Assignment 8

Alex Clemmer
Student number: u0458675

Problem 1:

So the way to calculate this is to take the total percent failure of the branch predictor (in the case of (a), this is 60%), and then multiply it by the total number of branch instructions (in the case of (a), this is 15%). This gives us the percentage of failed branch predictions that comprise the whole.

For each of these failed instructions, 2 instructions must be thrown out. So multiplying this result by the number of stall cycles gives us the total stall cycle increase.

KEY The formula on the right of these tables can be interpreted as: (stall cycles) (% failed instructions) (% branch instructions) = result

$$\begin{array}{lll} \textbf{4.23.1} & \text{instructions}(\mathbf{a}), \text{accuracy}(\mathbf{a}) & 2\cdot(1-0.4)\cdot 0.15 = 0.18 \\ & \text{instructions}(\mathbf{b}), \text{accuracy}(\mathbf{b}) & 2\cdot(1-0.6)\cdot 0.1 = 0.08 \\ & \text{instructions}(\mathbf{a}), \text{accuracy}(\mathbf{b}) & 2\cdot(1-0.6)\cdot 0.15 = 0.12 \\ & \text{instructions}(\mathbf{b}), \text{accuracy}(\mathbf{a}) & 2\cdot(1-0.4)\cdot 0.1 = 0.12 \end{array} \tag{1}$$

The failure cost of the always-*not*-taken predictor is also 2 cycles:

$$\begin{array}{lll} \textbf{4.23.2} & \text{instructions}(\mathbf{a}), \text{accuracy}(\mathbf{a}) & 2\cdot(1-0.6)\cdot 0.15 = 0.12 \\ & \text{instructions}(\mathbf{b}), \text{accuracy}(\mathbf{b}) & 2\cdot(1-0.4)\cdot 0.1 = 0.12 \\ & \text{instructions}(\mathbf{a}), \text{accuracy}(\mathbf{b}) & 2\cdot(1-0.4)\cdot 0.15 = 0.18 \\ & \text{instructions}(\mathbf{a}), \text{accuracy}(\mathbf{b}) & 2\cdot(1-0.6)\cdot 0.1 = 0.08 \end{array} \tag{2}$$

And finally, our 2-bit predictor will also cost 2 cycles upon failure:

$$\begin{array}{lll} \textbf{4.23.2} & \text{instructions}(\mathbf{a}), \text{accuracy}(\mathbf{a}) & 2\cdot(1-0.8)\cdot 0.15 = 0.06 \\ & \text{instructions}(\mathbf{b}), \text{accuracy}(\mathbf{b}) & 2\cdot(1-0.95)\cdot 0.1 = 0.01 \\ & \text{instructions}(\mathbf{a}), \text{accuracy}(\mathbf{b}) & 2\cdot(1-0.95)\cdot 0.15 = 0.015 \\ & \text{instructions}(\mathbf{b}), \text{accuracy}(\mathbf{a}) & 2\cdot(1-0.8)\cdot 0.1 = 0.04 \end{array} \tag{3}$$

Problem 2:

(a) First we identify the stalls. There's one between the first and second `lw`. That's about it. So we have two cases: the first is that we skip, and the second is that we don't. In the case of the skip, we have `9` cycles (one per instruction) + `1` for the delay between the `lw`s. That gives us `10`. When we don't skip, we have `10` cycles + `1` for the delay, which is `11`.

We don't skip when we loop for the second and fourth time, so that gives us `20` cycles. We *do* skip on the first, third, and fifth time, so that gives us `33` more cycles. Then we have to add `1` cycle each for both `ori`s at the beginning and end. Then we have to add `4` to compensate for the pipeline winding down.

This all gives us a total of `58` cycles.

(b)

```
Func:
        lw   $t1, 0($a1)
        ori  $t0, $a0, 0        # Start here
Loop:
        lw   $t2, 8($t1)
        add  $t3, $t0, $a2
        bne  $t2, $t3, Skip
        addi $t0, $t0, -1
        add  $t4, $t4, $t2
Skip:
        addi $a1, $a1, 4
        bne  $t0, $zero, Loop
        lw   $t1, 0($a1)
        ori  $v0, $t4, 0        # End here
        jr   $ra
```

(c) Eliminating all hazards, we can start by counting the number of instructions that get executed. In the loop, if we skip, we have `8` and if we don't we have `7`.

We skip three times and don't skip two, which gives us `37` total instructions. Then we account for the `lw` at the beginning and the `ori` at the beginning and end. That adds three for a total of `40`. Then we have four more instructions for loop wind-down, **which gives us a total of** `44`.

Problem 3:

For **4.38.1**, the energy consumption is exactly the same for both designs. The operation consists of an I-Mem read, two register reads, and a register written.

$$\begin{aligned} \textbf{4.38.1} \quad \textbf{(a)} \quad & \texttt{100pJ + } 2\cdot\texttt{60pJ + 70pJ = 290pJ} \\ \textbf{(b)} \quad & \texttt{200pJ + } 2\cdot\texttt{90pJ + 80pJ = 460pJ} \end{aligned} \tag{4}$$

For **4.38.2**, every instruction reads I-Mem and 2 registers (although careful attention should be paid to the fact that only one of these registers may be read, as I found out the hard way).

Store will require a D-Mem write, while load will require both a register write and a memory read. All other instructions will either manipulate only the registers (*e.g.*, `add`), or nothing (*e.g.*, `j`).

Since `lw` and `sw` will definitely consume more energy than their counterparts which do not require access to D-Mem, the contest is between them and them alone. Since the energy consumption of a read to D-Mem and a register write are more combined than a D-Mem write, `lw` is the clear winner.

$$\begin{aligned} \textbf{4.38.2} \quad \textbf{(a)} \quad & \texttt{100pJ + } 2\cdot\texttt{60pJ + 70pJ + 120pJ = 410pJ} \\ \textbf{(b)} \quad & \texttt{200pJ + } 2\cdot\texttt{90pJ + 80pJ + 300pJ = 760pJ} \end{aligned} \tag{5}$$

Lastly, in the case of **4.38.3**, we have to be a bit more creative. I start by looking for things that can be eliminated, either from every instruction, or even just from the critical instruction. The one that pops out is the register reads; we are always reading both registers, even if the instruction doesn't actually use registers, and even if we only need to read one. This might not mean much for instruction time, but it will mean something for energy consumption. We would have to implement separate control lines for each, but since the energy consumption for control is "negligible" without these control lines, I will also assume that it is negligible with them.

$$\begin{aligned} \textbf{4.38.3} \quad \textbf{(a)} \quad & \texttt{100pJ + 60pJ + 70pJ + 120pJ = 350pJ} \\ \textbf{(b)} \quad & \texttt{200pJ + 90pJ + 80pJ + 300pJ = 670pJ} \end{aligned} \tag{6}$$

As we can see, we save `60pJ` in the first case and `90pJ` in the second case. It doesn't sound huge, until you consider that that's about 15% and 11%, respectively.