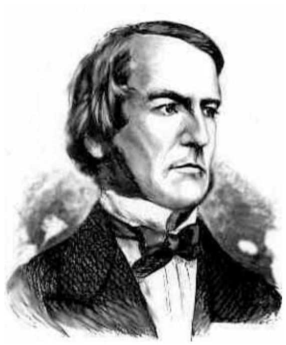
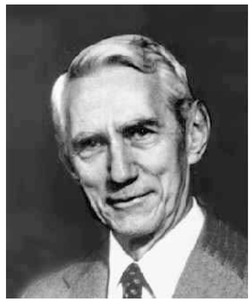
6.1 Combinational Circuits



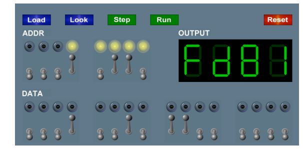
George Boole (1815 – 1864)



Claude Shannon (1916 – 2001)

Computer Architecture

Earlier lectures.
- TOY machine.



Next two lectures.
- Digital circuits.

Culminating lecture.
- Putting it all together and building a TOY machine.
  - (on paper, we mean)

Digital Circuits

What is a digital system?
- Digital: signals are 0 or 1.
- Analog: signals vary continuously.

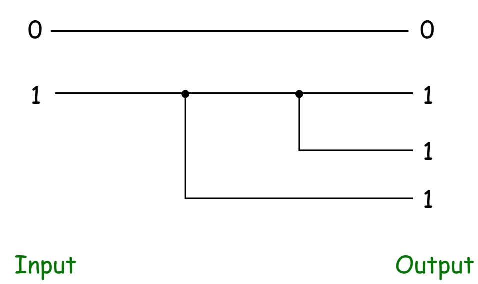
Why digital systems?
- Accuracy and reliability.
- Staggeringly fast and cheap.

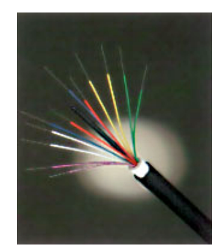
Basic abstractions.
- On, off.
- Switch that can turn something on or off.

Digital circuits and you.
- Computer microprocessors.
- Antilock brakes.
- Cell phones.
- Ipods
- . . .

Wires

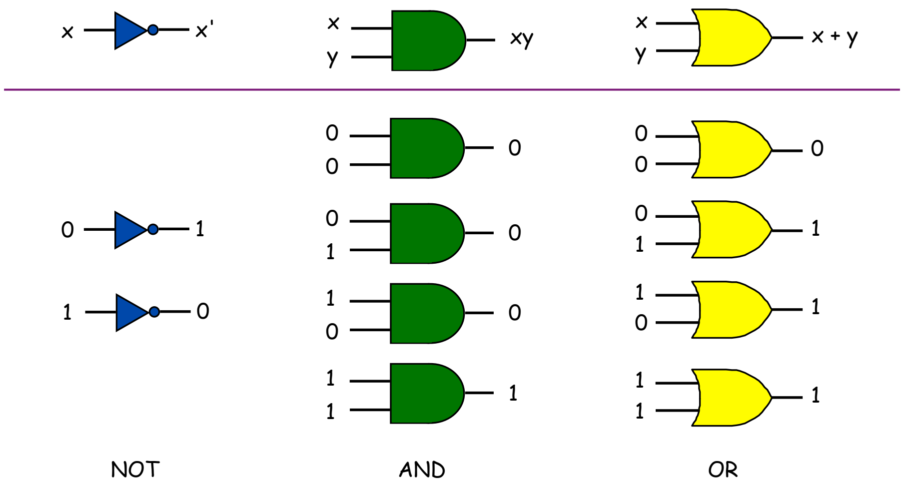
Wires.
- Propagate logical values from place to place.
- Signals "flow" from left to right.
  - A drawing convention, sometimes violated
  - Actually: flow from producer to consumer(s) of signal





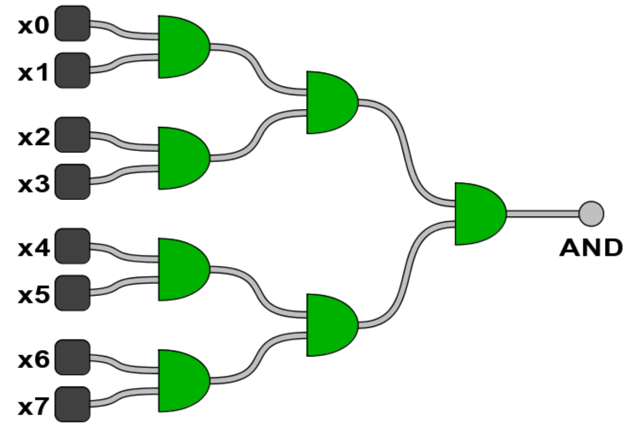
Logic Gates

Logical gates.
- Fundamental building blocks.



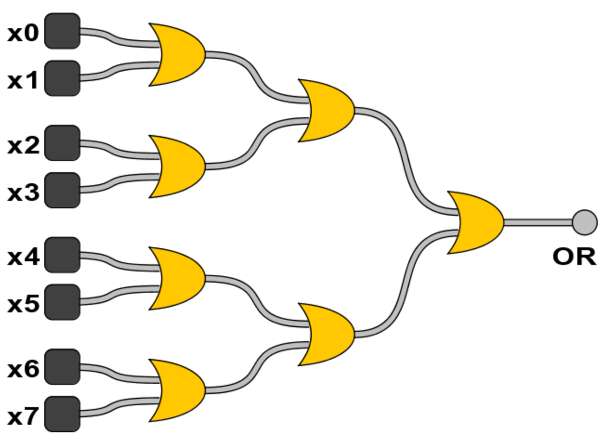
Multiway AND Gates

AND(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7).
- 1 if all inputs are 1.
- 0 otherwise.



Multiway OR Gates

OR(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7).
- 1 if at least one input is 1.
- 0 otherwise.



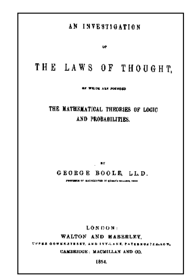
Boolean Algebra

History.
- Developed by Boole to solve mathematical logic problems (1847).
- Shannon master's thesis applied it to digital circuits (1937).

"possibly the most important, and also the most famous, master's thesis of the [20th] century" --Howard Gardner

Basics.
- Boolean variable: value is 0 or 1.
- Boolean function: function whose inputs and outputs are 0, 1.

Relationship to circuits.
- Boolean variables: signals.
- Boolean functions: circuits.


AN INVESTIGATION

THE LAWS OF THOUGHT,

THE MATHEMATICAL THEORIES OF LOGIC AND PROBABILITIES.

GEORGE BOOLE, LL.D.

LONDON:

WALTON AND MABERLY,

CAMBRIDGE: MACMILLAN AND CO.

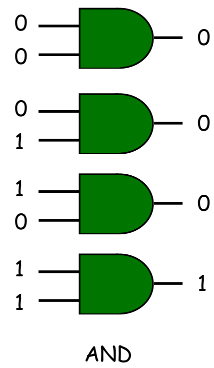
1854.

Truth Table

Truth table.

- Systematic method to describe Boolean function.
- One row for each possible input combination.
- N inputs $\Rightarrow$ 2^N rows.

AND Truth Table		
x	y	AND(x, y)
0	0	0
0	1	0
1	0	0
1	1	1



Truth Table for Functions of 2 Variables

Truth table.

- 16 Boolean functions of 2 variables.
  - every 4-bit value represents one

Truth Table for All Boolean Functions of 2 Variables									
x	y	ZERO	AND		x		y	XOR	OR
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

Truth Table for All Boolean Functions of 2 Variables									
x	y	NOR	EQ	y'		x'		NAND	ONE
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

Truth Table for Functions of 3 Variables

Truth table.

- 16 Boolean functions of 2 variables.
  - every 4-bit value represents one
- 256 Boolean functions of 3 variables.
  - every 8-bit value represents one
- 2^(2^N) Boolean functions of N variables!

Some Functions of 3 Variables						
x	y	z	AND	OR	MAJ	ODD
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	0	1
1	0	1	0	1	1	0
1	1	0	0	1	1	0
1	1	1	1	1	1	1

Universality of AND, OR, NOT

Any Boolean function can be expressed using AND, OR, NOT.

- "Universal."
- XOR(x,y) = xy' + x'y

Notation	Meaning
x'	NOT x
x y	x AND y
x + y	x OR y

Expressing XOR Using AND, OR, NOT							
x	y	x'	y'	x'y	xy'	x'y + xy'	XOR
0	0	1	1	0	0	0	0
0	1	1	0	1	0	1	1
1	0	0	1	0	1	1	1
1	1	0	0	0	0	0	0

Exercise. Show {AND, NOT}, {OR, NOT}, {NAND}, {AND, XOR} are universal.

Hint. Use DeMorgan's Law: (xy)' = (x' + y') and (x + y)' = (x'y')

Sum-of-Products

Any Boolean function can be expressed using AND, OR, NOT.

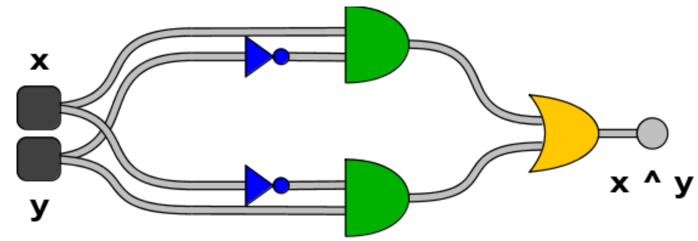
- Sum-of-products is systematic procedure.
  - form AND term for each 1 in truth table of Boolean function
  - OR terms together

Expressing MAJ Using Sum-of-Products								
x	y	z	MAJ	x'yz	xy'z	xyz'	xyz	x'yz + xy'z + xyz' + xyz
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	1
1	0	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	1
1	1	0	1	0	0	1	0	1
1	1	1	1	0	0	0	1	1

Translate Boolean Formula to Boolean Circuit

Use sum-of-products form.

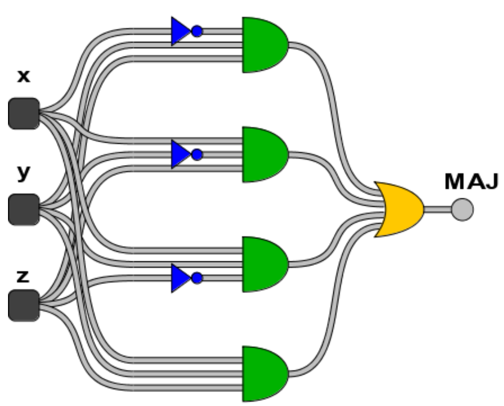
- XOR(x, y) = xy' + x'y.



Translate Boolean Formula to Boolean Circuit

Use sum-of-products form.

- MAJ(x, y, z) = x'yz + xy'z + xyz' + xyz.

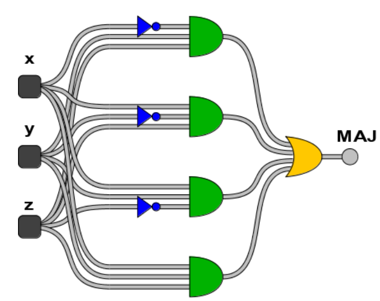


Simplification Using Boolean Algebra

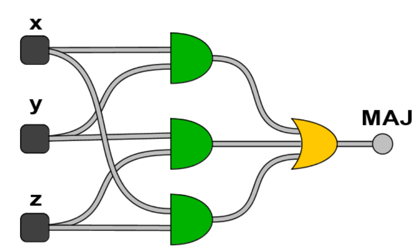
Many possible circuits for each Boolean function.

- Sum-of-products not necessarily optimal in:
  - number of gates (space)
  - depth of circuit (time)

- MAJ(x, y, z) = x'yz + xy'z + xyz' + xyz = xy + yz + xz.



size = 8, depth = 3



size = 4, depth = 2

Expressing a Boolean Function Using AND, OR, NOT

Ingredients.
- AND gates.
- OR gates.
- NOT gates.
- Wire.

Instructions.
- Step 1: represent input and output signals with Boolean variables.
- Step 2: construct truth table to carry out computation.
- Step 3: derive (simplified) Boolean expression using sum-of products.
- Step 4: transform Boolean expression into circuit.

ODD Parity Circuit

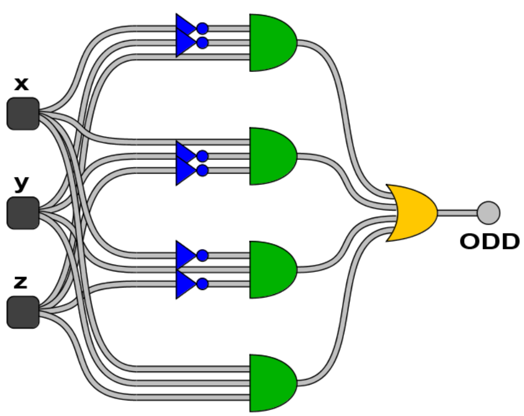
ODD(x, y, z).
- 1 if odd number of inputs are 1.
- 0 otherwise.

Expressing ODD Using Sum-of-Products

x	y	z	ODD	x'y'z	x'yz'	xy'z'	xyz	x'y'z + x'yz' + xy'z' + xyz
0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	1
0	1	0	1	0	1	0	0	1
0	1	1	0	0	0	0	0	0
1	0	0	1	0	0	1	0	1
1	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
1	1	1	1	0	0	0	1	1

ODD Parity Circuit

ODD(x, y, z).
- 1 if odd number of inputs are 1.
- 0 otherwise.



Let's Make an Adder Circuit

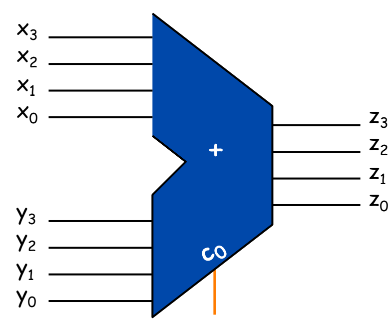
Goal: x + y = z for 4-bit integers.
- We build 4-bit adder: 9 inputs, 4 outputs.
- Same idea scales to 128-bit adder.
- Key computer component.

	1	1	1	0
	2	4	8	7
+	3	5	7	9
	6	0	6	6

Step 1.
- Represent input and output in binary.

	1	1	0	0
	0	0	1	0
+	0	1	1	1
	1	0	0	1

	x_3	x_2	x_1	x_0
+	y_3	y_2	y_1	y_0
	z_3	z_2	z_1	z_0



Let's Make an Adder Circuit

Goal: $x + y = z$ for 4-bit integers.

Step 2. (first attempt)
- Build truth table.
- Why is this a bad idea?
  - 128-bit adder: 2^{256+1} rows > # electrons in universe!

				c_0
	x_3	x_2	x_1	x_0
+	y_3	y_2	y_1	y_0
	z_3	z_2	z_1	z_0

4-Bit Adder Truth Table												
c_0	x_3	x_2	x_1	x_0	y_3	y_2	y_1	y_0	z_3	z_2	z_1	z_0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	1	1	0	0	1	1
0	0	0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	0	1	0	1	0	1	0	1
.	.	.	.	.	.	.	.	.	.	.	.	.
1	1	1	1	1	1	1	1	1	1	1	1	1

$2^{8+1} = 512$ rows!

Let's Make an Adder Circuit

Goal: $x + y = z$ for 4-bit integers.

Step 2. (do one bit at a time)
- Build truth table for carry bit.
- Build truth table for summand bit.

	c_3	c_2	c_1	$c_0 = 0$
	x_3	x_2	x_1	x_0
+	y_3	y_2	y_1	y_0
	z_3	z_2	z_1	z_0

Carry Bit			
x_i	y_i	c_i	c_{i+1}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Summand Bit			
x_i	y_i	c_i	z_i
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

Let's Make an Adder Circuit

Goal: $x + y = z$ for 4-bit integers.

Step 3.
- Derive (simplified) Boolean expression.

	c_3	c_2	c_1	$c_0 = 0$
	x_3	x_2	x_1	x_0
+	y_3	y_2	y_1	y_0
	z_3	z_2	z_1	z_0

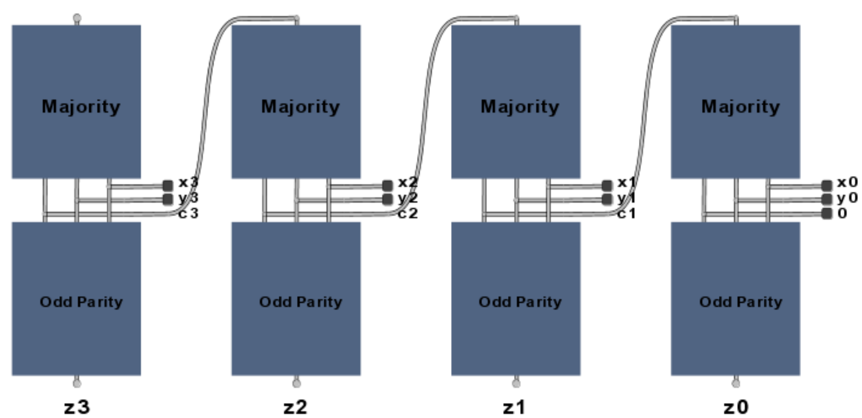
Carry Bit				
x_i	y_i	c_i	c_{i+1}	MAJ
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Summand Bit				
x_i	y_i	c_i	z_i	ODD
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Let's Make an Adder Circuit

Goal: $x + y = z$ for 4-bit integers.

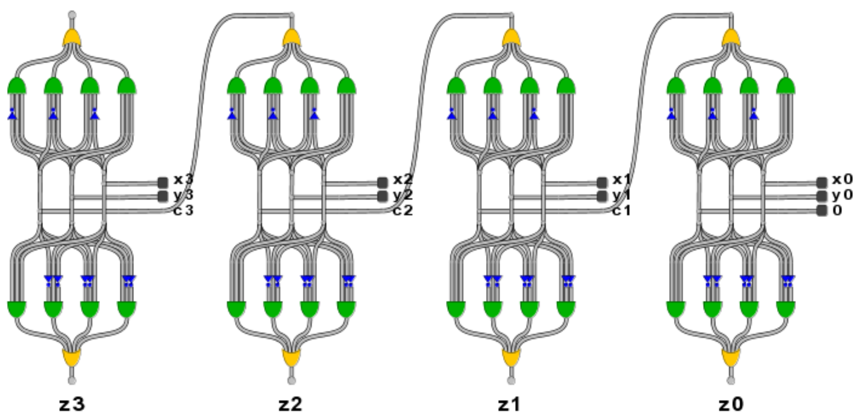
Step 4.
- Transform Boolean expression into circuit.
- Chain together 1-bit adders.



Let's Make an Adder Circuit

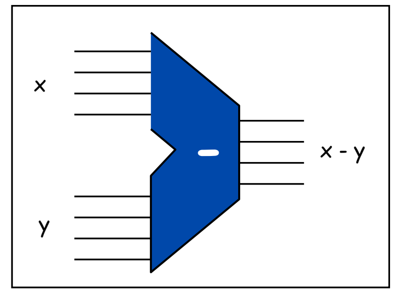
Goal: x + y = z for 4-bit integers.

Step 4.
- Transform Boolean expression into circuit.
- Chain together 1-bit adders.

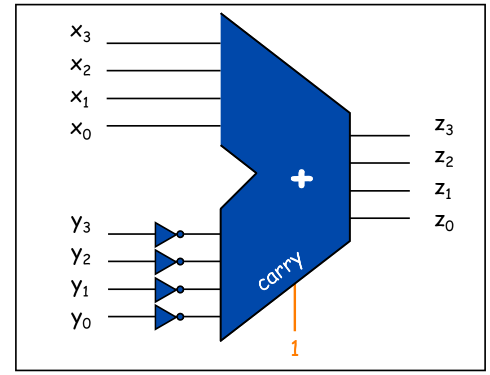


Subtractor

Subtractor circuit: z = x - y.
- One approach: design like adder circuit.
- Better idea: reuse adder circuit.
  - 2's complement: to negate an integer, flip bits, then add 1



4-Bit Subtractor Interface

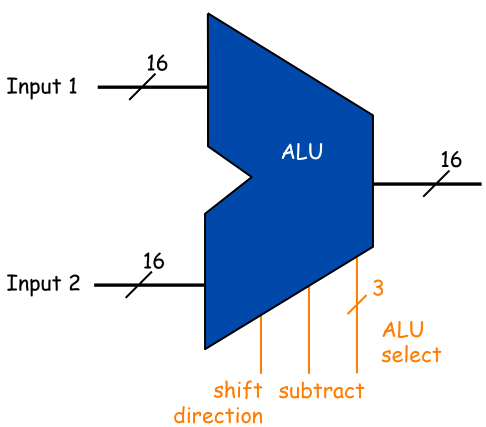


4-Bit Subtractor Implementation

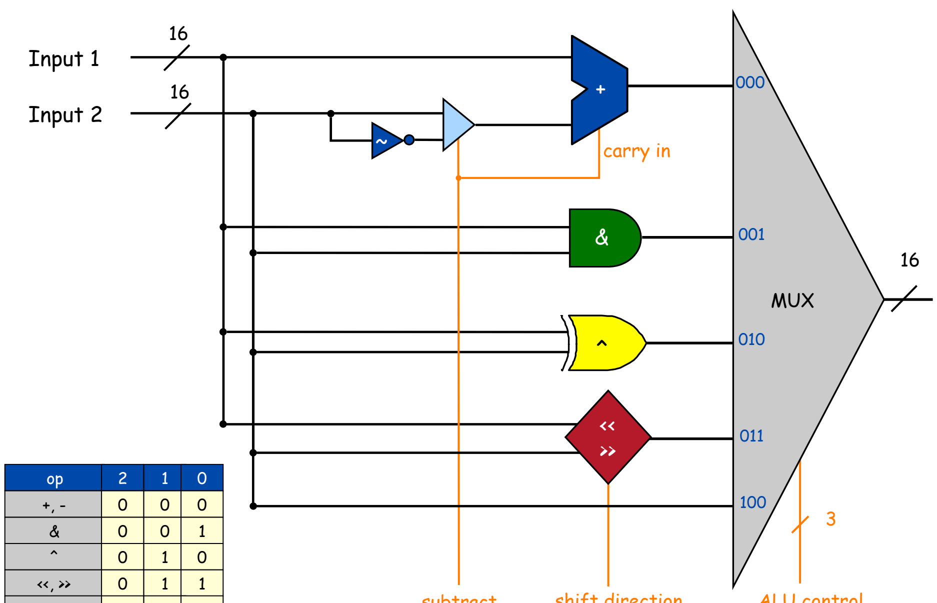
Arithmetic Logic Unit: Interface

ALU Interface.
- Add, subtract, bitwise and, bitwise xor, shift left, shift right, copy.
- Associate 3-bit integer with 5 primary ALU operations.
  - ALU performs operations in parallel
  - control wires select which result ALU outputs

op	2	1	0
+, -	0	0	0
&	0	0	1
^	0	1	0
<<, >>	0	1	1
input 2	1	0	0



Arithmetic Logic Unit: Implementation



op	2	1	0
+, -	0	0	0
&	0	0	1
^	0	1	0
<<, >>	0	1	1
input 2	1	0	0

Summary

Lessons for software design apply to hardware design!
- Interface describes behavior of circuit.
- Implementation gives details of how to build it.

Layers of abstraction apply with a vengeance!
- On/off.
- Controlled switch (transistor).
- Gates (AND, OR, NOT).
- Boolean circuit (MAJ, ODD).
- Adder.
- . . .
- Arithmetic logic unit.
- . . .
- TOY machine.