**Computer Architecture ALU project**

**Annie Brey**

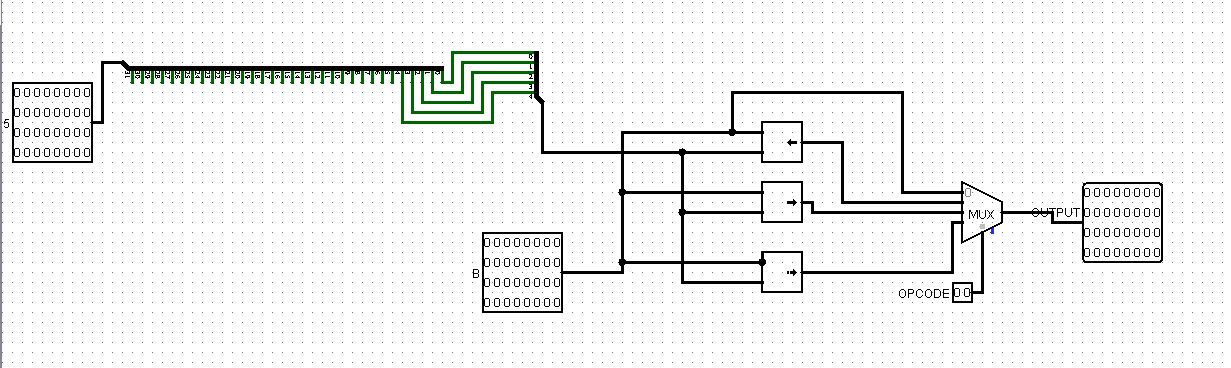
**Kenneth Neads**

**Shreyas Patil**

**Zabi Babar**

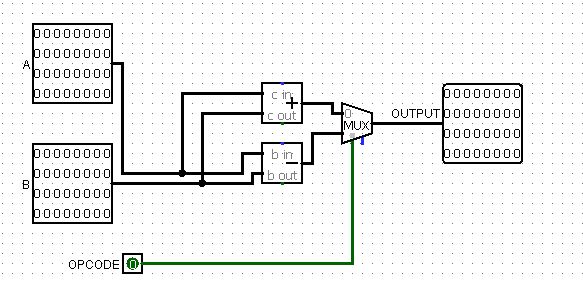
**1) Design and Implementation**

Shifter:



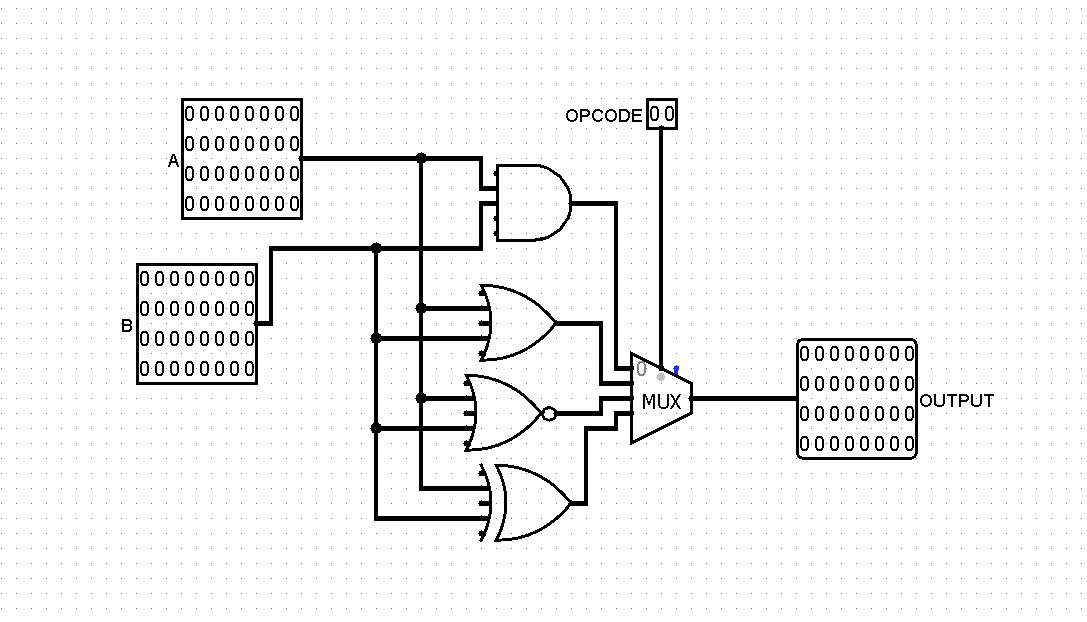
This is our shifter, we take a 32 bit input value (A) and extract the last 5 bits to choose how many bits the output gets shifted by. The other input (B) is the 32 bit number that gets shifted. The first input (00) to the Multiplexer is the logical left shifter where it shifts bits to the left and outputs that to the 32 bit output. The second input (01) to the Multiplexer is the logical right shifter where it shifts bits to the right and outputs that to the 32 bit output. The third input (10) to the Multiplexer is the arithmetic right shifter where it shifts bits to the right and fills the new left values with the MSB of the input; this number goes to the 32 bit output. The opcode for the MUX determines which shifting operation will be performed.

Adder:



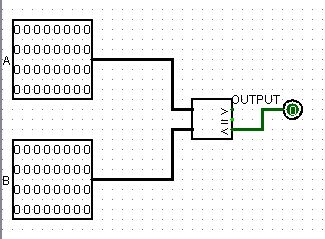
The adder can add or subtract 2 32 bit numbers depending on the opcode. The adder goes into the first input of the MUX and the subtractor goes into the second input of the MUX. If the opcode is 0, the 2 numbers will be added and that answer will be given to the 32 bit output. If the opcode is 1 the 2 numbers will be subtracted and that answer will be given to the 32 bit output.

Logic Unit:



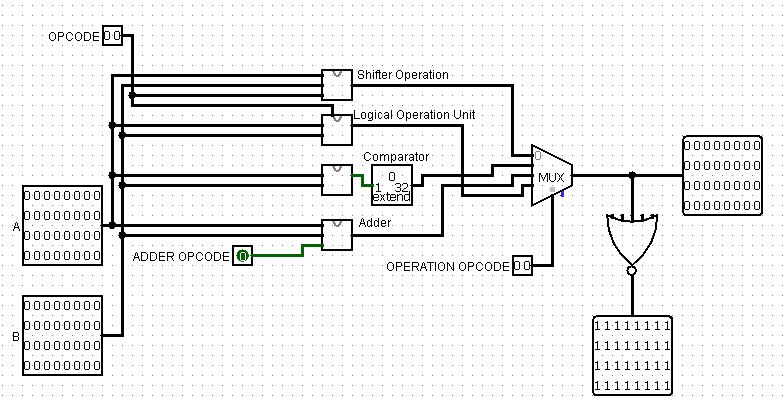
This is our logic unit, we designed it after the Logic unit in the lab instructions where a 32 bit input can either be AND, OR, NOR, or XOR with the 2nd 32 bit input. The first input (00) to the multiplexer is the AND operation. The second input to the MUX (01) is the OR operation. The third input to the MUX (10) is the NOR operation and the last input (11) to the MUX is the XOR operation . The opcode for the MUX selects the operation performed. The output is a 32 bit number.

Comparator:



This is our comparator, it compares 2 32 bit inputs (A and B) and if A is less than B then it outputs a single bit 1, which is later extended to a 32 bit number with the least significant bit being the solution to the comparator. If A is not less than B, the output is a 0.

ALU:



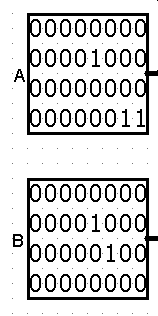
This is the final ALU, which has all of the above components. The 2 32 bit inputs are on the left as A and B

**2) Simulation and Testing**

ALU = 00 = shift

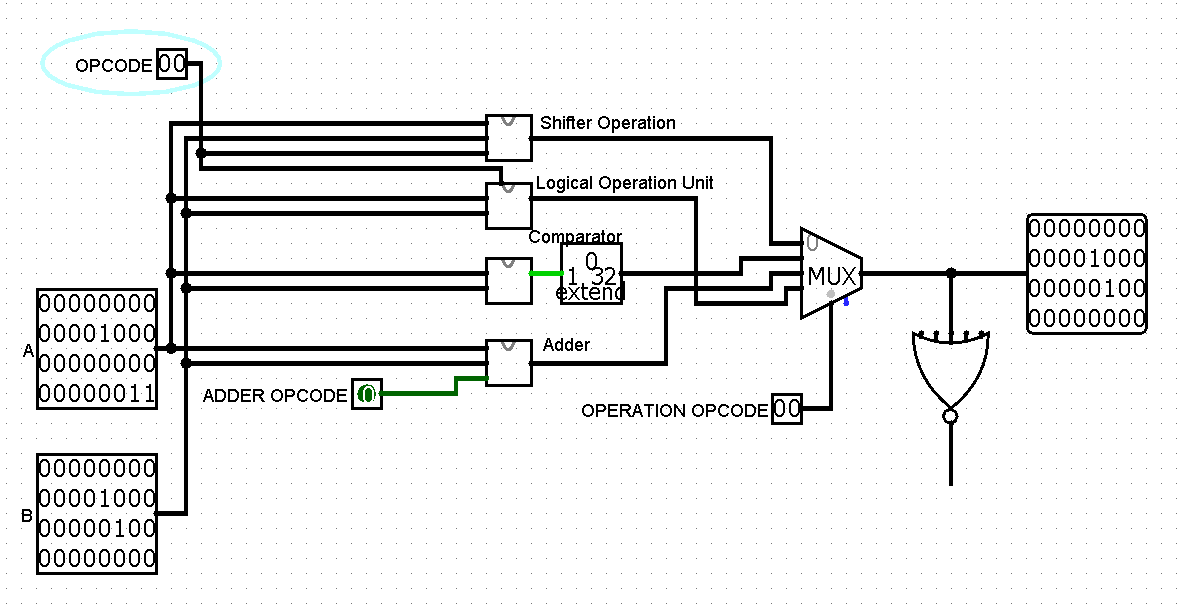
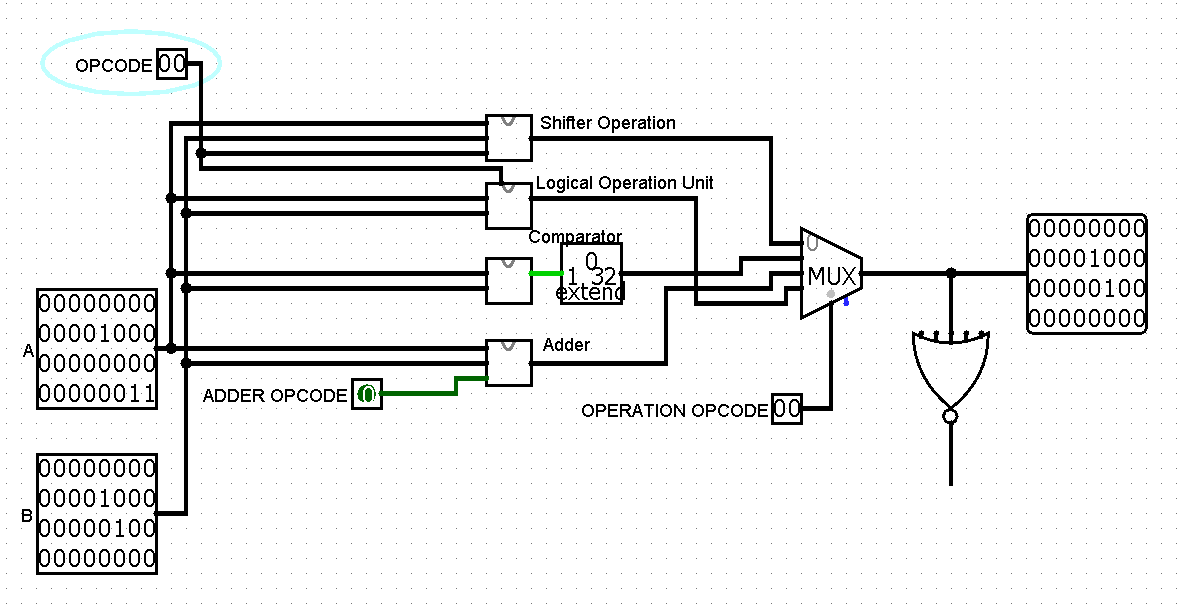
Shift Operation = 00 = None:

Input:



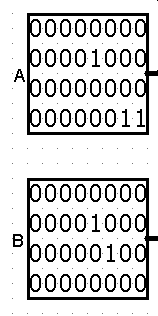
Expect Output: Since the shift opcode is 00, so the output should stay whatever input B was initially

Output:

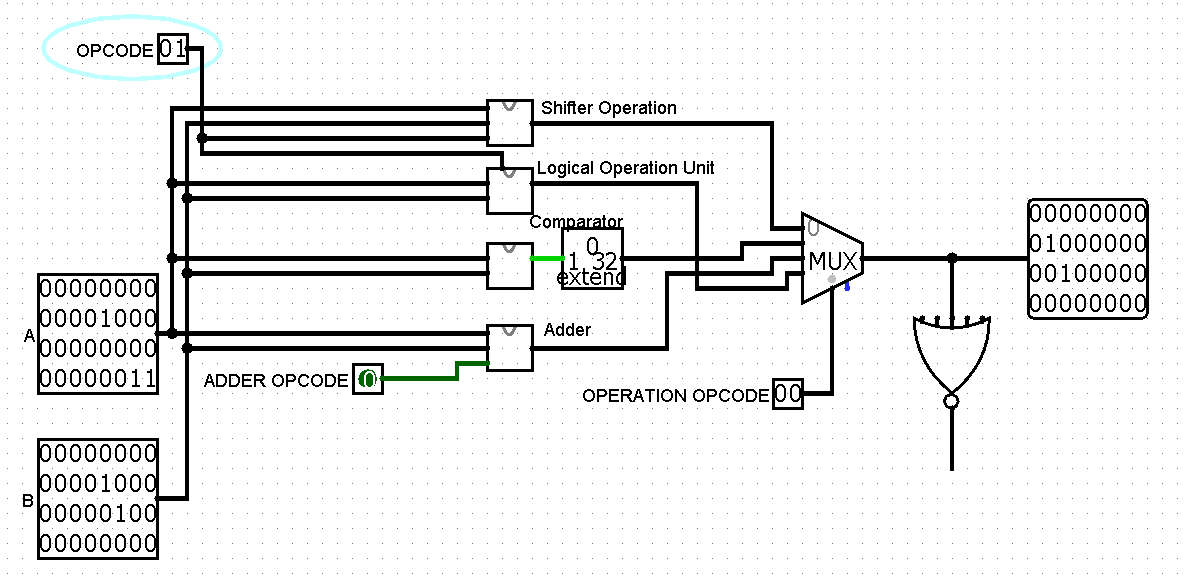
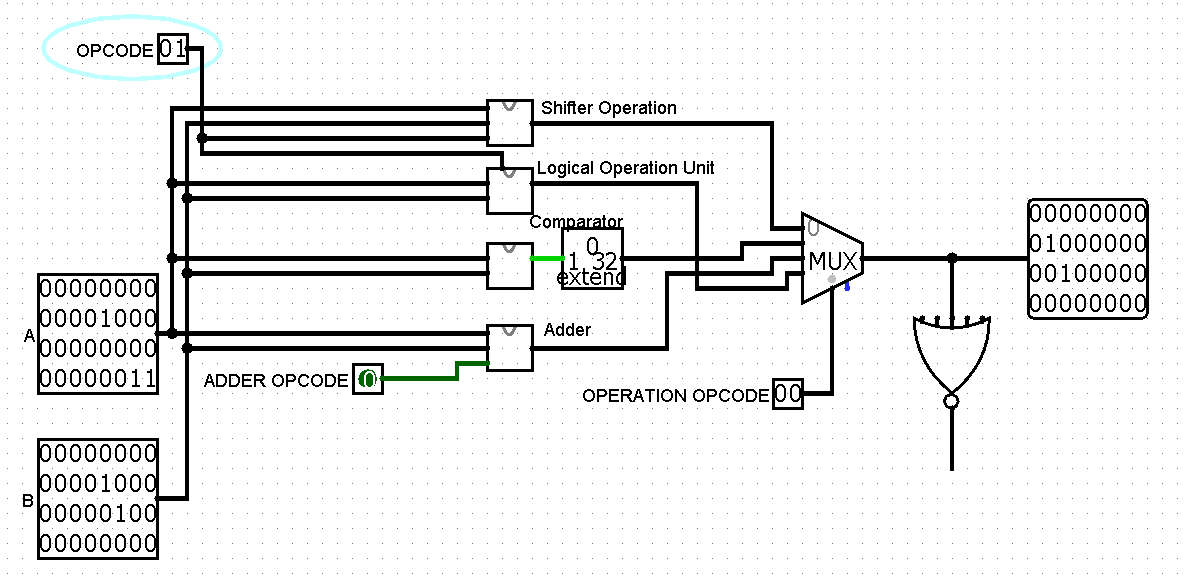
Shift Operation = 01 = SLL:

Input:



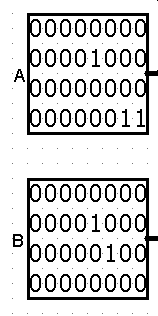
Expect Output: Since the shift opcode is 01, the bits of B should be shifted to the left, and the amount they are shifted corresponds to the last 5 bits of A. All values of B should be shifted 3 values to the left

Output:

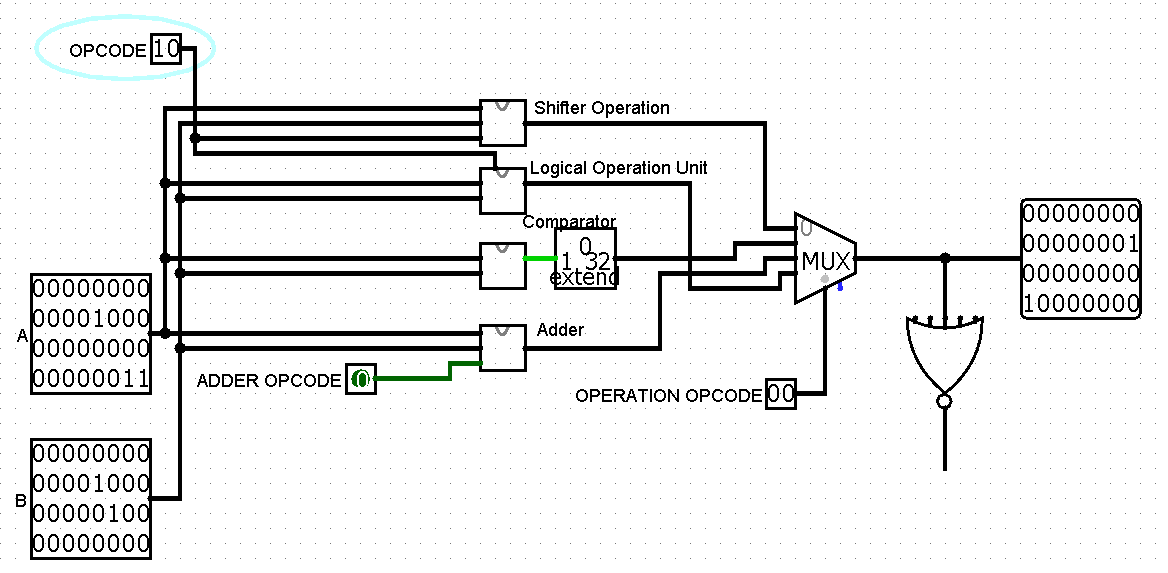
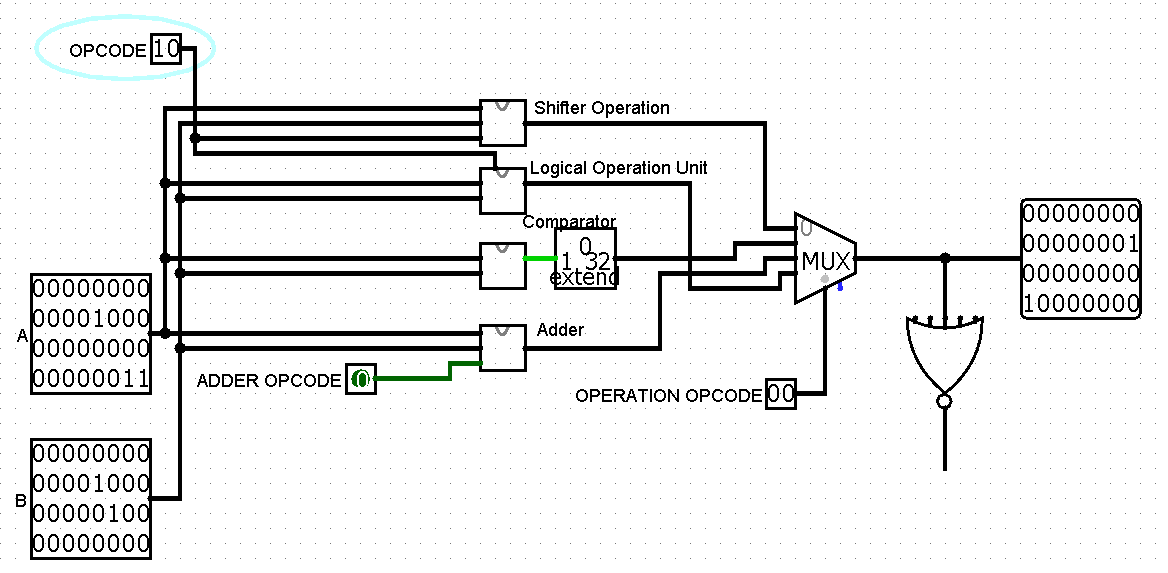
Shift Operation = 10 = SRL:

Input:



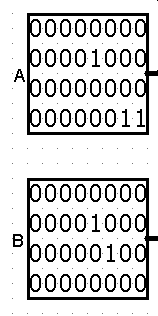
Expect Output: Since the shift opcode is 10, the bits in B should be shifted to the right. The amount they are shifted corresponds to the last 5 pits of A, so B should be shifted to the right by 3.

Output:

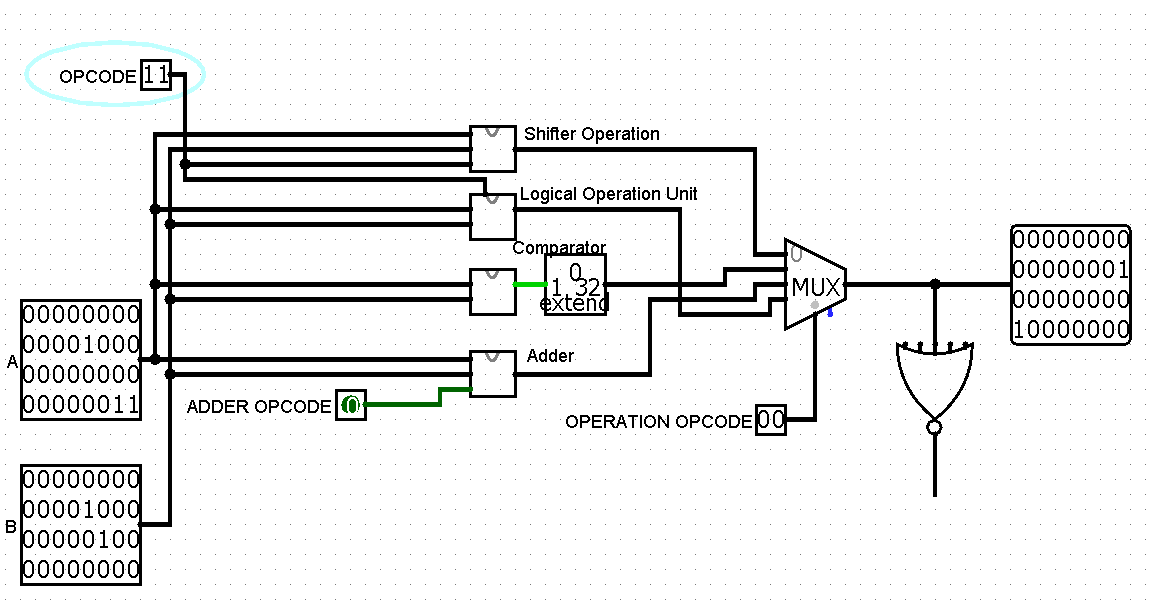
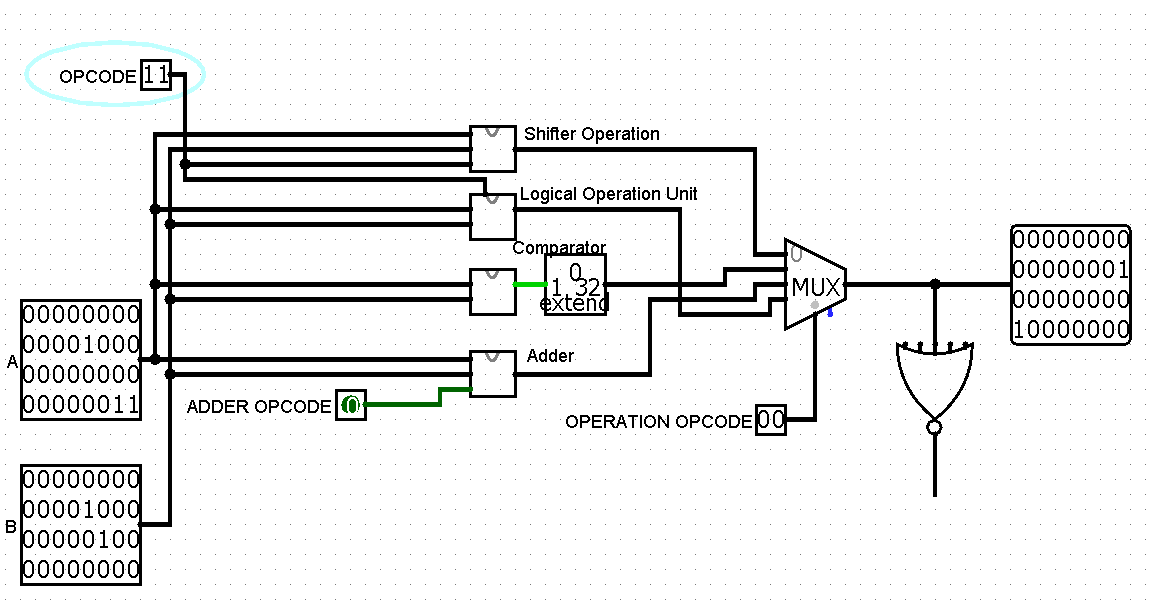
Shift Operation = 11 = SRA:

Input:



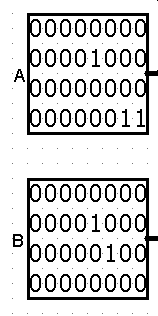
Expect Output: Since the shift opcode is 11, the values of B will be shifted to the right, but instead of 0s, the new inputs will be whatever the most significant bit is. The number of times B will be shifted is dependent on the last 5 bits of A, so B will be shifted to the right 3 times, and 3 0s will appear as the new shifted bits at the top of B.

Output:

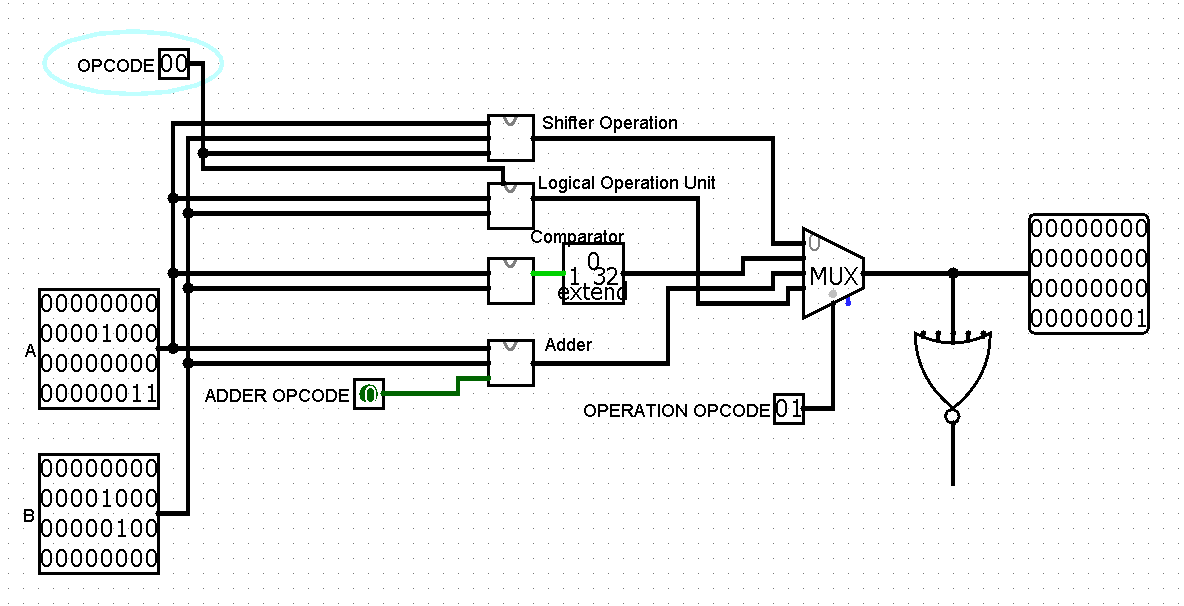
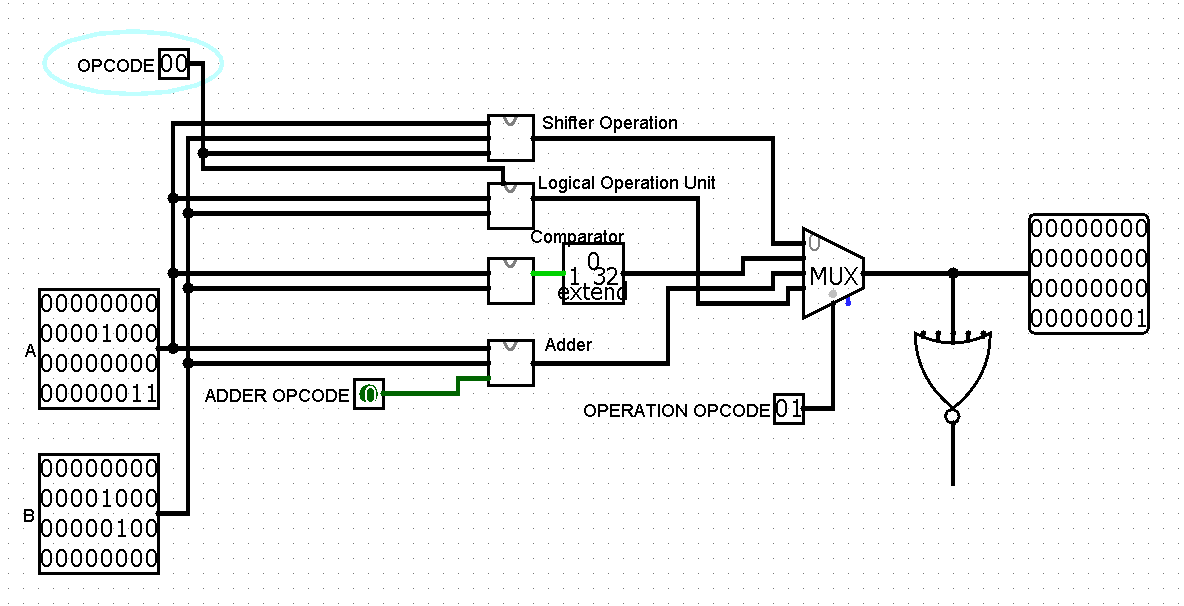
ALU = 01 = SLT

Input:



Expect Output: the output should be a 32 bit number, with the least significant bit existing as a 1 or a 0 depending on whether or not A < B or not. A < B in this case, so output should be a 1

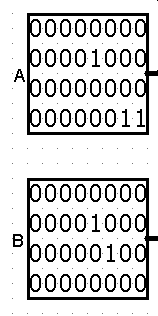
Output:

ALU = 10 = Arith

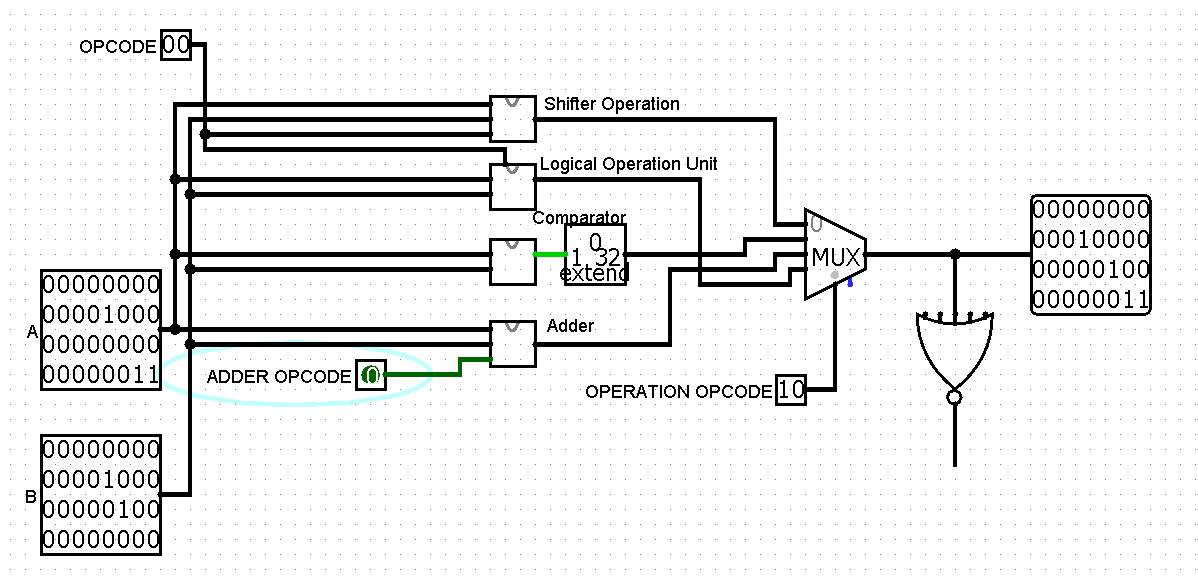
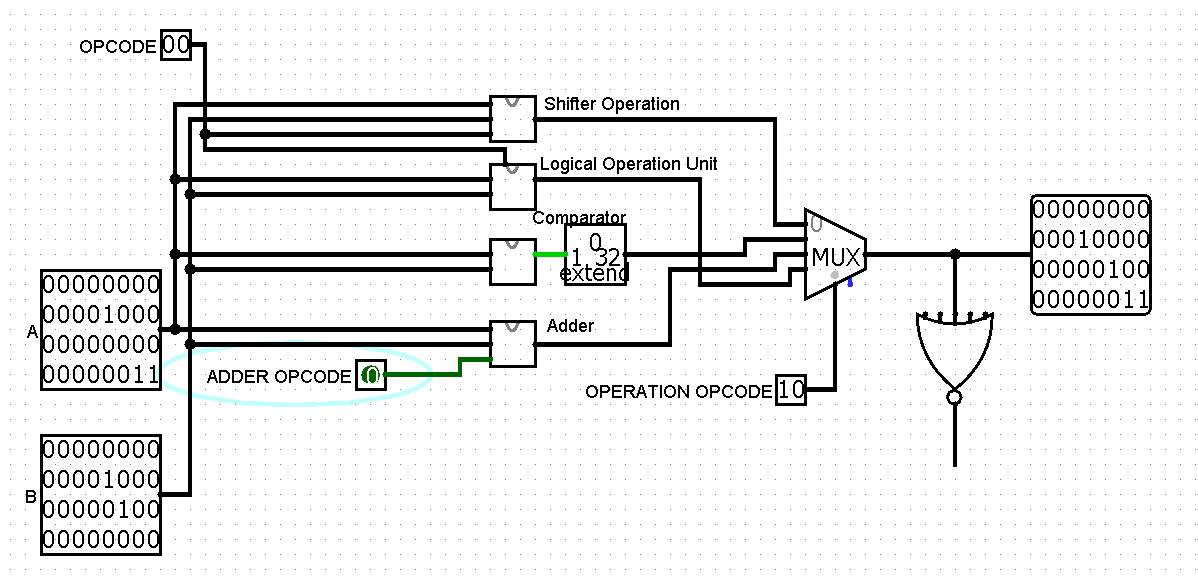
Arithmetic Operation = 0 = ADD

Input:



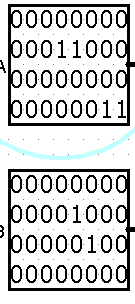
Expect Output: Since the arithmetic opcode is 0, it should add the two bits together.

Output:

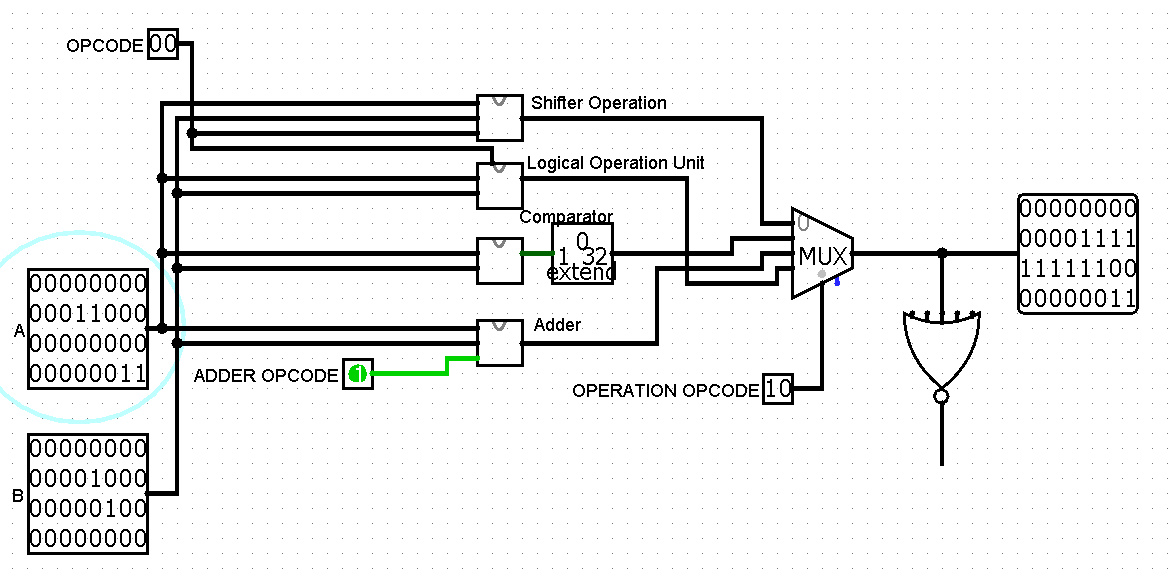
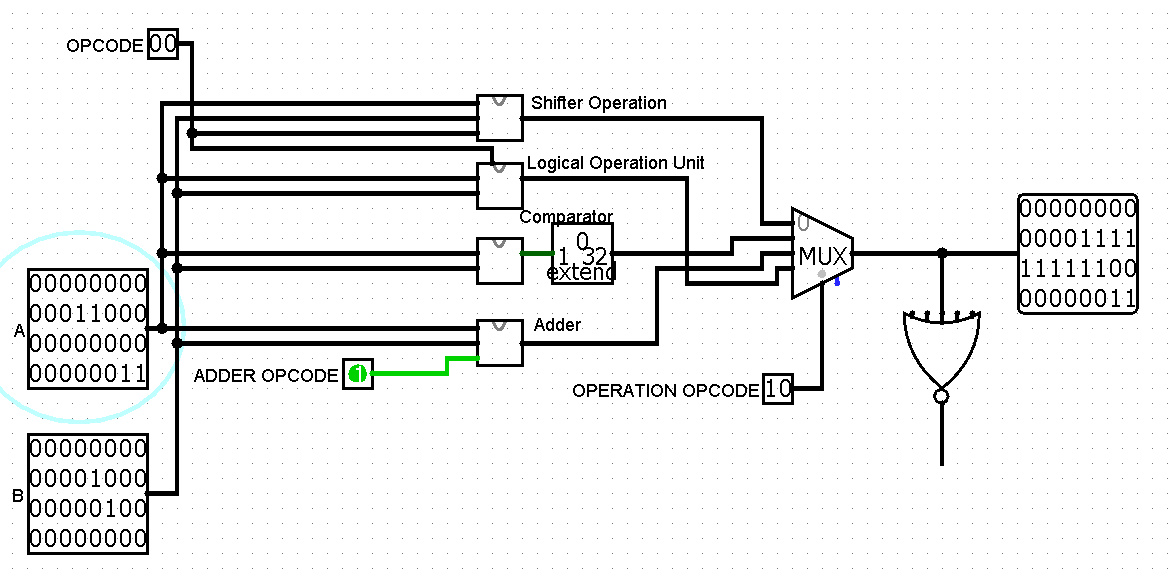
Arithmetic Operation = 1 = SUB

Input:



Expect Output: Since the arithmetic opcode is 1, it should subtract B from A.

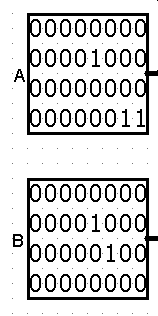
Ouptut:

ALU = 11 = Logic

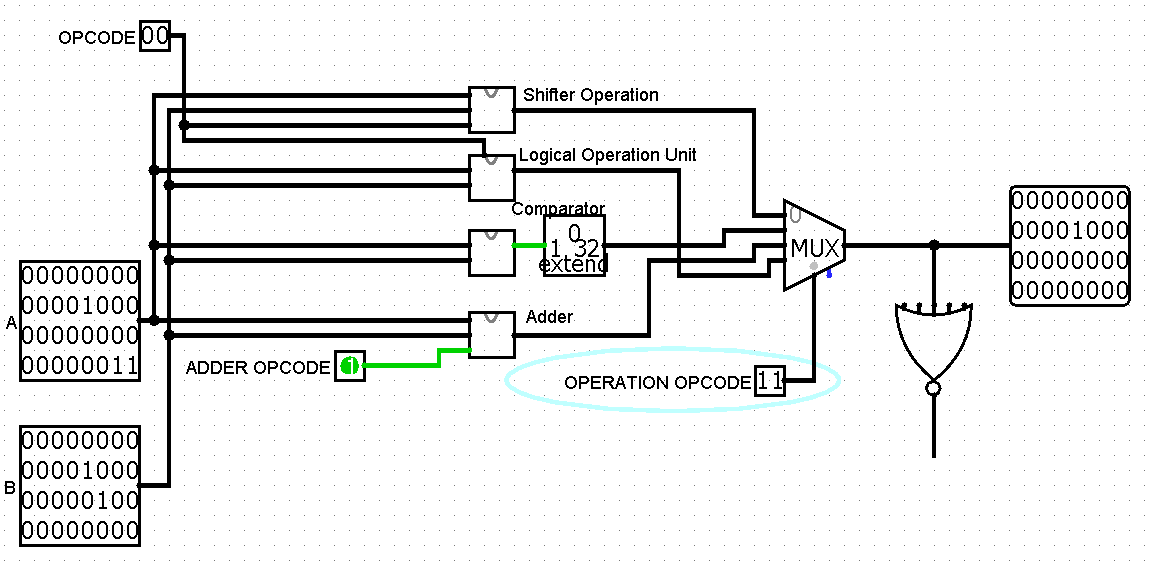
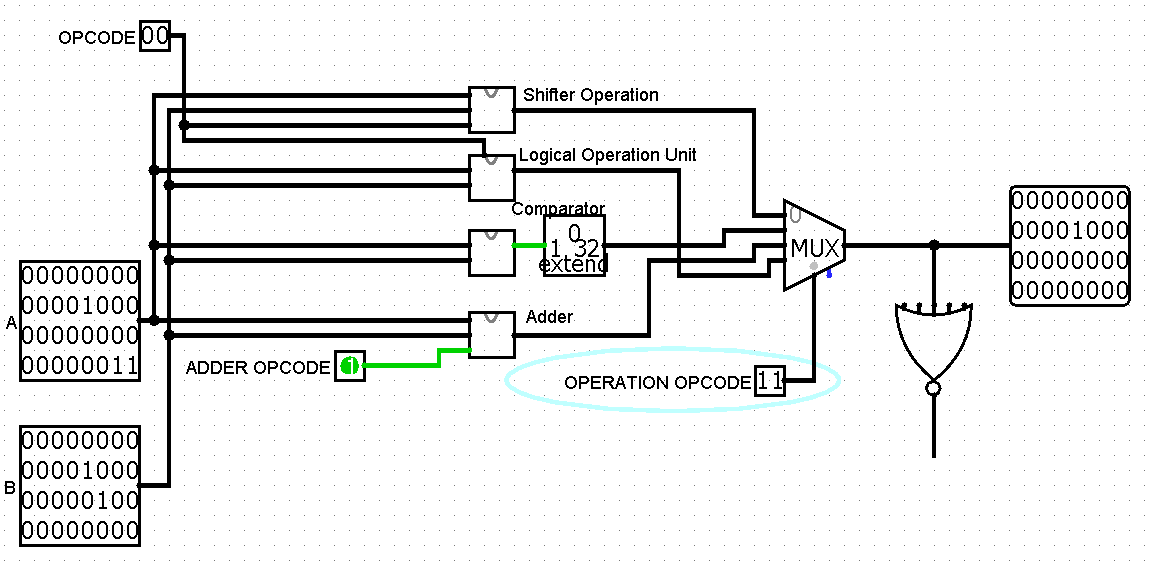
Logical Operation = 00 = AND

Input:



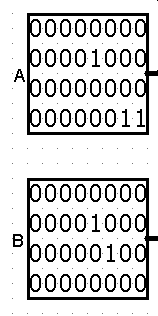
Expect Output: Since the logical opcode is 00, it should only output a 1, where both bits have the value of 1.

Output:

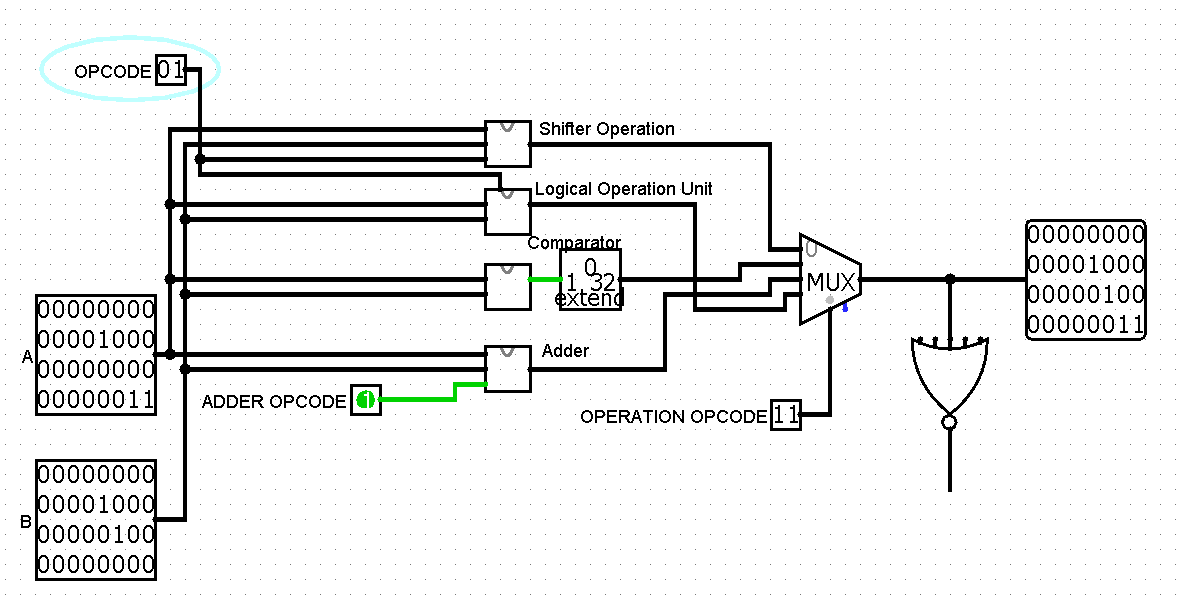
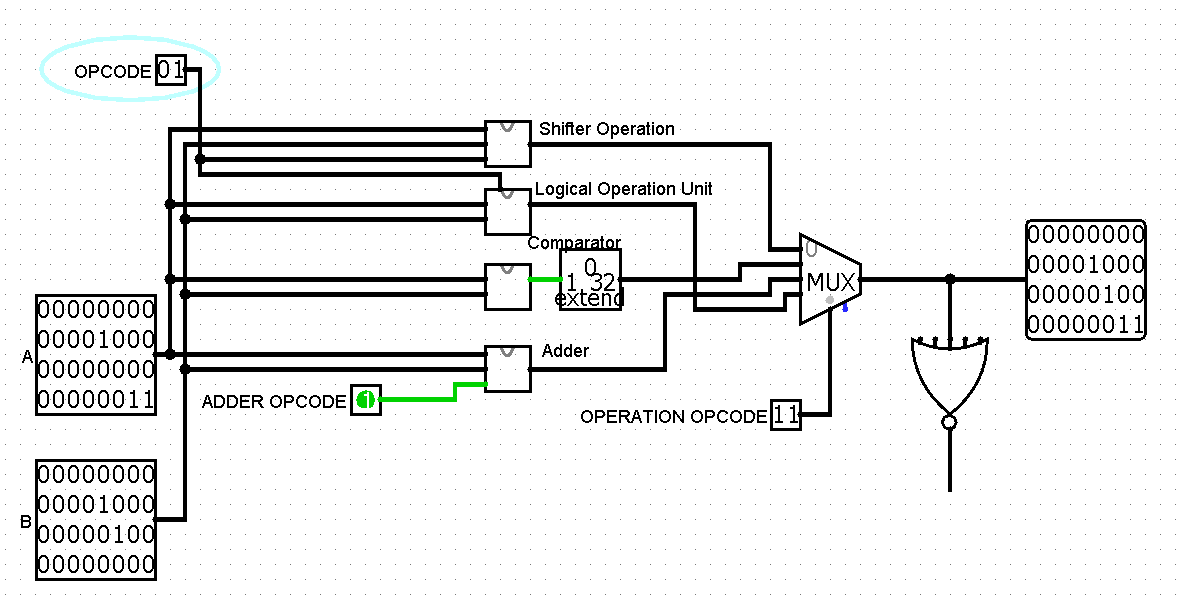
Logical Operation = 01 = OR

Input:



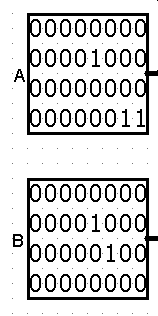
Expect Output: Since the logical opcode is 01, it should output a 1, wherever the location for either input is a 1.

Output:

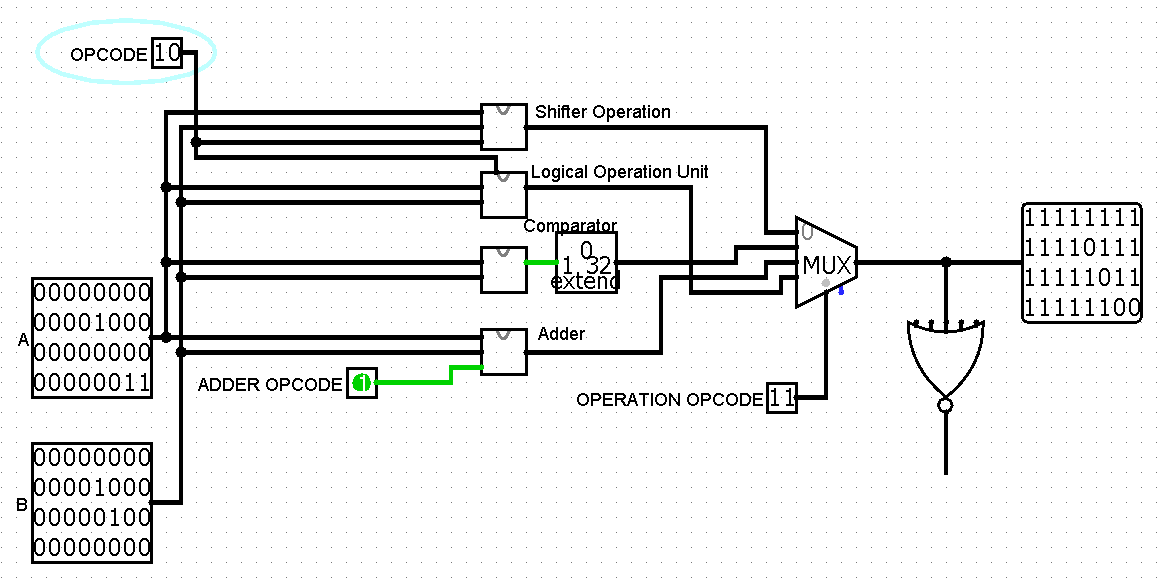
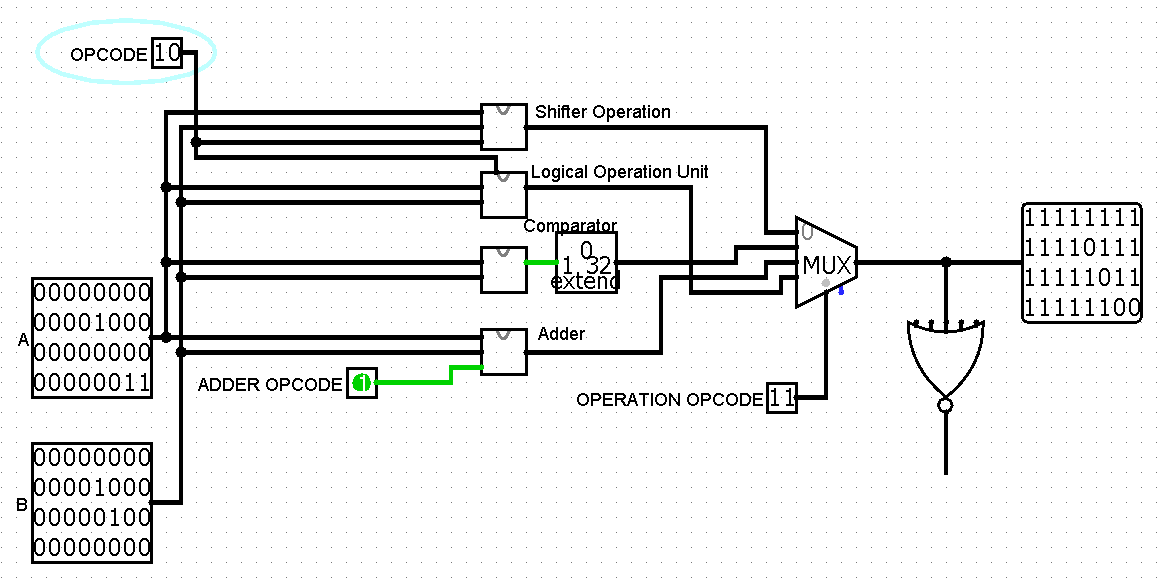
Logical Operation = 10 = NOR

Input:



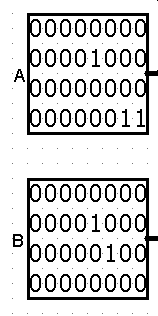
Expect Output: Since the logical opcode is 10, it should output a 1 wherever the values of A and B both do not equal 1.

Output:

Logical Operation = 11 = XOR

Input:



Expect Output: since the logical opcode is 11, the output bits should be a 1 wherever the bits of A and B are 1, but not both 1 at the same location.

Output:

