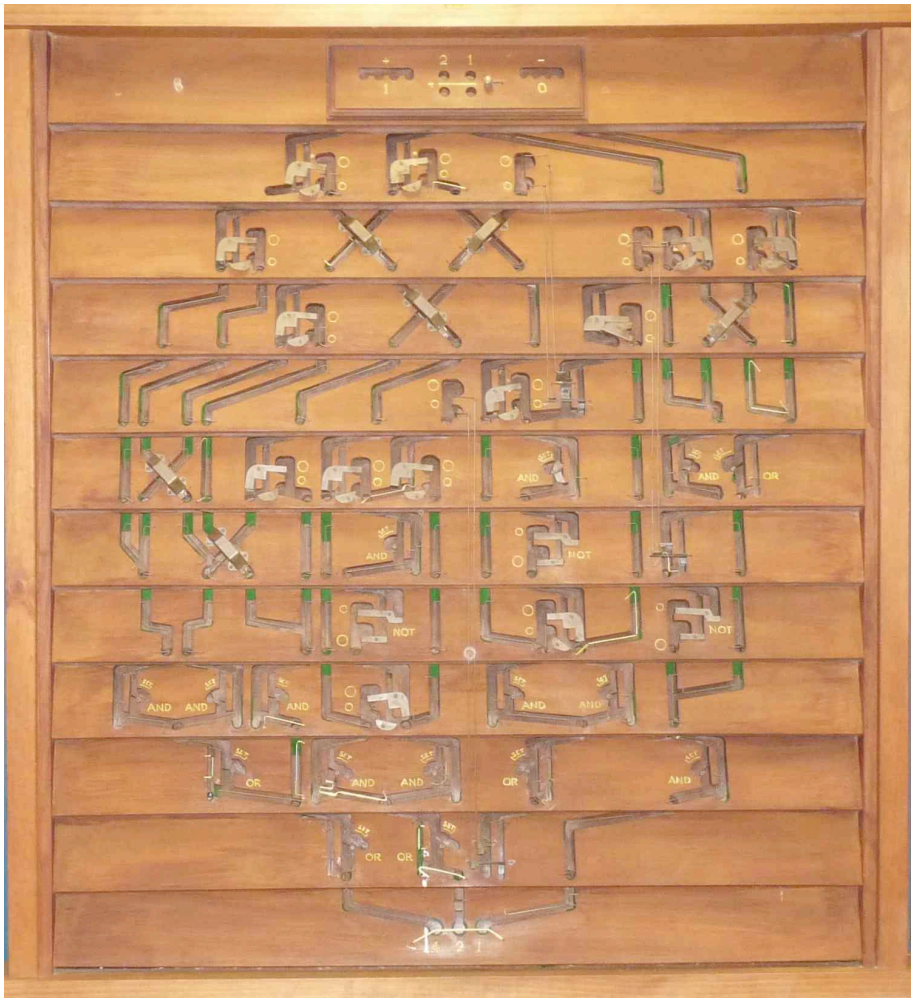
Leigh Christensen's Sculpture "Mainframe"



Leigh Christensen's computer sculpture was inspired by the "logic circuits" that are used in the design of computers. As well as looking great, the sculpture actually works! It is a circuit that adds together two small numbers to produce their sum.

Operating the Sculpture

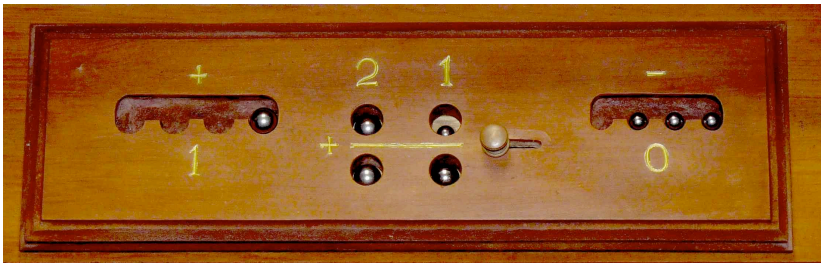
We have to go into computer concepts a little to understand what the sculpture actually accomplishes before we can operate it sensibly.

The sculpture, like a computer, deals with "binary" numbers. Binary numbers restrict the digits in numbers to just two – 0 and 1 – from the usual ten digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. These two "binary digits" are called "bits", for short. The sculpture adds any two binary numbers of only two bits each. With binary numbers, the rightmost bit represents 1, the next to the left represents decimal 2. There are only four 2-bit binary numbers 00, 01, 10, 11 representing decimal numbers 0 (0+0), 1 (0+1), 2 (2+0), 3 (2+1).

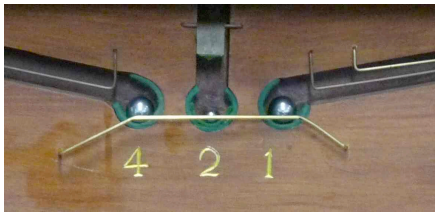
The sculpture can add any two of these 2-bit numbers. The largest possible two numbers to be added are both 3 (in decimal) so the largest possible sum can be 6 (and the smallest, of course, 0.) Decimal 6 is larger than the biggest 2-bit binary number so to the sum needs an extra leftmost bit, representing decimal 4. The "hardest" sum that the sculpture can undertake is, in decimal: 3+3 = 6, in binary: 11 + 11 = 110.

Where the sculpture differs from real computers is how the bits themselves are represented. Computers use electricity, voltages, or currents. The sculpture makes do with ball-bearings running in wooden grooves (slots or races or paths) routed into the planks from which the sculpture is constructed. To distinguish the two possibilities a large ball-bearing represents the bit 1, and a small ball-bearing the bit 0.

When not is use the ball-bearings of the two sizes are held in racks on each side of the panel on the top plank of the sculpture. When ready to work the two input number's ball-bearings are loaded into the holes in the top centre. The input numbers are one above the other, let's call them *top* and *bottom* with bits t2, t1 and b2, b1 respectively.

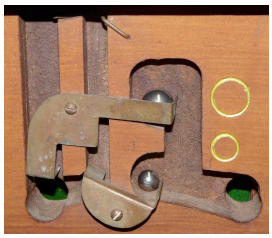


When the sculpture is operated the sum eventually will appear on the lowest plank. There will be three ball-bearings representing the sum - let's call the result bits s4, s2, s1 – representing 4 or 0, 2 or 0 and 1 or 0, respectively.



As there are two separate inputs, each of which can represent four different numbers 0, 1 2 or 3, there are 16 different distinct input combinations. We can summarize the sums expected for each possibility in a table:

top			**bottom**			**sum**			
decimal	binary		decimal	binary		decimal	binary		
	t2	t1		b2	b1		s4	s2	s1
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	1
0	0	0	2	0	0	2	0	1	0
0	0	0	3	0	0	3	0	1	1
1	0	0	0	0	0	1	0	0	1
1	0	0	1	0	0	2	0	1	0
1	0	0	2	0	0	3	0	1	1
1	0	0	3	0	0	4	1	0	0
2	0	0	0	0	0	2	0	1	0
2	0	0	1	0	0	3	0	1	1
2	0	0	2	0	0	4	1	0	0
2	0	0	3	0	0	5	1	0	1
3	0	0	0	0	0	3	0	1	1
3	0	0	1	0	0	4	1	0	0
3	0	0	2	0	0	5	1	0	1
3	0	0	3	0	0	6	1	1	0



Before the sculpture can be operated it has to be initialized if there is to be any hope of getting the correct answer. Some places - 19 in all - in the sculpture must be set up with ball bearings. All such places are marked quite clearly with two circles showing where large and small ball bearings must be located (except for one on the 3rd plank down on the right.)

There are also some elements in the circuit – 16 in all - that have a lever that needs to be "set". These are marked either "AND" or "OR", with the set position clearly indicated.

With all the initialization complete the sculpture is ready to work. Insert ball bearings for your selected top and bottom inputs. To release the ball-bearings move the knob next to the inputs to the right. Hopefully, after quite a period of time, the correct sum will appear on the bottom plank.

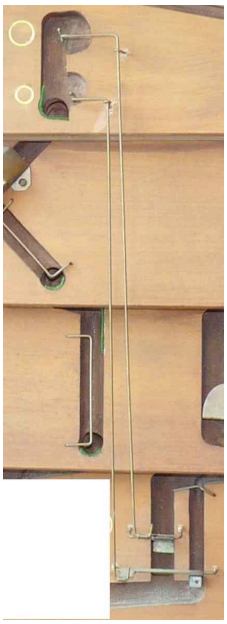
If all is OK you may try some other input numbers, but be sure to remove all ball bearings from the circuit where they are not supposed to be, initialize the 19 pairs of ball-bearings and check that all of the 16 gates are set. It is easy to miss removing one of the small ball-bearings from a hard-to-see place in the circuit – be guided by the total numbers of each size of ball-bearing – 19 for initializing plus the 4 in the full "input rack" – if you cannot initialize fully there will be a ball-bearing to be located somewhere.

How it works

The circuit elements

Because both 0 and 1 are represented by ball-bearings, every path in the circuit must eventually be traversed by only one ball bearing that represents the value that the corresponding "wire" would have in a real computer circuit. (The only

exception are the channels leading into “and” and “or” gates that are traversed by two ball-bearings, as we will explain later.) Because there are many more paths than ball bearings to start with, a large number of the circuit elements are “fan-outs” that duplicate the incoming ball bearing. Here is another example of a “fan-out:



The fan-out must pre-loaded with both a small and a large ball-bearing. The incoming bearing (of either size) rolls down the incoming slope from left to right (in this case, but some work from right to left). If the ball is a small one it falls down the narrow vertical shaft, if large it skips over the narrow shaft and falls down the wider shaft. In either case the ball hits a lever, releases a ball of the same size into the shaft on the left, and then continues to fall along the shaft to the right. In either case the affect is that there are now be two balls of the same size as the input ball, running down different paths.

The output balls are lower than the fan-out itself. In three places the sculpture needs the copy to be at a higher level. In these cases there is a modification using “rods” to release the copied ball at a higher level - as in the example alongside. In one case the same type of element is used to transfer a bit “up” in the circuit without duplication.

The real work of the circuit is done by the “and”, “or” and “not” gates. Consider the typical “and gate”. The two input balls both roll into the common single input slope that, in this example, runs down from left to right inside the wooden plank. As with the fan-out, a small ball will fall down the narrow shaft and a large ball down the wider shaft. The gate has to be set so that the first small ball to come along will fall straight through to the output giving a result of 0 – in passing the ball changes the setting so the any following small ball is trapped so that there is only one output ball.

The first large ball is trapped at the bottom of its shaft and goes nowhere. If there is a second large ball then the presence of the first ball will allow the second to

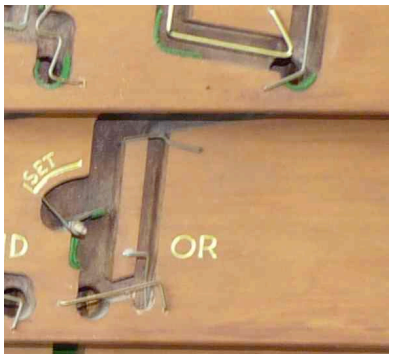
roll to the output. So, only in the case of two inputs of "1" will the output be "1". We can summarise the working of the "and gate" in a table.

and

inputs		output
0	0	0
0	1	0
1	0	0
1	1	1

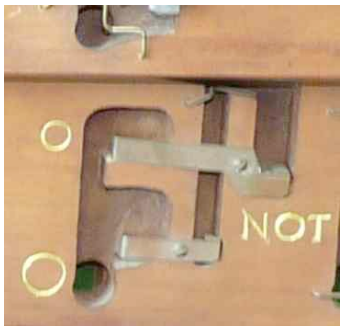
or

inputs		output
0	0	0
0	1	1
1	0	1
1	1	1



The or-gate is similar. But, in this case, the trap allows the first large ball through, catching any second large ball. The first small ball is held so that only a second small ball continues to the output. So, the only case when a small ball is output is when there are two small balls input – two 0s give a 0 output.

The not gate is like a fan-out in that it has to be pre-loaded with a ball of each size, but the input ball causes the output to be a ball of the other size. The input ball is retained.



not

input	output
0	1
1	0

The "circuit"

Considering the circuit as a whole, it is really quite straightforward. The section on the right calculates the bit s1 of the sum. This depends only on the rightmost bits t1 and b1 of the inputs. Ignoring bits b2 and t2, the table showing the sum is as on the right. The output bit is 1 only when the one of the inputs is 1, but not both. This is called an "exclusive or". We'll look into that a little later.

xor

t1	b1	s1
0	0	0
0	1	1
1	0	1
1	1	0

For now we need to note that the right side of the circuit also calculates whether it is necessary to carry 1 from the addition of t0 and b0. This only occurs when both t1 and b1 are 1. This can be obtained as the "and" of t0 and b0. Let's call this c2 (which actually represents either 2 or 0 to be added to t1 and b1.) The circuit on the right already finds this value so it is "fanned-out" but the copy is made at a higher level.

This value c2 is added to a2 and b2 to produce the bits s4 and s2 of the sum in the left hand section of the circuit. Multiple copies of c2 are needed so it immediately drops down to a fan-out, one branch of which is fanned-out again but transferred to the top.

If we look at the top left of the circuit:

we can see the three inputs t2, b2 and c2 that are to be summed. t2 and b2 are fanned out a number of times to produce multiple copies. Note the cross-overs to ensure that colliding balls do not get stuck. (Collisions are still possible at the inputs to the gates, however!)

The extreme left of the left part of the circuit is also quite simple and works out s4 which must be 1 only when at least two of t2, b2 and c2 are 1. The middle part of the circuit (the right of the left) has to find s2 and this is much more complicated. The goal is simple enough – s2 can only be true if one or three of t2, b2 and c2 are 1, as in the table. You might notice that the top four rows are just like the exclusive-or that was needed on the right hand side. In fact, what is needed is called the *exclusive or* of the three bits t2, b2 and c2.

inputs			**output**
t2	b2	c2	s2
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

More details

You can certainly test the sculpture on all inputs to show that it works correctly. More understanding probably needs a little knowledge of Boolean algebra, which is now assumed.

The right hand side, that takes the inputs t1 and b1 only, actually calculates:

(t1 **xor** b1) = ((**not** (t1 **and** b1)) **and** (t1 **or** b1)).

That is, firstly the inputs t1 and b1 are duplicated. One pair is used to find (t1 **and** b1), the other to find (t1 **or** b1). The carryout is found by duplicating (t1 **and** b1), sending one copy up to the left part of the circuit. The other copy of (t1 **and** b1) is inverted in a not-gate and its output is anded with the value (t1 **or** b1).

The left of the sculpture calculates s4 and s2 from the three inputs t2, b2 and c2. Four copies of each of the three inputs are created by fan-outs.

On the extreme left, to find s4 = (a2 **and** b2) **or** (b2 **and** c2) **or** (c2 **and** a2) the work is performed as:

s4 = (((a2 **and** b2) **or** (a2 **and** c2)) **or** (b2 **and** c2)).

The middle of the circuit, finding s2, is more complex. The right of the middle finds (b2 **xor** c2) which is transferred up to the top as input to another exclusive-or with a2 to find s2 = (a2 **xor** (b2 **xor** c2)) as required. These two central xors have the same form - the rightmost forms (b2 **and** c2), then (**not** (b2 **and** c2)) and then uses two ands and an or to find ((b2 **and** (**not** (b2 **and** c2))) **or** ((**not** (b2 **and** c2)) **and** c2)) which is (b2 **xor** c2) as required. This is then used in another xor cascade as input along with a2 to find, finally, s2 = (a2 **xor** (b2 **xor** c2)).

How realistic is the sculpture?

Clearly not! But how about the concepts involved? Certainly real computers use binary numbers, they use and, or and not gates so at this level the sculpture mirrors reality and illustrates it well. You will notice one aspect of the sculpture that is very life-like. Everything happens at once – computer circuits are extremely parallel devices. However, you will also notice that there is serial core to the circuit, with operations flowing from right to left, which constrains the speed of operation.

The circuit itself is perhaps typical of very early computers where the goal was getting it working and reducing cost rather than speed. Modern computers tend to use more circuitry so that addition can be much faster – the time to add two 64-bit binary numbers is often the same as the computer's cycle-time – a machine rated as 5 GigaHertz will complete such an addition in 200 picoseconds.

Real circuitry offers some advantages over ball-bearings. A modest fan-out of about 4 from any gate is possible without any special circuitry, the not of a circuit input is often available for no extra cost, it is possible for both and-gates and or-gates to have more than 2 inputs and it is often possible to just join wires together to make an "or" without a special or-gate being needed.

So, in modern terms the likely arrangement of a circuit would implement (writing the "wired-ors" as + and "not" as -):

Level 1:

s1 = ((t1 **and** (- b1)) + ((- t1) **and** b1))

s20 = ((t2 **and** (- b2)) + ((- t2) **and** b2))

s40 = ((t4 **and** (- b4)) + ((- t4) **and** b4))

c2 = t1 **and** b1

c4 = (t2 **and** b2) + (-t2 **and** b2 **and** t1 **and** b1) + (t2 **and** - b2 **and** t1 **and** b1)

Level 2:

s2 = ((s20 **and** (- c2)) + ((- s20) **and** c2))

s4 = ((s40 **and** (- c4)) + ((- s40) **and** c1))

The longest path in this involves only two gates as opposed to the ten, or so, in the sculpture. It would be a lot faster than the sculpture but nowhere near as spectacular!