Alex Oswald
EECE 2323
Due: 23 Oct 2020

Pre-Lab Assignment 5

§1 Load & Store Operations

With reference to a sequence of tasks in the `test_sequence_for_lab5v2.txt` file, which implements an example of store and load operations, the following two questions are answered:

1. What would be the contents of the registers in the register file after the steps described in the file?

The final contents of the registers in the register file are all zero after it is finally reset in step 8. However, just before this they had the following settings:

```
addr:  2'b00 (1)     value: 9'd2   = b000000010
addr:  2'b01 (2)     value: 9'd507 = b111111011 // from (4) 000000100
addr:  2'b10 (3)     value: 9'd21  = b000010101 // from 8'h15
addr:  2'b11 (4)     value: 9'd23  = b000010111 // from 2+21=23
```

2. Explain what these tests do functionally? (e.g. load register 1, store value x in memory location y, etc.)

Preliminarily, the registers and memory blocks are cleared to assure purity of the following operations. First, the value 2 is stored to memory locale 1, and the value 4 is stored to the memory locale 2. This is done by adding those integers with the zero register. Then, memory location 1 (storing the # 2) is loaded into reg1. Following, the inverse of memory location 2 is operated through the ~b ALU operator and stored in reg 2 as 8'b11111011 or 8'd251. Next, 8'h15 or 8'b00010101 or 8'd10 is stored in reg 3. Finally, reg 1 and reg 3 are added through the ALU and their output is stored in reg 4 (8'd23 b00010111). Ultimately, the registers are cleared at the final step so technically they all read 8'b00000000.

§2 Create Your Own Sequence!

In writing my own sequence of steps that test the datapath using different values and different ALU instructions, I came up with the following set of instructions:

```
1. reset
2. Store values 18 and 9 to memory locations 2 and 3.
3. Load the value of memory location 2 to reg 1.
4. Logical shift right the value of memory location 3 and store it in reg 2.
5. Compute the bitwise OR of reg 1 & reg 2 and store it in reg 3.
6. Write 8'hC8 to reg 4.
7. Invert the contents of reg 4.
8. Store the contents of reg 3 and 4 to memory locations 1 and 4 respectively.
9. reset
```

1. What are the expected values in the registers after your test is complete? What ALU operations have you not tested?

Anticipated Results:

```
Register (before clearing)
addr:  2'b00 (1)     value: 9'd18  = b000010010
addr:  2'b01 (2)     value: 9'd4   = b000000100 // from 9'd9 >>> 1
addr:  2'b10 (3)     value: 9'd22  = b000010110 // from 9'd18 | 9'd4
addr:  2'b11 (4)     value: 8'b55  = b000110111 // from ~9'd200 (invert)   (FINAL VALUE)

Memory (final)
addr: 1              value: 9'd22
addr: 2              value: 9'd18
addr: 3              value: 9'd9
addr: 4              value: 9'd55
```

The only ALU functions left untested would be the take_branch operations, if a==b or a!=b, a left shift (a<<1), and the bitwise AND (a & b) operations.

2. What do these tests do functionally?

First, the register and memory banks are cleared. Then, values 18 and 9 are stored to memory locations 2 and 3 respectively. After, the value of memory location 2 is loaded into reg 1 (#18). Next, the value of memory location 3 (#9) is effectively halved via a logical shift right one bit and stored in reg 2. Further, the bitwise or of reg 1 (18) and reg 2 (4) is taken (#22) and stored in reg 3. Next, the hex value of C8 (or 200) is written to reg 4, and subsequently inverted through the ALU and restored into reg 4. The last step is to store the contents of reg 3 and reg 4 to memory locations 1 and 4 respectively. Finally, the partial-processor has its reset toggled.