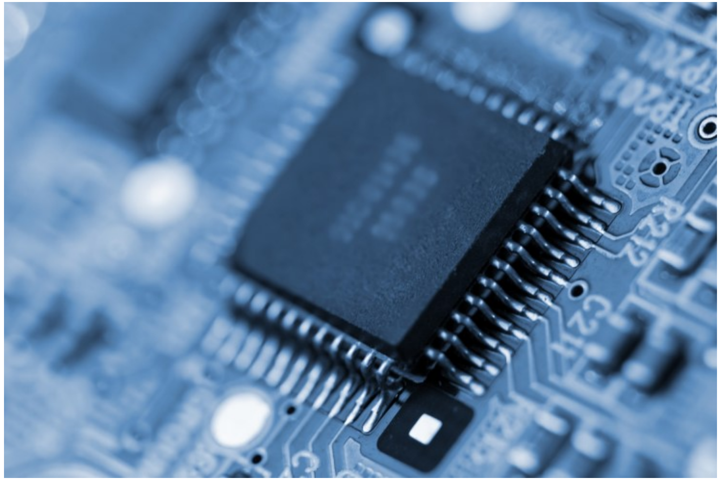
UNIVERSITY OF LEEDS

Computer Processors

Boolean Logic

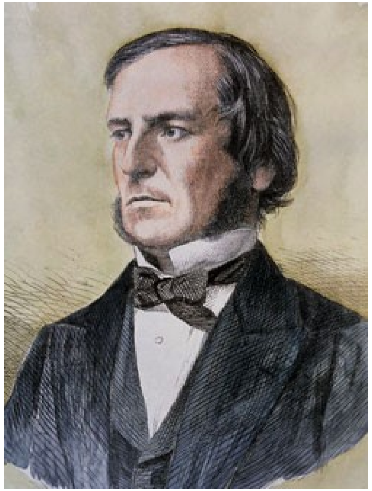


Lecture 3

Boolean algebra

UNIVERSITY OF LEEDS

- Boolean algebra was introduced by George Boole in 1847
- Boolean algebra is an algebra that has two value **true**/**false**, **high**/**low** or **1**/**0**
- The main operators in Boolean logic are **AND**, **OR** and **NOT**
- We have seen that logic seems to be an appropriate tool for computation



George Boole

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A v B
0	0	0
0	1	1
1	0	1
1	1	1

A	¬A
0	1
1	0

Truth tables

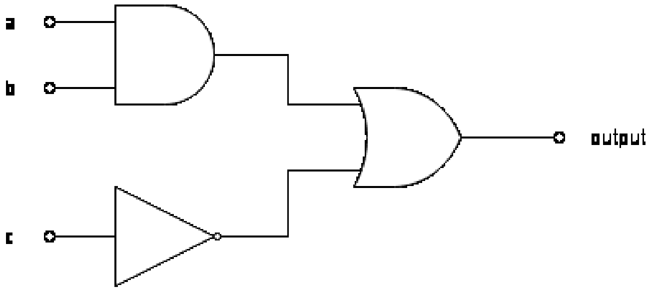
- Truth table are a means of communicating a Boolean function
- All possible values are enumerated with the result being placed in the final column
- All Boolean functions can be expressed as a truth table or as a functional expression

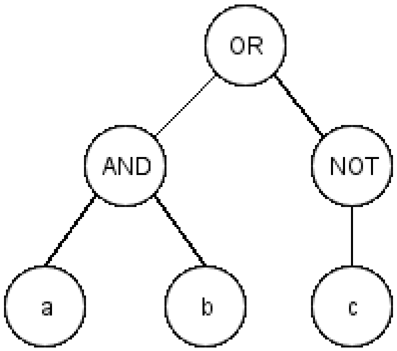
a	b	c	output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Representations of Boolean functions

- Truth tables
- Functional expressions
- Logic circuit diagram
- Expression tree

a **AND** b **OR NOT** c





a	b	c	output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Canonical representation (DNF)

UNIVERSITY OF LEEDS

- Every truth table can be expressed using at least one Boolean expression called the *canonical representation*
- May not be the most concise method of communicating a given Boolean function

1. Take each row of the truth table where ‘1’ appear as the output
2. Construct a logic expression for that row by **AND**’ing the variable where a ‘1’ appears in the column and the negation of a variable where ‘0’ appears in the column
3. **OR** each of the expression together

a	b	c	output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Canonical representation

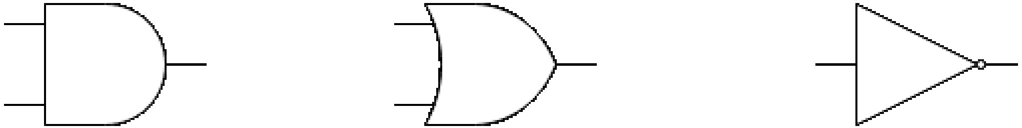
1. Take each row of the truth table where ‘1’ appear as the output
2. Construct a logic expression for that row by **AND**’ing the variable where a ‘1’ appear in the column and the negation of a variable where ‘0’ appears in the column
3. **OR** each of the expression together

$$(\neg a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge c)$$
$$\vee (\neg a \wedge b \wedge \neg c) \vee (\neg a \wedge b \wedge c)$$
$$\vee (a \wedge b \wedge c)$$

a	b	c	output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Canonical representation

- Every Boolean function can be expressed using the three Boolean operators **AND**, **OR** and **NOT**
- Might not be the most concise method of communicating the Boolean function
- It will come in handy when we design logic circuits



a	b	c	output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Canonical representation (CNF)

UNIVERSITY OF LEEDS

- CNF is an alternative normal form

- May not be the most concise method of communicating a given Boolean function

1. Take each row of the truth table where ‘0’ appear as the output
2. Construct a logic expression for that row by **OR**’ing the variable where a ‘0’ appears in the column and the negation of a variable where ‘1’ appears in the column
3. **AND** each of the expression together

a	b	c	output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Binary Boolean Functions

- There are 16 different binary Boolean functions
- Each binary Boolean function has a conventional name
- In general there are 2^m where $m = 2^n$

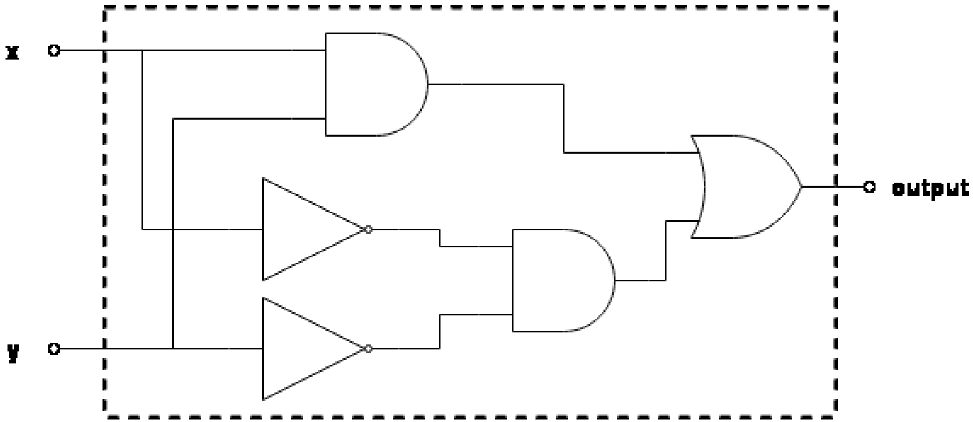
All of the binary Boolean functions are compositions of **AND**, **OR** and **NOT**

Function	x, y
Constant 0	0
And	$x \wedge y$
x And Not y	$x \wedge \neg y$
x	x
Not x And y	$\neg x \wedge y$
y	y
Exclusive Or	$x \wedge \neg y \vee \neg x \wedge y$
Or	$x \vee y$
Nor	$\neg(x \vee y)$
Equivalence	$x \wedge y \vee \neg x \wedge \neg y$
Not y	$\neg y$
If y then x	$x \vee \neg y$
Not x	$\neg x$
If x then y	$y \vee \neg x$
Nand	$\neg(x \wedge y)$
Constant 1	1

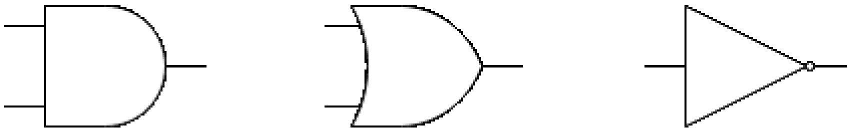
Logic gates

UNIVERSITY OF LEEDS

- A *gate* is a physical device that implements a Boolean function
- If a Boolean function operates on n variables and returns m binary results then the gate implementing the function has n input pins and m output pins
- Any gate, except the elementary gates, can be decomposed into elementary gates



Equivalence gate composed from elementary gates



Elementary gates

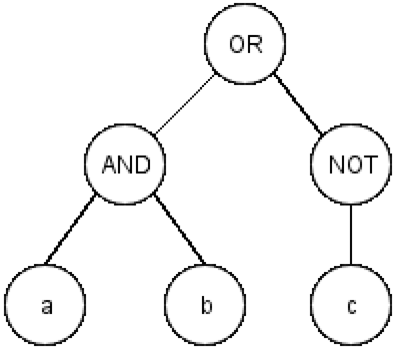
Logic gates

UNIVERSITY OF LEEDS

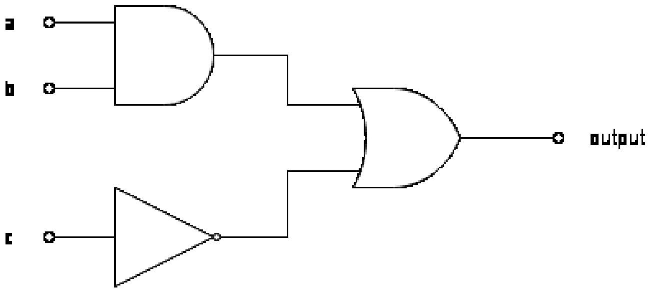
Convert from Boolean expression to Logic circuit.

1. Fully parenthesize the Boolean expression
2. Construct the tree expression for the Boolean expression
3. Start from the top of the tree working down recursively construct the logic circuit

a **AND** b **OR NOT** c

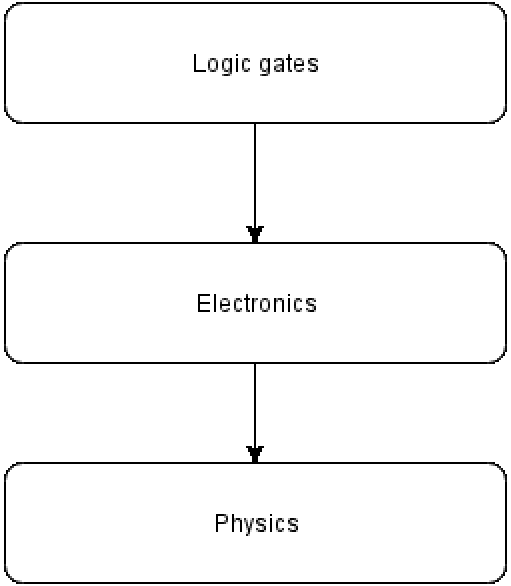


(a **AND** b) **OR** (**NOT** c)



Logic gates - Transistors

- We now know that all Boolean functions can be expressed in terms of **AND**, **OR** and **NOT** gates
- We know that **AND**, **OR** and **NOT** gates can be constructed from transistors
- Therefore all Boolean functions can be constructed from transistors



Layers of abstraction are important in computer science

Summary

UNIVERSITY OF LEEDS

- Introduced truth tables and canonical representation
- Introduced composite logic gates
- Demonstrated a technique for obtaining the canonical representation
- Demonstrated a technique for obtaining a logic circuit from a Boolean expression
- Demonstrated that abstraction is a powerful tool

Interested in logic? [The laws of thought]