CS 3410 P1 Design Documentation

Mario Xerri (max3)

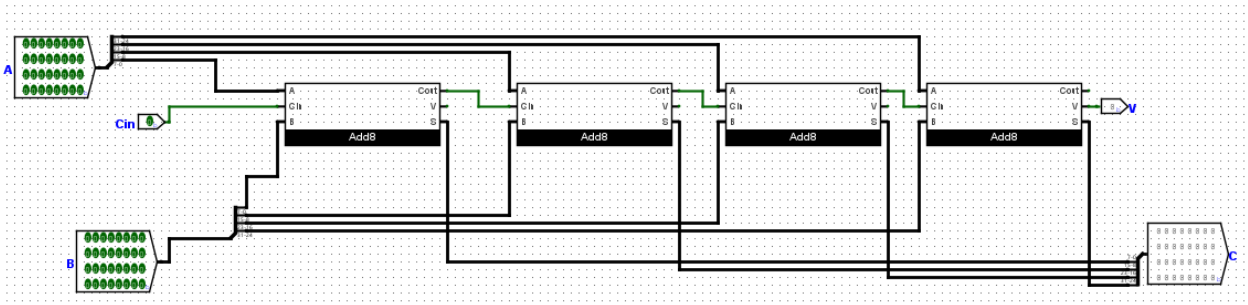
9/12/23

1. Overview:

We are designing a RISC-V ALU (Arithmetic and Logic Unit) which perform performs computation based on the operation which it is assigned to do based on instructions. This is a main component of a CPU and we will be designing its logical circuit using the Logisim software. The ALU is designed to handle bitwise operations between 32 bit numbers such as AND, OR, XOR, NAND; arithmetic operation such as addition and subtraction; equality operations such as Greater than or equal to, less than, equal to, and not equal; and bit shifting operations such as left logical shift, right logical shift and right arithmetic shift. The purpose of the project is to understand the structure of a CPU's ALU so we can understand the inner working of how a CPU performs computations.

2. Component Design Documentation

32 bit Adder:



Brief Description:

The 32 bit adder was designed to support the addition of two 32 bit signed inputs A and B with a carry in bit Cin. C is the output of the addition operation and V is a value which displays if there is overflow from the operation. This is necessary for the construction of the ALU.

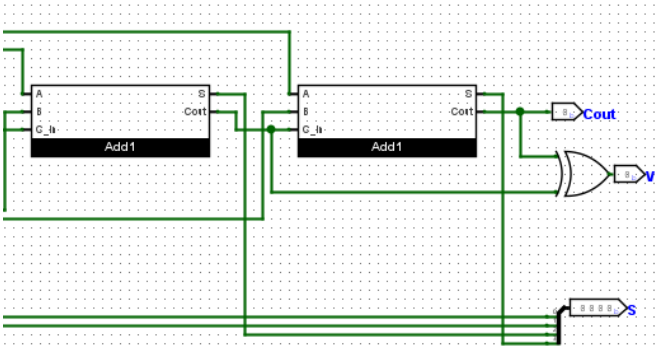
Control Logic and Functionality:

In this circuit, the each 8 bit adder is constructed from 4 bit adders which is constructed from 1 bit adders. To perform the addition, bit 0 from A and B, and the cin input, are sent to the first one bit adder and this one bit adder outputs a carryout and an output value which becomes bit 0 of the 32 bit sum C; then bit 1 from A and B and the carryout from the first adder are sent to the 2nd one bit adder which

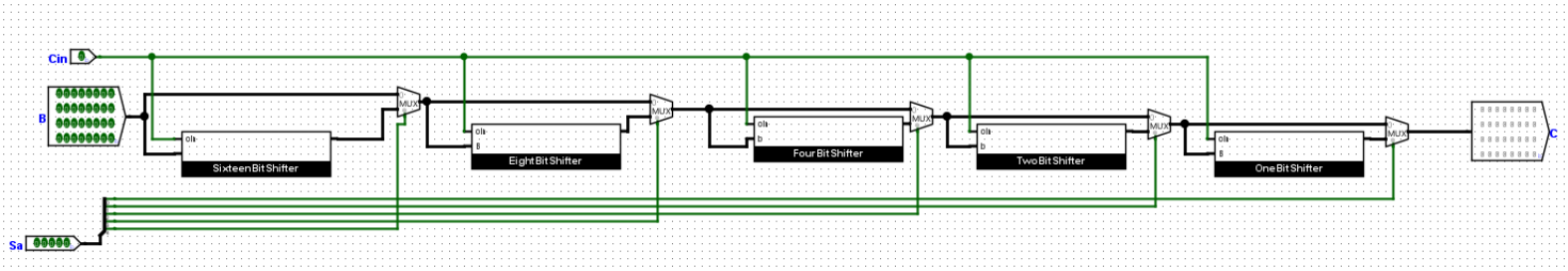
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 1: Truth table for 1bit adder

computes bit 1 of C and another carryout value, and so on until all of the one bit adders have finished their computation. Truth table of the 1 bit adder is shown above and to the right as a reference to this description. The bits from A and B are each sent to their respective adder using splitters which break up the bits of A and B. To determine if overflow occurs in the circuit we look at the carryout value from the last 2 one bit adders. These go into an XOR gate to determine if there is going to be overflow as shown from the snippet on the left of the last 2 1bit adders of a 4bit adder. Overflow for this 32 bit representation occurs when addition leads to a number out of the range of -2^(31) to 2^(31)- 1 inclusive.



32 Bit Left Shifter:



Brief Description:

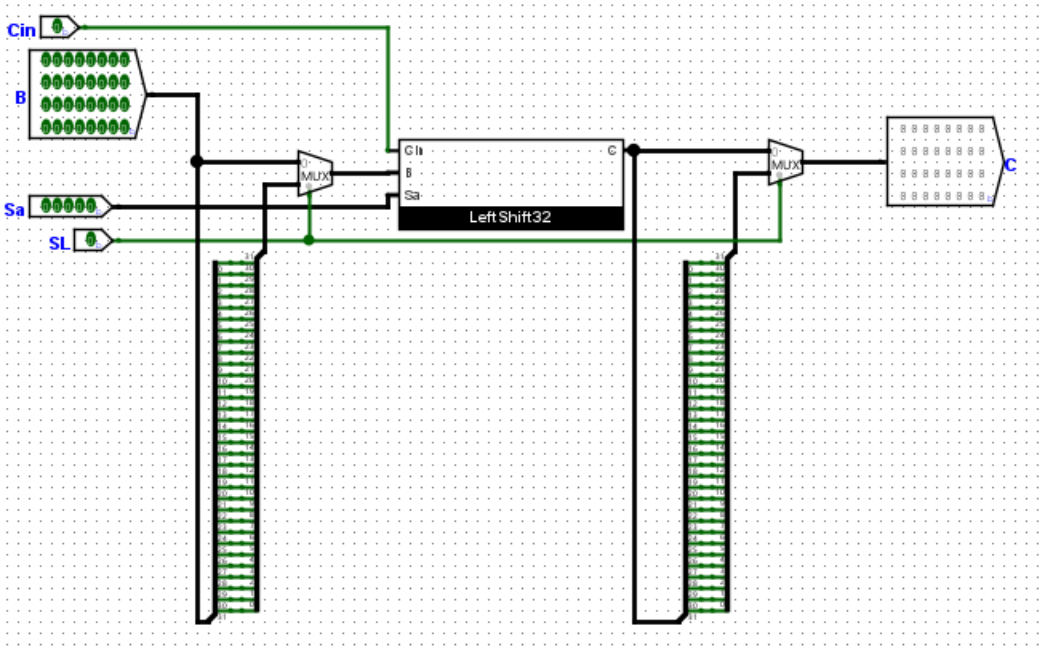
The 32 bit left shifter is designed to take in a 32bit number and shift each bit up to 31 positions to the left and fill each vacated position with a Cin value. This is necessary for construction of the ALU.

Control Logic and Functionality:

The inputs to this circuit is B: the 32 bit number to be shifted; Sa: the unsigned 5 bit number which defined how many positions to the left to shift; and Cin: the bit to fill the vacated bits with. And the output C: the shifted number. This circuit work by shifting in a series of stages where the 16 bit shifter is the first stage, followed by the 8bit, 4, 1 shifters. The Sa input is split into its 5 bits and each bit is used as the mux select bit which controls if that bits shifter should be called or not. A value of 1 in a bit of Sa means to use that respective but shifter. In our circuit we first pass B and B shifted 16 bits to the mux and the selection bit decides if the shifted or non shifted input should be passed. Then this is passed to the 8bit shifter in which the same process occurs. This process continues until the solution is output that goes out of the last mux

which controls the one but shifter. The combinations at different stages enabled by the muxes allow the circuit to shift B to the left by any desired amount. Additionally, if Cin is 0 the vacated bits will be filled with 0s, and id Cin is 1, the vacated bits will be filled with ones. This is performed on the level of the one bit shifter where we use a splitter to connect bit 0 of the output (vacated bit) with the cin and then you connect bit 0 of B with bit 1 of the C, bit 1 of B with bit 2 of C and so on until bit 31 of B is discarded.

32 Bit Shifter:



Brief Description:

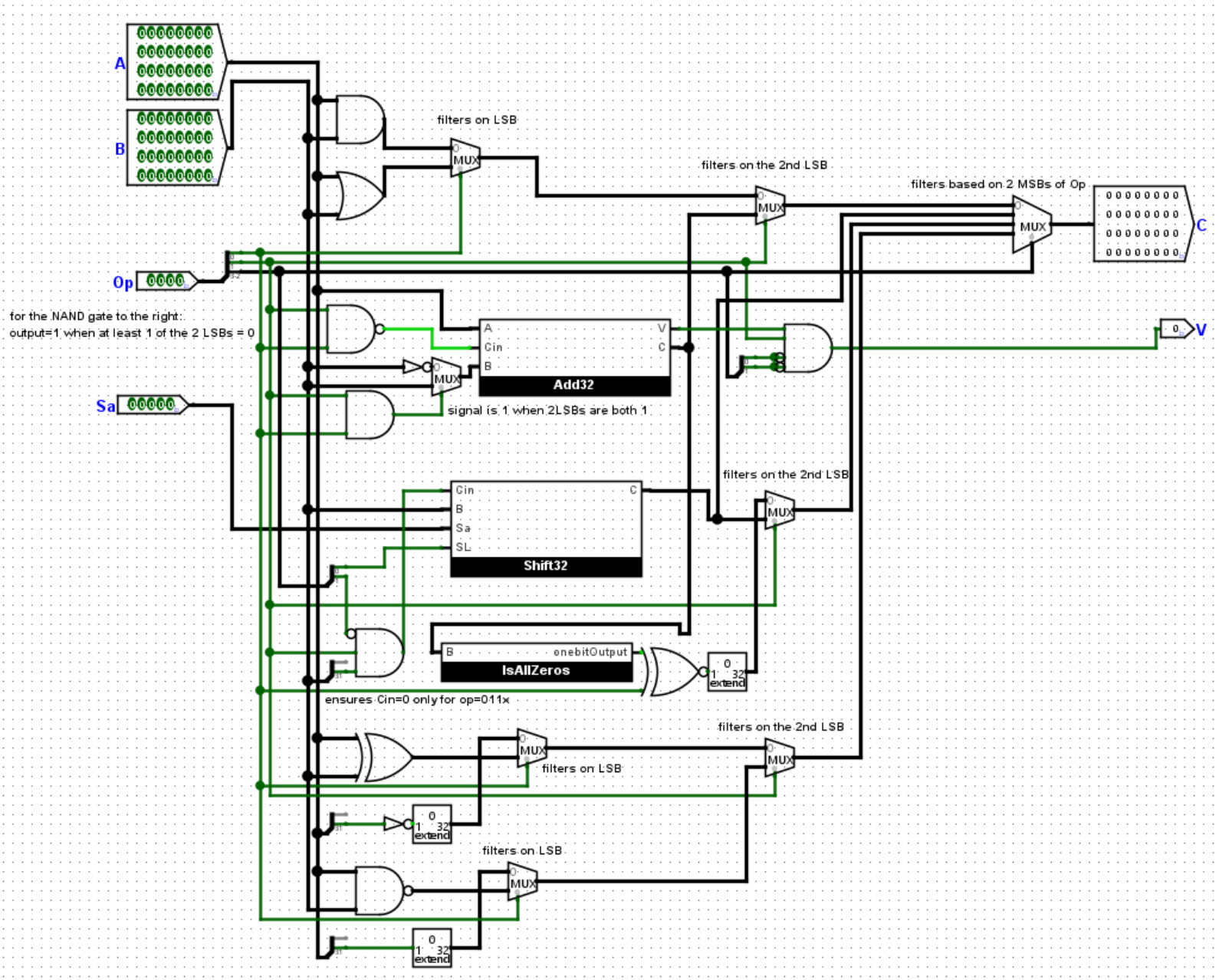
This circuit allows for right or left bit shifts of up to 31 positions where vacated bits are filled by either 0 or 1. This implementation relies on that of the 32 bit left shifter so this will have a shorter analysis. This is necessary for construction of the ALU.

Control Logic And Functionality:

Here B is the 32 bit input number, Sa is the amount that B will be shifted, Sl=1 indicates a right shift while sl=0 represents a left shift, and the value of cin is the bit that will fill the vacated bits in the shifting process. Lastly C is the output. In this circuit if Sl=0 then we have a regular left shift as described in the previous section. If Sl=1 then we have a right shift in which the B is flipped (bit 0 becomes 31, bit 1 becomes 30 and so forth), then is passed through leftshift32 and then is flipped again. This is the process which creates a right shift using a left shift. Sl is the selection bit for the 2 muxes above that will determine if the input and output into leftshift32 are flipped which would dictate whether or not a right shift (Sl=1) or left shift (Sl=0) is performed because it either choses the flipped or not flipped signal in both the input and

output cases. Lastly just like in left shift 32, cin fills the vacated bits and follow the same logic as described in the leftshift32 above.

32 Bit ALU:



Brief Description:

This circuit utilizes in a 4bit instruction and performs binary or unary operations on a 32 bit number(s). This is a very major part of a CPU which performs basic arithmetic and logical operations.

Inputs:

The inputs to the 32 bit ALU are A and B: 32 bit numbers for computation; op: the operation which needs to be computed represented as 4 bits; and Sa: the shift amount for shift operations (represented as 5 bits). The outputs are the 32 bit resultant value C and V which indicates if overflow occurred if its value is 1.

Mux Arrangement:

First I am going to explain basic logic flow of the circuit created by the muxes. First I grouped together operations by their 2 MSBs. Then within each group I broke operations into 2 sub-groups based on the 2nd LSB and then did the same for the LSB of each of these sub-groups (if the operation had a unique LSB). This would leave 1 operation in each group. First compute all of the operations (will go more into depth on this later) before passing anything into a mux to filter by operation. Then if there are 2 operations with a unique 4 bit op code and the same 3 MSB's but different LSB pass this through a mux with the select bit being the LSB of the op code to determine mux output. A selector bit of 1 would pass a the operation result that had a LSB of 1. I labeled these muxes in the picture as "filters on LSB". Then these muxes would pass the signal to another mux which filters the result of 2 operations which have the same 2 MSBs of their op codes but differing 2nd LSBs and utilizes the same select bit strategy as with the LSB filtering muxes except now the select bit is the 2nd LSB. Lastly these muxes labeled as "filters on 2nd LSB" pass their resultant signal to the mux which filters based on the 2 MSBs of the op codes. This mux thus has 2 select bits (the 2MSBs) and has 4 inputs to chose from, each coming from the signal that prevailed from the muxes that filtered signals by 2nd LSB for operations that had the same 2 MSBs. Then this larger mux is outputs the C.

There are only 3 2nd LSB filters and not 4 because one of the 4 groups of operations with the same 2MSBs consisted of just shift operations and the exact shift operation was determined before the signal was passed into the Shift32 circuit. Manipulating the signal to indicate addition or subtraction before passed into the adder also allowed for a mux not to be utilizes here. Same goes for all of the shift operations. I will talk about this more in depth later.

Logic Control and Functionality for All Operations:

* I will not explain the mux structure to make sure the right operation was executed as this was elaborated upon

AND – this operation passed A and B through one and gate which performed bitwise and on A and B and no preprocessing had to be done for this operation.

OR – this operation passed A and B through an or gate which performed bitwise or operation on them and no preprocessing had to be done for this operation.

The Shift Operations (Left Logical, right logical, right arithmetic)– There are a couple gates we use to determine the input for the shift32 circuit based on what shift we are performing and I will list some truth tables to illustrate the gates we used. The value of Cin should only be one when a right arithmetic shift occurs and B is negative (bit 31 is zero).

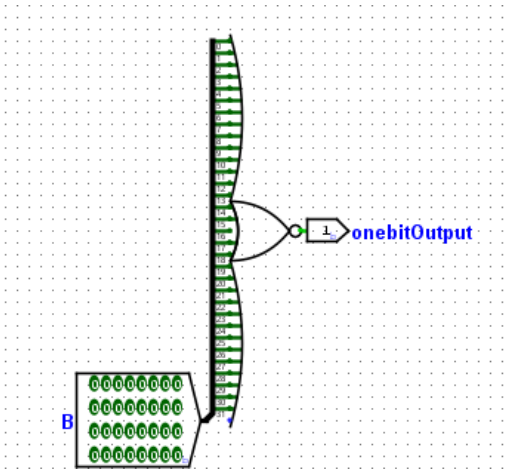
Op code bit 1	Op code bit 3	31st bit of A	Output (Cin)
1	0	1	1
1	1	1	0
1	0	0	0
0	0	1	0
0	1	0	0
0	0	0	0
0	1	1	0

This truth table justifies the 3 input and gate with the top input inverted which is to the left of the all zeros function (which checks if all bits are zero). The output of this gate goes to the Cin input of left shift32. Also, 2nd MSB of the opcode determines if the right shift or the left shift should be performed as it is the Sl input for Shift32 circuit. All other inputs to that circuit are straight forward as seen in the pic above (2nd MSB=1 indicates right shift and 0 indicates left).

Operations Using add32 circuit (ADD, SUB, EQ, NEQ) – We had to determine how to make a signal that would make Cin in the adder 1 when subtraction needed to be done (for operations of subtract, equality and inequality) and 0 when addition was occurring. A truth table was used to help us make out decision here:

Op code bit 3	Op code bit 2	Op code bit 1	Op code bit 0	Cin (output)
0	0	1	0	1
1	0	0	1	1
1	0	0	0	1
All	Other	Combinations	Yield	0

After quick analysis and running this through Logisim we used a NAND gate taking bit 0 and bit 1 as inputs. Additionally the output inverted was used as a selector bit for a mux that decided B should be inverted as it should be for any subtraction operation when inputted into Add32. The NAND gate we are referring to is the left of the Add32 circuit box. As addressed recently, the EQ and NEQ operation utilize the adder to perform subtraction. There is a function in the AlU circuit called isAllZeros and this splits B into all of its individual bits and connects all of them to a NOR gate to ensure that the output is one when all of the bits in B are zero. As shown from the picture to the right. For EQ and NEQ we get the value of A-B and then this number is sent to the isAllZeros function where it checks if A-B = 0. One or zero is outputted by this function and then the data is extended back to 32 bit.



LT & GE – For these functions we checked the 31st bit of A and the number is negative if that bit is 1 otherwise its positive. Then we use a bit extender from 1 bit -> 32 to get the signal back to 32 bits. No work done before the operation.

XOR, NAND – We passed signals A and B through these gates to perform the respective bitwise operation. No work done before the operation.

Overflow Logic:

The AND gate to the right of the Add32 box dictates if V (overflow) should be 1. Overflow should only occur during the addition of subtraction operation and it should only be 1 during those operations. This truth table describes the conditions for V to be 1.

Op code bit 3	Op code biit 2	Op code bit 1	V from Add32	V for ALU (output)
0	0	1	1	1
0	0	1	1	1
All	Other	Combinations	Yield	0

0010 and 0011 are the op codes for subtract and add so these are the only ones that should lead to an overflow output. For this reason we use the and gate to the right of the add32 box to make sure the conditions in the truth table hold to output overflow correctly.

3. Reference

The truth table of the one bit adder I took from the doc outline from canvas.