We acknowledge and pay our respects to the Kaurna people,
the traditional custodians whose ancestral lands we gather on.

We acknowledge the deep feelings of attachment and relationship of the Kaurna people to country and we respect and value their past, present and ongoing connection to the land and cultural beliefs.

Computer Systems

Lecture 02: Logic Gates

THE UNIVERSITY of ADELAIDE

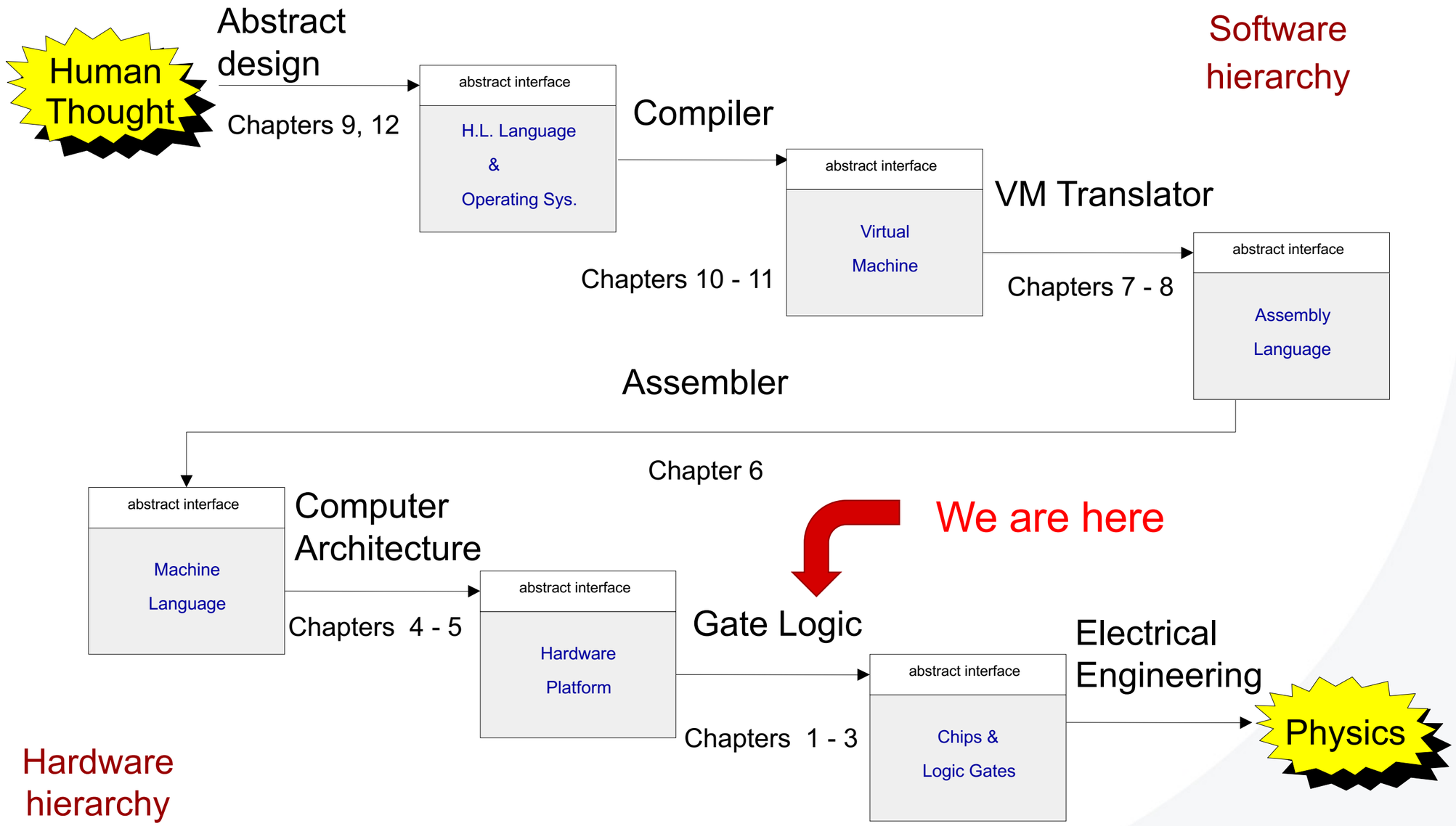
Course Summary

We’re going to show you how software and hardware work together to build a computer system.

Over the course, you will build parts of that system and get practice in combinational, sequential and gate logic, as well as learning how high-level languages make things happen in real systems.

THE UNIVERSITY of ADELAIDE

Our Journey



(Abstraction–implementation paradigm)

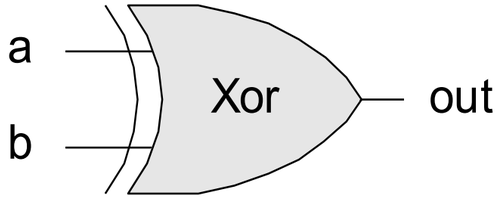
Poll

[open poll in browser]

THE UNIVERSITY of ADELAIDE

Review: Gate Logic

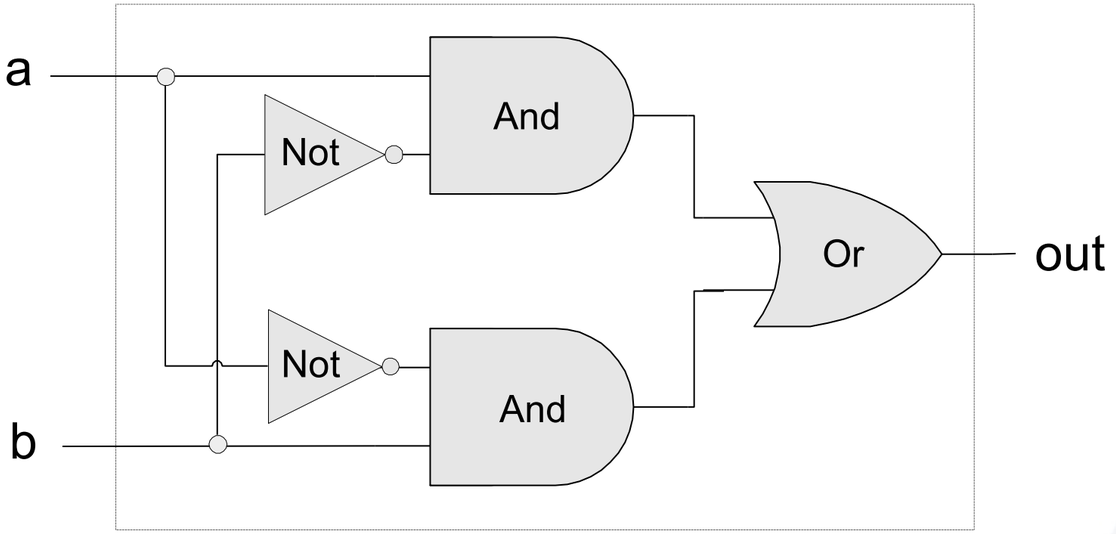
Interface



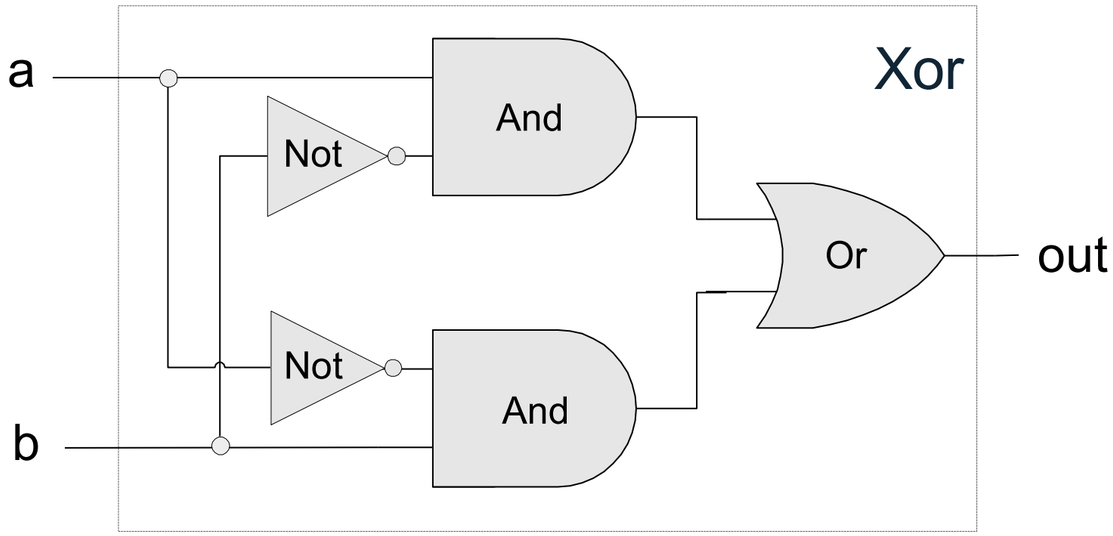
a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

(Truth table)

Implementation



In Workshops You Will Do:




```
CHIP Xor {
    IN a, b;
    OUT out;

    PARTS:
    Not(in=a,out=na);
    Not(in=b,out=nb);
    And(a=na,b=b,out=c);
    And(a=a,b=nb,out=d);
    Or(a=c,b=d,out=out);

}
```

All Boolean functions of 2 variables

Function	x	0	0	1	1
	y	0	1	0	1
Constant 0 $\leftrightarrow x \cdot \bar{x}$	0	0	0	0	0
x And y	$x \cdot y$	0	0	0	1
x And Not y	$x \cdot \bar{y}$	0	0	1	0
x	x	0	0	1	1
Not x And y	$\bar{x} \cdot y$	0	1	0	0
y	y	0	1	0	1
x Xor y (Add / Difference)	$x \cdot \bar{y} + \bar{x} \cdot y$	0	1	1	0
x Or y	$x + y$	0	1	1	1
x Nor y	$\overline{x + y}$	1	0	0	0
x Xnor y (Equivalence)	$x \cdot y + \bar{x} \cdot \bar{y}$	1	0	0	1
Not y	$\bar{y}$	1	0	1	0
x Or Not y (If y then x)	$x + \bar{y}$	1	0	1	1
Not x	$\bar{x}$	1	1	0	0
Not x Or y (If x then y)	$\bar{x} + y$	1	1	0	1
x Nand y	$\overline{x \cdot y}$	1	1	1	0
Constant 1 $\leftrightarrow x + \bar{x}$	1	1	1	1	1

THE UNIVERSITY of ADELAIDE

Canonical Form: representation of a Boolean function in terms of And, Or, and Not

We can construct a canonical representation of any Boolean function with:

- And (symbol "dot product" Example: $x \cdot y$)
- Or (symbol "plus sign" Example: $x + y$)
- Not (symbol "bar on top" Example: $\overline{x}$)

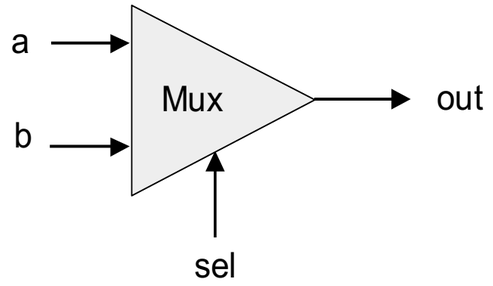
How to write a canonical form?

- Sum of Product (SoP) ← focus in this course
- Product of Sum (PoS)
- Karnaugh Map (K-map)

…

Canonical Form – Sum of Product (SoP)

a	b	sel	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



Exercise at the end!

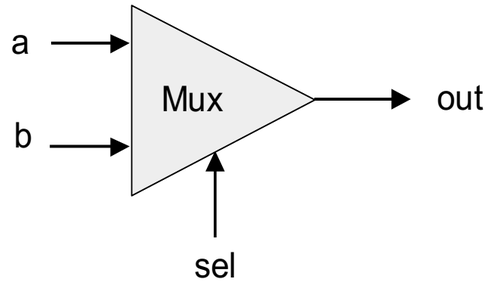
$$\text{out} = (\bar{a} \cdot b \cdot \text{sel}) + (a \cdot \bar{b} \cdot \overline{\text{sel}}) + (a \cdot b \cdot \overline{\text{sel}}) + (a \cdot b \cdot \text{sel})$$

1. Find all lines with out = 1, ignore all lines with out = 0.
2. Write inputs in product (And), if the input is 1 write as it is, otherwise, write in bar (Not) form.
3. Repeat 2. for each line with out = 1.
4. Use sum (Or) to connect each term.

THE UNIVERSITY of ADELAIDE

Canonical Form – Product of Sum (PoS)

a	b	sel	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

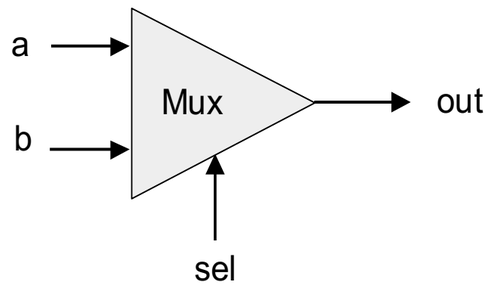


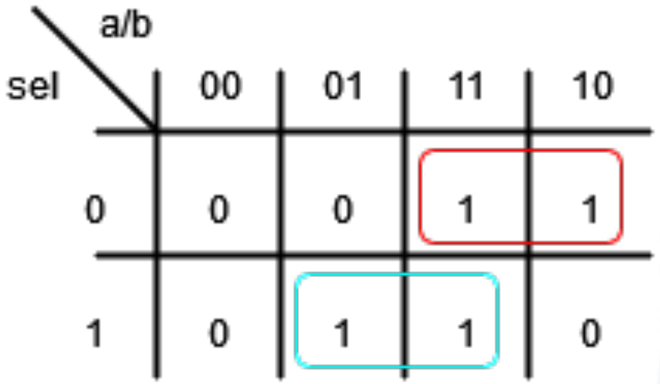
$$\mathbf{out} = (\mathbf{a} + \mathbf{b} + \mathbf{sel}) \cdot (\mathbf{a} + \mathbf{b} + \overline{\mathbf{sel}}) \cdot (\mathbf{a} + \bar{\mathbf{b}} + \mathbf{sel}) \cdot (\bar{\mathbf{a}} + \mathbf{b} + \overline{\mathbf{sel}})$$

1. Find all lines with out = 0, ignore all lines with out = 1.
2. Write inputs in sum (Or), if the input is 0 write as it is, otherwise, write in bar (Not).
3. Repeat 2. for each line with out = 0.
4. Use product (And) to connect each term.

Canonical Form – Karnaugh Map (K-map)

a	b	sel	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



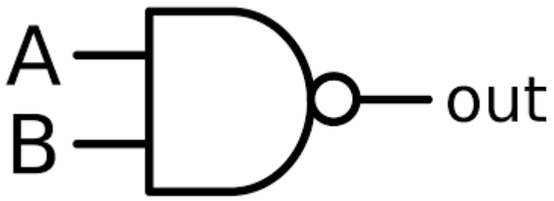


$$\text{out} = (a.\overline{\text{sel}}) + (b.\text{sel})$$

K-map is one of the systematic ways of simplifying Boolean expressions.
In this course, we will not ask for efficient/simplified Boolean expressions.
Extra reading: https://www.allaboutcircuits.com/textbook/digital/chpt-8/logic-simplification-karnaugh-maps/

How to construct And, Or and Not from Nand

From this



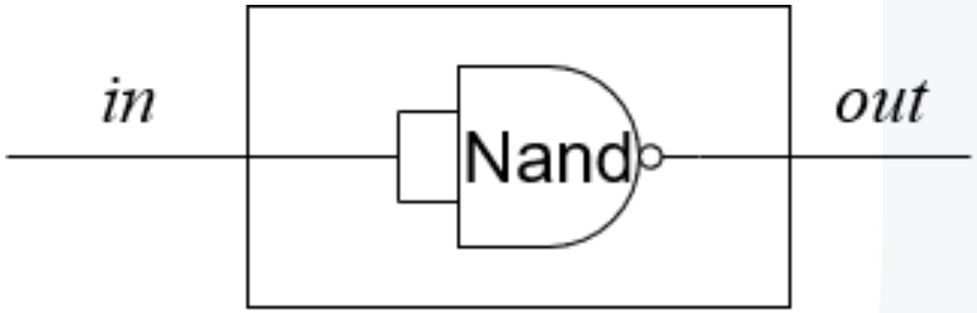
A	B	out
0	0	1
0	1	1
1	0	1
1	1	0

$$out = \overline{A \cdot B}$$

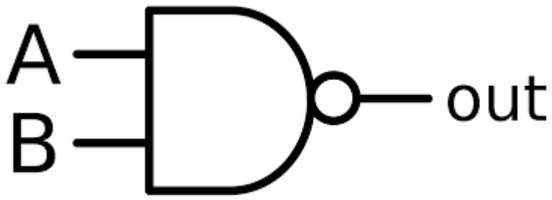
Formal proof in Textbook Appendix A1.3

How to construct And, Or and Not from Nand

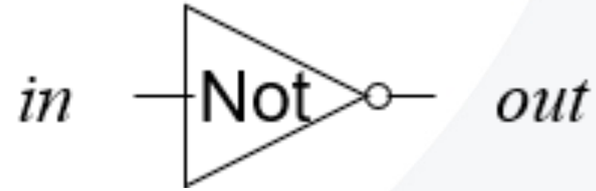
From this



A	B	out
0	0	1
0	1	1
1	0	1
1	1	0



Have A and B commonly connected, A and B always the same

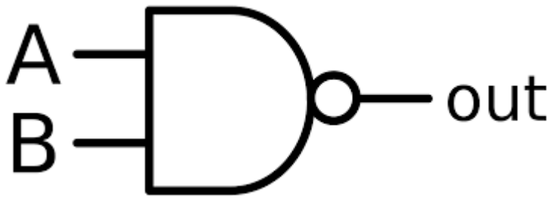


$$out = \overline{A \cdot B}$$

Formal proof in Textbook Appendix A1.3

How to construct And, Or and Not from Nand

From this



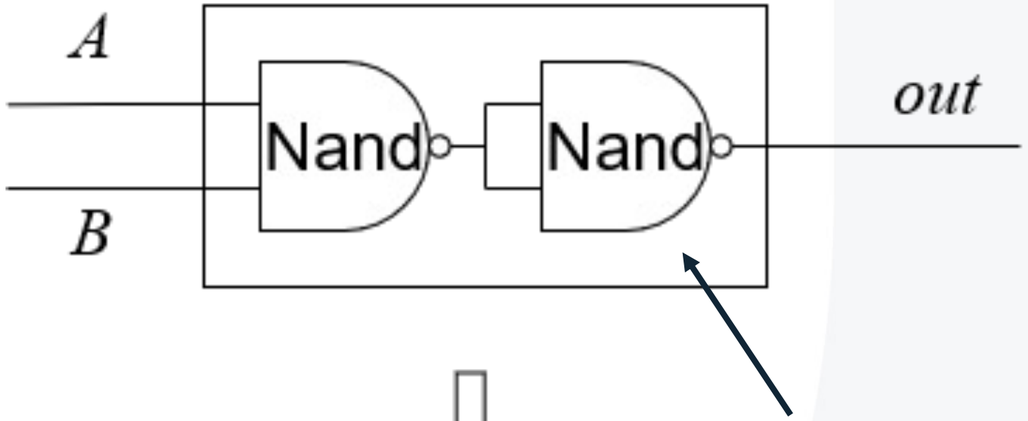
A	B	out
0	0	1
0	1	1
1	0	1
1	1	0

$$out = \overline{A \cdot B}$$

Formal proof in Textbook Appendix A1.3

How to construct And, Or and Not from Nand

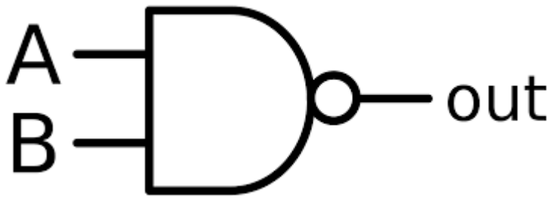
From this

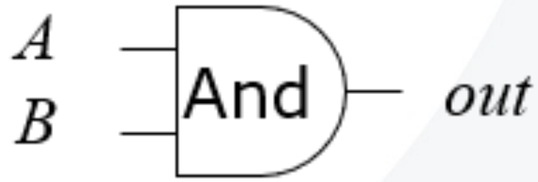


Not gate we built in the last slide

A	B	out	$\overline{out}$
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

Add one Not gate to the out



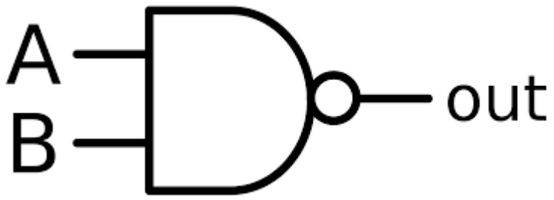


$$out = \overline{A \cdot B}$$

Formal proof in Textbook Appendix A1.3

How to construct And, Or and Not from Nand

From this



A	B	out
0	0	1
0	1	1
1	0	1
1	1	0

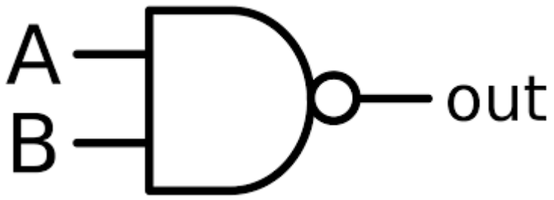
$$out = \overline{A \cdot B}$$

Formal proof in Textbook Appendix A1.3

How to construct And, Or and Not from Nand

From this

A	B	out
0	0	1
0	1	1
1	0	1
1	1	0



$$out = \overline{A \cdot B}$$

In propositional logic and Boolean algebra, **De Morgan's laws**,[1][2][3] also known as **De Morgan's theorem**,[4] are a pair of transformation rules that are both valid rules of inference. They are named after Augustus De Morgan, a 19th-century British mathematician. The rules allow the expression of conjunctions and disjunctions purely in terms of each other via negation.

The rules can be expressed in English as:

- The negation of a disjunction is the conjunction of the negations
- The negation of a conjunction is the disjunction of the negations

or

- The complement of the union of two sets is the same as the intersection of their complements
- The complement of the intersection of two sets is the same as the union of their complements

or

- not (A or B) = (not A) and (not B)
- not (A and B) = (not A) or (not B)

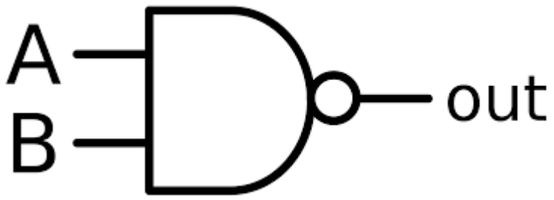
where "A or B" is an "inclusive or" meaning *at least* one of A or B rather than an "exclusive or" that means *exactly* one of A or B.

Formal proof in Textbook Appendix A1.3

How to construct And, Or and Not from Nand

From this

A	B	out
0	0	1
0	1	1
1	0	1
1	1	0



In electrical and computer engineering, De Morgan's laws are commonly written as:

$$\overline{(A \cdot B)} \equiv (\overline{A} + \overline{B})$$

and

$$\overline{A + B} \equiv \overline{A} \cdot \overline{B},$$

where:

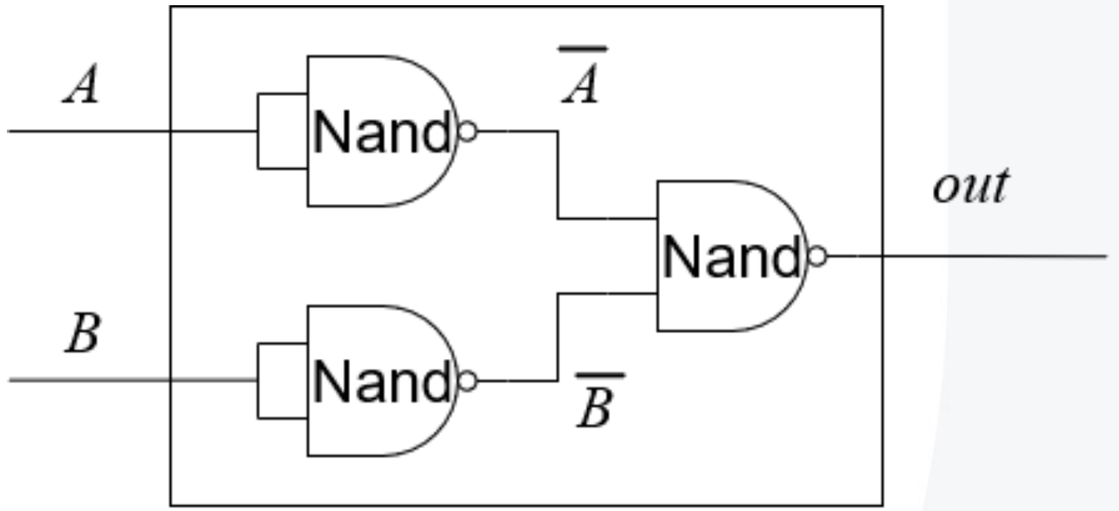
- $\cdot$ is the logical AND,
- $+$ is the logical OR,
- the $\overline{\text{overbar}}$ is the logical NOT of what is underneath the overbar.

$$out = \overline{A \cdot B}$$

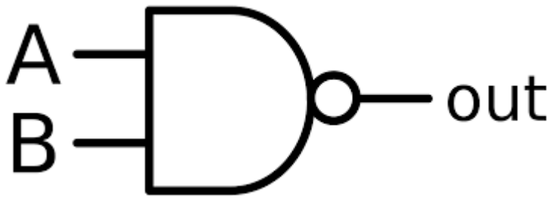
Formal proof in Textbook Appendix A1.3

How to construct And, Or and Not from Nand

From this



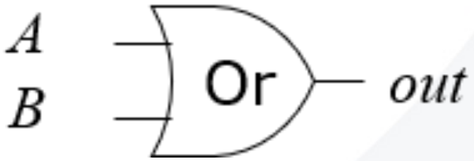
A	B	$\overline{A}$	$\overline{B}$	out
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0



Add one Not gate to each of the A and B

$$x+y = \overline{\overline{x} \cdot \overline{y}}$$

De Morgan's Theorem

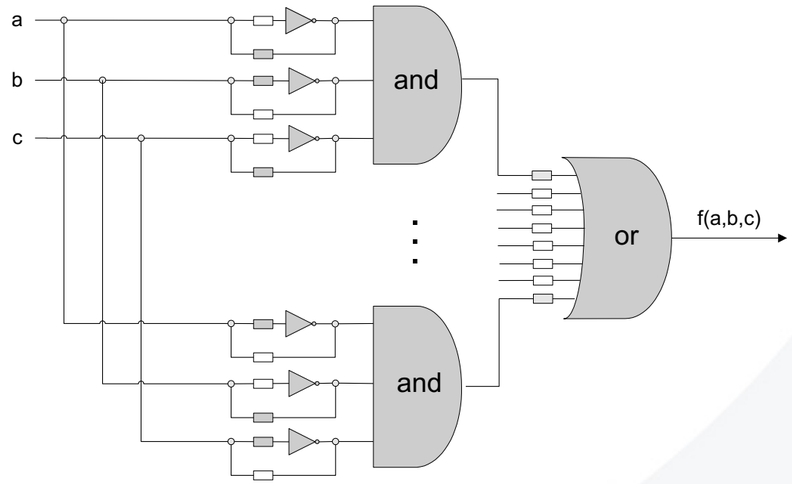


$$out = \overline{A \cdot B}$$

Formal proof in Textbook Appendix A1.3

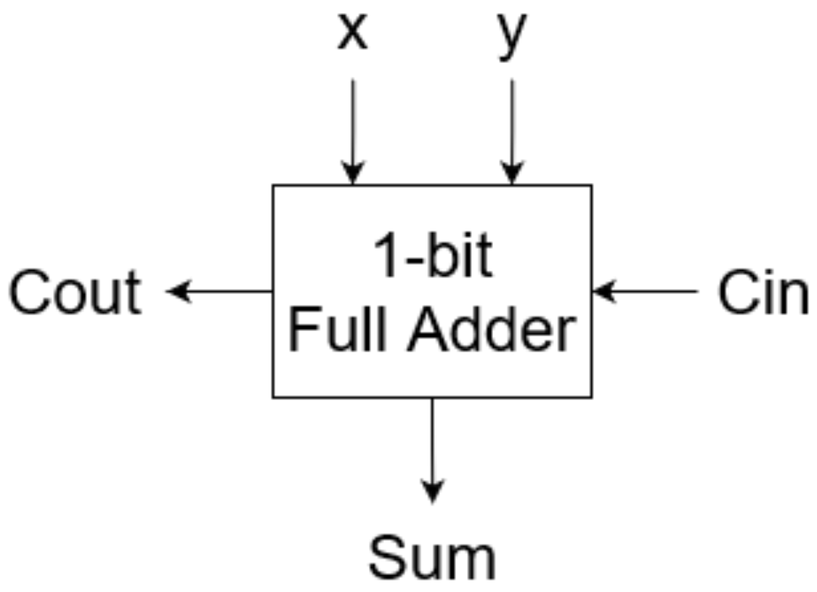
Boolean Functions: Summary

- Each Boolean function has a canonical representation
- The canonical representation is expressed in terms of And, Or, Not
- And, Not, Or can be expressed in terms of Nand alone (or Nor)
- Every Boolean function can be implemented by a standard circuit consisting of Nand gates only
- Mass production
- Universal building blocks, unique topology



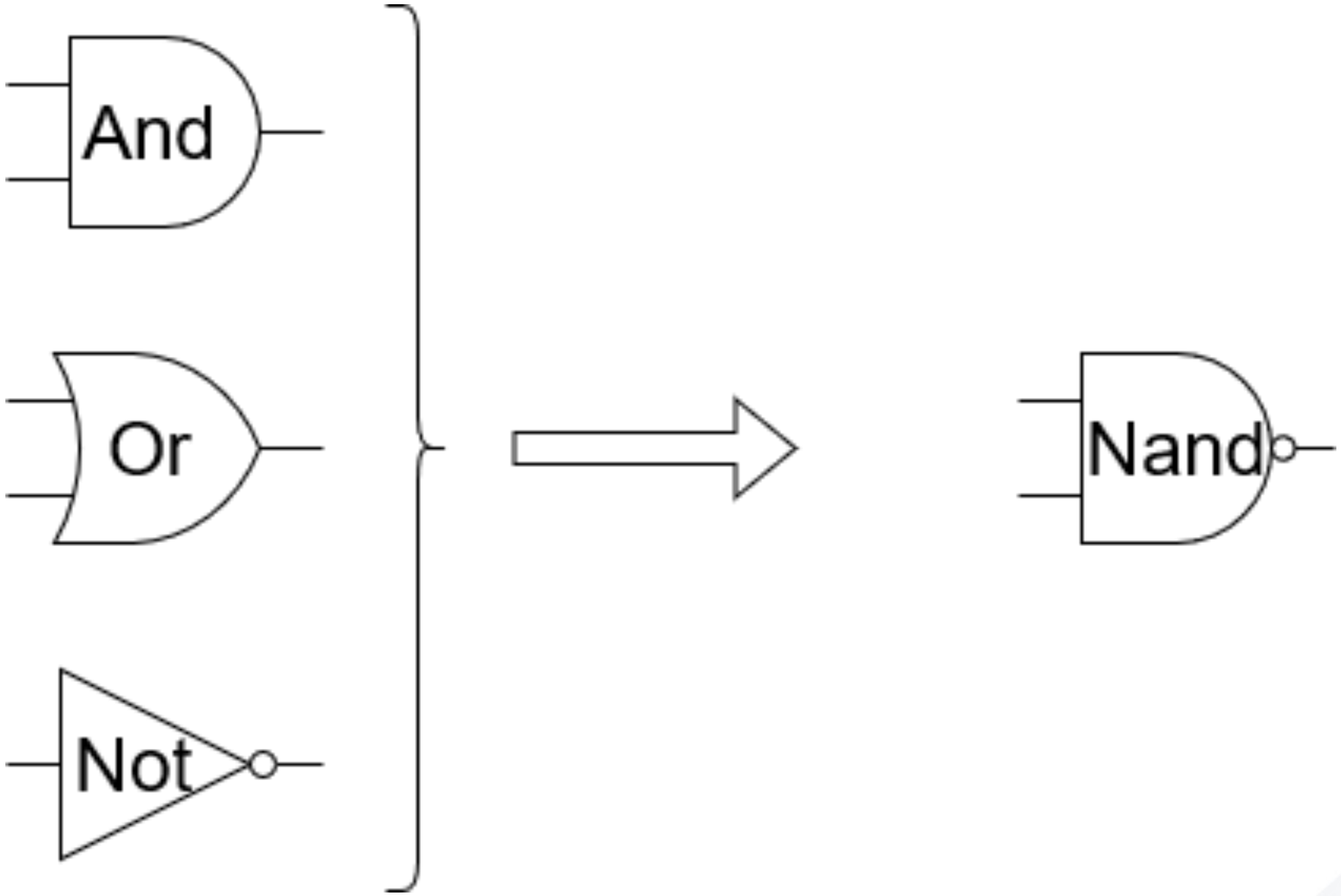
THE UNIVERSITY of ADELAIDE

Exercise: Canonical Form Boolean Expression for 1-bit Full Adder

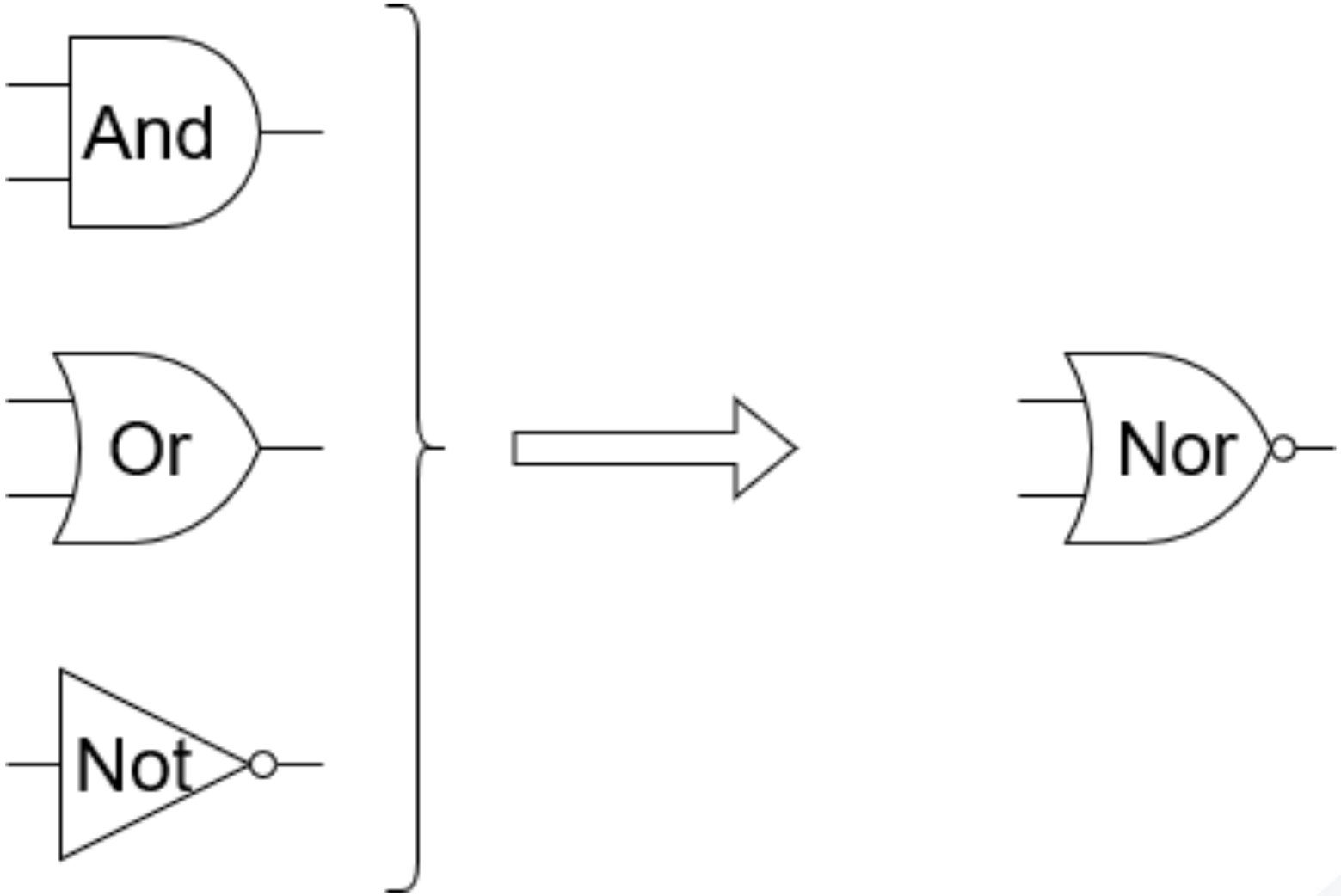


x	y	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

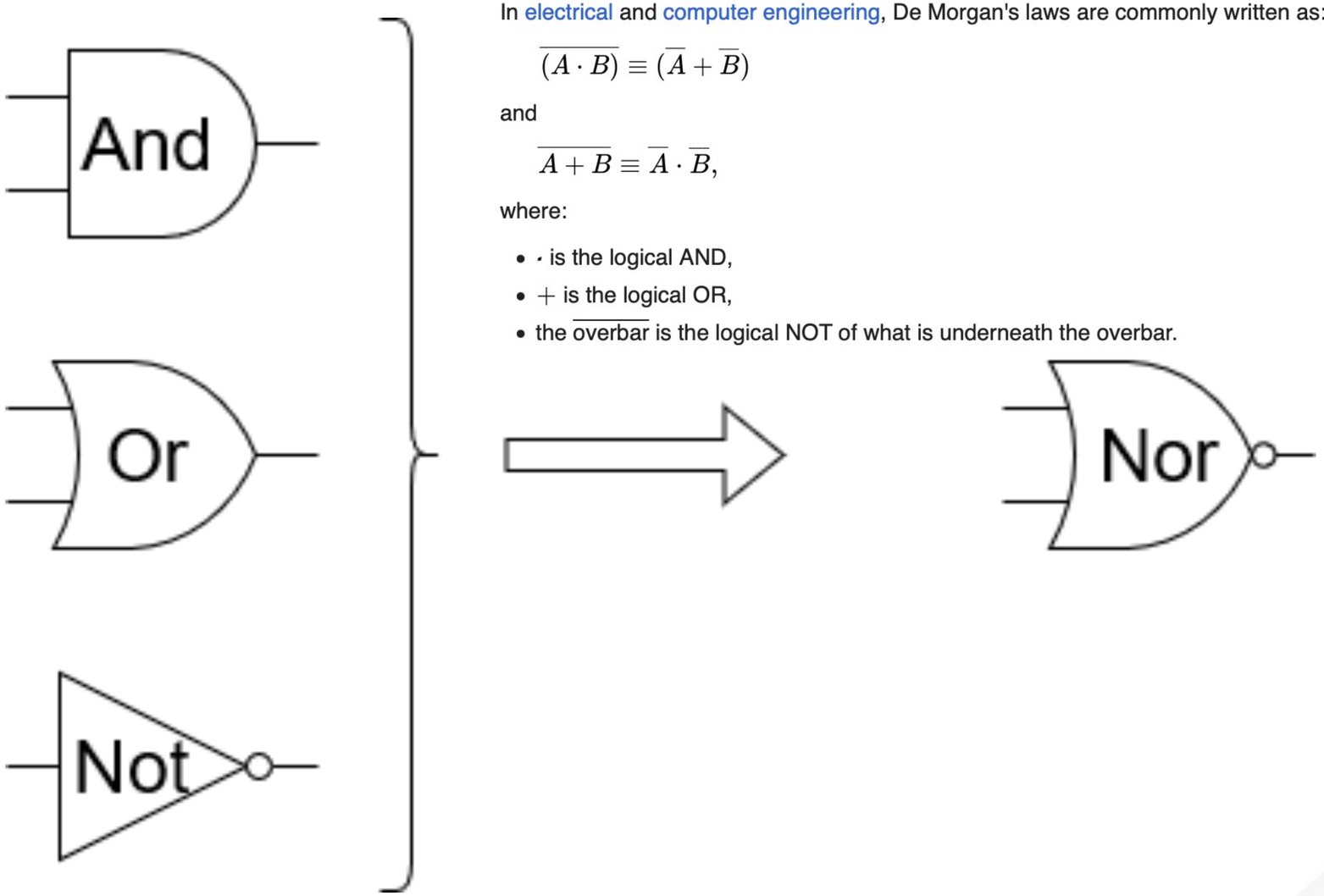
Exercise: Build Nand using And, Or, Not



Exercise: Build Nor using And, Or, Not



Exercise: Build Nor using And, Or, Not



In electrical and computer engineering, De Morgan's laws are commonly written as:

$$\overline{(A \cdot B)} \equiv (\overline{A} + \overline{B})$$

and

$$\overline{A + B} \equiv \overline{A} \cdot \overline{B},$$

where:

- $\cdot$ is the logical AND,
- $+$ is the logical OR,
- the $\overline{\text{overbar}}$ is the logical NOT of what is underneath the overbar.

This Week

- Review Chapter 1 of the Textbook
- Start Assignment 1 (available Now)
- Workshops start in week 2
- Week 2 quiz due before Thursday lecture.
- Please use Piazza for communication
  - Email me for personal issues only (start subject with [CS])
  - Email me about prerequisites extensions

THE UNIVERSITY of ADELAIDE