**EE 310 – Lab 7 Report**

**NAU, 17 April 2020**

**By Alenn Wright**

**Problem Description**

In this lab, we are going to start with a modified version of MIPSzy we used in Lab6. This version has support for offsets for lw and sw instructions, i.e. you can now use non-zero offsets, such as lw $t1, 5000($t0) and sw $t2, 5016($zero). You will then implement add instructions on this version of MIPSzy, i.e. you will modify it to have the ability to execute sub instruction in addition to the existing lw, sw, addi, add instructions.

The other specifications are still the same:

• Its data memory space is limited to 1024 32-bit words, from addresses 4096 to 8188 (recall that addresses are incremented by 4, i.e. 4096, 4100, 4104, …),

• It can store programs of up to 1024 lines long (more than enough for our 10-20 line programs),

• It can NOT implement any branches, jumps, comparisons, and hence NO loops or subroutines are possible at this time.

|  |
| --- |
|  |

Figure 1. Expected behavior of the circuit

**Solution Plan**

In order to solve the problem explained above, I will create 2 registers to hold the value of x and y. Then one register to hold the value of the operations. Then i will use the running hold value and manipulate it to reduce instructons

|  |
| --- |
|  |

Figure 2. State diagram for the proposed solution

**Implementation and Test Plan**

I have implemented the solution plan explained above, by *the code is exactly what I set out to do and ran the first time after I added all the implementations of add\_sub.*

|  |
| --- |
| addi $t6, $zero, 5000  lw $t1, 0($t6)  lw $t2, 4($t6)  sub $t3, $t2, $t1  sw $t3, 0($t6)  add $t3, $t3, $t2  sw $t3, 4($t6)  add $t3, $t3, $t2  sw $t3, 8($t6)  sub $t3, $t3, $t1  sw $t3, 12($t6)  add $t3, $t3, $t2  add $t3, $t3, $t1  sw $t3, 16($t6)  sub $t3, $t3, $t1  sub $t3, $t3, $t1  sw $t3, 20($t6)  add $t3, $t3, $t1  add $t3, $t3, $t1  add $t3, $t3, $t2  sw $t3, 24($t6)  sub $t3, $t3, $t1  sub $t3, $t3, $t1  sw $t3, 28($t6)  MIPSzy\_0.IM.memory[0] = 'b00100000000011100001001110001000;  MIPSzy\_0.IM.memory[1] = 'b10001101110010010000000000000000;  MIPSzy\_0.IM.memory[2] = 'b10001101110010100000000000000100;  MIPSzy\_0.IM.memory[3] = 'b00000001010010010101100000100010;  MIPSzy\_0.IM.memory[4] = 'b10101101110010110000000000000000;  MIPSzy\_0.IM.memory[5] = 'b00000001011010100101100000100000;  MIPSzy\_0.IM.memory[6] = 'b10101101110010110000000000000100;  MIPSzy\_0.IM.memory[7] = 'b00000001011010100101100000100000;  MIPSzy\_0.IM.memory[8] = 'b10101101110010110000000000001000;  MIPSzy\_0.IM.memory[9] = 'b00000001011010010101100000100010;  MIPSzy\_0.IM.memory[10] = 'b10101101110010110000000000001100;  MIPSzy\_0.IM.memory[11] = 'b00000001011010100101100000100000;  MIPSzy\_0.IM.memory[12] = 'b00000001011010010101100000100000;  MIPSzy\_0.IM.memory[13] = 'b10101101110010110000000000010000;  MIPSzy\_0.IM.memory[14] = 'b00000001011010010101100000100010;  MIPSzy\_0.IM.memory[15] = 'b00000001011010010101100000100010;  MIPSzy\_0.IM.memory[16] = 'b10101101110010110000000000010100;  MIPSzy\_0.IM.memory[17] = 'b00000001011010010101100000100000;  MIPSzy\_0.IM.memory[18] = 'b00000001011010010101100000100000;  MIPSzy\_0.IM.memory[19] = 'b00000001011010100101100000100000;  MIPSzy\_0.IM.memory[20] = 'b10101101110010110000000000011000;  MIPSzy\_0.IM.memory[21] = 'b00000001011010010101100000100010;  MIPSzy\_0.IM.memory[22] = 'b00000001011010010101100000100010;  MIPSzy\_0.IM.memory[23] = 'b10101101110010110000000000011100; |

Figure 3. Verilog code for the proposed solution

|  |
| --- |
|  |

Figure 5. Lab pictures of the running solution