Introduction

This document will show the design process of a 4 function, 4 bit ALU. It will cover the following components:

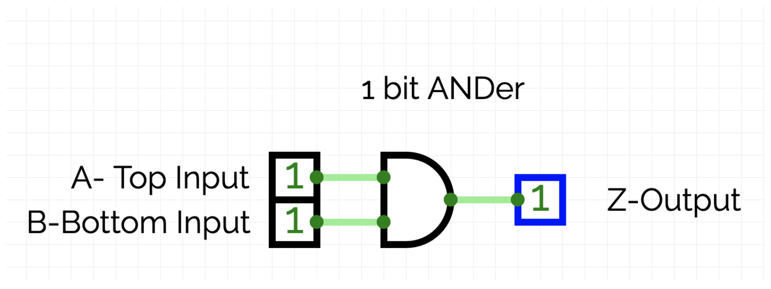
AND

For this ALU, AND just performs one AND operation to the two 4 bit inputs. Lets sketch the truth table, equation, and diagram for a 1 bit AND logic unit here:

A- Top Input	B-Bottom Input	Z-Output
0	0	0
0	1	0
1	0	0
1	1	1

In this case, there is only one condition for a true output, A and B must be true.
Here is the diagram and equation:

$$Z = A \cdot B$$



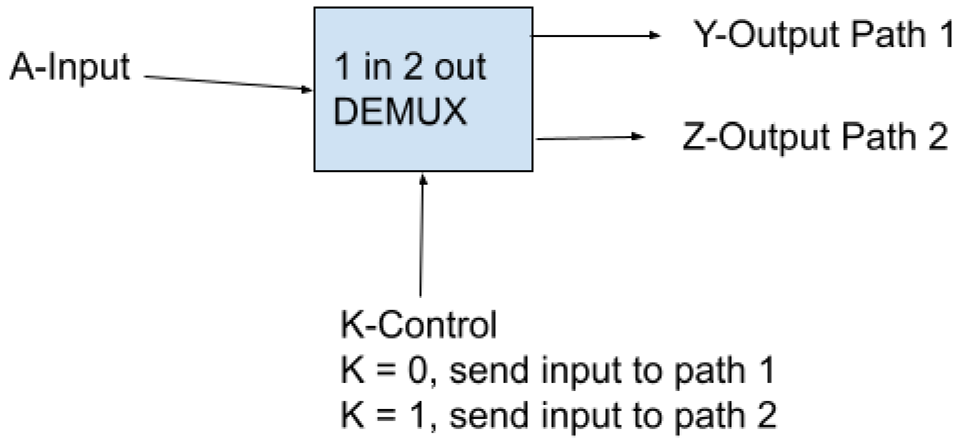
As you can see, this is the easiest logic unit out of all the other 4 in the ALU.

DEMUX

In an ALU, DEMUX takes one input and decides which path it should take for the output based on the control lines. This is really like going for a hike. You can come into an intersection where you have 3 options. In this case, your control signal to take the right path will be based on a map and your goal (e.g. taking the shortcut to leave or keep exercising through the longest route back to your car).

In this document, it will demonstrate the design process for a 1 bit, 1 in 2 out DEMUX (Y and Z). It has 2 outputs and 2 inputs (1 for the actual data (A) and 1 for control (K) which is enough to handle 2^1 (2) possible paths).

Here is a rough overview drawing of the DEMUX



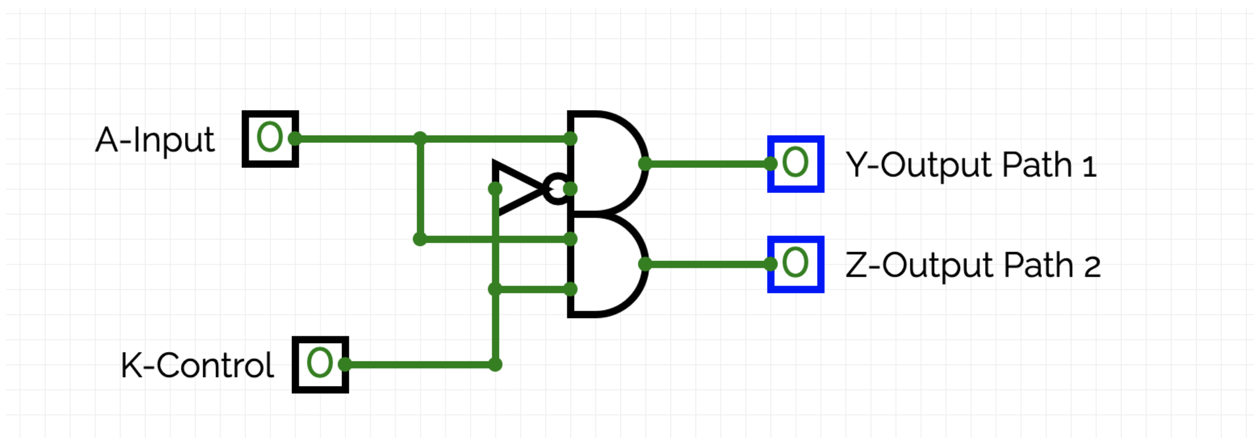
Here is the truth table:

A-Input	K-Control	Y- Output Path 1	Z- Output Path 2

0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

Highlighted rows are the input conditions that make the Y or Z output true. My logic gate layout will be built based on those conditions that results in a **true** output for Y or Z.

Based off the truth table, here is the drawing and equation for the circuit:



$$Y = A\bar{K} \quad Z = AK$$

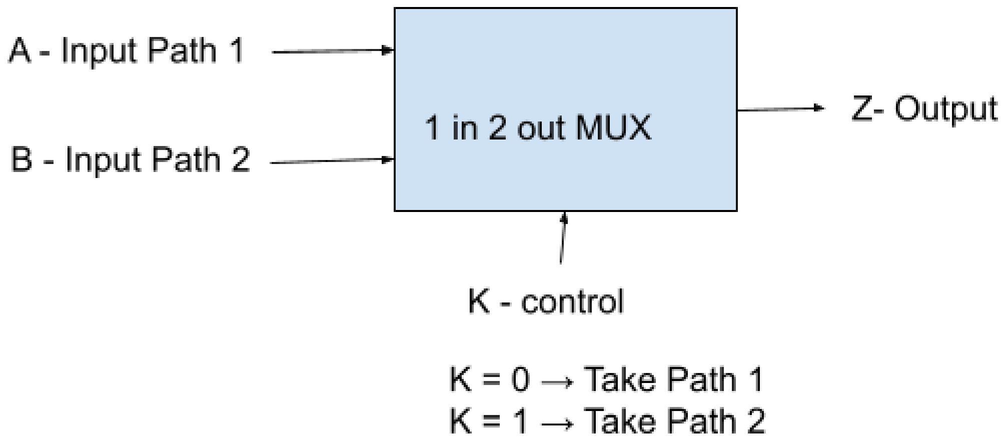
Just like the AND logic unit, this Demux circuit is built straight from the truth table in the simplest form.

MUX

In an ALU, MUX decides which one of all the input paths it routes to the output based on the control lines. This is really like a redundant dual propane supply switch valve. Your propane gas from 2 tanks need to enter into the furnace’s supply. In this case, your control signal is the propane level which maximizes performance for the furnace.

In this document, it will demonstrate the design process for a 1 bit, 2 in 1 out MUX (Z). It has 1 output(Z) and 3 inputs (2 for the actual data (A and B) and 1 for control (K) which is enough to handle 2^1 (2) possible paths).

Here is a rough overview drawing of the MUX:



Here is the truth table:

A- Input Path 1	B-Input Path 2	K-Control	Z- Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1

1	1	1	1

Here is an equation based on the input conditions that results in a **true** output. This one however can be simplified which is shown here:

$$Z = \bar{A}BK + A\bar{B}\bar{K} + AB\bar{K} + ABK$$

Commutative law

$$Z = \bar{A}BK + ABK + A\bar{B}\bar{K} + AB\bar{K}$$

Distributive law

$$Z = BK(\bar{A} + A) + A\bar{K}(\bar{B} + B)$$

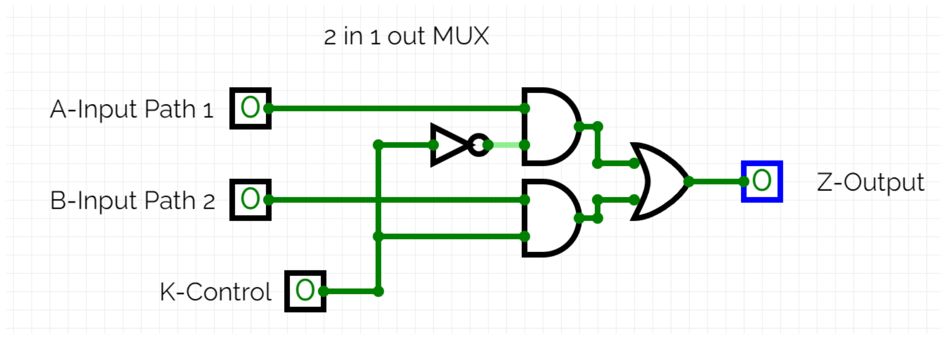
Complementary law

$$Z = 1BK + 1A\bar{K}$$

Law of intersection

$$\boxed{Z = BK + A\bar{K}}$$

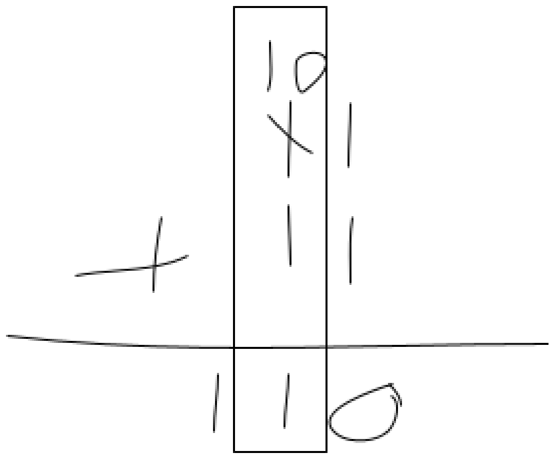
With this simplified equation, here is the circuit diagram for this MUX:



Full Adder

Full Adder is an adder that can handle a carry out output from the previous adder math units. This adder is very like dealing beyond the first digit in elementary school math. You now have to take into account the carry outs from the previous digits.

Example:



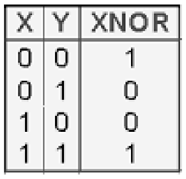
In this case, the second digit (inside the rectangle) not only had to deal with the two input values, but also the carry out from the first digit which is why there is also a third digit in the sum.

Taking this into account, lets build the truth table for the full adder.

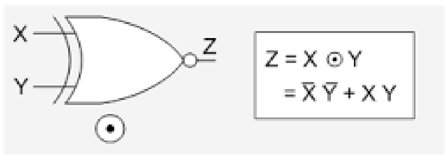
A- Top Number	B- Bottom Number	C- Carry In	Y - Sum	Z- Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

From this truth table here is the equation in simplified form:

Note : External online source is used for the XNOR identity:



X	Y	XNOR
0	0	1
0	1	0
1	0	0
1	1	1



$$Z = X \odot Y$$
$$= \bar{X}\,\bar{Y} + X\,Y$$

$$Y = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

$$Y = \bar{A}\bar{B}C + ABC + \bar{A}B\bar{C} + A\bar{B}\bar{C}$$

Commutative law

$$Y = C(\bar{A}\bar{B} + AB) + \bar{C}(\bar{A}B + A\bar{B})$$

Distributive Law

$$Y = C(\bar{A}\bar{B} + AB) + \bar{C}(A \oplus B)$$

XOR identity

$$Y = C(\overline{A \oplus B}) + \bar{C}(A \oplus B)$$

XNOR identity

$$Y = C \oplus (A \oplus B)$$ XOR identity

$$Z = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

$$Z = C(\bar{A}B + A\bar{B}) + AB(\bar{C}+C)$$

Distributive Law

$$Z = C(A \oplus B) + AB(1)$$

XOR identity

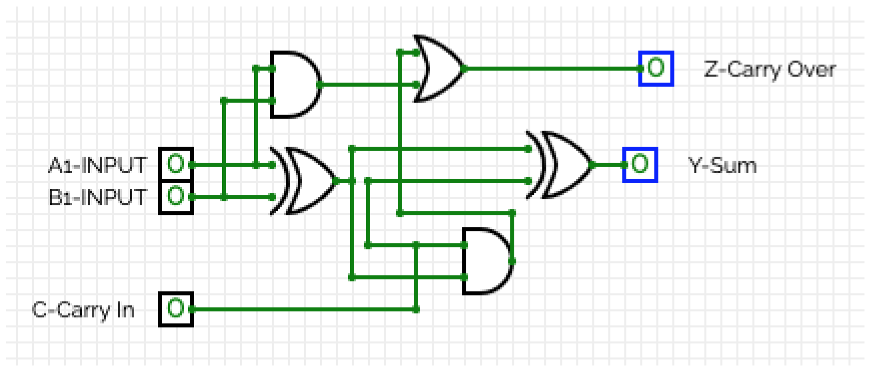
Complementary Law

$$Z = C(A \oplus B) + AB$$

Law of Intersection

With the simplified equations for the 1 bit Full Adder: Here is the circuit diagram for the 1 bit Full adder.

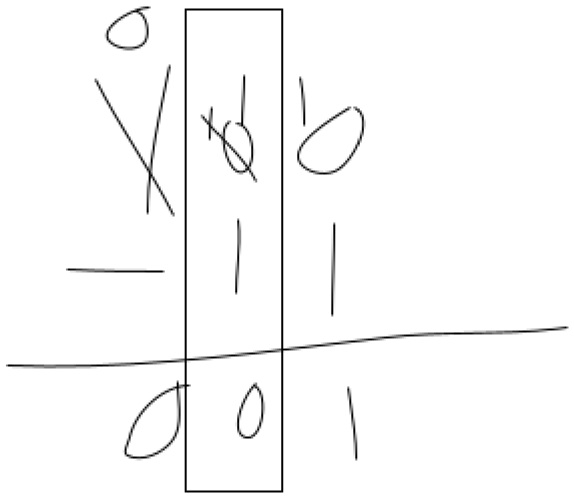
Note: Both the Y-Sum and the Z-Carry out requires the output of the subcircuit A **XOR** B. Thus, only one of this subcircuit is used as both outputs will share this subcircuit.



Full Subtractor

Full Subtractor is a subtractor that can handle a borrow request output from the previous subtractor math units. This subtractor is very like dealing beyond the first digit in elementary school math. You now have to take into account the borrow request from the previous digits.

Example:



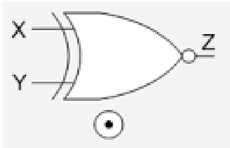
In this case, the second digit (inside the rectangle) not only had to deal with the two input values, but also the borrow request from the first digit.

Taking this into account, let's build the truth table for the full adder.

A- Top Number	B- Bottom Number	C- Borrow In request	Y - Difference	Z- Borrow Out Request
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

From this truth table here is the equation in simplified form:
Note : External online source is used for the XNOR identity:

X	Y	XNOR
0	0	1
0	1	0
1	0	0
1	1	1



$Z = X \odot Y$
$= \bar{X}\bar{Y} + XY$

$Y = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$

$Y = \bar{A}\bar{B}C + ABC + \bar{A}B\bar{C} + A\bar{B}\bar{C}$

Commutative law

$Y = C(\bar{A}\bar{B} + AB) + \bar{C}(\bar{A}B + A\bar{B})$

Distributive Law

$Y = C(\bar{A}\bar{B} + AB) + \bar{C}(A \oplus B)$

XOR identity

$Y = C(\overline{A \oplus B}) + \bar{C}(A \oplus B)$

XNOR identity

$Y = C \oplus (A \oplus B)$ XOR identity

$$Z = \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + ABC$$

$$Z = \bar{A}\bar{B}C + ABC + \bar{A}B\bar{C} + \bar{A}BC$$

Commutative Law

$$Z = C(\bar{A}\bar{B} + AB) + \bar{A}B(\bar{C} + C)$$

Distributive identity

$$Z = C\overline{(A \oplus B)} + \bar{A}B1$$

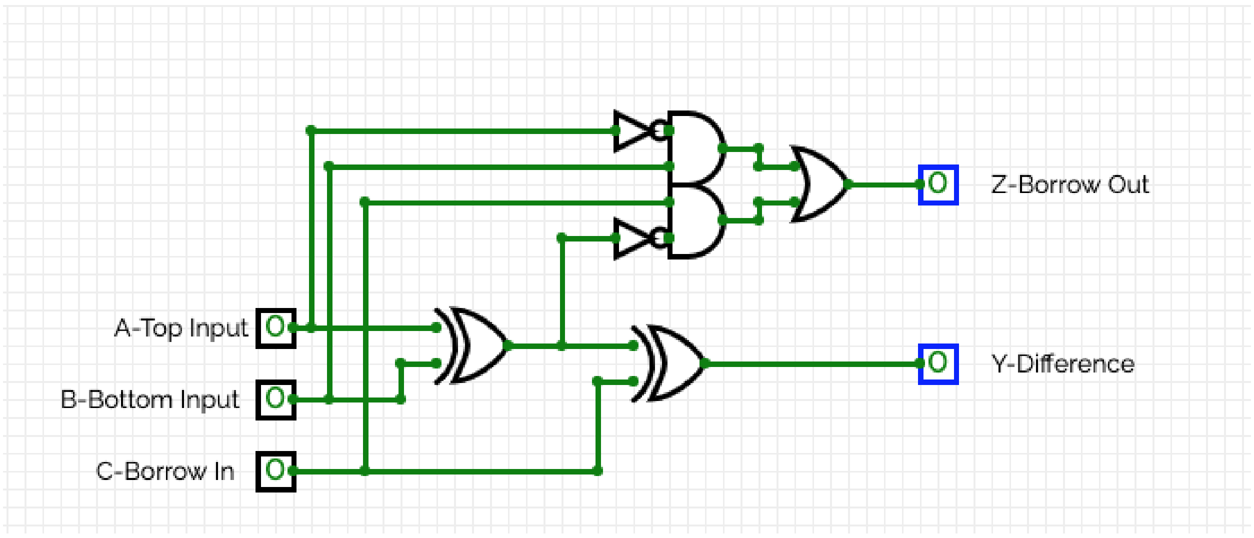
XNOR Identity Complementary Law

$$Z = C\overline{(A \oplus B)} + \bar{A}B$$

Law Of Intersection

With the simplified equations for the 1 bit Full Subtractor: Here is the circuit diagram for the 1 bit Full subtractor.

Note: Both the Y-Difference and the Z-Borrow out requires the output of the subcircuit A **XOR** B. Thus, only one of this subcircuit is used as both outputs will share this subcircuit.



Less Thander

LESThander is a logic gate that returns true if top number < lower number. Otherwise, it returns false.

Just like in class examples on ALU design, we can use truth tables and equations to build the LESThander… but there is a quicker way instead of doing this much work.

We learned what a subtractor is…

The subtractor features the borrow request when the top number < bottom number for one digit. This can be used to my advantage to build a Lessthander.

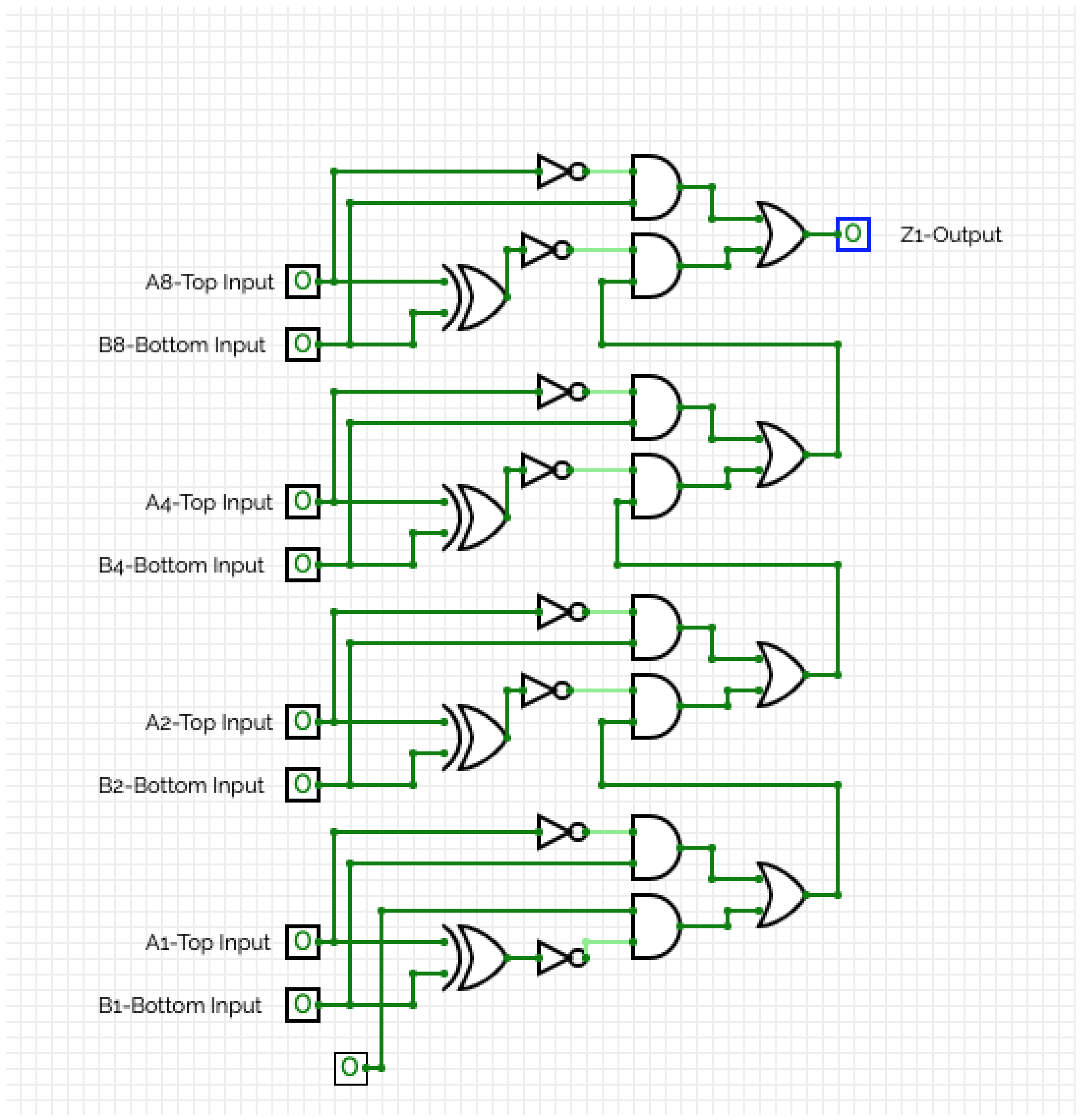
Just like how you can quickly see if a base 10 number (common number used in math and everyday lives) is larger than another by checking the sign of a subtraction result (if a-b results in a negative value, a<b, and is a-b is does not result in a negative value, a>= b) which is essentially binary state as output, same with the borrow request. It also outputs a binary state (1 - borrowing from the next digit **is required**. 0- borrowing from the next digit is **not required**).

Specifically, I care about the borrow request value for the 4th digit's (leftmost digit) borrow request. This is because… if the top number is greater than the bottom number or is equal to, there is no need to borrow from a digit that does not exist. (e.g. if I subtract 100 by 99, the difference is 1 and it only borrows up to the third digit.)

Thus, that is the sign analysis equivalent of the a-b example.

However, there are gates that we do not need which I will remove. It includes the xor gates that will be used for the difference output. Who cares about the difference in a Less Than operation?

Here is the full 4 bit Less Thander Circuit:



Note for the Final ALU: Since a Less Thunder only returns one binary digit, the remaining 3 digits (D2, D4, and D8) on the 4 bit MUX input will be permanently attached to a zero output as placeholders for the input of the MUX. Thus, If a<b, the ALU will output 0001 while a>=b will cause the ALU to output 0000

Conclusion

This is how you build a 4 function, 4 bit ALU.