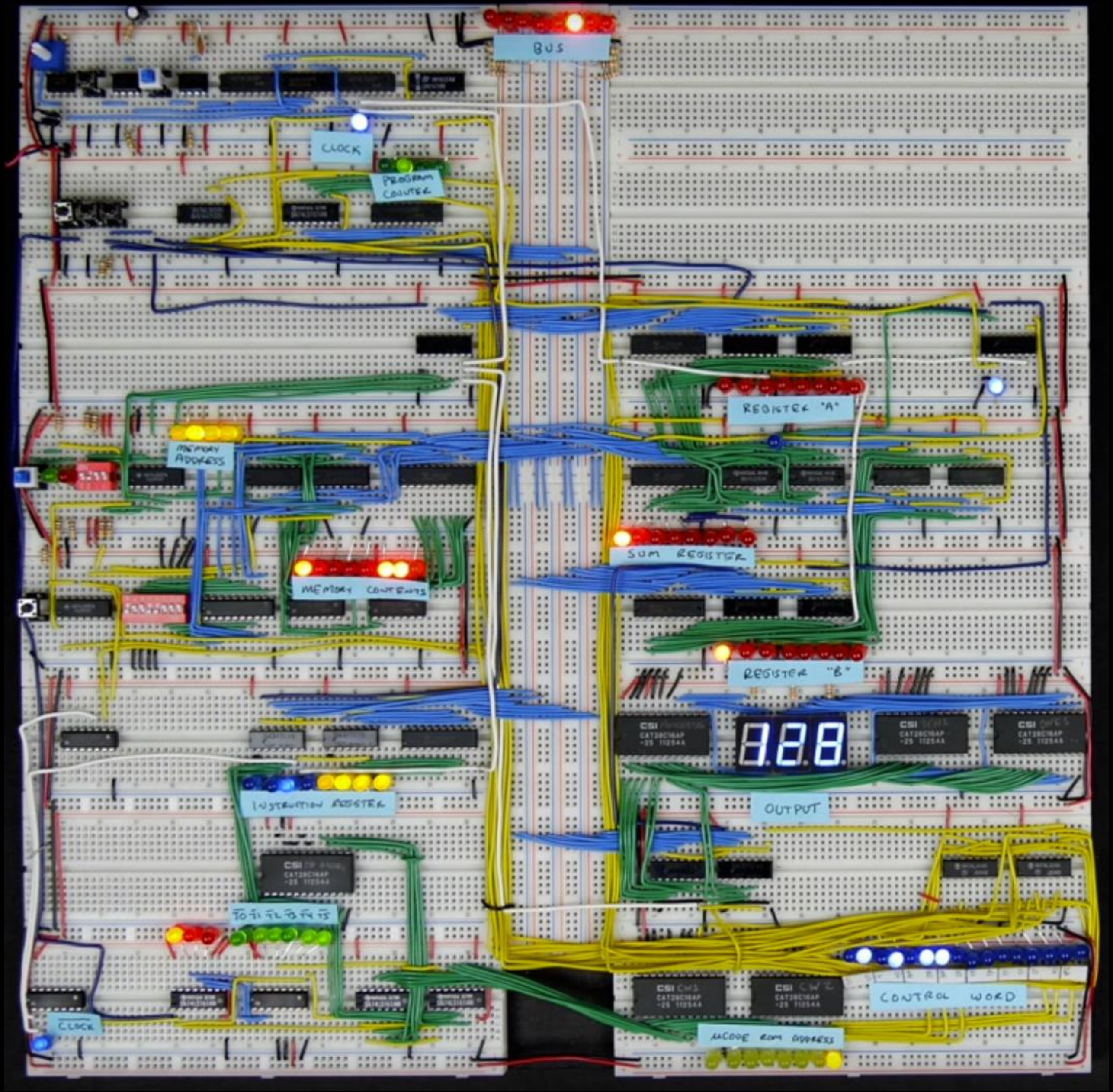
Intro to Computer Logic – Part I



CPSC 240-09
John Overton

Section Outline

- Boolean Algebra and Logic Design
- Simplification of Boolean Functions using K-maps
- Combinatorial Circuits (decoders and multiplexors)
- Sequential Circuits (Flip-Flops, latches)
- Look at two other computer architectures: The ATMega and ARM Cortex M

Binary Addition

- Starting with the LSB, add each pair of digits, include the carry if present.

	carry:					1			
	0	0	0	0	0	1	0	0	(4)
+	0	0	0	0	0	1	1	1	(7)
	0	0	0	0	1	0	1	1	(11)
bit position:	7	6	5	4	3	2	1	0	

Forming the Two's Complement

- Negative numbers are stored in two's complement notation
- Represents the additive Inverse

Starting value	00000001
Step 1: reverse the bits	11111110
Step 2: add 1 to the value from Step 1	11111110 +00000001
Sum: two’s complement representation	11111111

Binary Subtraction

- When subtracting A – B, convert B to its two's complement
- Add A to (–B)

$$\begin{array}{r} 0\,0\,0\,0\,1\,1\,0\,0 \\ -\,0\,0\,0\,0\,0\,0\,1\,1 \\ \hline \end{array} \longrightarrow \begin{array}{r} 0\,0\,0\,0\,1\,1\,0\,0 \\ +\,1\,1\,1\,1\,1\,1\,0\,1 \\ \hline 0\,0\,0\,0\,1\,0\,0\,1 \end{array}$$

Practice: Subtract 0101 from 1001.

Boolean Operations

- NOT ¬, the overbar, ′ (prime)
- AND ∧ (cap), ·
- OR ∨ (cup), +
- Operator Precedence
- Truth Tables

Boolean Algebra

- Based on symbolic logic, designed by George Boole
- Boolean expressions created from:
  - NOT, AND, OR

Expression	Description
$\neg X$	NOT X
$X \wedge Y$	X AND Y
$X \vee Y$	X OR Y
$\neg X \vee Y$	(NOT X) OR Y
$\neg(X \wedge Y)$	NOT (X AND Y)
$X \wedge \neg Y$	X AND (NOT Y)

Operator Precedence

- Examples showing the order of operations:

Expression	Order of Operations
$\neg X \vee Y$	NOT, then OR
$\neg(X \vee Y)$	OR, then NOT
$X \vee (Y \wedge Z)$	AND, then OR

Truth Tables (1 of 3)

- A Boolean function has one or more Boolean inputs, and returns a single Boolean output.
- A truth table shows all the inputs and outputs of a Boolean function

Example: $\neg X \vee Y$

X	$\neg X$	Y	$\neg X \vee Y$
F	T	F	T
F	T	T	T
T	F	F	F
T	F	T	T

Truth Tables (2 of 3)

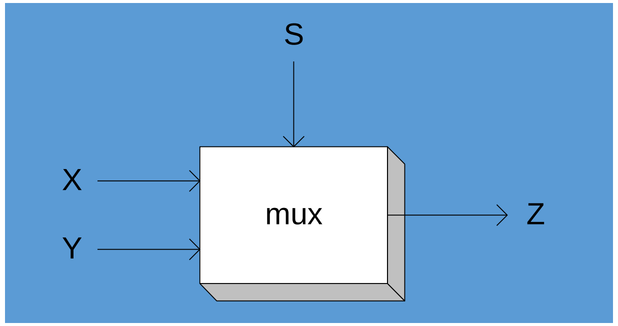
- Example: $X \land \neg Y$

X	Y	$\neg$**Y**	**X** $\land$ $\neg$**Y**
F	F	T	F
F	T	F	F
T	F	T	T
T	T	F	F

Truth Tables (3 of 3)

- Example: $(Y \wedge S) \vee (X \wedge \neg S)$

X	Y	S	$Y \wedge S$	$\neg S$	$X \wedge \neg S$	$(Y \wedge S) \vee (X \wedge \neg S)$
F	F	F	F	T	F	F
F	T	F	F	T	F	F
T	F	F	F	T	T	T
T	T	F	F	T	T	T
F	F	T	F	F	F	F
F	T	T	T	F	F	T
T	F	T	F	F	F	F
T	T	T	T	F	F	T



Two-input multiplexer

TTL Circuits

- Towards the 60's, common logic circuits were packages manufactured by Motorola, Texas Instruments, Fairchild and other manufacturers
- These devices packaged different gates in different combinations with the thought that they could be combined to create many different types of digital products (such as computers)
- These devices are still manufactured today, but not in the quantities manufactured through the 80's.
- However, they are still needed and used in many digital products (such as computers or computing systems)
- We will look at a few 5 volt TTL parts (from the 74LSnn series).
- Modern computer chips run at lower voltages (3.3 volts, 1.8 volts, etc.) There are newer TTL parts that run at lower voltages

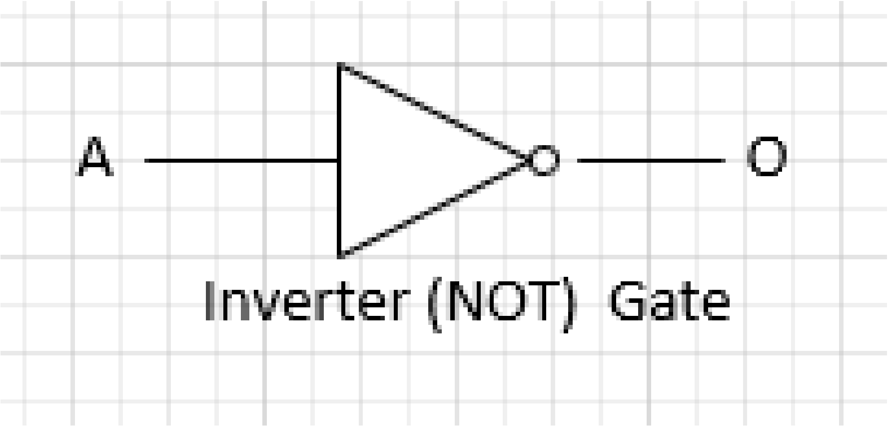
Programmable Gate Arrays

- Programmable logic devices are devices that can be programmed to take on the functionality of millions of gates – PLDs (PAL, GAL, CPLD), Field Programmable gate arrays (FPGA).
- Usually these programmable gate arrays are programmed using a language such as Verilog or VHDL
- Cell libraries and IP libraries can be purchased that will assist in creating a larger system on an FPGA chip

NOT

- Inverts (reverses) a boolean value
- Truth table for Boolean NOT operator:

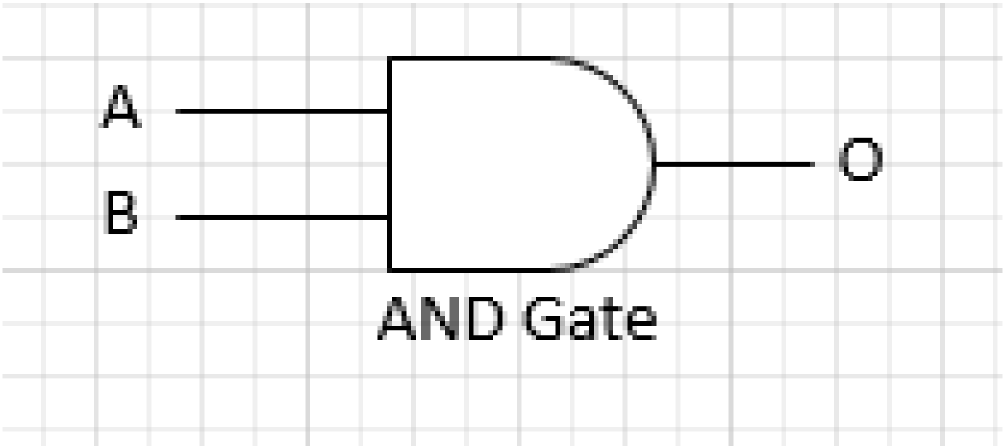
X	¬X
F	T
T	F



AND

- Truth table for Boolean AND operator:

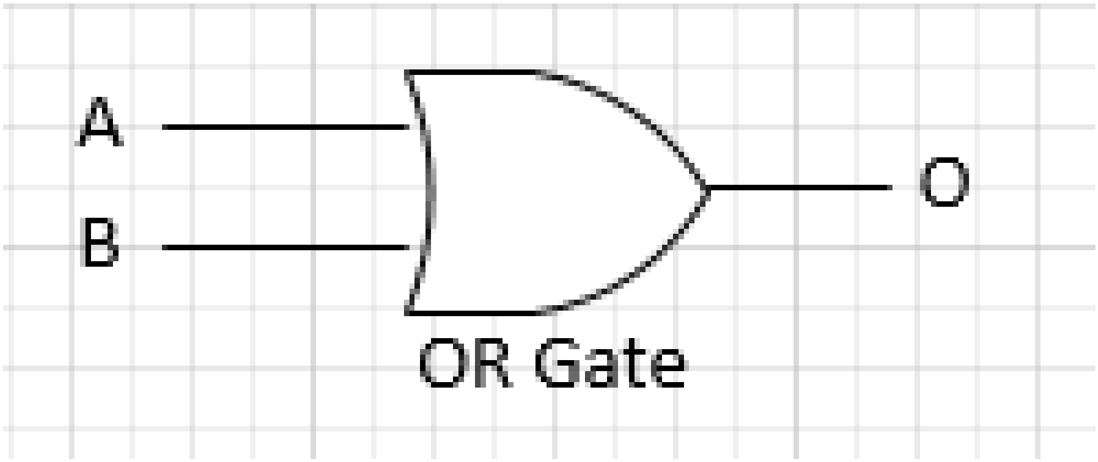
X	Y	X ∧ Y
F	F	F
F	T	F
T	F	F
T	T	T



OR

- Truth table for Boolean OR operator:

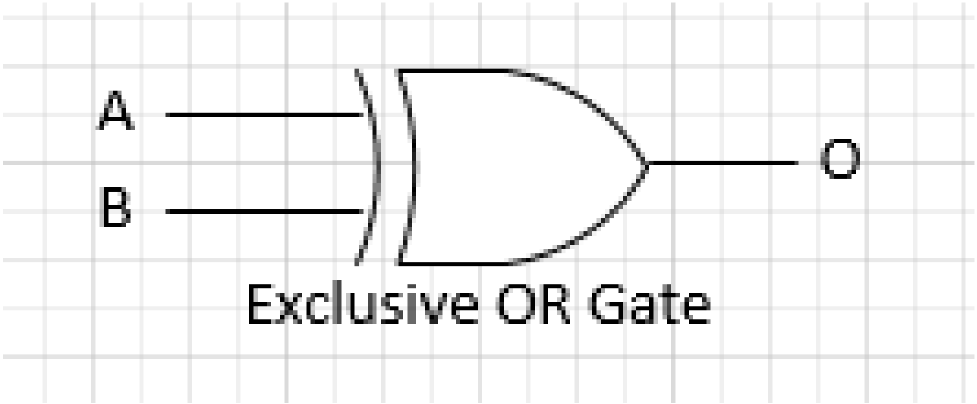
X	Y	X ∨ Y
F	F	F
F	T	T
T	F	T
T	T	T



Exclusive OR

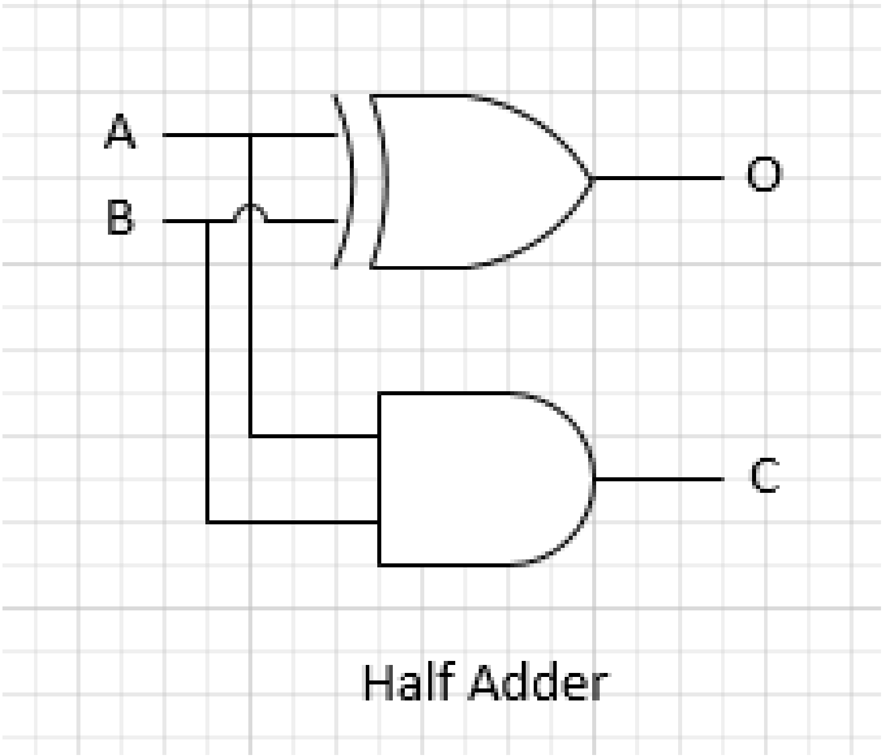
- Truth table for Boolean XOR operator:

Input		Output
A	B	O
0	0	0
0	1	1
1	0	1
1	1	0



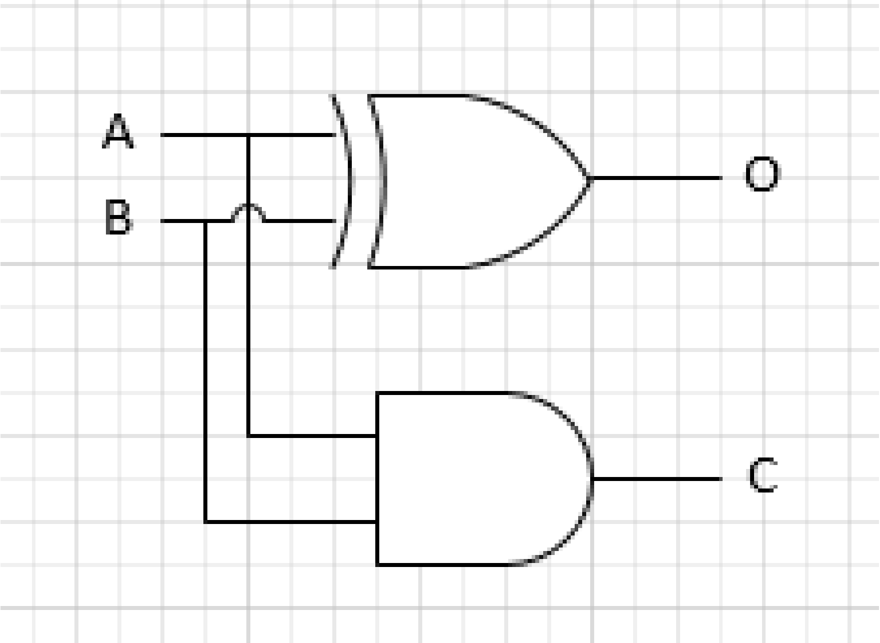
Building an Adder

Input		Output	
A	B	O	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Building an Adder

Input		Output	
A	B	O	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Half Adder

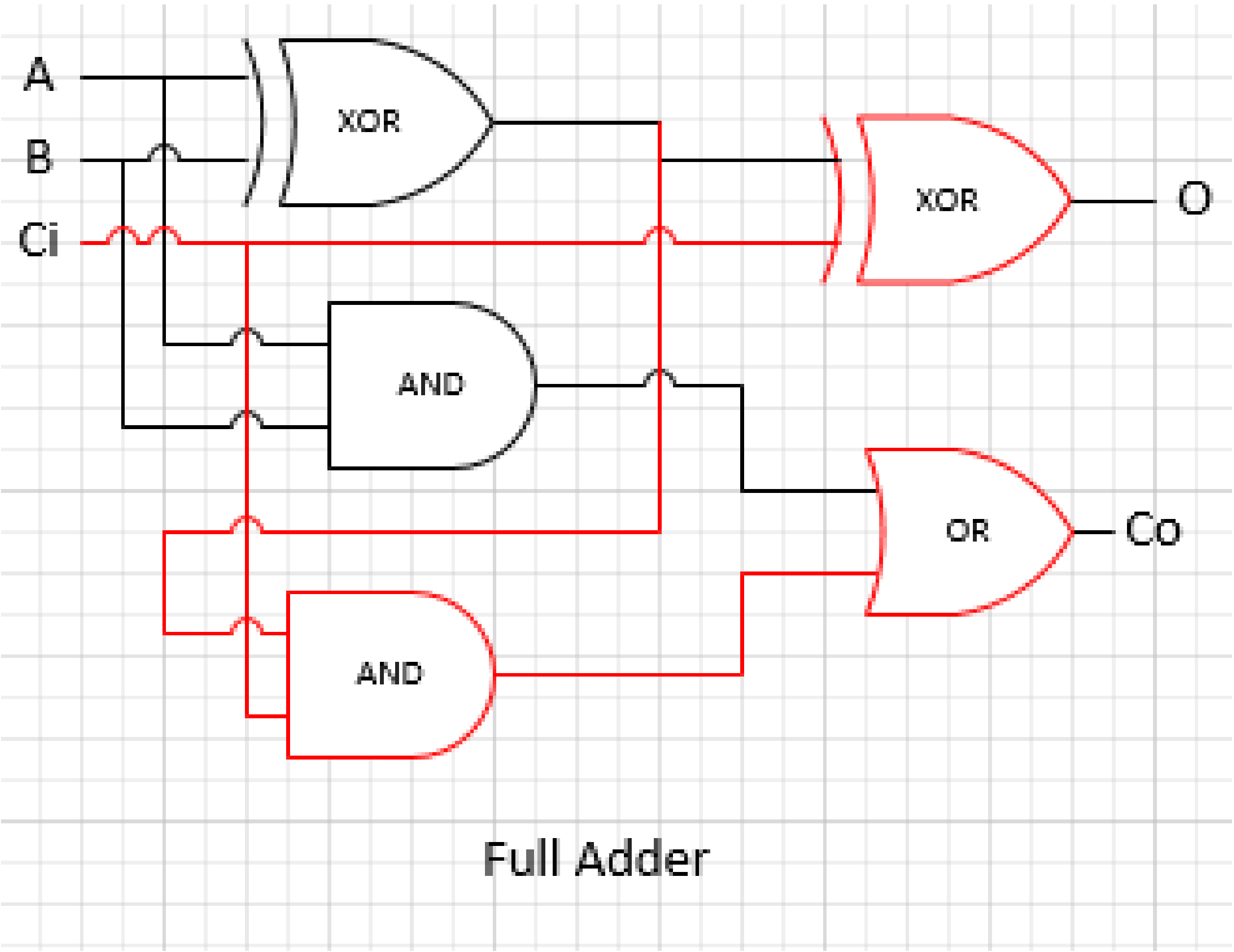
Building an Adder

	Input		Output	
Cin	A	B	O	Cout
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

If we can build a full adder for one bit, then we can chain as many together as we need (for example 8 for 8-bit addition, 32 for 32-bit addition or even 64 for 64-bit addition

Building an Adder

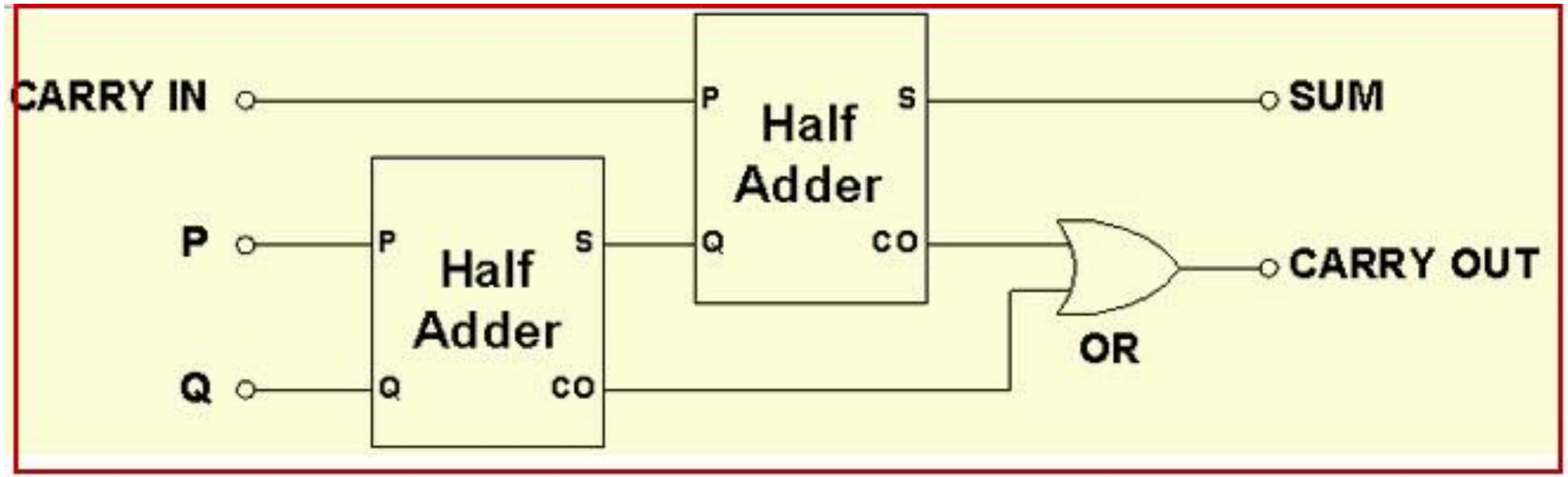
	Input		Output	
Cin	A	B	O	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



The black lines in the schematic on the left are from the original half-adder. The red lines and components are added to make a full adder.

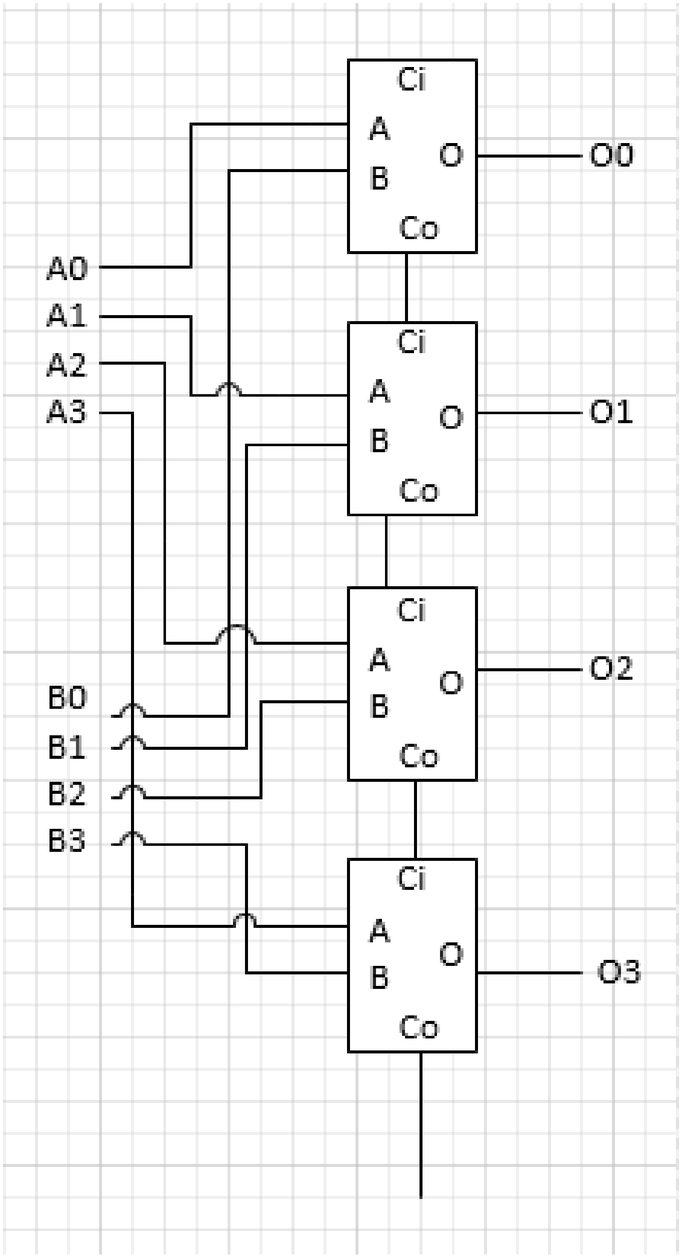
Building an Adder

- Here's another view of the full adder – it is actually two half adders put together plus an additional OR gate for Cout



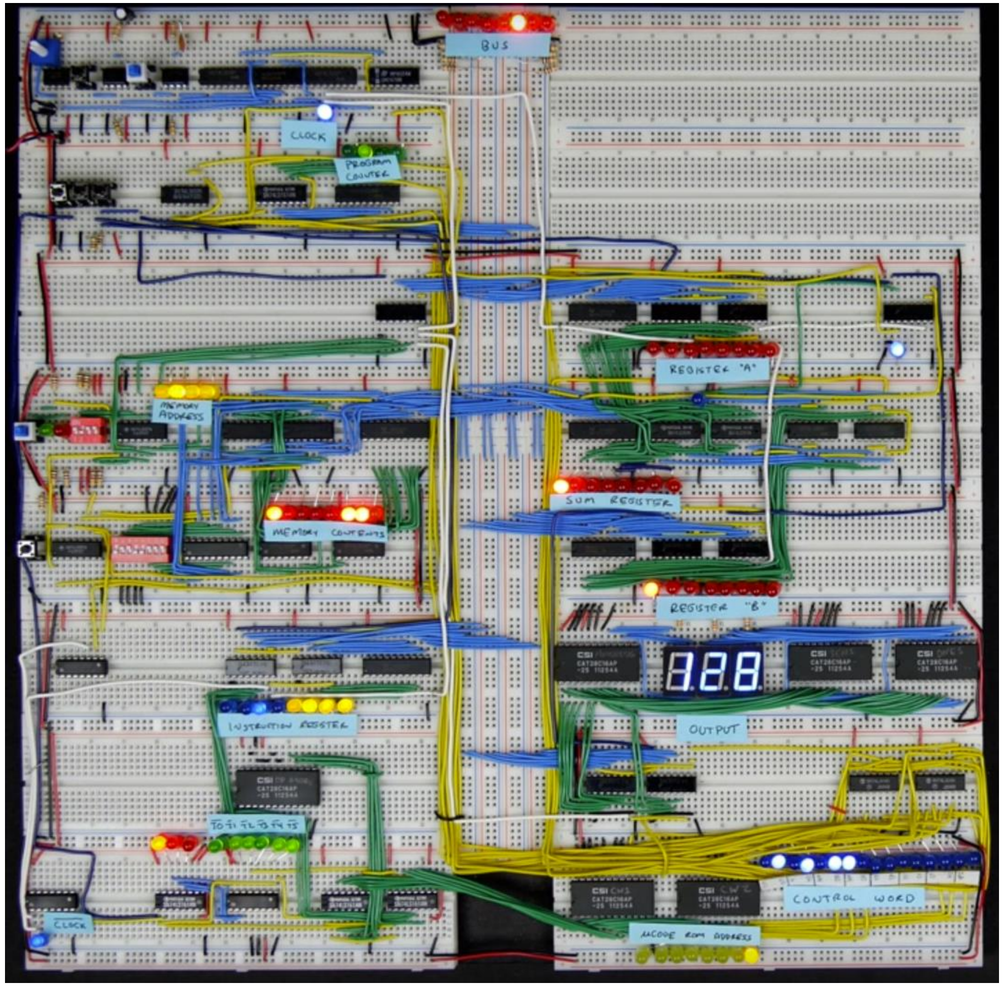
Combining 1-bit adders

- We can combine 1-bit adders to get the number of bits that we need

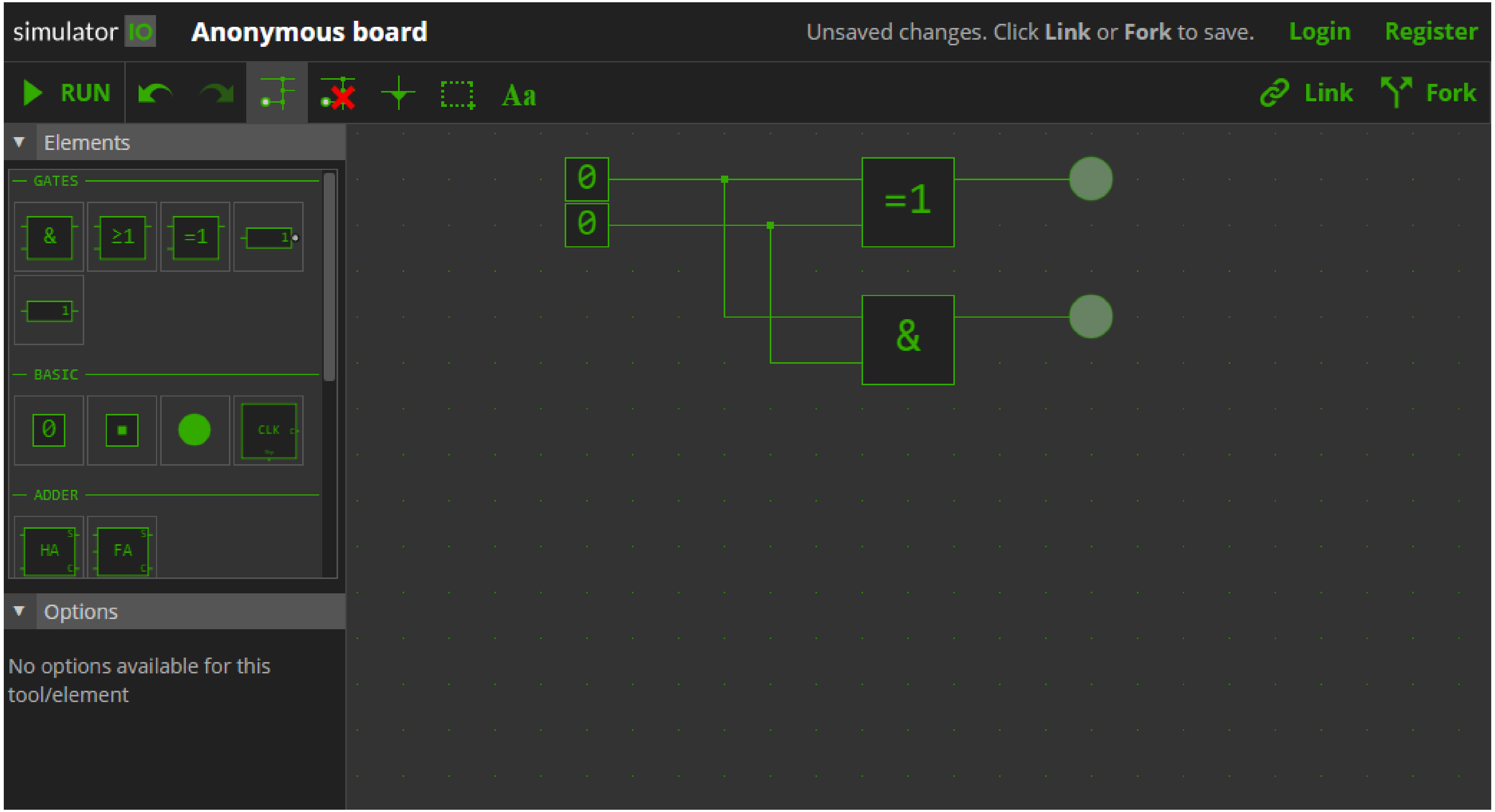


Building a computer

https://www.youtube.com/watch?v=HyznrdDSSGM&list=PLowKtXNTBypGqImE405J2565dvjafglHU



https://simulator.io/



https://circuitverse.org/simulator