CS 3410 Project 1 Design Documentation

Rachel Nash

September 11, 2017

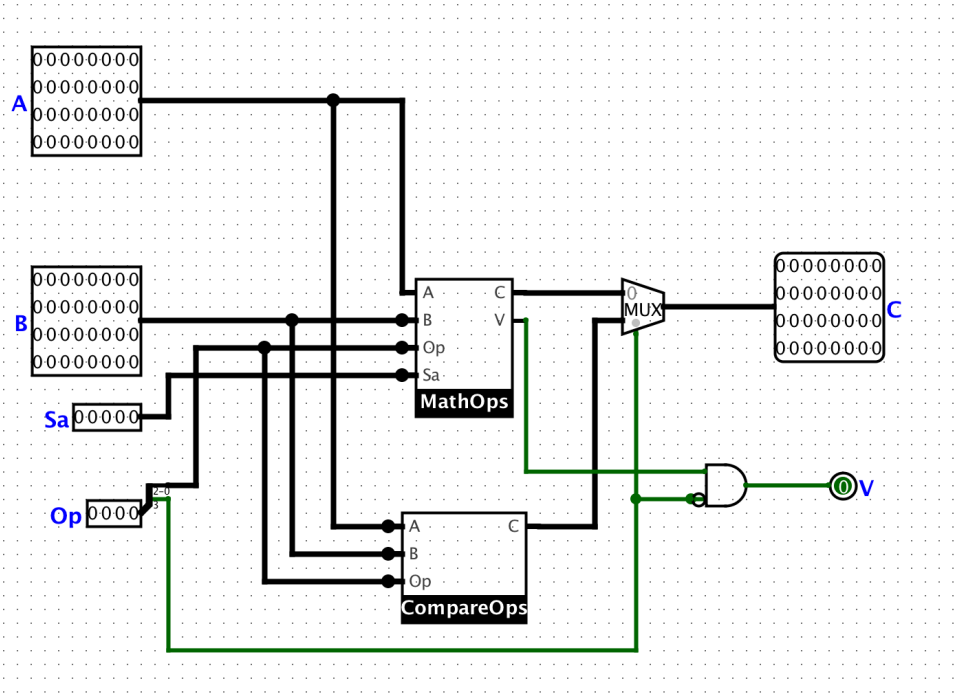
1 Overview

The purpose of this project was to design a MIPS ALU using Logisim software. Factors that went into consideration include gate count, critical path, and readability.

2 Component Design Documentation

2.1 Diagrams and Descriptions

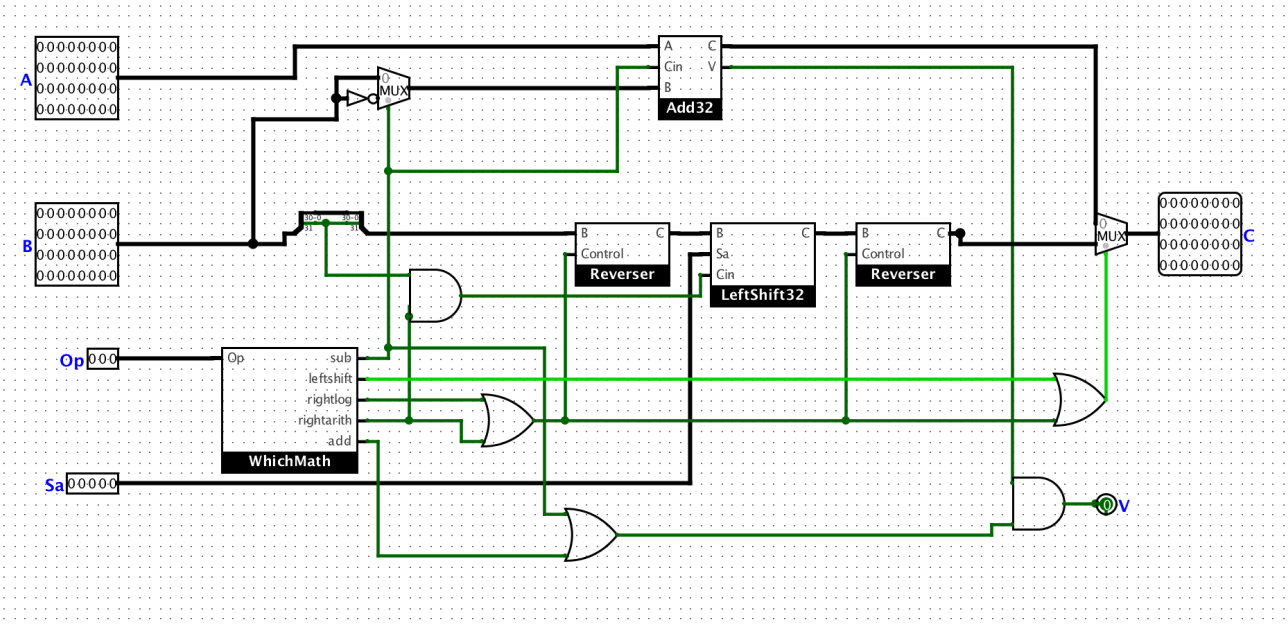
2.1.1 ALU32



To make this complicated circuit easy to read, I split the operations into two main sub-circuits: MathOps, which contains all math functions and all have the first bit 0, and CompareOps, which contains comparative/logic operators and all

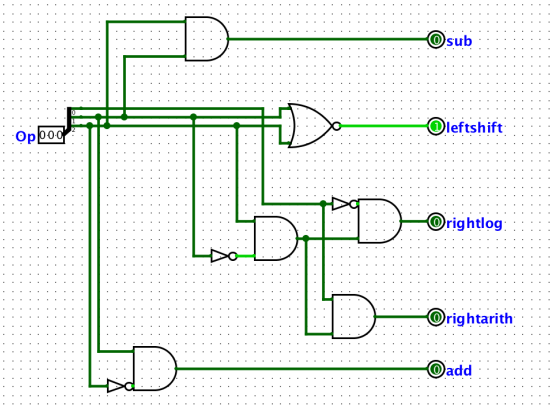
have the first bit 1. MathOps has a V output because only addition and subtraction will possibly have overflow, represented as V. V will be 1 only if MathOps outputs V=1 AND the last bit is 0. MathOps also takes Sa whereas CompareOps does not, but both output C[32], and then the last bit will select, using the mux, which of these outputs is correct for the op code.

2.2.1 MathOps



Since this circuit is only relevant when the first bit is 0, it only uses a 3-bit operator. To provide clarity, a sub-circuit chooses which math operation to perform.

2.2.1.1 WhichMath



This simply selects which of the functions is needed based on the last 3 digits of the opcode. While it may not be the most efficient way to do this, I chose this option to improve readability.

2.2.1.2 Shifts

If any type of shift is output from WhichMath, the final mux chooses the second path, where B has been put through LeftShift32. If leftshift is the output, neither of the rightshifts will be on (checked with OR gate), so the control of the Reverser will stay off and B will not be reversed at any point. When the operation is a rightshift, the control will be on, so B will be reversed before and after entering the LeftShift32. If a rightshift is arithmetic and the last bit of B is a 1 (checked by the AND gate near WhichMath) then Cin for LeftShift32 is a 1, so the new digits made by shifting will be 1.

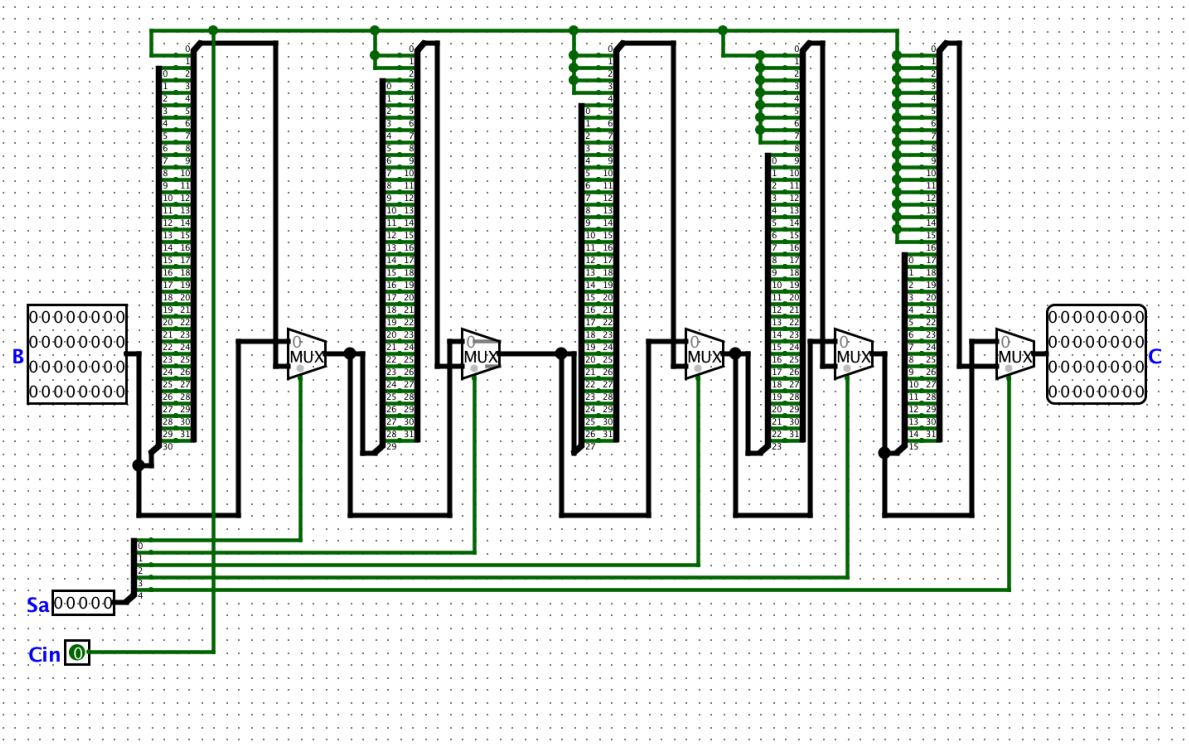
Bit 2	Bit 1	Bit 0	Function
0	0	x	left logical
1	0	0	right logical
1	0	1	right arithmetic

2.2.1.3 Addition and Subtraction

If subtraction is on, then the mux in the top left selects NOT B instead of B, which finds its' one's complement (negated) representation. Then, if subtraction is on, there is an input of 1 for Cin of the Add32, which adds 1 to the final number, creating a two's complement. V is calculated to be 1 only if the V output of Add32 is 1 and either addition or subtraction is on (see AND gate next to V).

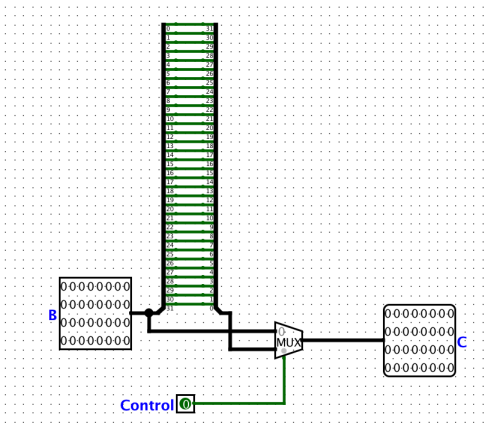
Bit 2	Function
0	addition
1	subtraction

2.2.2 LeftShift32



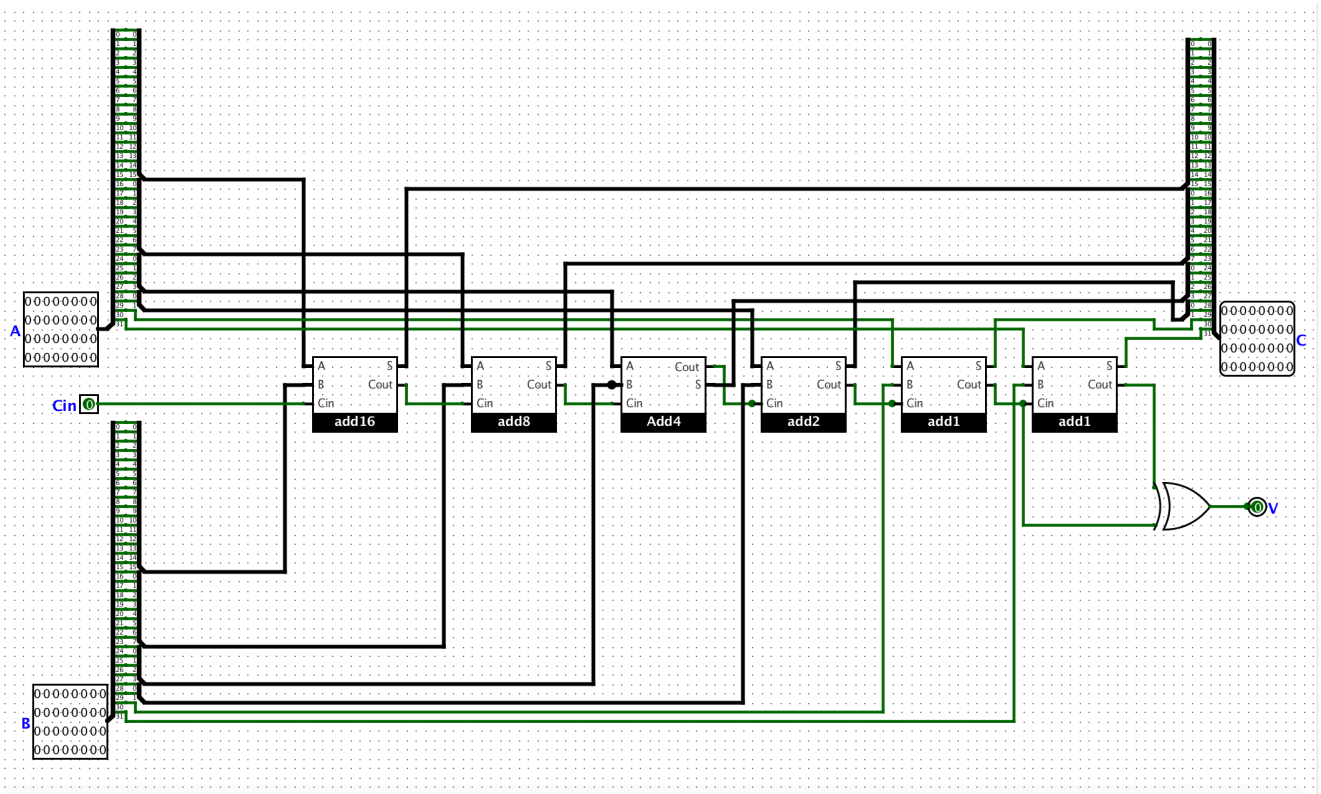
This sub-circuit shifts B[32] left by Sa digits. Each bit of Sa is a control for a mux which selects whether or not B is shifted by that amount. For instance, bit 3 of Sa corresponds to a decimal value of 8 if it is on, so if Bit3=1, then B needs to be shifted to the left 8 bits. You can see that bit 3 is connected to the 4th mux, which chooses between a new version of B that is 8 bits to the left, or whatever B was before this shift. Cin determines which bit is used to pad.

2.2.3 Reverser



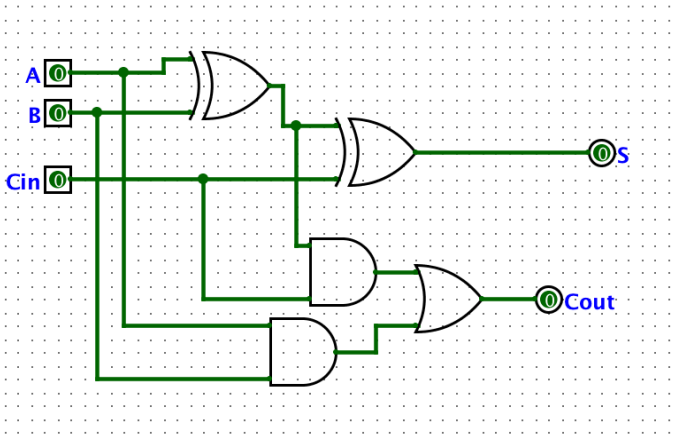
This simple circuit uses 32-bit splitters to reverse B if the control is on. The purpose of this is to be able to do a right shift with the LeftShift32.

2.2.4 Add32



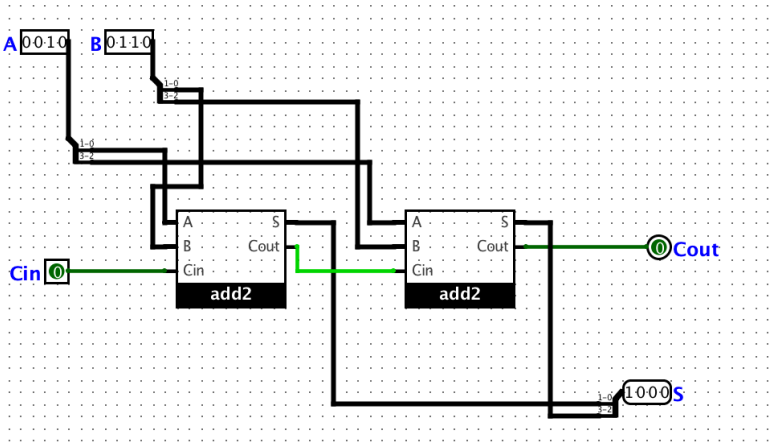
Add32 makes use of smaller, similar sub-circuits to handle addition. It is very similar to the unsigned adder, except we calculate overflow. This is done by putting the last two carry bits into an XOR gate.

2.2.4.1 Add1



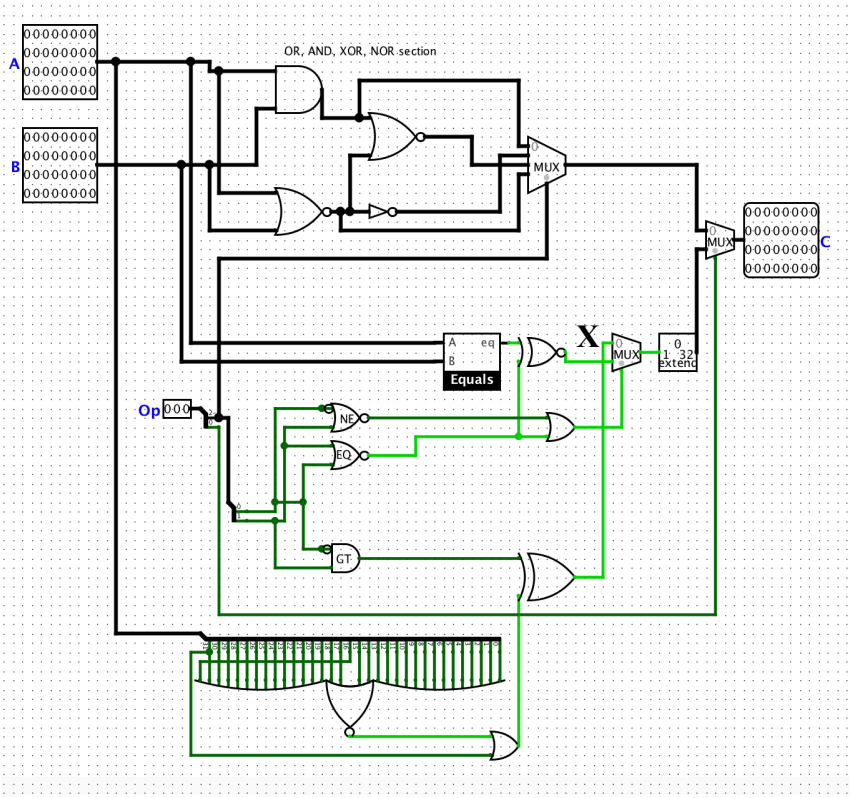
This is the full adder, the optimal design for adding two bits. S is the output and Cout is the carry-out bit.

2.2.4.2 Add4



Add4 combines two add2 circuits to add bits 0-1 then bits 2-3. Add2, add8, and add16 work the same way. They could have all been formed using many add1 circuits, but this is easier to read and wire.

2.3.1 CompareOps



This is the other main sub-circuit of the ALU and is used for comparative/logic operations. All of these operations have the first bit 1, so only a 3-digit Op is needed and, unlike MathOps, no Sa or V is needed.

2.3.1.1 OR, AND, XOR, NOR

These operations are primarily handled near the top of the circuit. Each of these goes into a 4-bit mux.

Mux input for Bit 0 = 0

Bit 1	Bit 2	Function
1	0	OR
1	1	NOR
0	0	AND
0	1	XOR

2.3.1.2 NE, EQ, LE, GT

Now I will discussion what happens when Bit0 = 1. I check if it is not equal, equal, or greater than (less than option is not explicitly used because it will be checked if the others opcode options are off).

Bit 0 = 1

Bit 1	Bit 2	Function
1	0	NE
0	0	EQ
0	1	GT
1	1	LE

I check for equality using the Equals sub-circuit (explained later) which outputs 1 if A and B are equal and 0 otherwise. I will show in the truth table below

how I calculate NE and EQ. I use the label 'X' on my circuit above to show the value at this point.

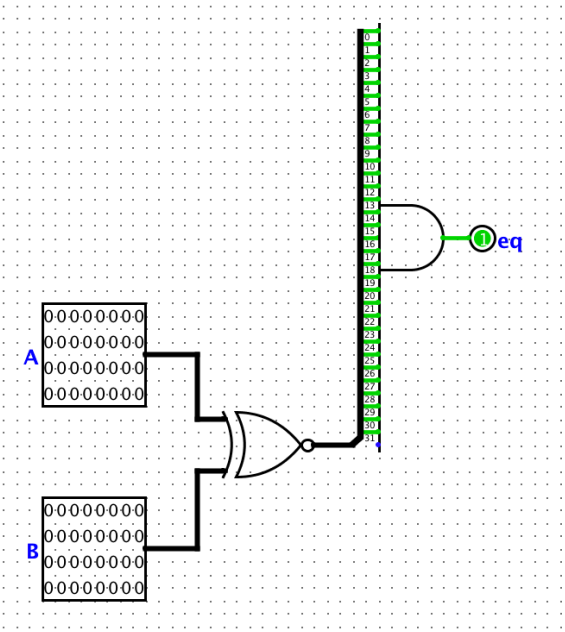
Output of 'Equals'	Is the 'EQ' opcode on?	X
1	1	1
0	1	0
0	0	1
1	0	0

This way, X is 1 if the opcode is EQ and A and B or equal, or if the opcode is NE and A and B are not equal, but not otherwise. After this, a mux will select if one of these was the opcode and should be used as output. Since all the Bit 0 = 1 opcodes have an output of either 1 or 0, the mux is followed by an extend, which pads it with zeros on the left.

I check if A <= 0 using the 32-input NOR gate at the bottom of CompareOps. If the last bit is 1, then A is negative, or if every single bit is 0 (true if putting all bits into NOR is on), then A is zero, so the OR gate at the very bottom is 1 if A<=0 and 0 otherwise, which means A>0. I then use an XOR gate to see if LE or GT is satisfied (keep in mind that if GT is off, we assume for now that LE is on, and later check with a mux):

GT is on	A<=0	output of XOR
1	0	1
0	1	1
1	1	0
0	0	0

2.3.2 Equals



This simple sub-circuit of CompareOps outputs 1 if A and B are equal and 0 otherwise. The XNOR creates a 32-bit number where a 1 means A and B had the same bit, then AND checks to make sure they are all 1.

3. References

All work was completed by myself, Rachel Nash.