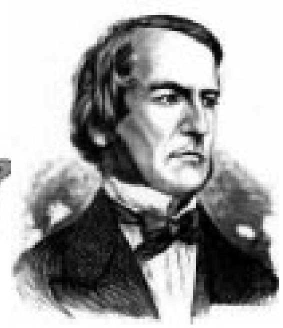
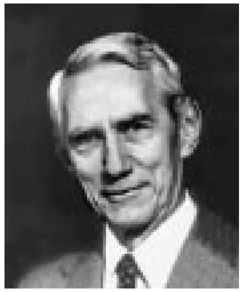
Lecture A3: Boolean Circuits



George Boole
(1815 – 1864)



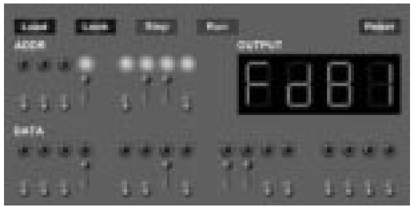
Claude Shannon
(1916 – 2001)

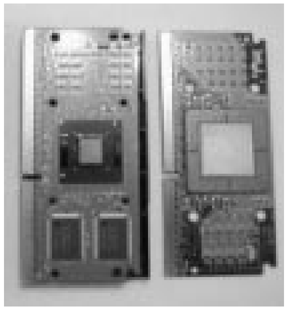
Computer Architecture

Lecture A1 – A2.
- TOY machine.

Lecture A3 – A4.
- Digital circuits.

Lecture A5.
- Putting it all together.





Digital Circuits

What is a digital system?

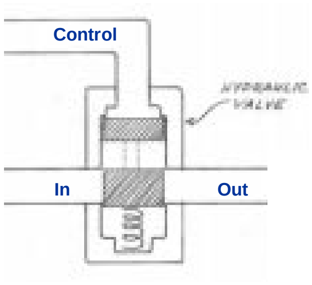
Why digital systems?

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

Digital circuits and you.
- Computer microprocessors.
- Antilock brakes.
- VCR.
- Cell phone.

Abstract Switch

Abstract switch.
- Electrical (transistor).
- Electro-mechanical (relay).
- Hydraulic (water valve).

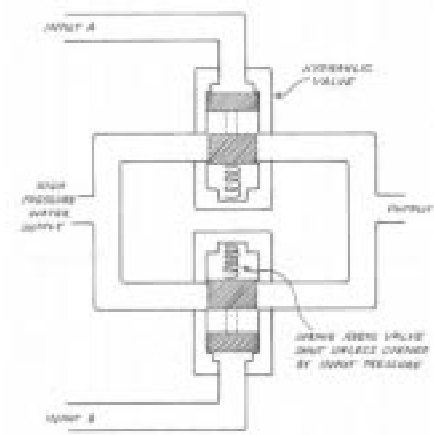


Hydraulic valve.
- 3 connections: input, output, control.
- Pressure on control pushes on a piston that turns on water flow from input to output.
  - valve is always entirely open or closed (digital)
  - amplification, restoring logic design
- Control pipe affects output pipe, but output does not affect control.
  - establishes forward flow of information over time

Computer

Hydraulic computer. (The Pattern on the Stone)

- Build computer by connecting hydraulic valves and pipe.



Hydraulic OR Gate

Computer

Electrical computer.

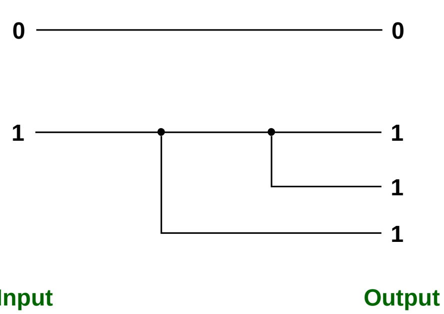
- Same fundamental abstraction.

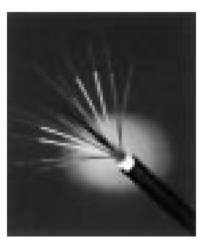
	Hydraulic Computer	Electrical Computer
Connector	Pipe	Wire
Signal	Water pressure	Voltage
Switch	Hydraulic valve	Metal-oxide transistor

Wires

Wires.

- Propagate logical values from place to place.
- Signals "flow" from left to right and top to bottom.

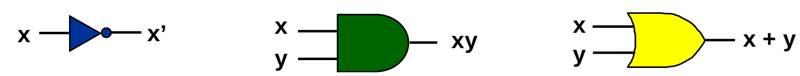


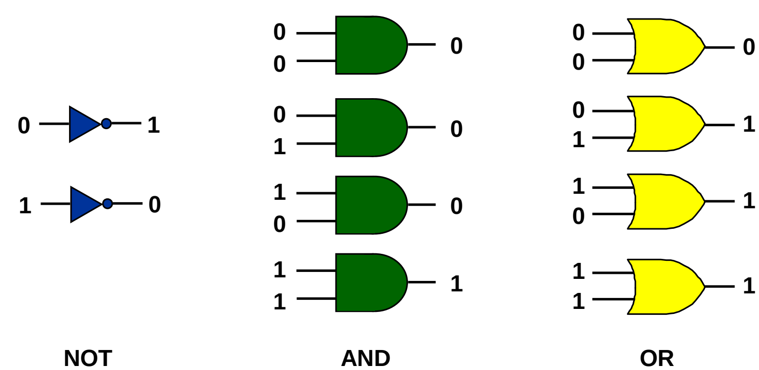


Logic Gates

Logical gates.

- Fundamental building blocks.

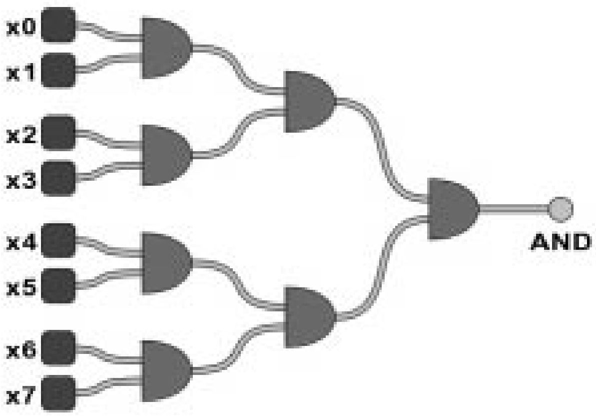




Multiway AND Gates

$AND(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7)$.

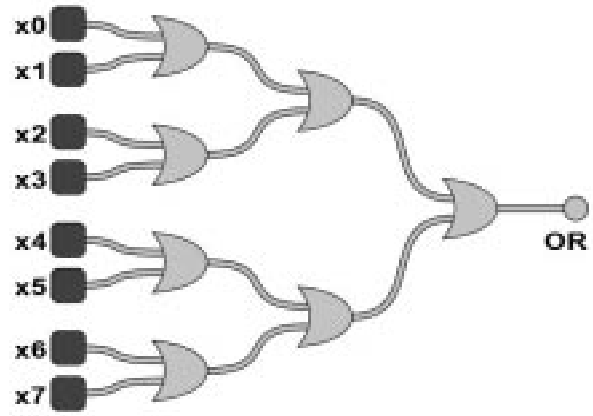
- **1 if all inputs are 1.**
- **0 otherwise.**



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.

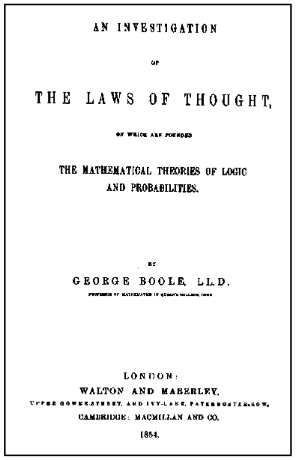
- **Developed by Boole to solve mathematical logic problems (1847).**
- **Shannon first applied to digital circuits (1937).**

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

OF

THE LAWS OF THOUGHT,

THE MATHEMATICAL THEORIES OF LOGIC AND PROBABILITIES.

BY

GEORGE BOOLE, LL.D.

LONDON:

WALTON AND MABERLEY,

CAMBRIDGE: MACMILLAN AND CO.

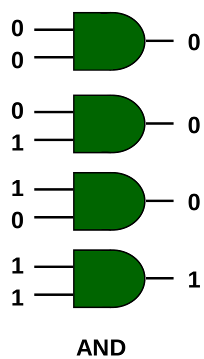
1854.

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



Truth Table

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

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	MUX
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	1	0	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	0
1	1	0	0	1	1	0	1
1	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

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: {AND, NOT}, {OR, NOT}, {NAND}, {AND, XOR} are universal.

Sum-of-Products

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

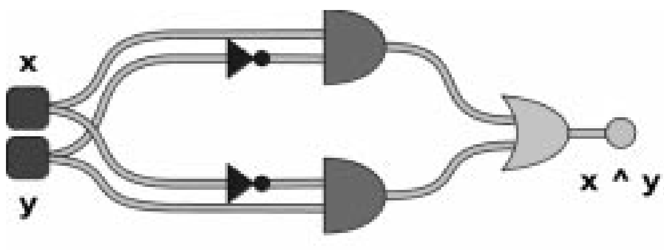
- Sum-of-products is systematic procedure.
  - form AND term for each 1 in Boolean function
  - OR terms together

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.

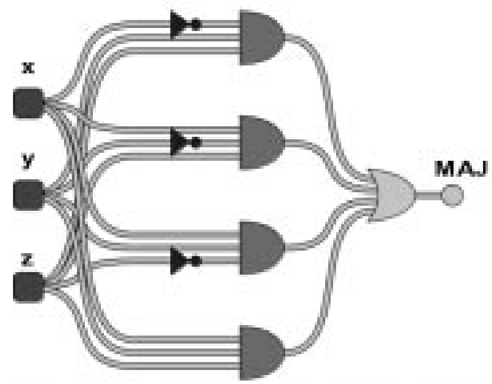
- XOR(x, y) = xy' + x'y.



Translate Boolean Formula to Boolean Circuit

Use sum-of-products form.

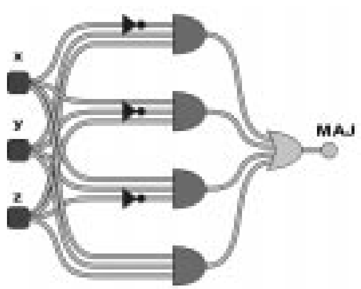
- MAJ(x, y, z) = x'yz + xy'z + xyz' + xyz.



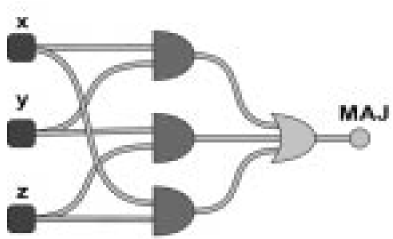
Simplification Using Boolean Algebra

Many possible circuits for each Boolean function.

- Sum-of-products not necessarily optimal in:
- MAJ(x, y, z) = x'yz + xy'z + xyz' + xyz = xy + yz + xz.



size = 8, depth = 4



size = 4, depth = 3

Recipe for Making Combinational Circuit

Ingredients.

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

White Chocolate Mousse Cake, Balducci's

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

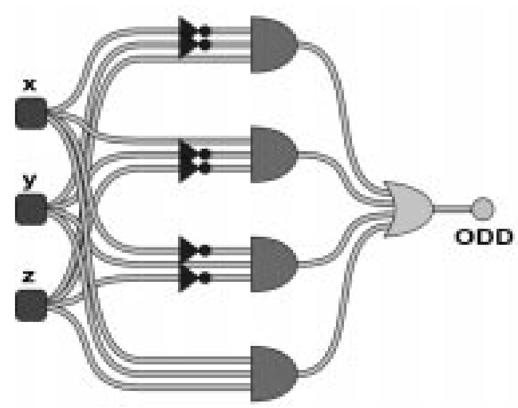
ODD(x, y, z).
- 1 if odd number of inputs are 1.
- 0 otherwise.

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

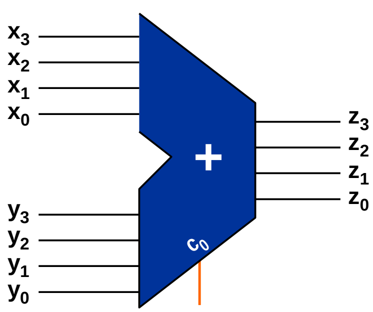
ODD(x, y, z).
- 1 if odd number of inputs are 1.
- 0 otherwise.



Let's Make an Adder Circuit

Goal: x + y = z.
- We build 4-bit adder: 8 inputs, 4 outputs. (Same idea scales to 128-bit adder.)
- Key computer component.

Step 1.
- Represent input and output in binary.



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

	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.

Step 2.
- Build truth table.
- Why is this a bad idea?

				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: $x + y = z$.

	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

Step 2.
- Build truth table for carry bit.
- Build truth table for summand bit.

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: $x + y = z$.

	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

Step 3.
- Derive (simplified) Boolean expression.

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

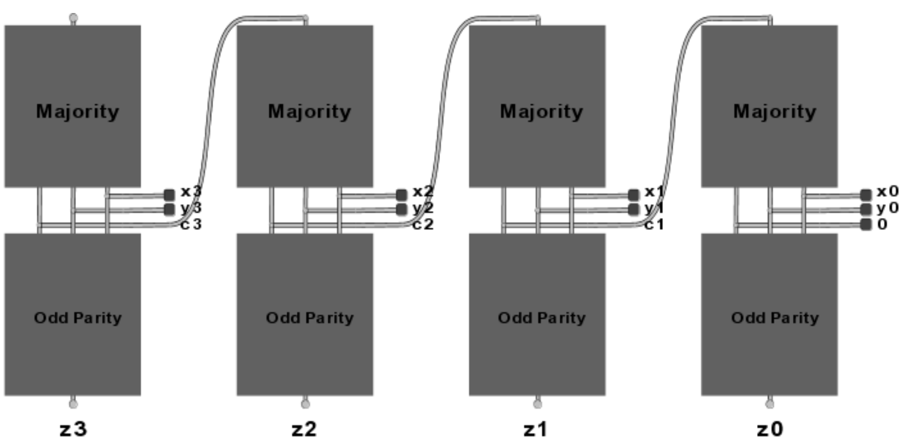
Summand Bit

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: $x + y = z$.

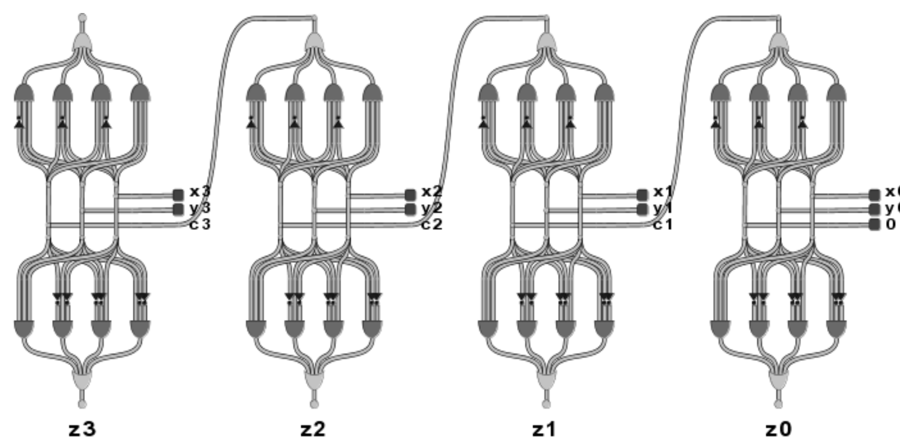
Step 4.
- Transform Boolean expression into circuit.



Let's Make an Adder Circuit

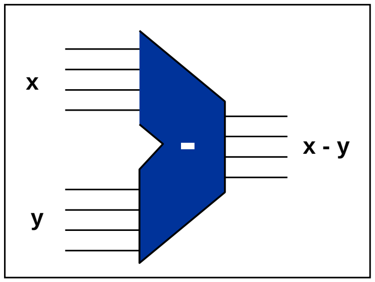
Goal: $x + y = z$.

Step 4.
- Transform Boolean expression into circuit.

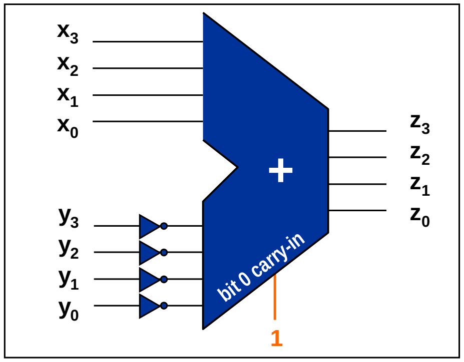


Subtractor

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



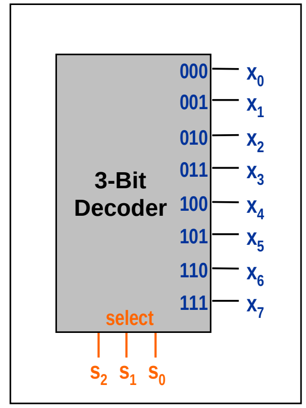
4-Bit Subtractor Interface



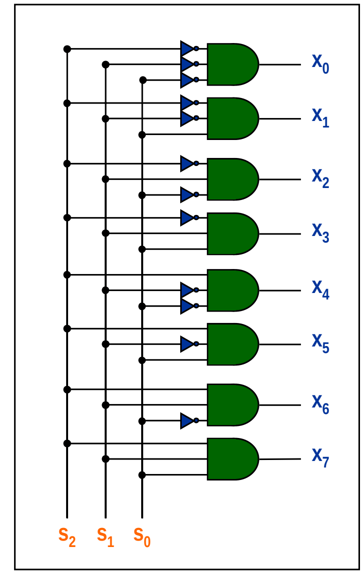
4-Bit Subtractor Implementation

N-Bit Decoder

N-bit decoder.
- N address inputs, 2^N data outputs.
- Addressed output bit is 1; all others are 0.



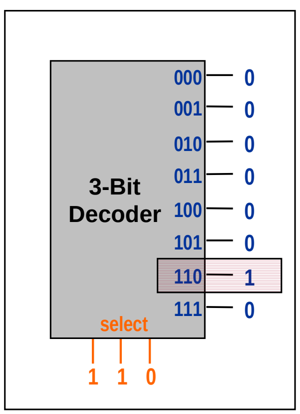
3-Bit Decoder Interface



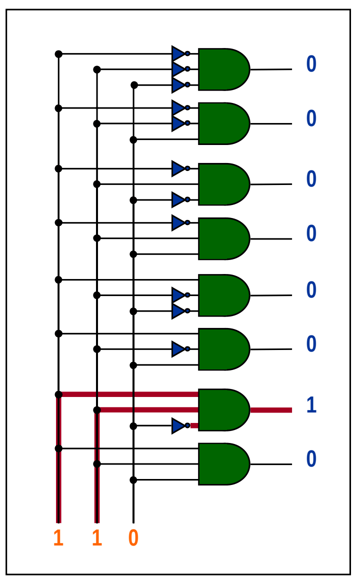
3-Bit Decoder Implementation

N-Bit Decoder

N-bit decoder.
- N address inputs, 2^N data outputs.
- Addressed output bit is 1; all others are 0.



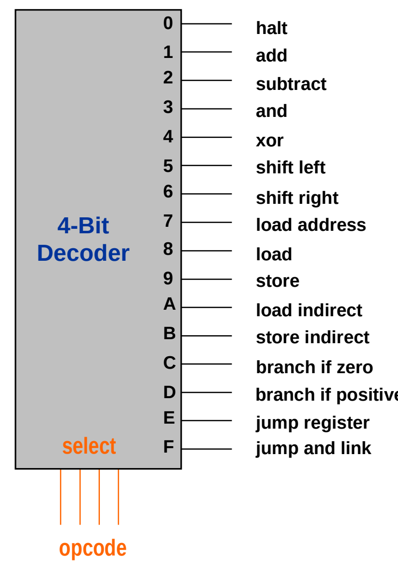
3-Bit Decoder Interface



3-Bit Decoder Implementation

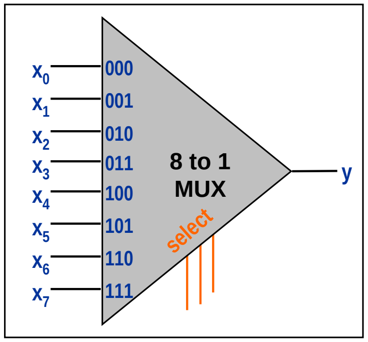
N-Bit Decoder

Application.
- Convert from binary to "unary."
- Decode opcode into instruction type.

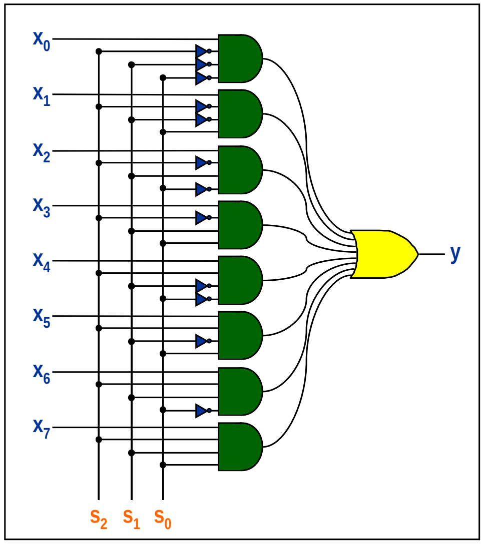


8-to-1 Multiplexer

2^N-to-1 multiplexer.
- N select inputs, 2^N data inputs, 1 output.
- Copies "selected" data input bit to output.



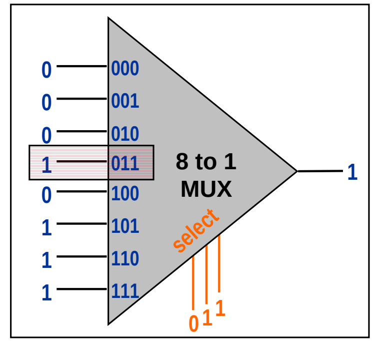
8-to-1 Mux Interface



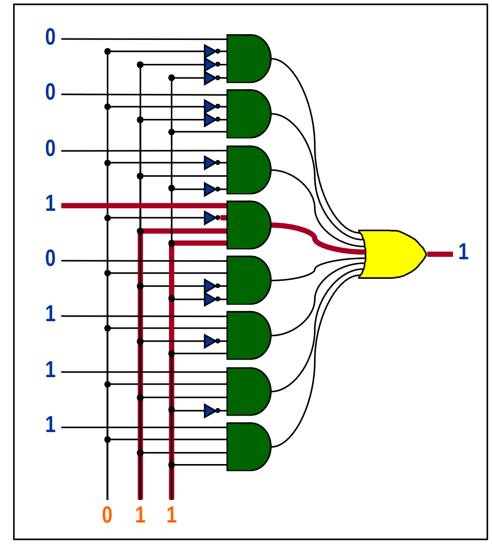
8-to-1 Mux Implementation

8-to-1 Multiplexer

2^N-to-1 multiplexer.
- N select inputs, 2^N data inputs, 1 output.
- Copies "selected" data input bit to output.



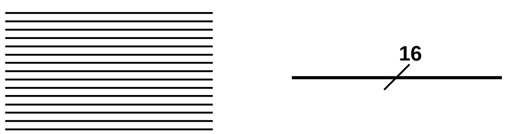
8-to-1 Mux Interface



8-to-1 Mux Implementation

Bus

16-bit bus.
- Bundle of 16 wires.
- Memory transfer, register transfer.

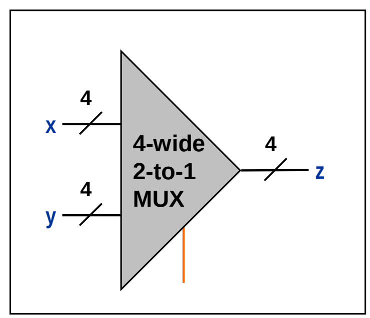


8-bit bus.
- Bundle of 8 wires.
- TOY memory address.

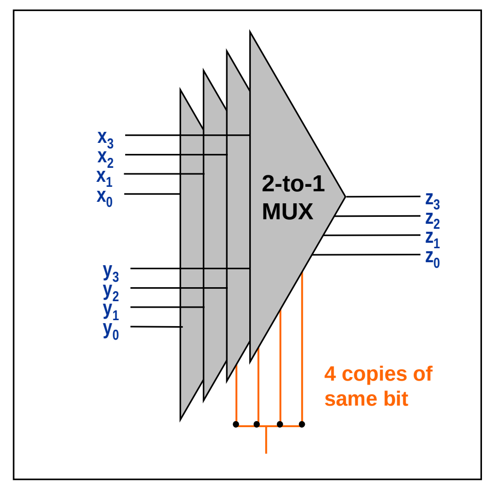
4-bit bus.
- Bundle of 4 wires.
- TOY register address.

4-Wide 2-to-1 Multiplexer

Goal: select from one of two 4-bit buses.
- Implement by layering 4 2-to-1 multiplexers.



Interface

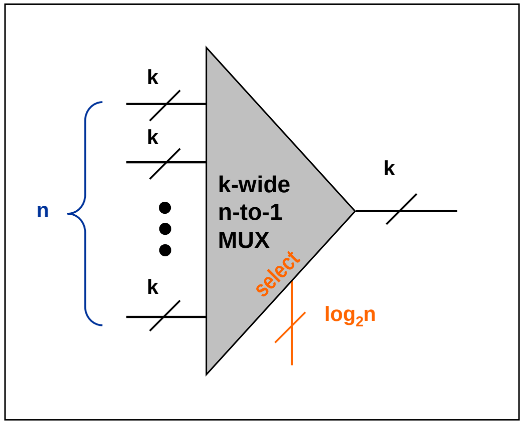


Implementation

k-Wide n-to-1 Multiplexer

Goal: select from one of n k-bit buses.

- Implement by layering k n-bit muxes.



Interface

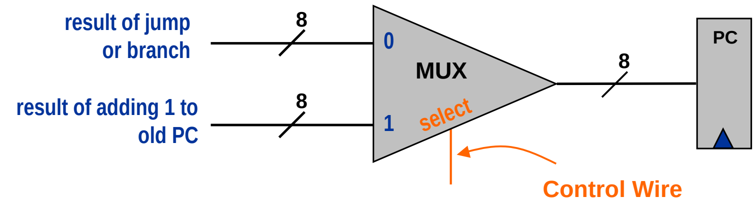
Multiplexer: Application

Program counter

- Result of jump or branch instruction.
- Adding 1 to old program counter.

Use 8-wide 2-to-1 mux to route appropriate 8-bit address to PC.

- Unspecified detail: how to set control wire?

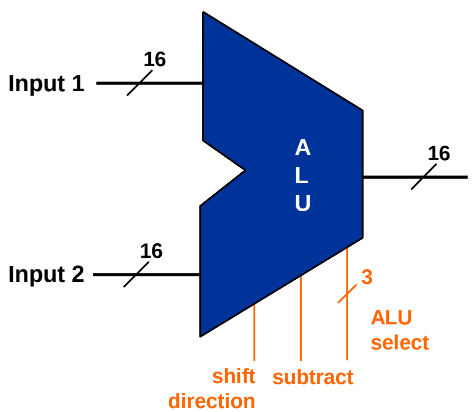


Arithmetic Logic Unit: Interface

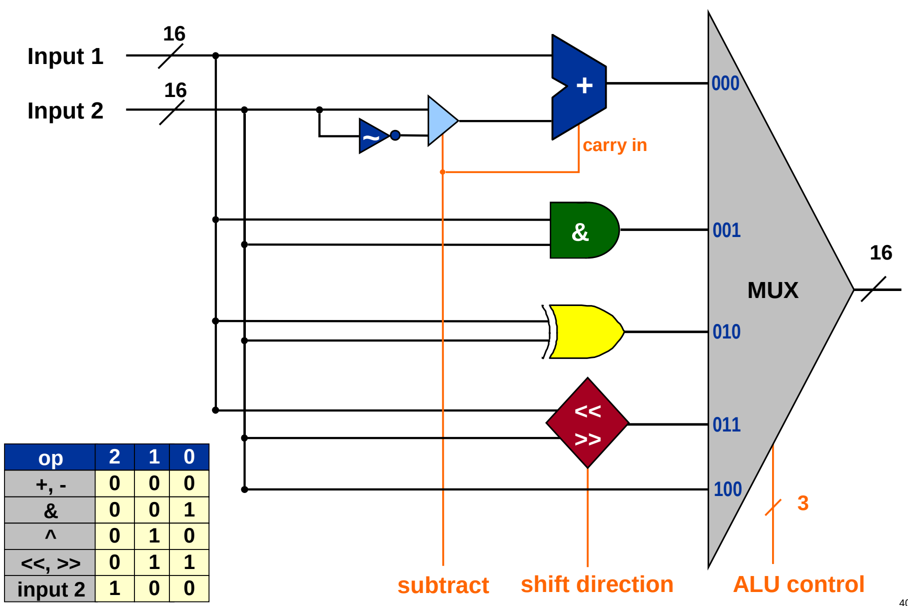
ALU Interface.

- 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

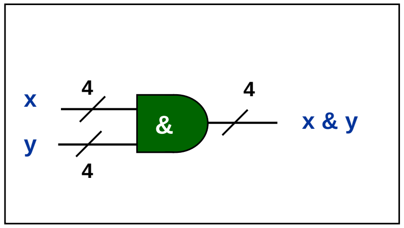


op	2	1	0
+, -	0	0	0
&	0	0	1
^	0	1	0
<<, >>	0	1	1
input 2	1	0	0

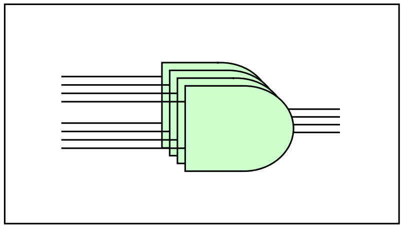
Bitwise AND, XOR, NOT

Bitwise logical operations.

- Inputs x and y: n-bits each.
- Output z: n-bits.
- Apply logical operation to each corresponding pair of bits.



Bitwise And Interface



Bitwise And Implementation

Abstraction and Encapsulation

Lessons for ADT apply to hardware!

- Interface describes behavior of circuit.
- Implementation gives details of how to build it.

Layers of abstraction apply with a vengeance!

- TOY ALU is made of:
  - multiplexer, which is made of:
  - adder, which is made of:
  - and some other things also made of gates
- TOY ALU will itself be a component of TOY computer (Lecture A5).