CS 3410 Project 1 Design Document

Zaibo Wang

September 12, 2017

1 Overview

This project is a 32-bit ALU which has the following capabilities: add, subtract, right shift arithmetic, left/right shift logical, equal, not equal, and, or, nor, xor, less than or equal to, greater than. The ALU is one component of a MIPS processor which is a latter project.

2 Adder

A one-bit full adder was designed to support the addition of two 1-bit inputs with a carry bit. This circuit is necessary for the construction of more advanced full adders.

2.1 Implementation Details

In the above diagram, A and B are the 1-bit inputs, while C_{in} is the carry-in bit. S is the output bit, while C_{out} is the carry-out bit. This is the optimal circuit for the following truth table:

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

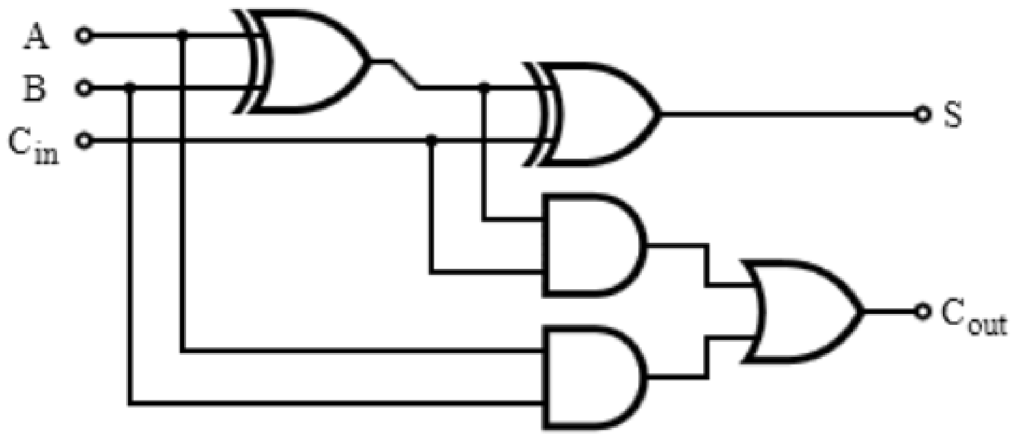

Figure 1 and 2: All other higher order adders were made using abstractions of the 1-bit adder. Because the 32-bit adder requires an output V which indicators overflow, an alternate 2-bit adder called adder2v was made which outputs V instead of Cout. This pattern is repeated for all the adders up to 16-bit (for sake of brevity, those will not be included.)

Figure 3: For decoding logic, the control comes from bit 2 of the Op input. If Op is 1, A-B is evaluated. If Op is 0, A+B is evaluated. For subtraction, since B is a two's compliment number, the negative negative is found by taking the inverse and adding one. In circuit form, this is done with a NOT gate on B, and 1 in Cin. The output for V uses an AND gate with Op bit 1 and NOT Op bit 3 because these two conditions must be satisfied for addition/subtraction to be the designated operation. For any other operation, V should be 0.

Add/Subtract Logic

Op bit 2	*Cin*	*B*
0	0	B
1	1	Not B

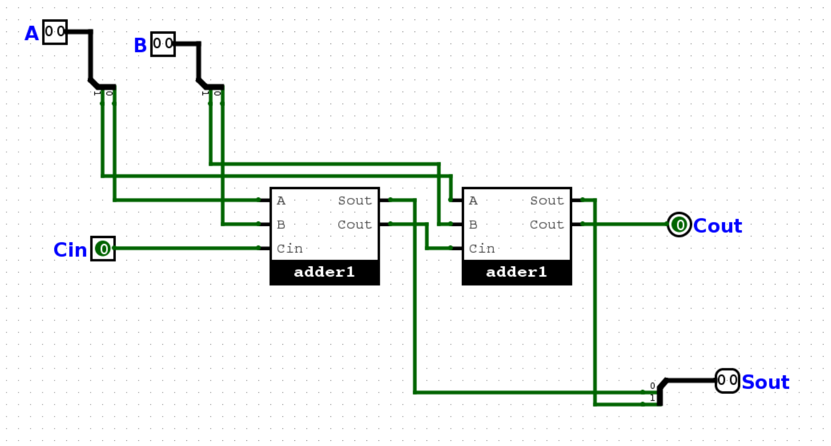

Figure 1: 2-bit Adder

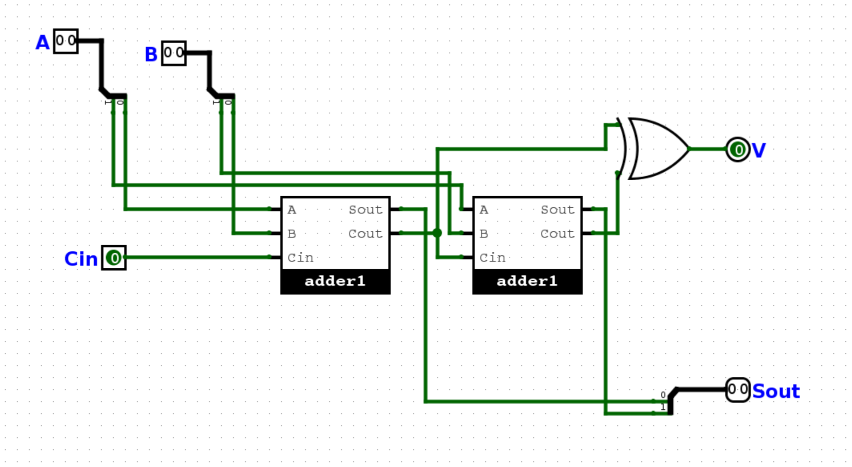

Figure 2: 2-bit Adder V

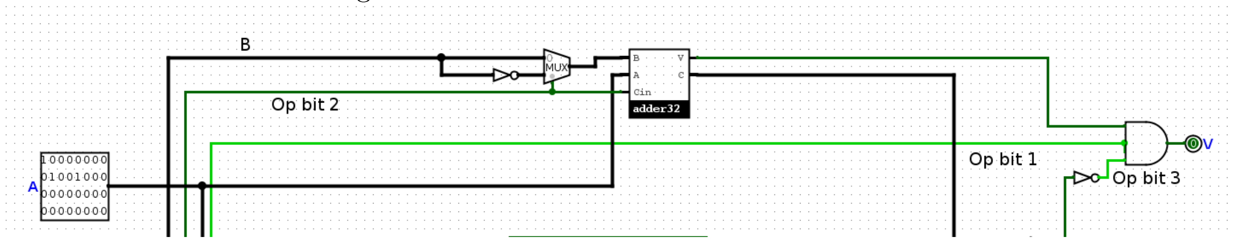

Figure 3: Circuit for adding and subtracting

V Logic

Op_bit_1	*Op_Bit_3*	*V from Adder*	*V*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

2.2 Evaluation

The circuit implements the optimal logical expression for both the resulting bit and the carry-out bit. One flaw in this design is each subcircuit for the adder required two different types of circuits. While this does not affect performance, a more modular design would be better for circuitry in general. The TA told me after the circuit was created that there was an easier way to get V rather than use alternate circuits. This circuit also uses a ripple carry adder which as described in the lab write-up, is too slow for a real processor.

3 Shifter

3.1 Implementation Details

Figure 4: The bit reverser maps 32 bits in reverse order. This is used in right shifting: reversing bits, doing a left shift, and reversing bits again is equivalent to a right shift. This allows right and left shifting with just a left shifter. While a simple operation, making it a subcircuit reduces space

Figure 5: The left shifter takes in a 32-bit value B, Cin and Sa, and outputs C = (B << Sa) — carrybits. This implementation splits Sa into 5 1-bit components. Each of these 1-bit wires signal a shift of 16, 8, 4, 2 or 1 bits. Combinations of these shifts allows B to shift up to 31 bits.

Figure 6: The logic for controlling the type of shift is decided through two muxes conviently labeled Mux 1 and Mux 2. Mux 1 Takes input B and the reverse of B. It is controlled by Op bit 2 because this bit controls whether the operation is a left or right shift. If it is a left shift, B is inputted without change. If the operation is a right shift, the reversed B is inputted. There is a corresponding Mux after the shifter. This will re-reverse B in case right shift was operated. Mux 2 Takes Op bit 1 and the MSB of B. Op bit 1 was

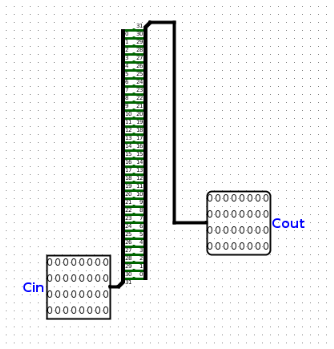

Figure 4: Bit reverser

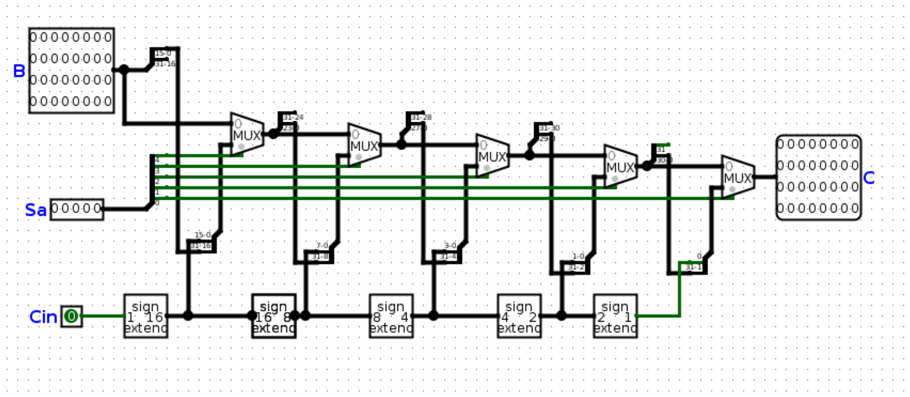

Figure 5: Left Shifter

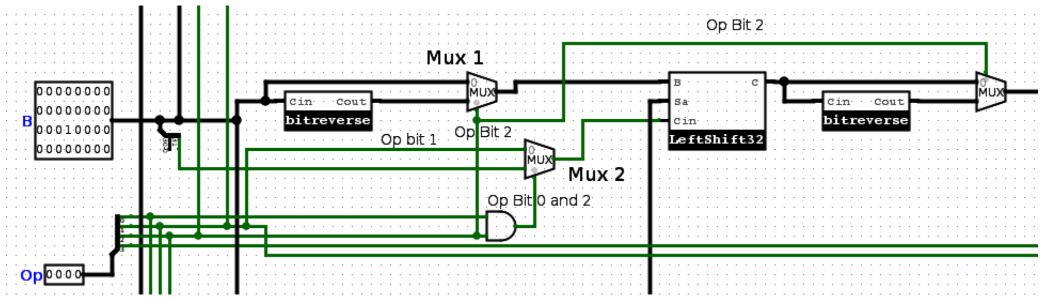

Figure 6: Complete shifter

chosen because for any shifting operation, Op bit 1 must always be 0. This removes the need of a constant. The control signal is Op bit 0 AND Op bit 2. The reason is if these are both true, a right shift arithmetic is signaled. In such a shift, the newly added values must be the same sign as the MSB. Thus, the MSB is sent to Cin if Op bit 0 and 2 are true. Otherwise, 0 is sent to Cin.

	Op_bit_2	*InputtoB*
Mux 1	0	B
	0	B Reversed

	Op_bit_0	*Op_Bit_2*	*InputtoCin*
	0	0	0
Mux 2	0	1	0
	1	0	0
	1	1	1

3.2 Evaluation

One design choice was to use two reversals for right shifting. This is more efficient than having both a right and a left shifter. The left shifter could be used for 3 different operations. I found that for Cin, the logical shifts required a particular Cin while the arithmetic shift required a different Cin. I decided to use a mux for this. A similar case applied to the B: the left shift required no change to be while the two right shifts required a reversal. I believe using two muxes in such a way is an efficient way to choose the inputs to the left shifter. This solution also only uses a single left shifter.

4 Logical

4.1 Implementation Details

There are four logical operators that the ALU needs to process: AND, OR, XOR, NOR. There are only three gates because NOR can be evaluated using the NOT of OR. The control for these operations are signaled by Op bits 1 and 2. The two Op bits are split into one wire and used as the input signal for a 2x4 mux.

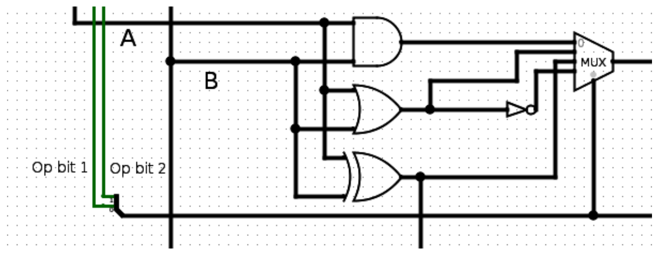

Figure 7: Logical Operators

Logical Operators

Op_bit_1	*Op_Bit_2*	*Output*
0	0	A AND B
0	1	A OR B
1	0	A EXOR B
1	1	A NOR B

4.2 Evaluation

The biggest design choice was splitting the wire after OR instead of using a separate NOR gate. While another NOR gate might make the circuit more clear, using a NOT gate is more efficient.

5 Value Comparisons

5.1 Design Implementation

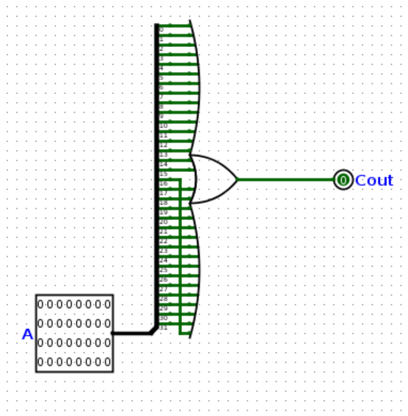

Figure 8: Or on all bits

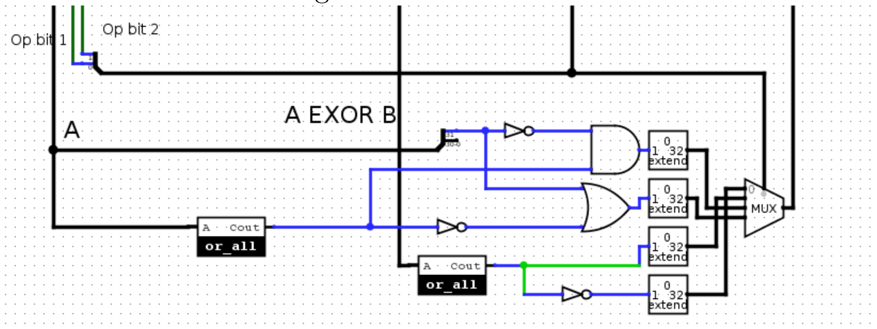

Figure 9: Value Comparisons

Figure 8: This subcircuit was made because it takes up too much space (similar to the bit reverser.) It splits all bits and applies OR to all the bits. This is used to detect if the input is equal to 0 (more specifically that all the bits have value 0.) This operation is necessary for the 0 case for the gt/le operations.

Figure 9: Similar to the design for logical circuits, the control comes from Op bits 1 and 2 which determine which value comparison is sent from the MUX. Instead of using A and B as inputs, the circuit uses A and A EXOR B from a previous circuit.

Value Comparisons

Op_bit_1	*Op_Bit_2*	*Output*
0	0	A == B
0	1	A != B
1	0	A > 0
1	1	A $\leq$ 0

To evaluate gt/le, the first bit is identified since the values are in two's complement. A leading 1 in two's complement means negative while a leading 0 means positive. The exception to this rule is the value 0. This is why the or_all circuit must be used.

Compare to 0

MSB of A	or_all of A	$A > 0$	$A \leq 0$
0	0	0	1
0	1	1	0
1	0	0	1
1	1	0	1

To evaluate equality of A and B, the circuit takes input A EXOR B and applies the or_all circuit. If all bits in A and B are equal, then EXOR will output all 0 and or_all will evaluate to 0. If there are any bits that are different between A and B, EXOR will have some bits output 1 and or_all will evaluate to 1.

Equality

or_all of A EXOR B	A == B	A != B
0	1	0
1	0	1

5.2 Evaluation

By using A EXOR B from a previous circuit, it saves the use of an additional EXOR gate. As previously explained, the or_all subcircuit was made to check the edge case of 0 for gt/le. It was also used in equality because if two values are equal, their EXOR would evaluate to all 0s. or_all is the best way I could think of to these for all 0s.

6 Output C

6.1 Implementation Details

At the end of the circuit, a series of muxes are set up to ultimately determine which operation is used. They are labeled A, B, C for clarity

Mux A: This mux takes input Op bit 0 because this bit signals the difference between a logical comparison and a value comparison. 0 signals logical and 1 signals value.

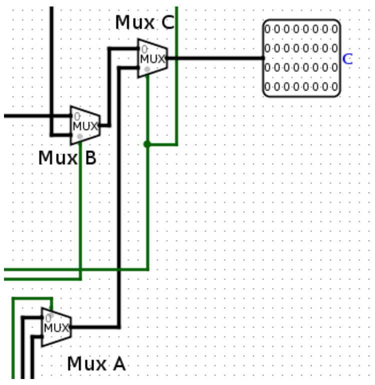

Figure 10: Output Muxes

Mux B: This mux takes Op bit 1 to because this bit signals the difference between using the adder or the shifter. 0 signals shifter and 1 signals adder.

Mux C: This mux takes input Op bit 3 to determine if the output C should come from Mux A output or Mux B output. 1 signals Mux A and 0 signals Mux B. I will not include truth tables because I believe it does not enhance clarity.

6.2 Evaluation

There are 4 total input wires of which 1 is chosen. A 2x4 mux seems logical. However, I chose to use 3 1x2 muxes. The reasoning behind this decision is that there are actually 3 different input bits that are used to evaluate which input is chosen. From above, Op bits 0, 1, 3 are used to evaluate what C is. Thus it wouldn't be possible to use a 2x4 mux for such an operation and 3 1x2 muxes is better.