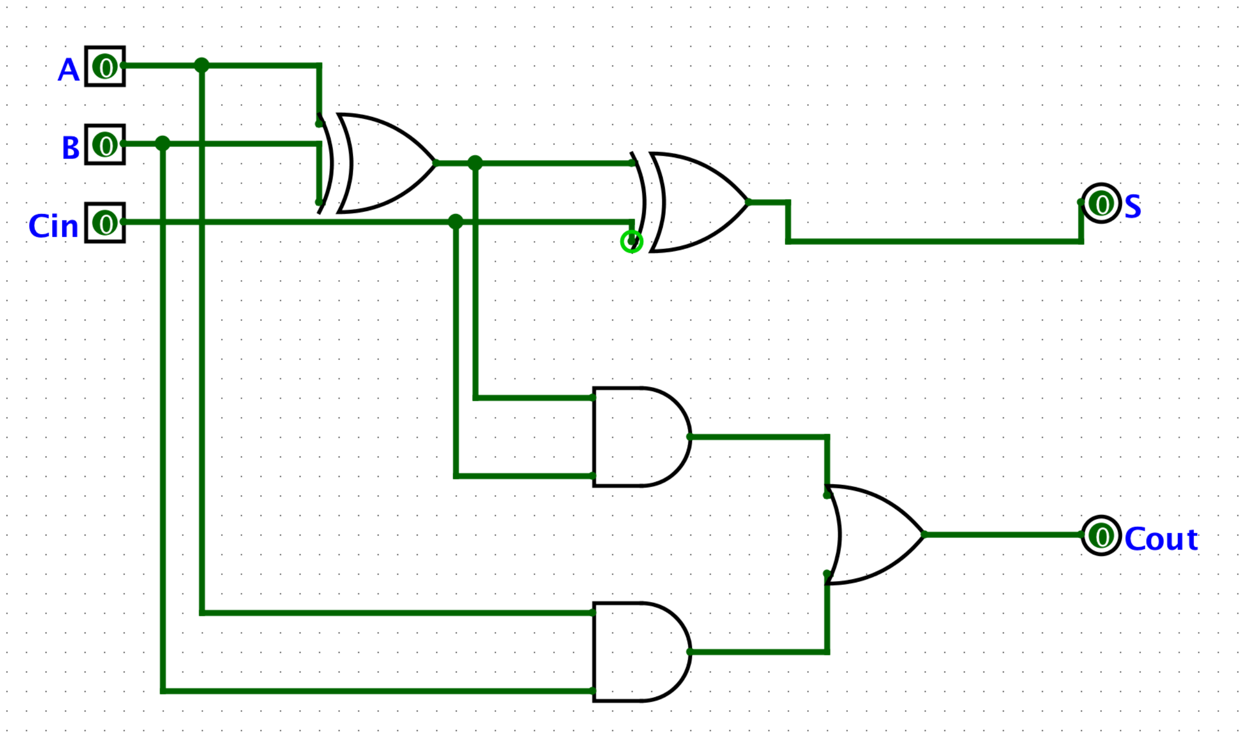
CS 3410 ALU Design Documentation
Ellaine Chou
ec695

1-bit full adder

Purpose: Add two 1-bit inputs with a carry bit. Used for the two-bit adder, and ultimately, the 32-bit adder. A and B are the 1-bit inputs, Cin is the carry-in bit, S is the 1-bit output, Cout is the carry-out bit.

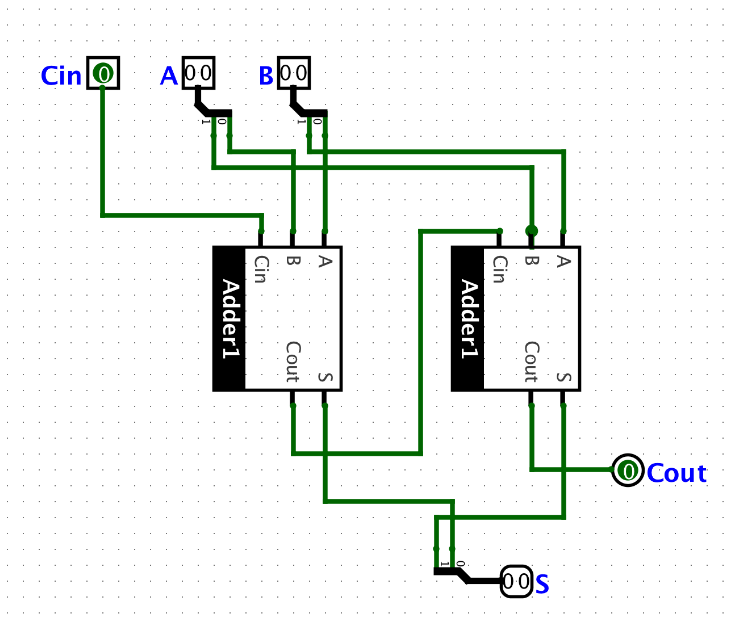
Truth Table:

A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
1	0	0	0	1
0	1	0	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1



2,4,8,16-bit Full Adders

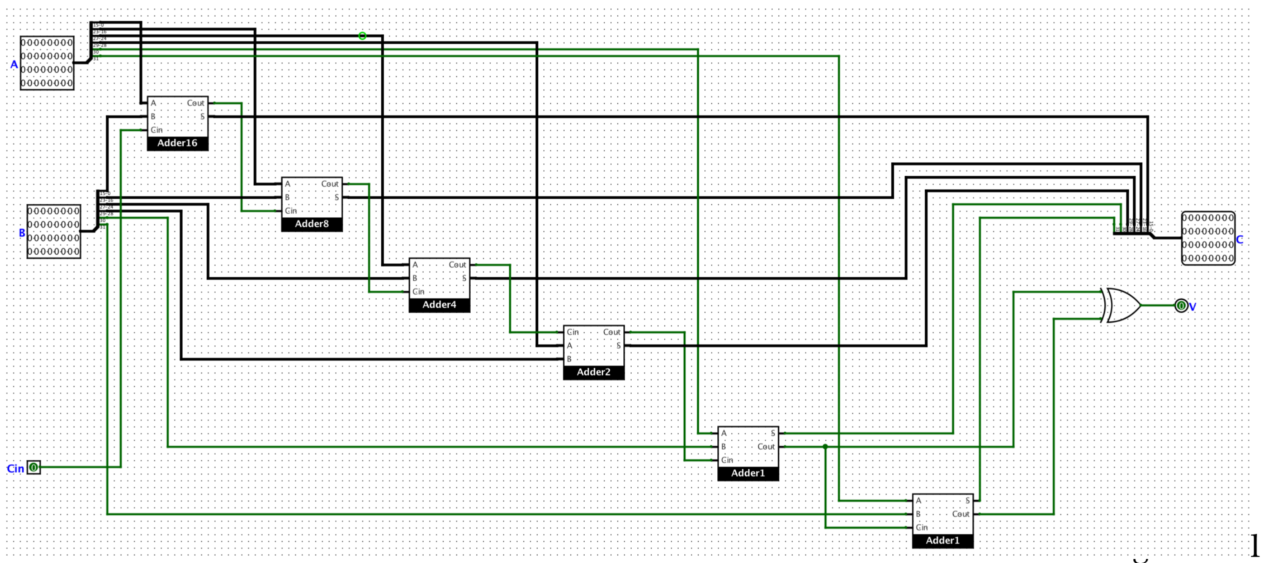
Purpose: Add two 2,4,8, or 16 bits inputs. The logic and schematics for all adders are the same. Two x—bit inputs, a one-bit carry-in Cin, a x-bit S output, and a one-bit carry-out Cout. A 2-bit adder is comprised of two 1-bit adders, a 4-bit adder is comprised of two 2-bit adders, and so on.



2-bit adder

32-bit Adder

Purpose: Add two 32-bit 2’s complement inputs. V is asserted if 2’s complements addition (or subtraction) causes overflow. Comprised of combinations of n-bit adders to add up to 32 and to have two 1-bit adders to be able to calculate the overflow. Overflow is calculated by XOR’ing the carryout for the msb and second msb bit addition/ subtraction.

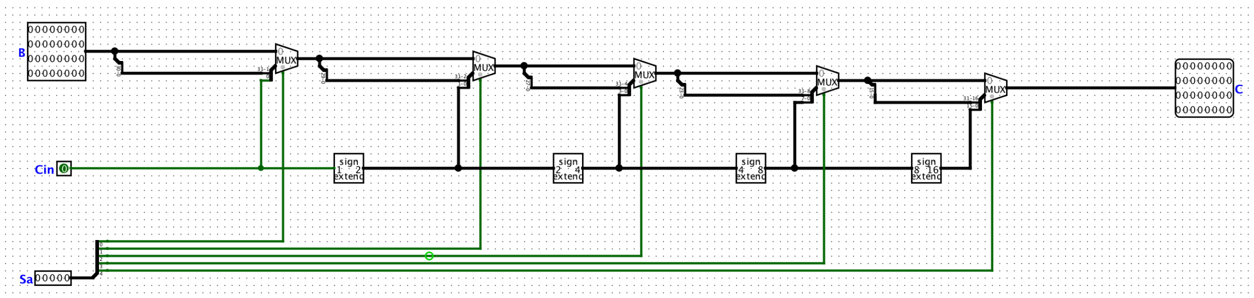


Truth Table for the XOR

A	B	Cin
0	0	0
0	1	1
1	0	0
1	1	0
1	0	1
0	1	1
1	1	1

<u>**LeftShift-32**</u>

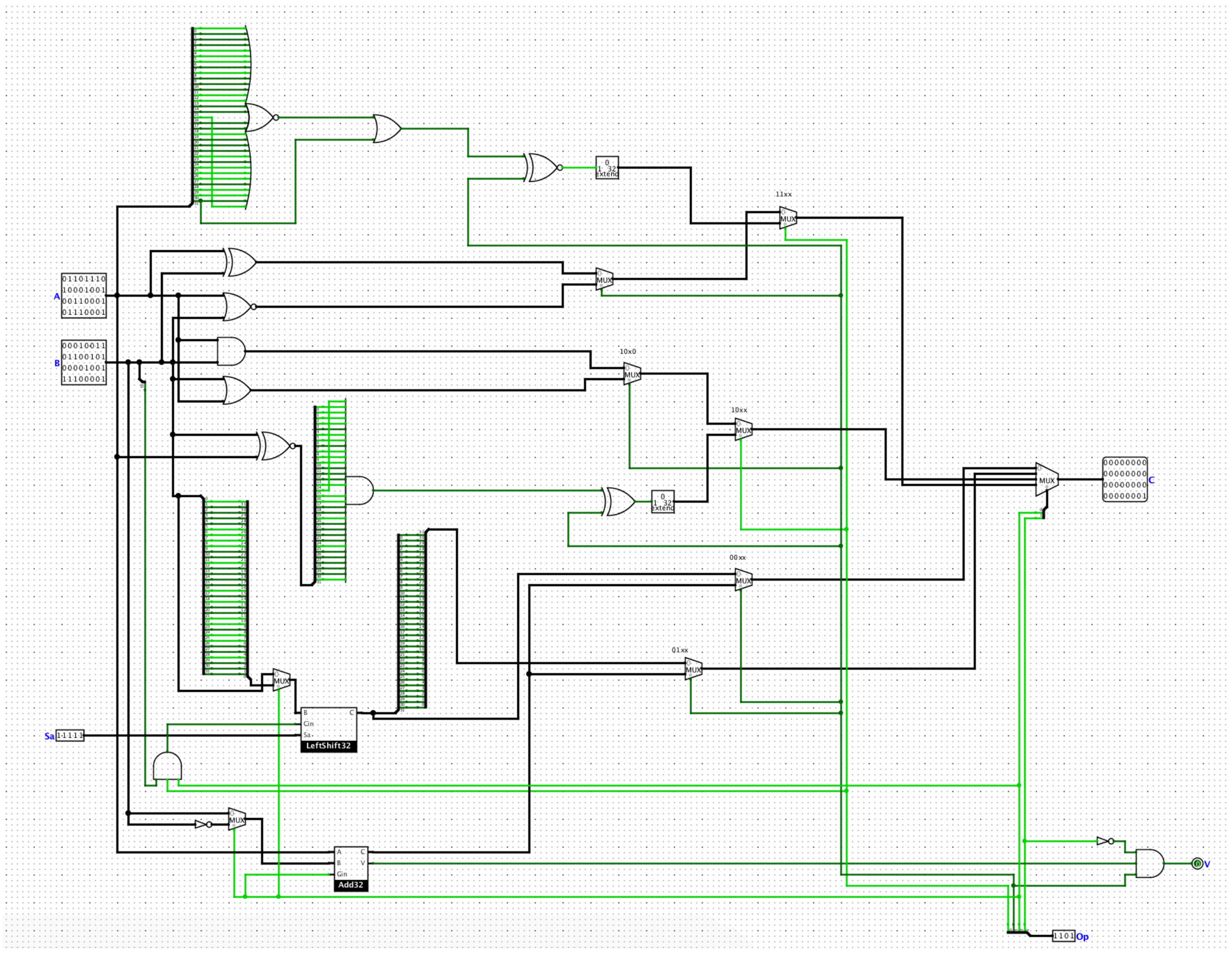
Purpose: Shift a 32-bit input B[32] left by Sa[5] amount of bits, with Cin[1] fillers.



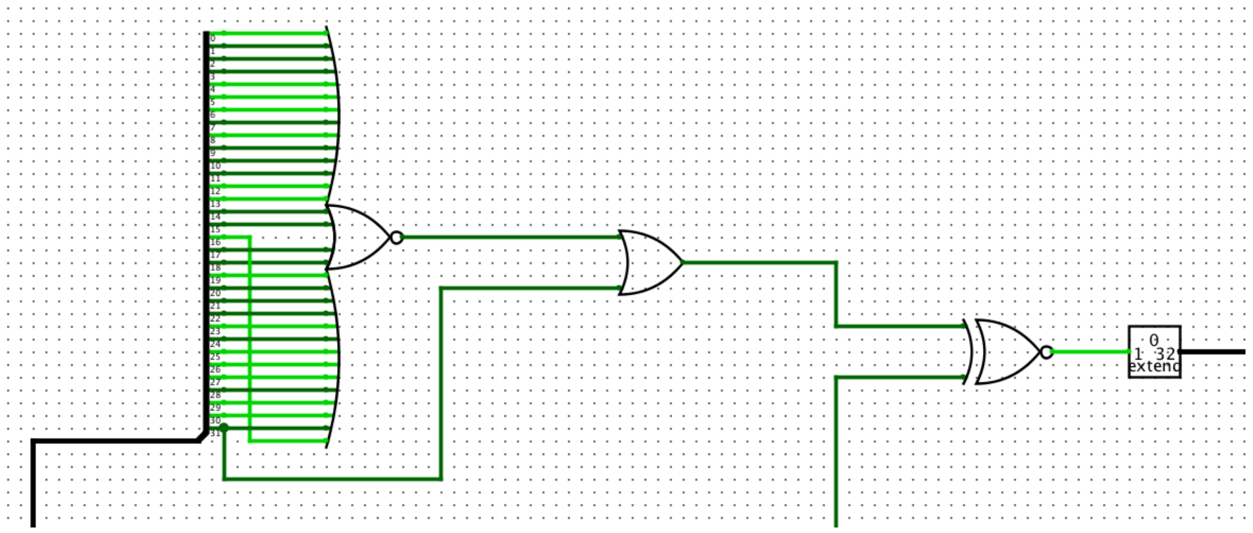
The leftmost mux uses the lsb of Sa[5] to determine whether or not to shift the input by 2^0 bits. The second-left mux, if a 1, shifts the input by 2^1 and so forth. Wire splitters and rearranging of the input bits allow for the shifts to occur. For example, if the leftmost mux is a 1, then a 2^0 bit shift occurs, and the lsb of the input is now replaced by the Cin filler, while the 1st - 31st bit are replaced with the input's 0-30 bits. The bit extender is needed if there is more than a one-bit shift, hence more than 1 bit of Cin needed to fill in the output.

ALU

Purpose: To perform basic logic functions, logic left-shift and right-shift, arithmetic right-shift, and addition/subtraction. A[32] and B[32] are the two inputs to apply the function on, Sa[5] determines the number of bits to be shifted if a shift function is chosen, Op[4] determines the function to perform, C[32] is the result of the function, V[1] determines whether overflow has occurred if addition/subtraction was the function (1 if overflow occurred, 0 if no overflow).



Functions: gt and le

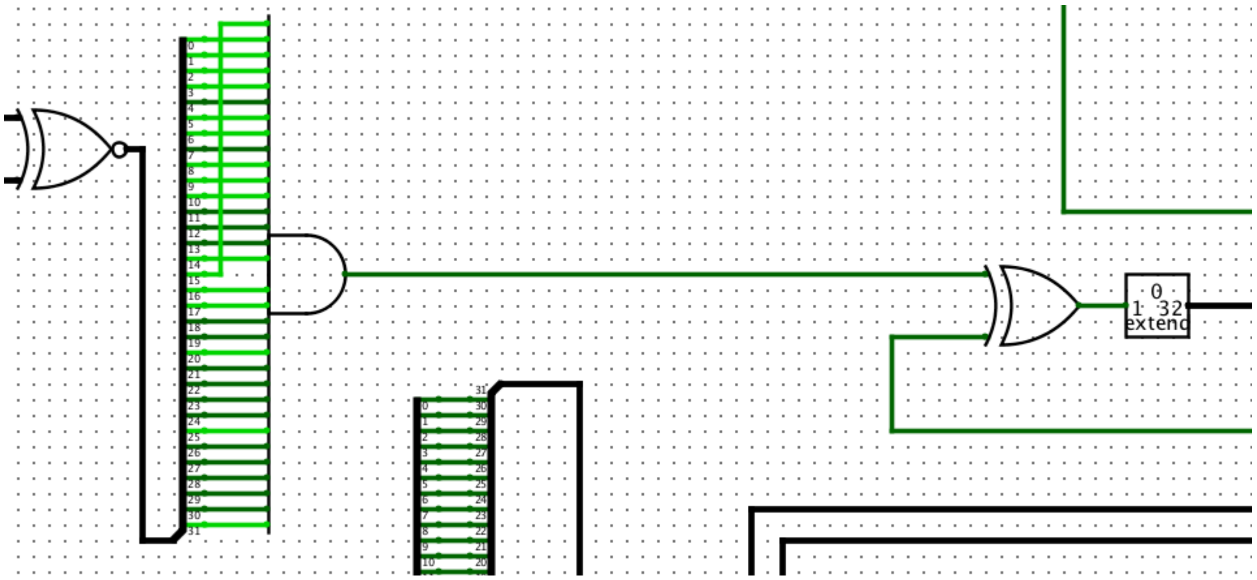


To determine if B[32] is less than or equal to zero, a NOR gate to the left checks to see if all the bits of B[32] are zero. An OR gate takes in the value from the NOR gate and the value of the msb of B[32] to determine if B is <= 0. If all bits are zero (NOR gate value is one), then B is equal to 0. If msb of B is 1, then B is < 0. Both scenario are mutually exclusive, thus inputs of 1 and 1 through the OR gate is never possible.

The XNOR determines whether or not the opcode function is gt or le since the output for each function is the inverse of one another. The second input of the XNOR is 1st bit of the opcode, since the 1st bit is the differing element between the opcodes for gt and le.

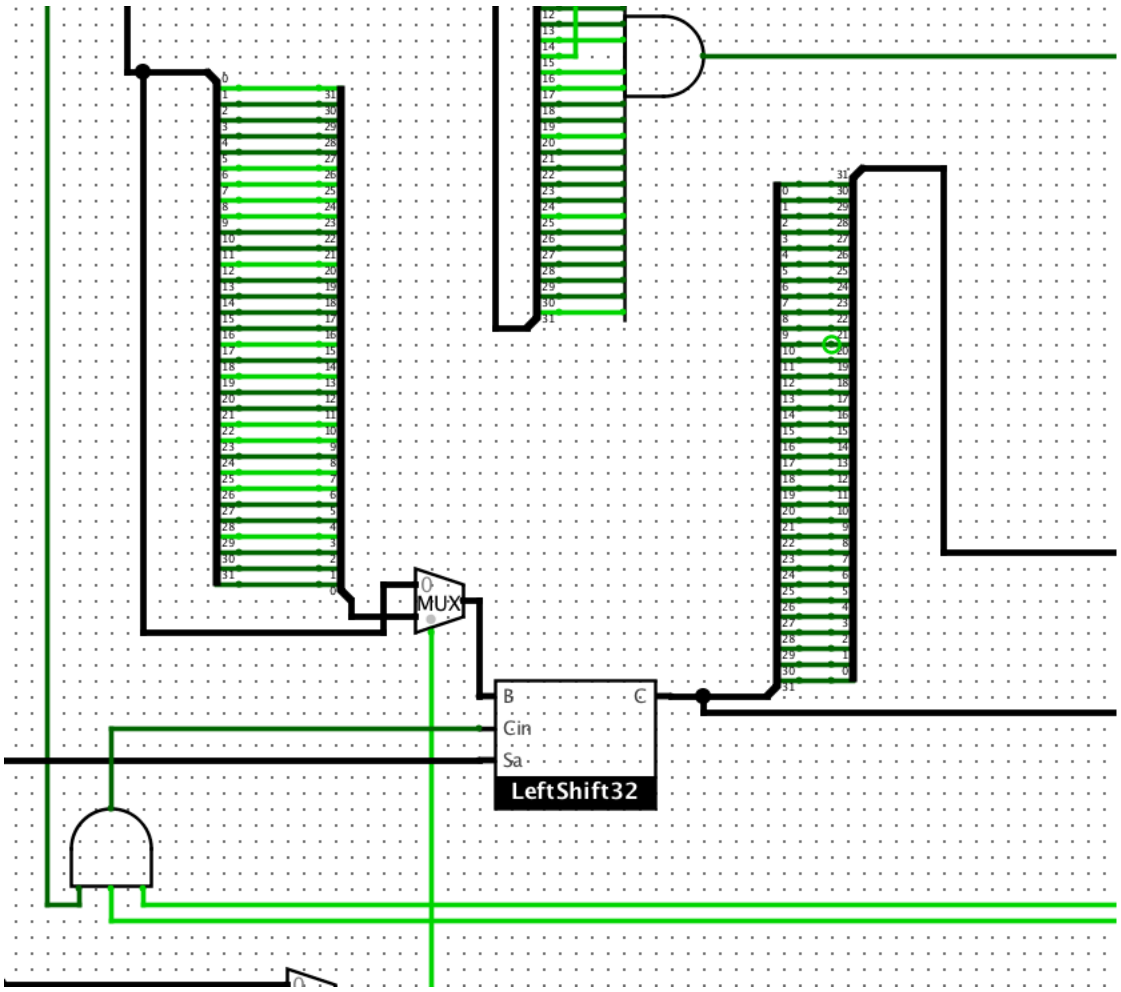
Output of OR	B[1]	Output
0	0	1
1	0	0
1	1	1
0	1	0

Function: eq and ne



A 32-input XNOR is used to test the equality of A and B. An AND gate is used to check if all 32-bits of the XNOR output is 1 — if they are, then A and B are bit-wised equal. The rightmost XOR gate determines whether or not the opcode function is asking for eq or ne, since the output for each function is the inverse of the other. The two inputs for the XOR are the value of the AND gate, and the 1st bit of the opcode since the two opcodes differ by the 1st bit (1 being ne and 0 being eq).

Functions: Shift left logical, shift right logical, and shift right arithmetic



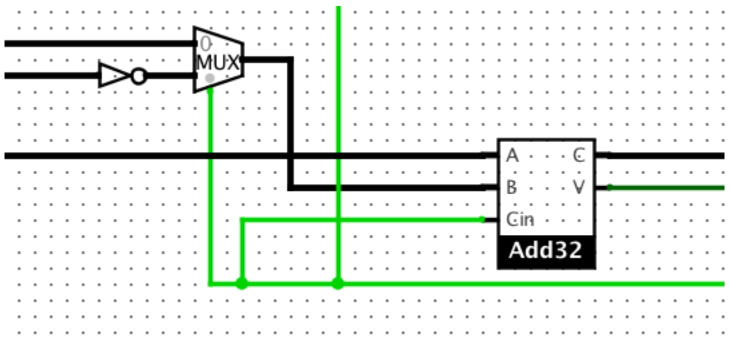
The wire splitter to the left inverts the B input if an right shift is to be performed. To achieve a right shift using a left-shifter, B is inverted, left-shifted by Sa bits, then inverted again. The mux to the left determines whether or not to invert B based on the 2nd bit of the opcode. The 2nd bit is the determining element between a right shift or a left shift (1 being a right shift, 0 being a left shift).

The AND gate at the bottom left determines the Cin value based on B's msb, the zeroth and second bits of the opcode. If both bits of opcode are 1, then the function is an arithmetic right shift, and Cin takes on the value of B's msb. Otherwise, Cin is always zero. See truth table on next page.

AND Gate Truth Table

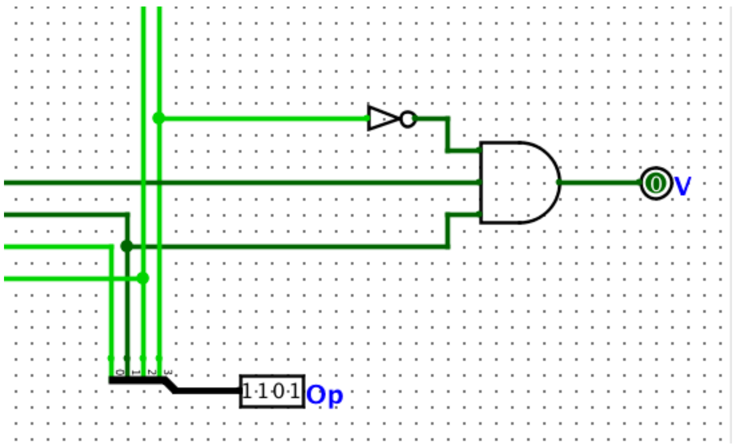
B’s msb	Opcode[0]	Opcode[2]	Cin
1	0	0	0
0	0	0	0
1	0	1	0
0	0	1	0
1	1	0	0
0	1	0	0
1	1	1	1
0	1	1	0

Functions: Add and Subtract



The mux determines whether to add or subtract A and B (since subtracting is basically the addition of A and the negative of B in two’s complement). The control input of the mux is second bit of the opcode. (1 being subtraction, and 0 being addition).

Determining V:

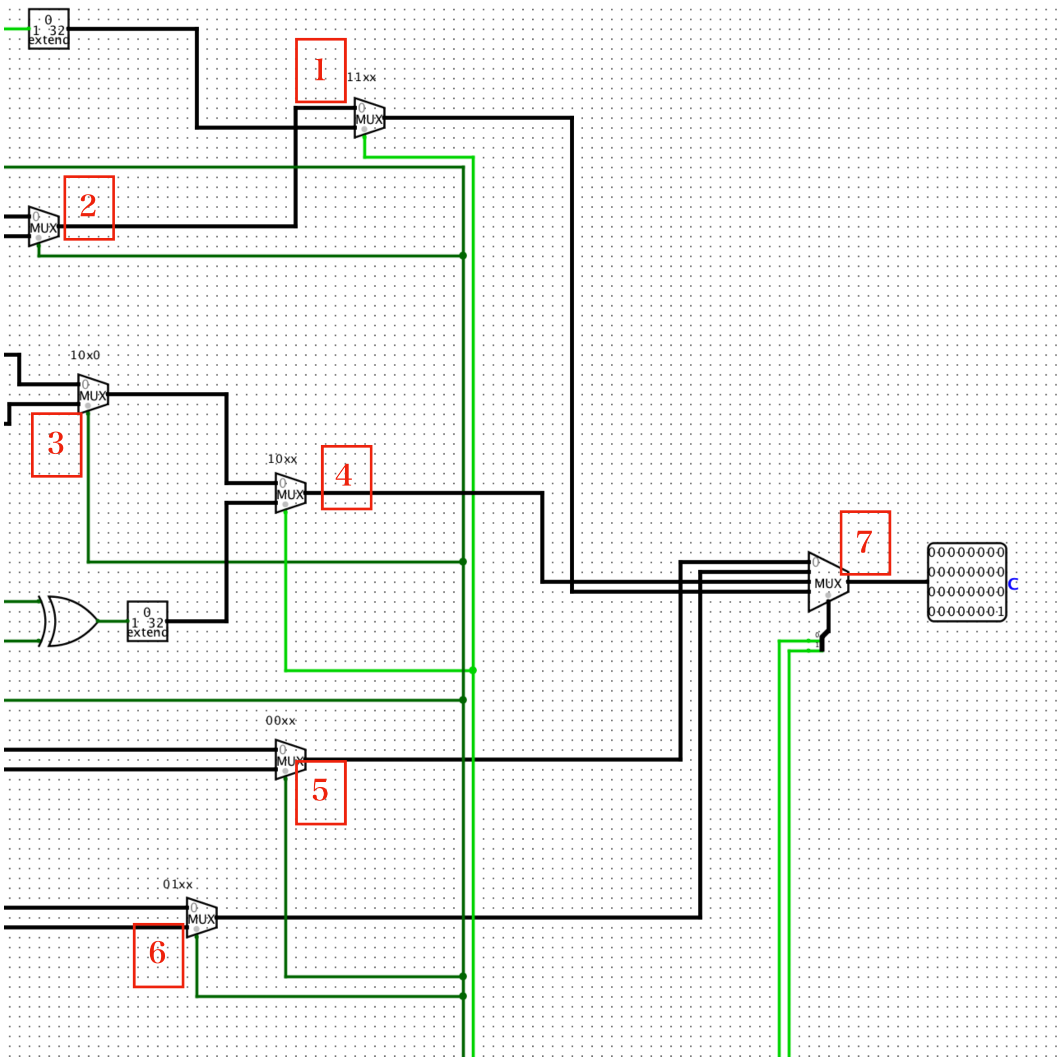


Since V only takes on the 32 adder's overflow value when the opcode asks for addition or subtraction, an AND gate taking in the value of the 32 adder overflow, the first bit of the opcode, and the inverted of the 3rd opcode bit. We invert Op[3] because Op[3] is always 0 for add and subtract function, and always 1 for Op[1].

AND Gate Truth Table

Op [1]	Op[3]'	Add32 C	V
1	1	0	0
1	1	1	1
1	0	0	0
1	0	1	0
0	1	0	0
0	1	1	0
0	0	0	0
0	0	1	0

Logic behind the Muxes



Mux 1: Muxes between function le/gt and output of Mux 2 by using Op[0] as select input. Both le and gt has 1 for Op[0], both XOR and NOR has 0 for Op[0]. Thus, the XNOR gate after the le/gt “subcircuit” chooses between le and gt, and Mux 2 chooses between XOR and NOR.

Mux 2: Muxes between function XOR and NOR using Op[1] as select input. 0 being XOR and 1 being NOR.

Mux 3: Muxes between AND and OR using Op[1] as select input. 0 being AND, 1 being OR.

Mux 4: Muxes between value of Mux 3 and the value of the XOR gate that chooses between eq/ne. Uses Op[0] as select input. Both ne and eq has Op[0] as 1. Both AND and OR as Op[0] as 0. The XOR gate first chooses between eq and ne, mux 3 chooses between AND and OR.

Mux 5: Muxes between left shift and add using Op[1] as the select input. Both left shift and add have the same Op[3] and Op[2] bits, thus Op[1] is the determining element.

Mux 6: Muxes between shift right arithmetic and subtract using Op[1] as the select input.

Mux 7: 4x2 mux that takes Op[3] and Op[2] as the select inputs. 00 would be whatever output mux 5 had selected (Shift left logical or add); 01 would be whatever output mux 6 had selected (shift right arithmetic or subtract); and so forth.