Ok guys, (long) story time.

I never ended up cutting the large board for the 4-bit design I was working on because I was more than a little worried about committing the mill to a 5 hour (or more) cut and drill. Also, boards in that size are expensive - $20 for two whereas I can get the smaller 150mm x 100mm for $20 for 10. Not to mention the mill only takes about an hour to cut and drill one of the smaller boards. If I make a mistake in the design, it’s a lot easier to fix one smaller board than the bigger ones.

So, that was the first line of thinking that sent me down the path that lead to this wall of text. The second was a simple question I had. In all the 4-bit designs I’d been working on (and even the 4-bit ALU I showed you guys earlier), I had a data bus that was 4-bits wide, meaning I could only do math on numbers up to 15. That seemed massively limiting. Even with an 8-bit data bus, that only allows math on numbers up to 255.

However, there are tons of 8-bit computers and video games out there. So, how did they do math on large numbers? Loops! That’s how. Instead of adding or subtracting the entire number at once, they break it up into 8-bit sections, and then loop through until they have the answer. Here’s an example with some random numbers:

3,692 + 4,503

3,692 in binary is 0000 1110 0110 1100‬

4,503 in binary is 0001 0001 1001 0111‬

Now, you just do the math on the first eight bits and then loop around do the math on the second 8 bits. The first 8 bits of 3,692 are 0110 1100 and the first eight bits of 4,503 are 1001 0111‬. When doing math in binary, if you add two ones, you can’t right 2, so you write 0 and carry the 1. Here’s how that’s done.

0110 1100

1001 0111

From right to left adding the top number to the bottom number:

0+1 = 1

0+1 = 1

1+1 = 0 (1) > The parenthesis is our carry

1+0+(1) = 0 (1)

0+1+(1) = 0 (1)

1+0+(1) = 0 (1)

1+0+(1) = 0 (1)

0+1+(1) = 0 (1)

So, the first 8 bits of our answer should be 0000 0011. Then we do the same procedure for the second either bits. The actual answer is 8,195, which is 0010 0000 0000 0011. You can see that the first eight bits (from right to left) are the same as those the we just figured out doing the math manually.

Okay, that’s how addition in binary works. I won’t go into subtraction because it’s a little more complicated, but you get the idea.

Now, to make something 8-bits wide with relays is difficult. It takes a massive number of relays, which takes up a lot of space and money. 4-bits is much easier, but since we’re just doing loops, what if we only did 1-bit? Sure, it’d be a lot slower, but these are relays we’re talking about, speed went out the window a long time ago. Enter my idea for 10-bit addition/subtraction calculator that uses a 1-bit ALU (Arithmetic Logic Unit).

I started designing and came up with these six boards to achieve what I wanted:

PICTURES

Here’s how it works.

You input your numbers in binary on the top two rows. Each row supports up to 10-bits, which means any number from 0 to 1023. Then you press the little button and it spits out the answer on the bottom row of LEDs. There are 11 LEDs, so it’ll display an answer up to 2,047 (which is 1023+1023).

When you push the button, the sequencer (the 2nd board) starts. The sequencer steps through each of the 10 bits one by one all the way to the end.

When the sequencer steps through a bit, it sends 5V into the corresponding bit for the first and second number. If the switch is on, that info is sent to the ALU (the last board).

The ALU does the math. It outputs the sum into one of the two sum storage units (the 3rd or 4th board) and outputs the carry into the storage board (the 5th board) for use on the next bit.

The LEDs on the bottom just show the contents of the two storage units.

It sounds ridiculous and when you see it completed, it is absolutely one hundred percent ridiculous.

PICTURES

And here’s a video of me using it (this one will take a while to upload):

VIDEO