CS 3410 Design Document
Milica Mandic
February 3, 2013

1 Overview

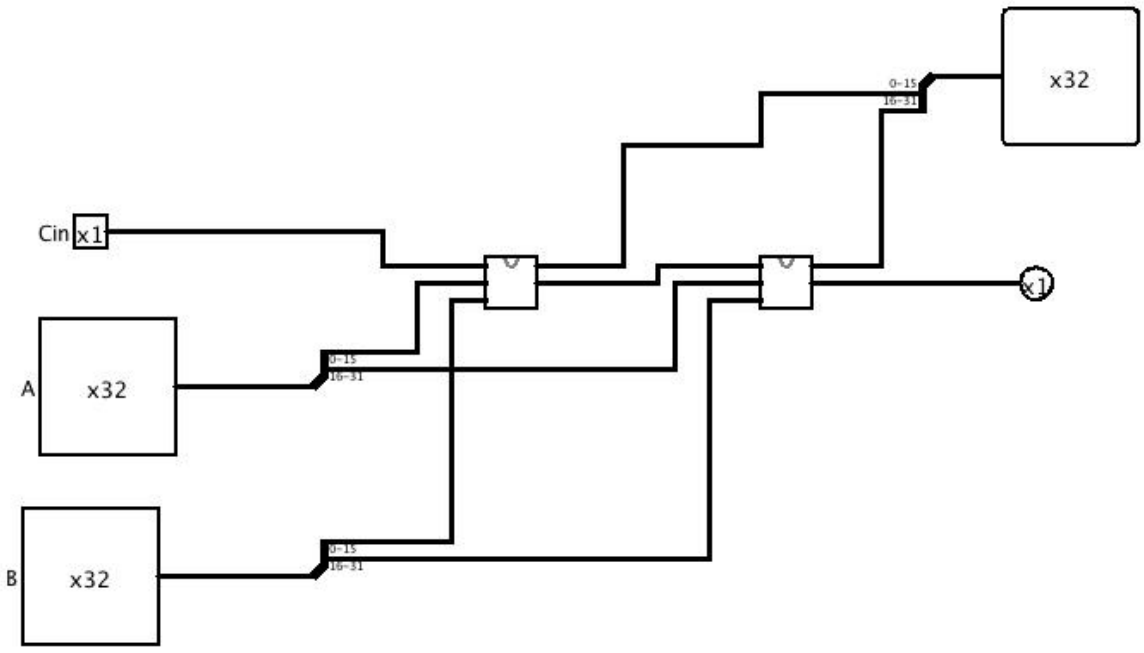
The purpose of this document is to outline the design of an ALU in Logisim. The circuit implements a couple of basic arithmetic and logic operations on 32-bit numbers.

2 Component Design

2.1 Main components used

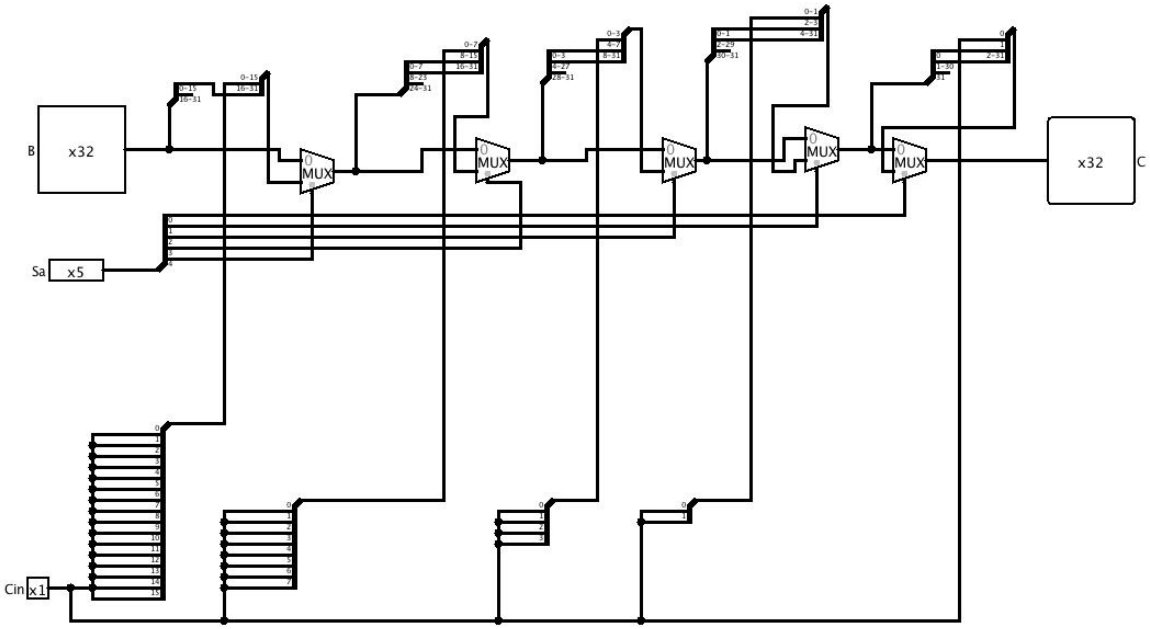
Add32

This circuit adds two 32-bit numbers, returning the sum and detected overflow. Inputs that are given to it are the 2 numbers and a carry in- this carry in is 0 for addition and 1 for subtraction (if the second number is previously bitwise negated) . This component is made out of 5 smaller components: 1-bit adder, 4-bit adder, 8-bit adder, 16-bit adder; these are all buiding blocks for the larger circuit. For an example, 32-bit adder uses 2 16-bit adders to do the addition:



LeftShift32

This circuit shifts Sa number of bits to the left and substitutes them with Cin. Since Sa is a 5-bit number we can divide the process into shifting 16,8,4,2 or 1 to the left based on the 1s in this binary number. This circuit is used to implement right shifts as well, which is done by flipping all the bits (31-> 0, 30-> 1 etc), doing the shifting and then reflipping the bits back. The difference between logical and arithmetic right shifts is that logical shift always adds 0s to the front, while the arithmetic one will shift in the MSB. This is an example of the general LeftShift32:



2.2 Operations implemented

- **Shift Left Logical**- Shifts B left by Sa spaces, while the rest is filled with 0s (Cin is 0). Uses LeftShift32.

- **Add**- Adds A and B and detects overflow V, which check for by checking whether A and B have the same sign bit which is opposite from the sum's sign bit. Carry in is 0. Uses Add32.

- **Shift Right Logical-** Flips all the bits of B, uses ShiftLeft32 on them and then flips all the bits back into the initial position. This way

shifting left is done from the opposite side and results in right shift. Carry-ins (inserted bits) are all 0s.

- **Shift Right Arithmetic-** The same as Logical only now the Carry-ins have the value of the MSB of B. This way if B was negative initially (MSB=1), it will stay negative even after the shifting.

- **Subtract-** Uses Add32, but previously gets a bitwise negation of B and uses Cin of 1. This way we get the 2's complement negated value of B. In other words A- B= A+ (-B)

- **And-** Use AND gate with 2 32-bit inputs A and B

- **Or-** Use OR gate with 2 32-bit inputs A and B

- **XOr-** Use XOR gate with 2 32-bit inputs A and B

- **NOr-** Use OR gate with 2 32-bit inputs A and B, and then NOT gate on the output

- **Equals-** Subtract A and B- if output is 0 then return 1 otherwise 0

- **NotEqual-** Negated value of Equals

- **Greater-** Subtract A and B, if MSB of the output is 0 then return 1 otherwise 0

- **LessOrEqual-** Negated value of Greater.

2.3 General ALU implementation

All values are calculated for all operations and then they are chosen by a Multiplexer based on the Opcode that was requested. For an example if "1000" is selected, then the Multiplexer should return the result of ANDing A and B together. The second, smaller mux will choose the value of V, the overflow- this should be determined for "001x" and "011x" only, since addition and subtraction are the only ones that result in overflow. For all the other Opcodes the Multiplexer should return 0.