Remy Ren

Using two late tokens on this assignment.

Lab Intro and Objectives

The main objective of this lab is to take the many components we have made in previous labs and combine them along with some additional simple logic to create a Single Stage CPU!

Brief Component Design Descriptions

All components from previous lab entries will not be given a brief design description. There are a heck of a lot of components and it'd kinda just fill space when you can just look back.

1. Components from previous labs: MUX5, MUX64, ShiftLeft2, SignExtend (modified to account for different instruction types), FA (FullAdder), ADD, ALU, ALUControl (modified to account for I-format ADDI, SUBI), CPUControl, DMEM, registers

2. PC: The program counter for the CPU! This uses behavioral modeling on the account of the clock determining when PC passes the next address. It takes in a clock, write enable, reset signal, as well as some input address for the next PC address. It will output the current PC address upon any rising clock edge.

3. singlecyclecpu: Uses structural modeling, combining all previous components to construct a single cycle CPU. The logic is derived primarily from Figure 4.23 in the textbook, or from the Green Card for referencing instructions.

   It requires a clock and reset signal, but as for output only has various debug signals.

4. sscpu_testbench: Uses structural modeling. Takes in IMEM from the assignment and allows programs to be run from there.

A Brief Explainer on My Testbenches

All testbenches are modeled using structural as they require their to be tested components in their architecture. In this case, as singlecyclecpu is a “black box” in terms of its outputs, only giving debug outputs, I didn’t bother with assertion statements. Instead, the primary purpose of my testbench, sscpu_testbench, was to run a program from IMEM, then allow me to run GTKWave to be able to see signals within the CPU for debugging.

Results and Waveforms

Test Program 1: Computation (imem_comp)

```
ADDI X10, X11, 1
ADDI X10, X11, 2
ADDI X9, X9, 1
SUBI X9, X9, 1
ADD  X10, X9, X11
-------- Assembly Translation ---------
16'h0000:  out = 32'b10010001000000000000010101101010; // ADDI X10, X11, 1
16'h0001:  out = 32'b10010001000000000000100101101010; // ADDI X10, X11, 2
16'h0002:  out = 32'b10010001000000000000010100101001; // ADDI X9, X9, 1
16'h0003:  out = 32'b11010001000000000000010100101001; // SUBI X9, X9, 1
16'h0004:  out = 32'b10001011000010110000000100101010; // ADD  X10, X9, X11
```

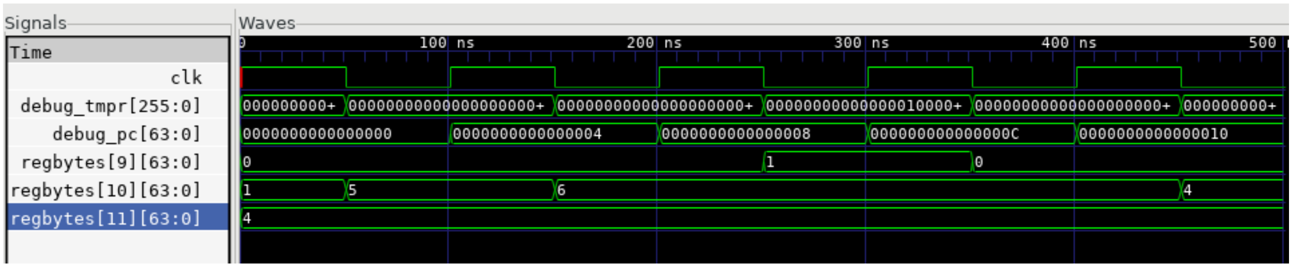


Figure 1: Computation Test in GTKWave

Here is a breakdown of the computation test in **Figure 1:**

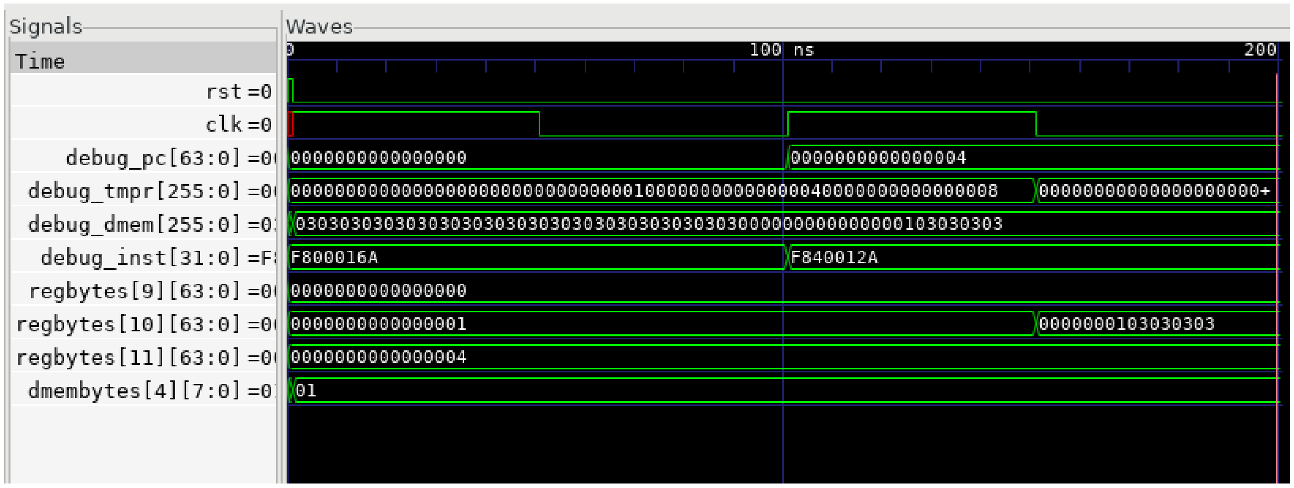
1. At the start, X9=64d”0”, X10=64d”1”, and X11=64d”4”. In this case, regbytes[9],[10],[11] correspond with X9,10,11.

2. (1st Instruction) X10 gets X11 + d”1”, so X10=”5” after first falling edge of clock.
3. X10=X11 + d”2”, so X10=”6” after second falling edge.
4. X9=X9 + d”1”, so X9=”1” after third falling edge.
5. X9=X9 - d”1”, so X9=”0” after fourth falling edge.
6. X10=X9 + X11, so X10=”4” after fifth falling edge.

As all waveforms and registers correspond with the expected results of the computation test, it is a success!

Test Program 2: Load and Store (imem_ldstr)

```
STUR X10, [X11, 0]
LDUR X10, [X9, 0]
----- Assembly Translation ----
16'h0000:  out = 32'b11111000000000000000000101101010; // STUR X10, [X11, 0]
16'h0001:  out = 32'b11111000010000000000000100101010; // LDUR X10, [X9, 0]
```



```
Signals                 Waves
Time                    0                                 100 ns                           200
rst =0
clk =0
debug_pc[63:0] =0(      0000000000000000                  0000000000000004
debug_tmpr[255:0] =0(   0000000000000000000000000000000100000000000000004000000000000000008   000000000000000000000+
debug_dmem[255:0] =0:   03030303030303030303030303030303030303030303000000000000000103030303
debug_inst[31:0] =F:    F800016A                          F840012A
regbytes[9][63:0] =0(   0000000000000000
regbytes[10][63:0] =0(  0000000000000001                                                   0000000103030303
regbytes[11][63:0] =0(  0000000000000004
dmembytes[4][7:0] =0:   01
```

Figure 2: Load and Store Test in GTKWave

Here is a breakdown of the computation test in **Figure 2:**

1. All bytes in DMEM are initialized to d"3". Additionally, X9=64d"0", X10=64d"1", and X11=64d"4".
2. On the first rising edge of the clock, X10's contents are stored within the contents at the memory address held in X11. In this case it is at DMEM(4) as X11=d"4". Thus, DMEM(4)=d"01".
3. On the second rising edge of the clock, the next instruction is executed. This loads the contents of what's at the memory address held in X9=d"0" into X10. Therefore X10=DMEM(0), which as all bytes in DMEM were initialized to d"3", X10=x"0000000103030303"

As all waveforms and registers correspond with the expected results of the computation test, it is a success!

Test Program 3: p1 (imem_p1)

```
ADDI X9, X9, 1
ADD  X10,X9,X11
STUR X10, [X11,0]
LDUR X12, [X11, 0]
CBZ X9, 2
B 3
ADD X9, X10, X11
ADDI  X9, X9, 1
ADD   X21, X10, X9
----- Assembly Translation ----
16'h0000:  out = 32'b10010001000000000000010100101001; // ADDI X9, X9, 1
16'h0001:  out = 32'b10001011000010110000000100101010; // ADD  X10,X9,X11
16'h0002:  out = 32'b11111000000000000000000101101010; // STUR X10, [X11,0]
16'h0003:  out = 32'b11111000010000000000000101101100; // LDUR X12, [X11, 0]
16'h0004:  out = 32'b10110100000000000000000001001001; // CBZ X9, 2
16'h0005:  out = 32'b00010100000000000000000000000011; // B 3
16'h0006:  out = 32'b10001011000010110000000101001001; // ADD X9, X10, X11
16'h0007:  out = 32'b10010001000000000000010100101001; // ADDI  X9, X9, 1
16'h0008:  out = 32'b10001011000010010000000101010101; // ADD   X21, X10, X9
```

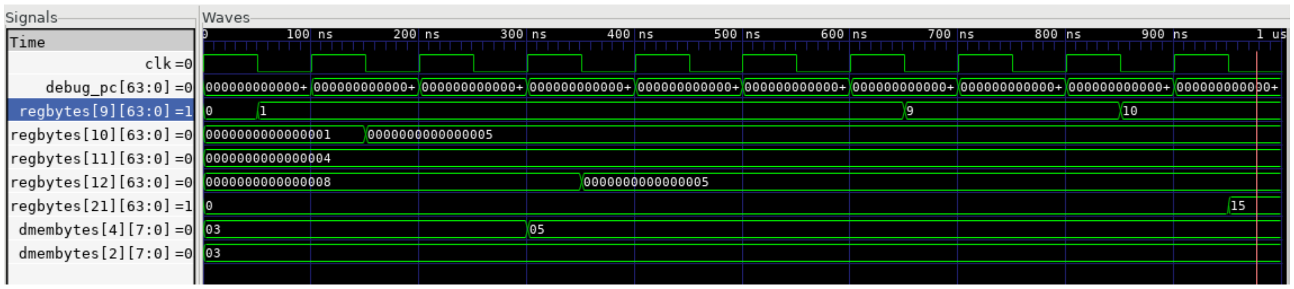


Figure 3: p1 Test in GTKWave

Here is a breakdown of the computation test in **Figure 3:**

All bytes in DMEM are still initialized to d”3” and X9=64d”0”, X10=64d”1”, and X11=64d”4”.

I will now follow the program and test results by the nth clock cycle. eg. 1. is the first clock cycle.

1. X9=X9 + d”1”, so X9=d”1”
2. X10=X9 + X11, so X10=d”1” + d”4” = d”5”
3. X10’s contents are stored within the contents at the memory address held in X11, DMEM(4). This is seen in the falling edge of this clock cycle where dmembytes[4] = 05.
4. X12 gets the contents stored at the memory address of X11, DMEM(4). Therefore X12=d”5”, reflected in the waveform.
5. Here, if X9 is 0, CBZ will skip two instructions. As X9=d”1”, it does not. Sorry that the program counter isn’t really visible in its values, but it occurs.
6. Unconditional branch three instructions, arriving at the second to last ADDI.
7. X9=X9 + d”1” = d”10”, I might have an odd bug? X9 should be getting just d”1”.
8. The final ADD runs, where X21=X10 + X9 = d”10” + d”5” = d”15”

It seems like I have an error in my either my branching or ADDI, not sure why the load is occurring as such. Unsuccessful test for now, will work on this in between now and lab4.

<u>**Test Program 4: p1 (imem_p1), BUT NOW X9 IS -1!**</u> Assuming that having X9 is supposed to be run with the final test, p1, here are the results.

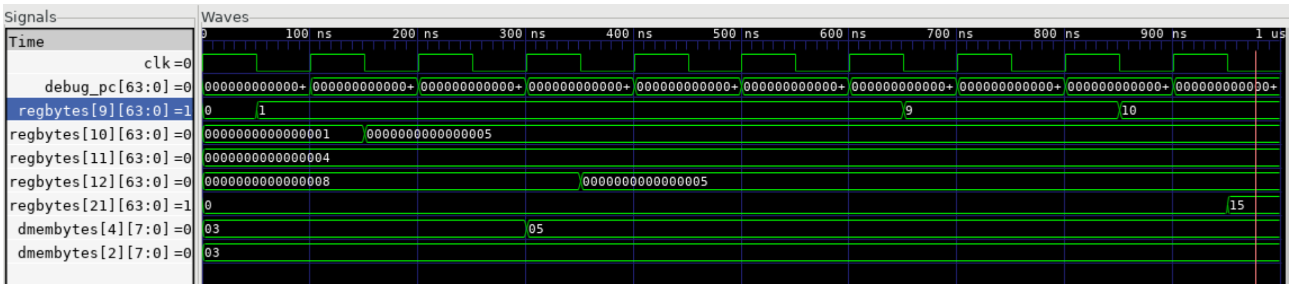


Figure 4: p1 Test with X9 = d"-1" in GTKWave

Here is a breakdown of the computation test in **Figure 4:**

All bytes in DMEM are still initialized to d"3" and X9=64d"-1"= All Fs Under Two's Complement, X10=64d"1", and X11=64d"4".

1. X9=X9 + d"1", so X9=d"0" from overflow.
2. X10=X9 + X11, so X10=d"0" + d"4" = d"4"
3. X10's contents are stored within the contents at the memory address held in X11, DMEM(4). This is seen in the falling edge of this clock cycle where dmembytes[4] = 04.
4. X12 gets the contents stored at the memory address of X11, DMEM(4). Therefore X12=d"4", reflected in the waveform.
5. Here, X9 is 0, CBZ will skip two instructions.
6. Should be X9=X9 + d"1" = d"2"... But for some reason the same load error occurs. I unfortunately don't have time at the moment to look into this further.

7. The final ADD runs, where X21=X10 + X9 = d"10" + d"5" = d"15"

It seems like I have an error in my either my branching or ADDI, not sure why the load is occurring as such. Unsuccessful test for now, will work on this in between now and lab4.

The answer to the question as to what value would be written to memory if X9 = -1 is that it would be d"4" to DMEM(4).