Homework Quiz (HW #4)
March 15, 1999
CS152 Computer Architecture and Engineering

This quiz covers one of the problems from homework #4.
Good Luck!

<table>
<thead>
<tr>
<th>Your Name:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SID Number:</td>
<td></td>
</tr>
<tr>
<td>Discussion Section:</td>
<td></td>
</tr>
</tbody>
</table>

| Total: |  |
The VSCALE instruction

In your homework, you implemented the bcp (block copy) instruction. Here is the pseudo-code for a related instruction, called vscale. It is the same as block copy, except that it multiplies every source word by a constant before copying it to its destination. Let \( v1 \) be the starting address of the input vector, \( v2 \) be the starting address of the destination vector, \( length \) be the length of both vectors (in words), and \( scale \) be the scale factor:

\[
\text{vscale}(v1,v2,\text{length},\text{scale}) \{
\text{int index} = (\text{length}-1)\times4; \quad /* \text{mult} \times4 \text{ since we want byte addr */}
\]

\[
\text{while} \ (\text{index} \geq 0) \ \{ \\
\quad \text{M}[v2+\text{index}] = \text{scale} \times \text{M}[v1+\text{index}]; \\
\quad \text{index} = \text{index} - 4;
\}
\]

Your job is to implement the vscale instruction in the multicycle data path. Here is the coding of vscale instruction:

\[
\text{opcode} = \text{Instruction}[31:26] = 0x31 \quad (\text{an unused opcode}) \\
\text{R[rs]}\Rightarrow v1, \text{R[rt]}\Rightarrow v2, \text{R[rd]}\Rightarrow \text{length}-1, \text{Instruction}[10:0]\Rightarrow \text{scale}
\]

The \( scale \) value is hard-coded into the lower 11 bits of the instruction. Assume that it is a signed number. Note that we have to read three registers to make this work, i.e., rs, rt, and rd. Since scaling a vector of zero length isn’t very interesting, assume that R[rd] contains the length –1.
Problem 1a: Start with Figure 1. Assume that you are not allowed to add additional ALUs or adders, but that the current ALU supports multiply. What is the lowest number of cycles that you might expect to spend per vector element? Why? Hint: use the ALU every cycle.

Ans: 4. Each loop has 4 ALU operations: \( v1+\text{index}, v2+\text{index}, \text{scale} \times M[v1+\text{index}], \) and \( \text{index} = \text{index} - 4. \)

Problem 1b: What needs to be added to the datapath to implement \( v\text{scale} \) with this number of cycles in the inner loop? Add only registers, muxes, and simple logic. You can modify existing components. Assume that the ALU supports a multiply operation and ignore overflow. Note that the ALU takes a complete cycle to compute and memory takes a complete cycle to read or write. Hint: you will need to add new registers and may need to add enable lines to existing registers.

Place enables on registers A and B so that they can hold their values \( (v1 \text{ and } v2) \) indefinitely. We need a register to hold “index”. Note that we will do the “length\(\times4\)” functionality with wires. Further, this register cannot change every cycle (we need to add it to both \( v1 \) and \( v2 \) at minimum), so it needs an enable. In order to add both \( v1 \) and \( v2 \) to index, the index register needs to be muxed into both of the ALU inputs. To get the initial value of index (from \( R[rd] \times 4 \)), we need to mux in the “rd” register ID into one of the register file inputs (we chose input A).

In order to get the full overlap of the ALU, we need to compute \( (v2+\text{index}) \) while we are busy reading \( M[v1+\text{index}] \). Then, we need to wait a cycle, while we compute \( (\text{scale}\times M[v1+\text{index}]) \) The simplest thing to do is to add an enable on the ALUout register, so that you can preserve its contents for a cycle. Note that we also need to latch the output of the ALU and route it to the data input of the memory. So, include another ALU register, call it MemComp and connect it to the output of the ALU. Then, you need a new mux that places either MemComp or the B register into the memory.

Finally, we need to make the sign extender have a new function, namely “sign-extend 11 bits”. This will produce a sign-extended 32-bit value which is placed into ALU mux B. Note that this simply amounts to ignoring 5 of the 16 immediate bits. Further, we need to mux in the data input register (from memory) to the ALU mux A.

Note that you weren’t really expected to say all this (or even draw something as complete as Figure 1 on the next page).
Figure 1: Major parts of datapath for vscale instruction.
Some parts of existing datapath were left out.
Problem 1c: Use pseudo-code and register-transfer language to describe what is happening in each cycle of your instruction. Be very clear about which things are happening at the same time!

```
1  vscale:  index ← (R[rd] << 2)
2        A ← R[rs]; B←R[rt]
3  loop:   ALUout ← A+index
4        MemDataReg ← M[ALUout]; ALUout←B+index
5        MemComp←MemDataReg×Extend[Imm11]
6        M[ALUout]←MemComp; index←index-4; branch ≥ 0 to line 2
7  done:
```

Notice that the ALU is used on every cycle of the inner loop (3—6).