 1.2 \)

So, the overall CPI of this loop is a weighted sum of these two scenarios:
\[
0.2 \times 1.33 + 0.8 \times 1.2 = 1.226.
\]
4. (10 pts.) Assuming an array of length 6 found at address 0x0000C0D8, list the stream of addresses referenced by countlocalminima.

The six integers are located at 0x0000C0D8, 0x0000C0DC, 0x0000C0E0, 0x0000C0E4, 0x0000C0E8, 0x0000C0EC. The references for each of the four iterations of the loop are:

- 0x0000C0D8, 0x0000C0DC, 0x0000C0E0
- 0x0000C0DC, 0x0000C0E0, 0x0000C0E4
- 0x0000C0E0, 0x0000C0E4, 0x0000C0E8
- 0x0000C0E4, 0x0000C0E8, 0x0000C0EC
5. (15 pts.) For each of the caches listed below, show how a 32-bit addresses breaks into *tag*, *set index*, and *byte offset* fields.

**Cache A: 1024B, 2-way set-associative, 16B lines (32B per set, so 32 sets in cache)**

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

tag (23 bits) set index (5 bits) byte offset (4 bits)

**Cache B: 4096B, direct-mapped, 16B lines (16B per set, so 256 sets in cache)**

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

tag (20 bits) set index (8 bits) byte offset (4 bits)
6. (15 pts.) Assuming the address references to an initially empty 2-level cache hierarchy with Cache A as the L1 and Cache B as the L2, fill in the table below indicating the set for each reference and whether it resulted in a hit or a miss. If there is no access, just mark the squares with a “-”.

<table>
<thead>
<tr>
<th>Address</th>
<th>L1 (Cache A)</th>
<th>L2 (Cache B)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set Result</td>
<td>Set Result</td>
</tr>
<tr>
<td>0x0000C0DE</td>
<td>0x0D miss</td>
<td>0x0D miss</td>
</tr>
<tr>
<td>0x0000C0E2</td>
<td>0x0E miss</td>
<td>0x0E miss</td>
</tr>
<tr>
<td>0x0000C0E6</td>
<td>0x0E hit</td>
<td>-</td>
</tr>
<tr>
<td>0x0000C0EC</td>
<td>0x0E hit</td>
<td>-</td>
</tr>
<tr>
<td>0x0000C0DE</td>
<td>0x0D hit</td>
<td>-</td>
</tr>
<tr>
<td>0x0000C0E2</td>
<td>0x0E hit</td>
<td>-</td>
</tr>
</tbody>
</table>
7. (5 pts.) Assume that Cache A’s access time is 10ns with a miss rate of 10%, Cache B’s access time is 500ns with a miss rate of 20%, and Memory’s access time is 5000ns. What is the expected access time for the overall cache hierarchy?

\[10 + 0.10 \times (500 + 0.20 \times 5000) = 160\]