Zach Dubinsky

CSC-270

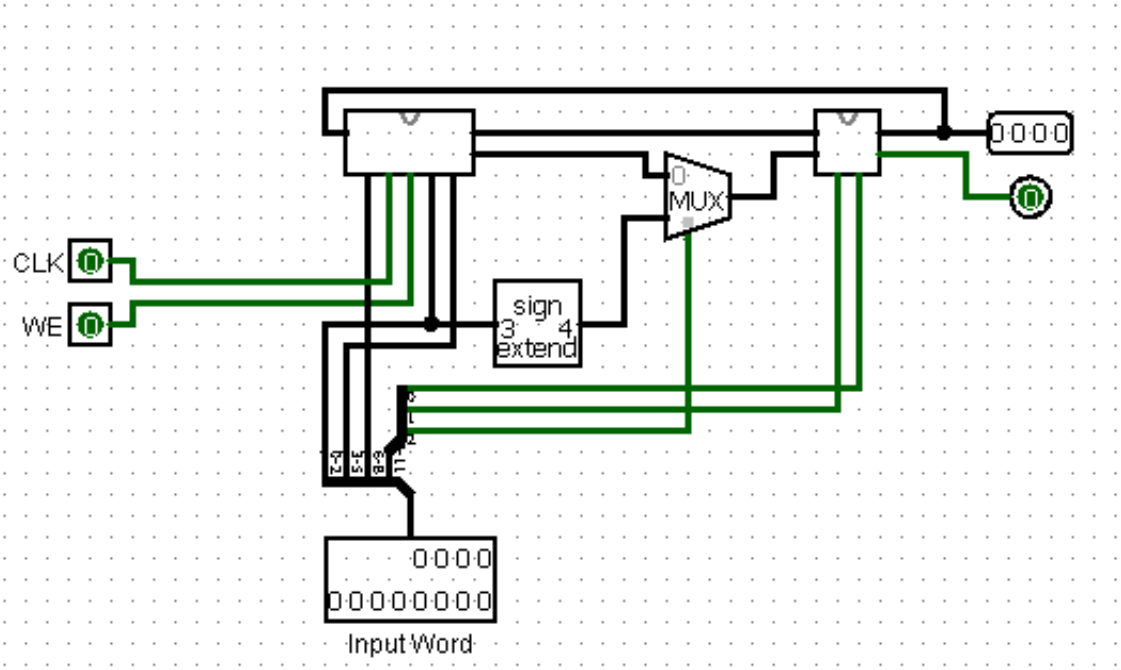
Prof.Rieffel

1/25/2022

Lab 3

I affirm that all included work is my own in accordance with the class syllabus and the Union College Honor Code. [Signed: Zachary Dubinsky, 1/25/2022].

1. My Logism Circuit



2. Description of the Machine Language.

I designed my machine language in accordance with the assignment, sadly it does not align with MIPS. The input word accepts twelve bits. For R-type instructions the form of the input word is,

<op(3)>, <wa(3)>,<ra1(3)>,<ra2(3)>.

For I-type instructions the form of the input word is,

<op(3)>, <wa(3)>,<ra1(3)>,<imm(3)>.

The particular type of instruction is denoted by the first digit in the input word, in the op-code. A leading zero indicates an R-type instruction and sends data from the register to the ALU. A leading one indicates an I-type instruction and switches the multiplexor to accept a three-bit input value from the instruction word, where the second register address would be. A sign extender is used to provide the "missing bit", so immediate values are treated as two's complement binary encodings. The input terminal would not resize, so there are two rows of input. I chose to have the upper and leftmost bit to be the most significant, and the lower rightmost bit to be the least significant. Since each component of the input word is three bits I was able to use a wire splitter that takes the twelve-bit input word and splits it into four three-bit binary values, the digits of which are adjacent to each other in the input word. The op-codes were split once more into one-bit binary values that are sent to the op-code terminals of the ALU, and one that is sent to a multiplexor that accommodates immediate values. The read and write address inputs are simply wired into the appropriate register terminal. RA2 is also wired into a sign extender, and then a multiplexor, to allow immediate values.

It should be noted that the "clock" and "write-enabled" inputs are not in the input word because that just wouldn't work. There are two, independent, one-bit inputs for these actions.

3. More Questions

The largest immediate value that can be placed in the register is 1111 and the smallest value that can be placed in the register is 0000. I tested these with

ADDI $s0 $s0 0d15

ADDI $s0 $s0 0d0,

and

ADDI $s0 $s0 0d0

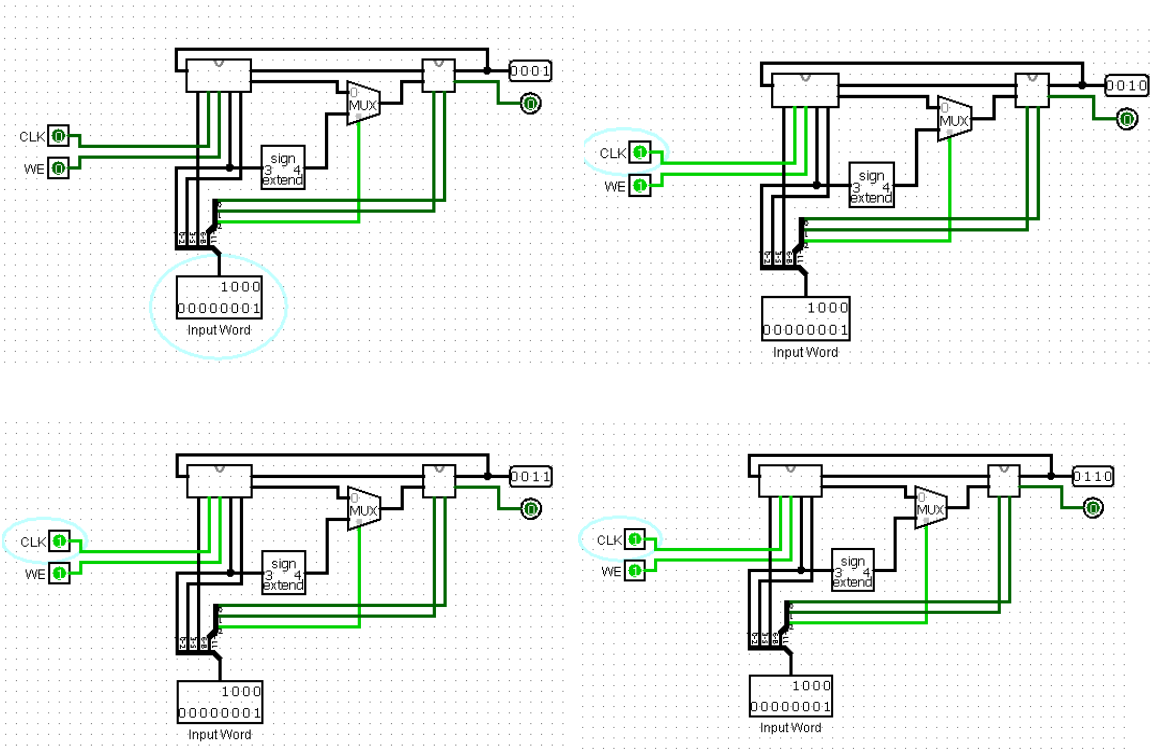
ADDI $s0 $s0 0d1,

respectively. Both outputs were correct. I find this range to be useful in the sense that the entirety of a four-bit binary value may be stored in the four-bit D-register cells of the register file. To improve the range, the size of the D-register cells could be made larger. The logic of the modularized circuit would not change, only the size of inputs and wiring. The ALU and register file would need to be adjusted similarly to accommodate this, but these changes would not be hard.

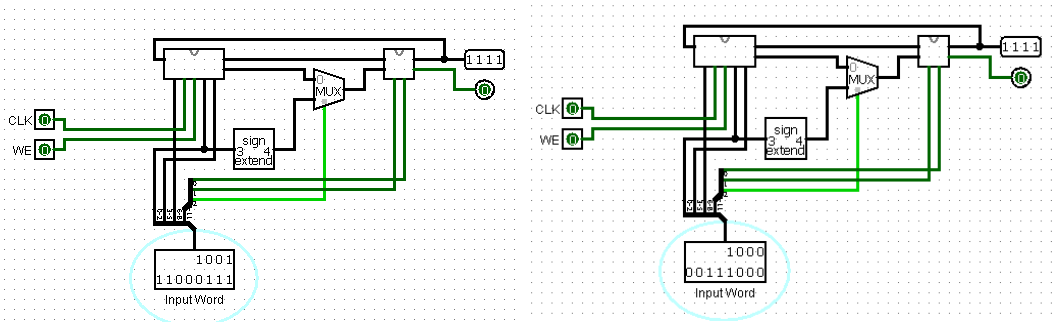
4. Proof of Functionality

I performed the iterative addition tests that were given as tips on Nexus. I tested immediate inputs to ensure that all values were allowed, which they were. I also performed a series of other additions as a sanity check. I've included screenshots of some tests below, the order of commands is left-to-right and from top-to-bottom.

Ex 1. The Nexus iterative test. The command was executed twice between the second-to-last and last image.



Ex 2. Storing a maximum value in register 111. Implicit addition of an empty register too.



Ex 3. Some nice addition between two registers.