Diep (Emma) Vu

Lab 3 Report

I. Description of Machine Language

The Machine Language is a 12-bit number with separate functions for each bit or groups of bits. We partition a 12-bit number into sections, and the bits are indexed in increasing order (the least significant bit is 0 while the most significant bit is 11)

- 11 (Most Significant Bit): This bit represents the datapath type, with 0 for R-type (arithmetic logic between values stored in registers, if any) and 1 for I-type (arithmetic logic between a value of a specified register with an immediate)
- 10 → 9: This pair of bits represents the function to be performed by the ALU, with the rules described in the table below.

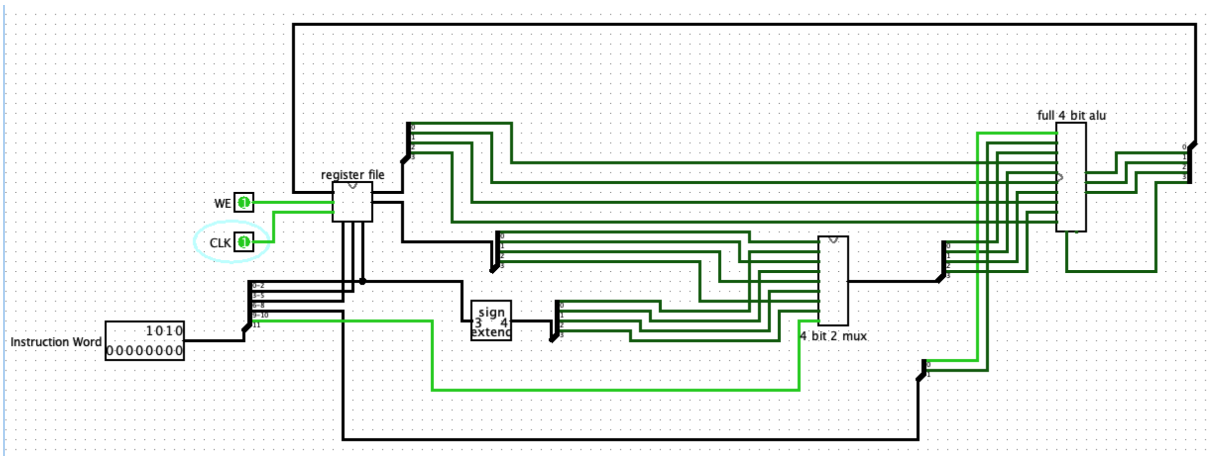
Bit 10	Bit 9	Function
0	0	RA_1 + RA_2
0	1	RA_1 AND RA_2
1	0	RA_1 OR RA_2
1	1	RA_1 – RA_2

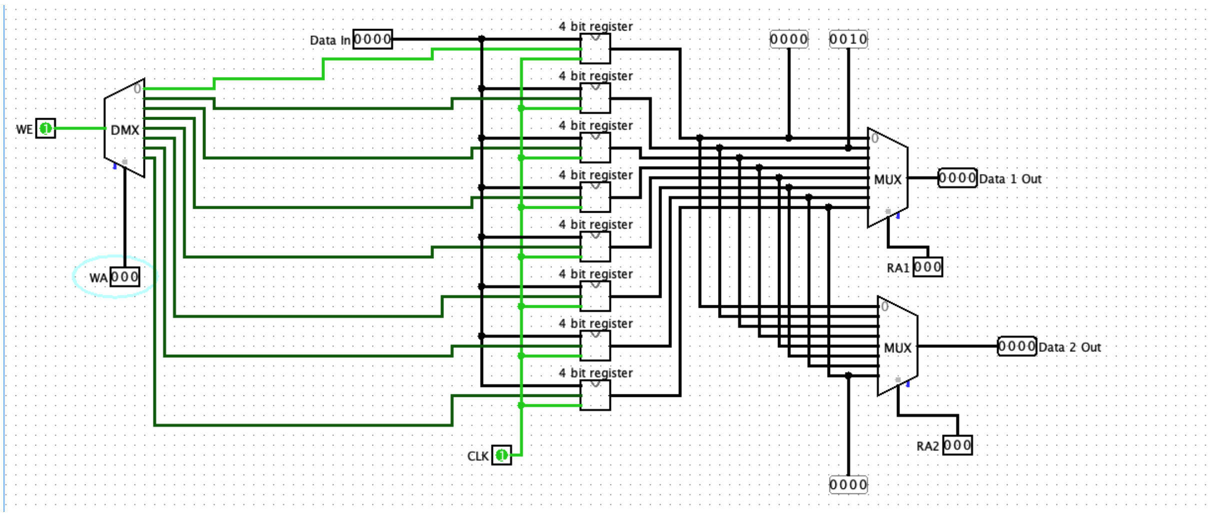
- 8 → 6: This group of bits represents the data path's Writing Address (WA), which register will be assigned with data passing from the ALU. We have 8 registers, so a 3-bit value that covers 8 numbers from 0 (000_2) to 7 is required (111_2). This 3-bit number and a bit representing whether or not writing to a register is enabled are connected to DMX to determine whether or not to assign data from the ALU and where to assign it. Once writing is enabled, data is only assigned when the Clock "goes high" - that is, when the bit representing the Clock changes from 0 to 1.
- 5 → 3: This group of bits specifies the datapath's first Reading Address (RA_1), which register the register file should use to feed to the ALU. We need 3 bits to represent which register to read from, just like the WA. This number is then fed into a MUX to determine which register to read from.
- 2 → 0: This group of bits represents the datapath's second Reading Address (RA_2). This group in the R-type datapath represents which register the register file should take to feed to the ALU, which is why it has 3 bits. This group represents the value of the immediate in the I-type datapath. However, because the immediate has only 3 bits and

the data provided to the ALU must have 4, we plug the immediate into a signed 3-to-4 bit extender to convert it to a 4-bit number.To decide whether to take the value from the register file or the immediate, we plug the second value and the immediate into a 4-bit 2 MUX, controlled by bit 11 that tells us the type of this data path.

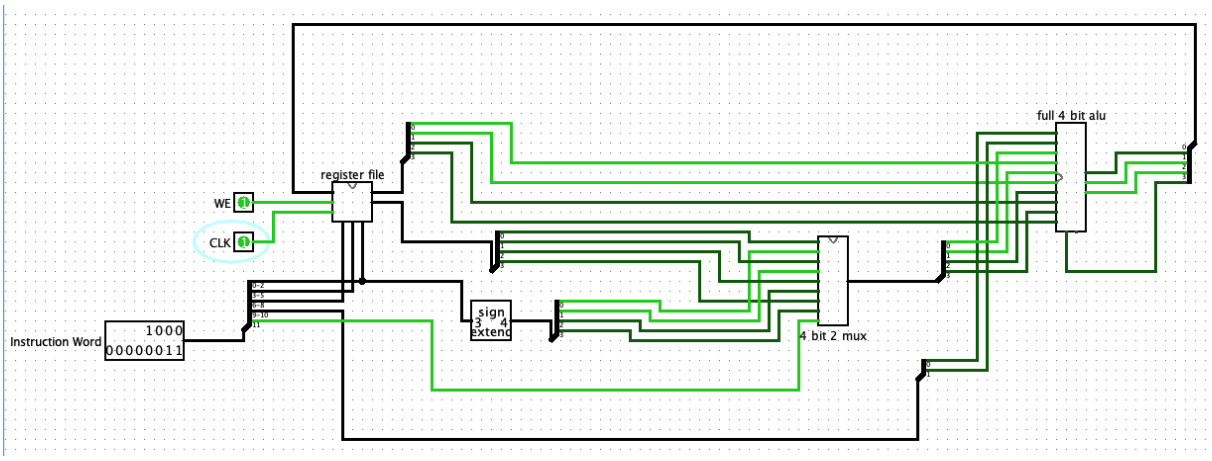
II. Testing

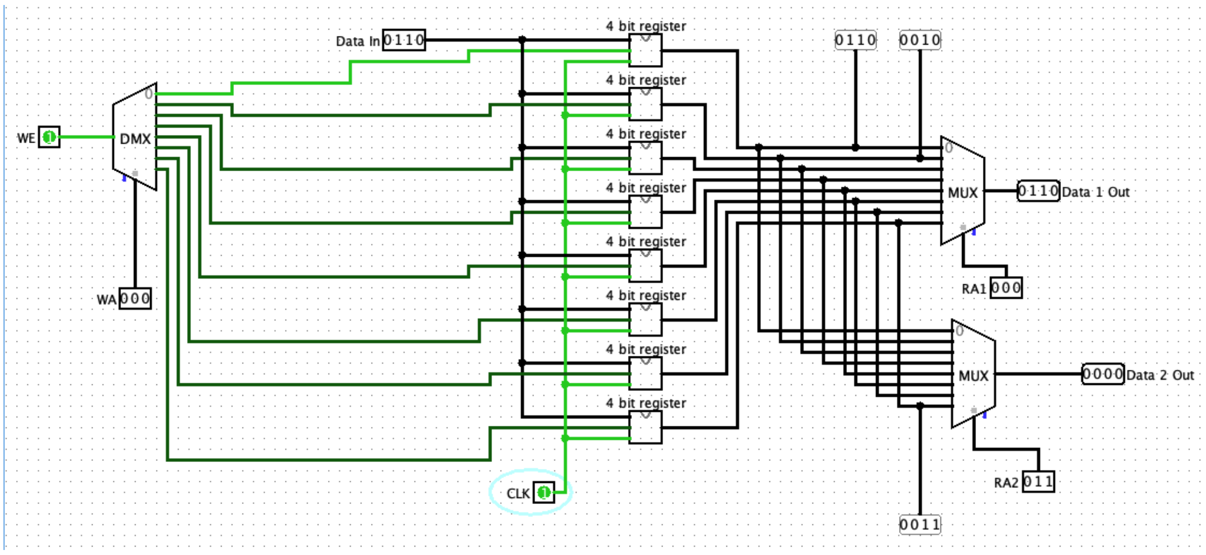
First, we assign 0 to R_0 via ANDI \$0 \$0 0 (that is 101 000 000 000) so R_0 has a value of 0. We can use a probe to check the value of R_0 after this.



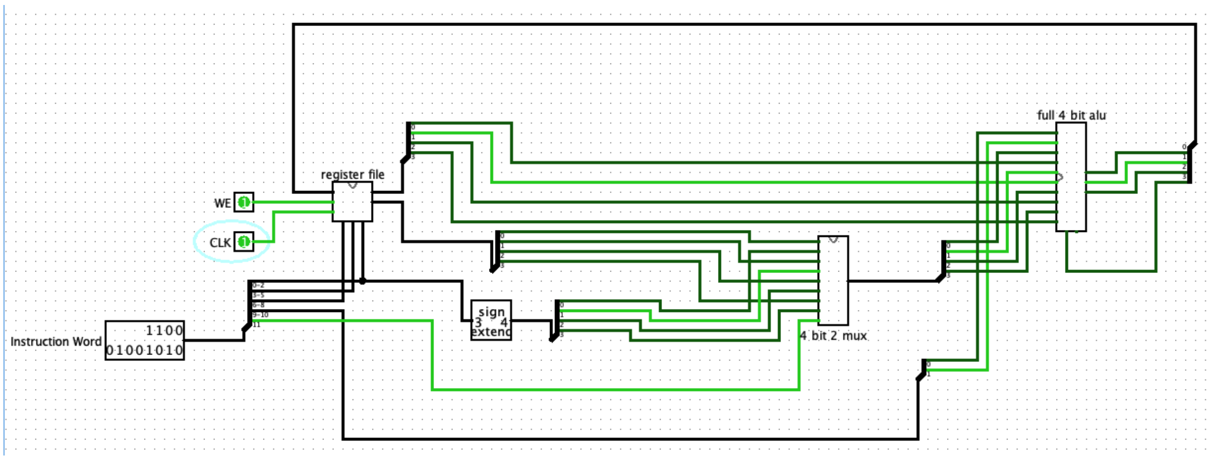


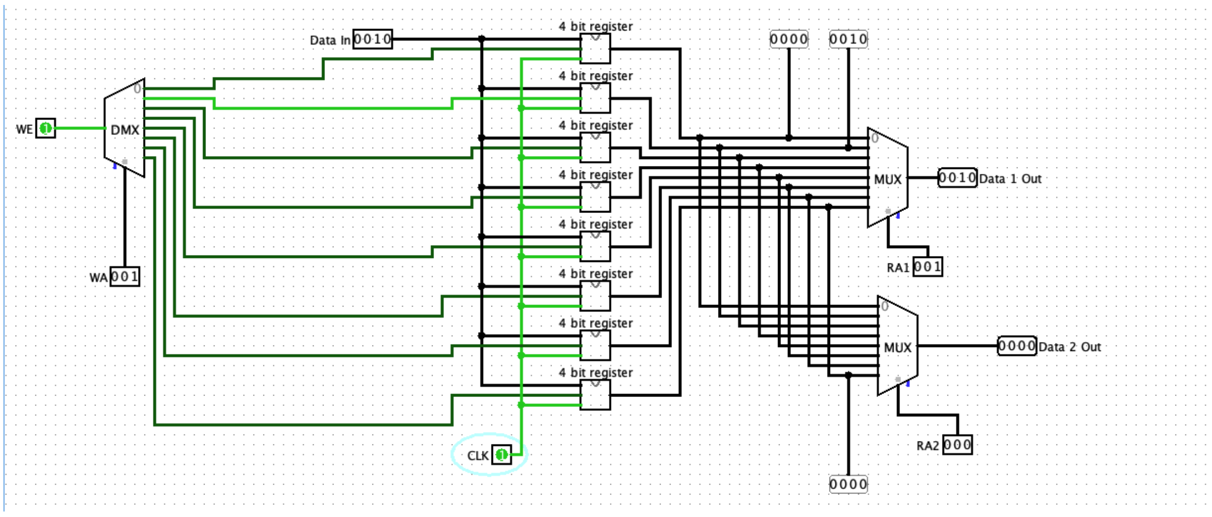
Then we add a random value 3 to R_0 via ADDI \$0 \$0 3 (which is 100 000 000 011)





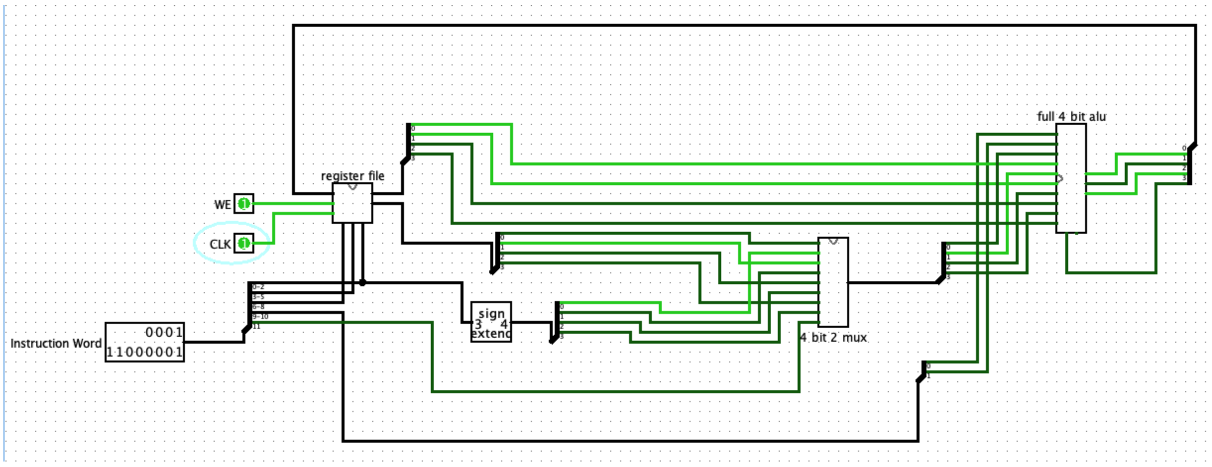
Then we assign 2 to R_1 by repeating the same process, ANDI \$1 \$1 0 (101 001 001 000) followed by ORI \$1 \$1 2 (110 001 001 010)

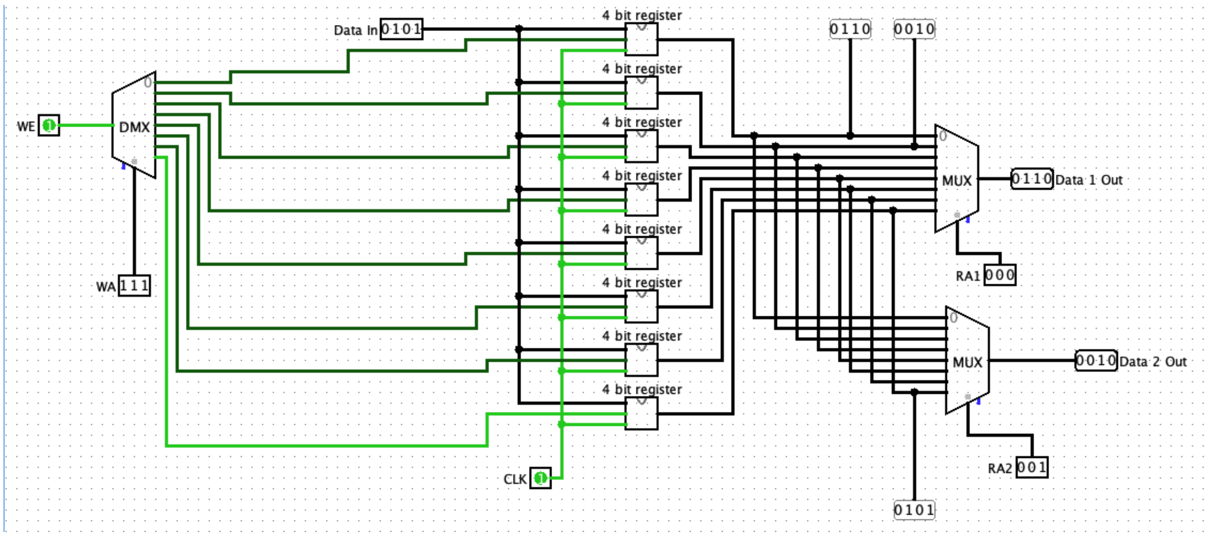




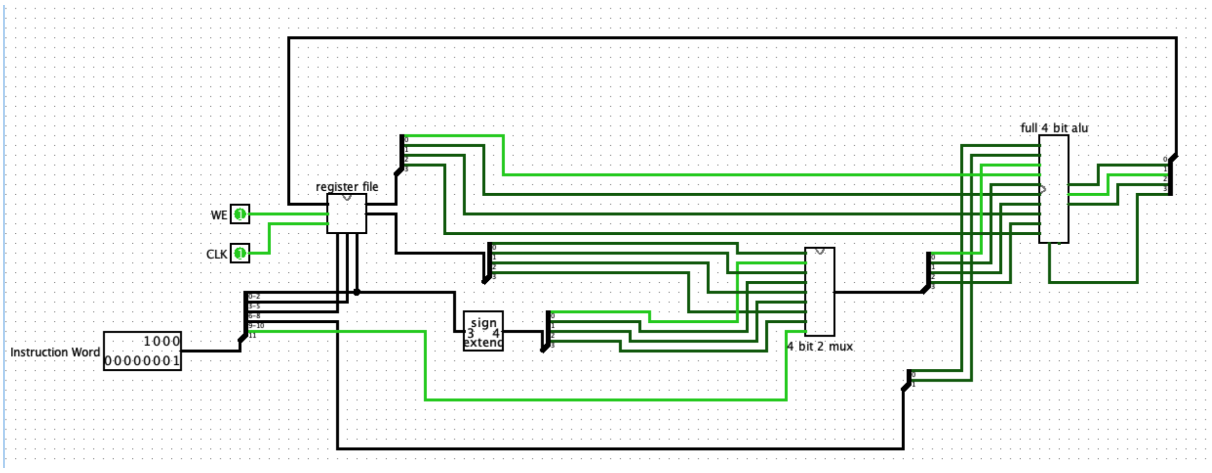
Now we have 2 registers with value, it is possible to test R-type instructions: ADD, SUB, AND, OR:

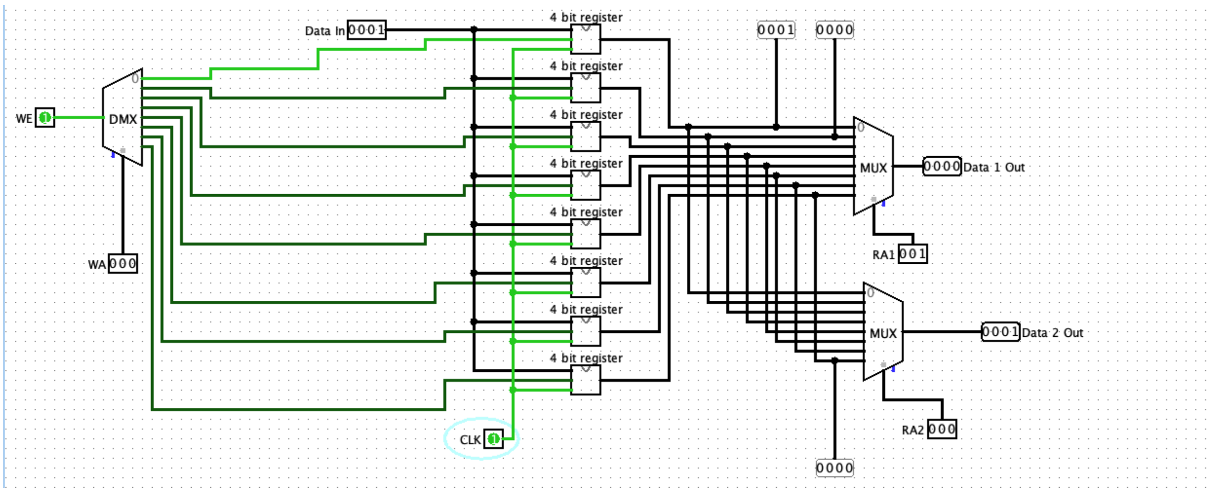
- ADD $7 $0 $1 (000 111 000 001):
  => Expect to see 0101_2 = 5 at register 7



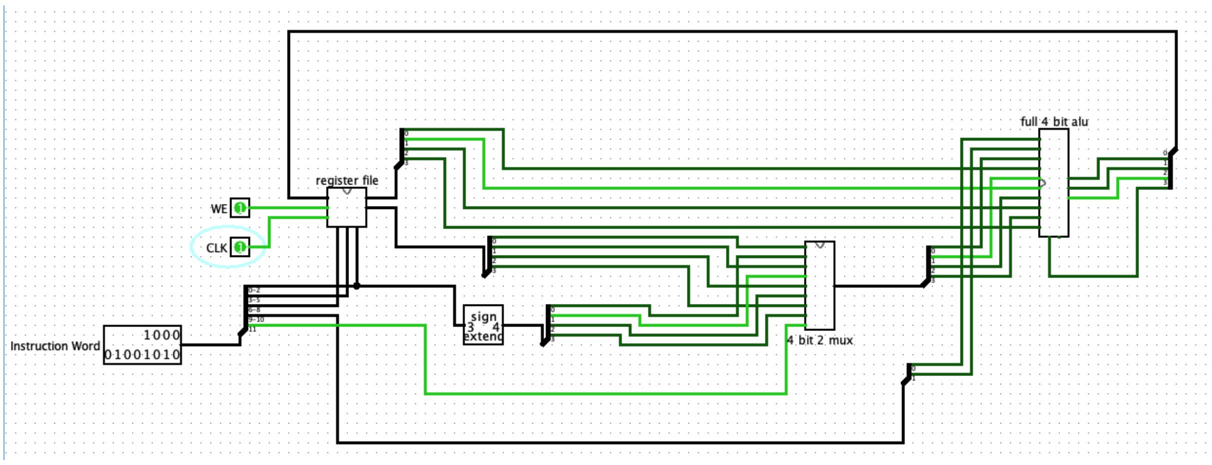


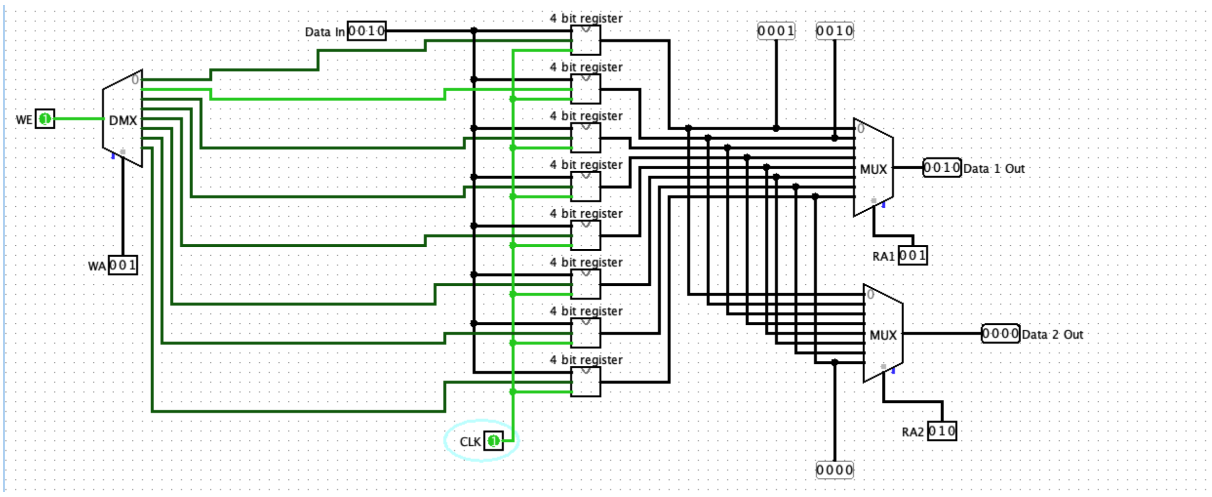
- ADDI \$0 \$0 1 (100 000 000 001):
  => Expect to see 0001_2 = 1 at register 0



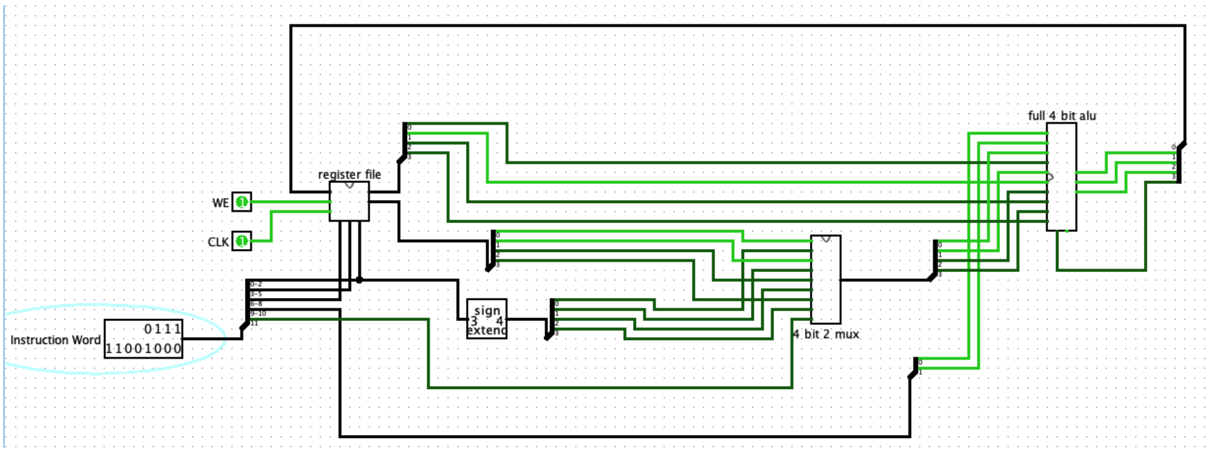


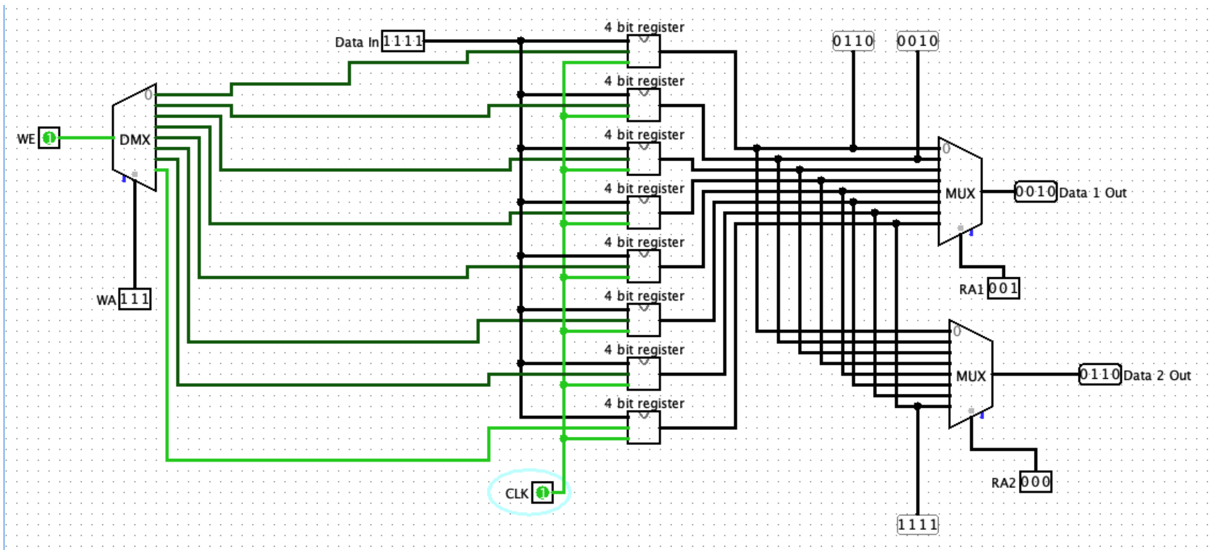
- ADDI \$1 \$1 2 (100 001 001 010):
  => Expect to see 0010_2 = 2 at register 1



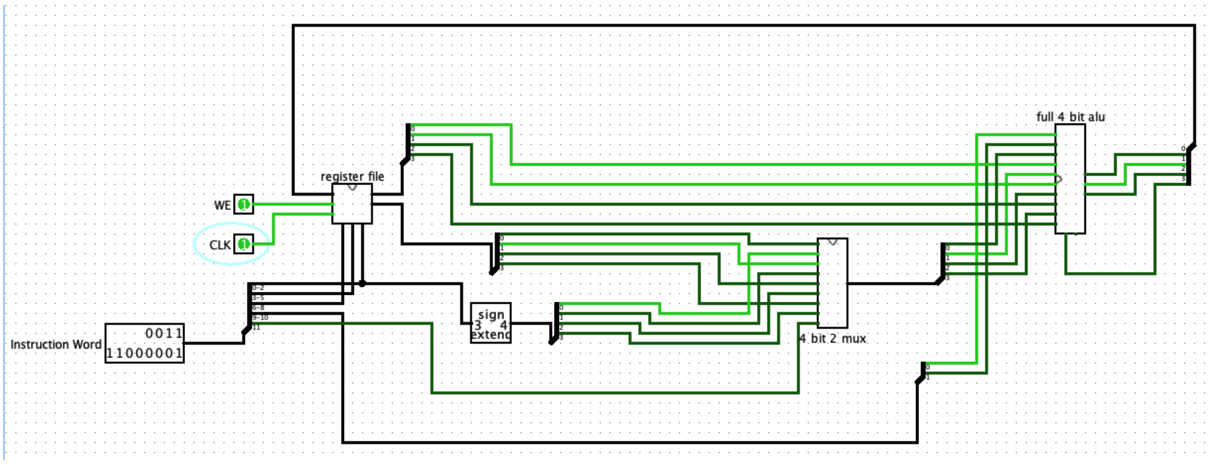


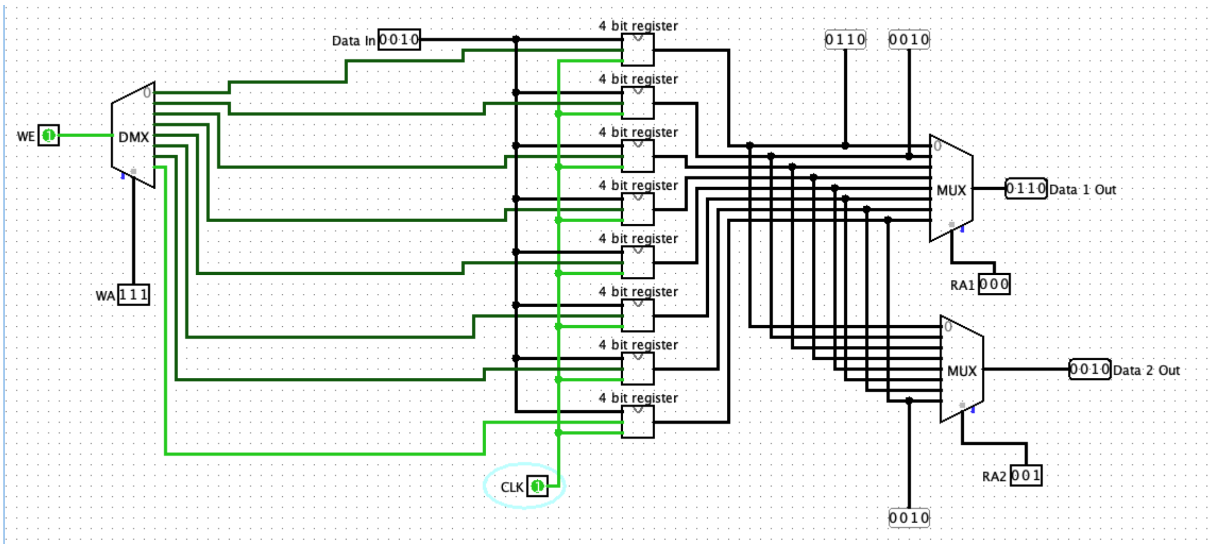
- SUB \$7 \$1 \$0 (011 111 001 000):
  => Expect to see 1111_2 = -1 at register 7



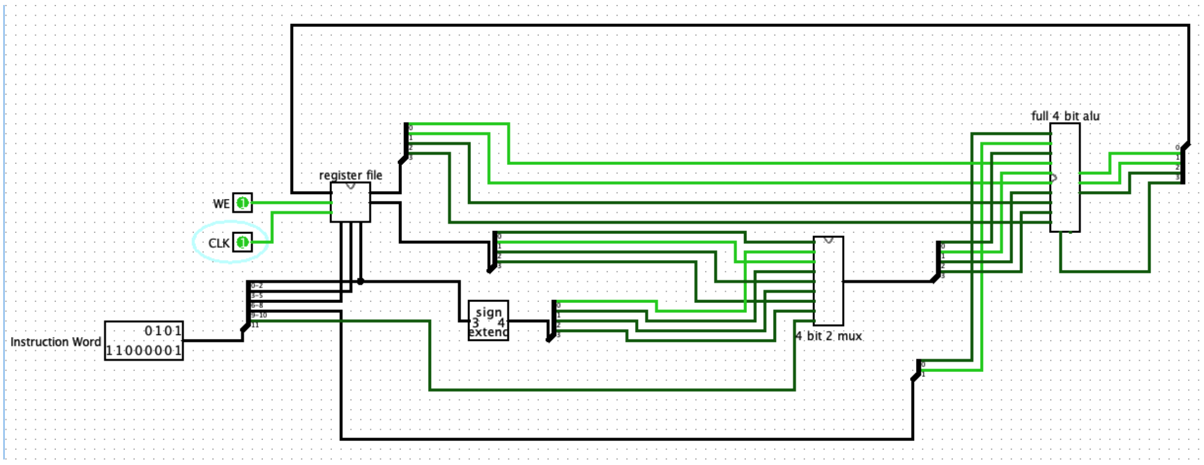


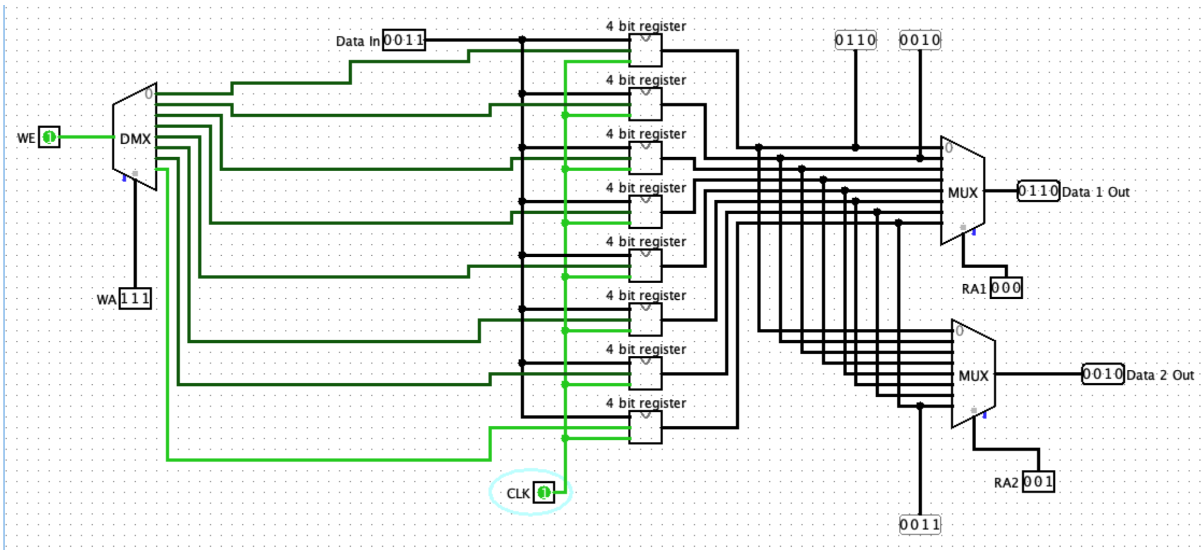
- AND \$7 \$0 \$1 (001 111 000 001):
  => Expect to see 0010_2 = 2 at register 7





- OR \$7 \$0 \$1 (010 111 000 001):
  => Expect to see 0011_2 = 3 at register 7





III. Analysis

- Part 6 has a missing bit since we are using only 3 bits to represent both the index of the register to read from and the value of the immediate, when the immediate should be a 4-bit number to be fed into the 4-bit ALU.
- The maximum immediate value that can be put into the register via ADDI is $011_2 = 3$.
- The minimum immediate value that can be put into the register via ADDI is $100_2 = -4$.
- The [-4, 3] is too small to be useful because it requires us to run ADDI numerous times to initialize a register with a larger value.
- However, the maximum 4-bit two's complement value for this lab is $0111_2 = 7$, while the smallest 4-bit two's complement value is $1000_2 = -8$. As a result, the range of values that a register can take is [-8, 7], hence [-4,3] is considered as well. To improve this, we can lengthen the instruction word to assign more bits for RA_2, which is the immediate in I-type, and increase the number of bits a register can carry to hold the value that can be sent from the immediate (for example a register in the MIPS architecture can hold 16 bits, which is also the number of bits allocated for the immediate).