Lecture 2

Fri: SPS.

Tues: HW by 5pm

Tues: B 2.1, 2.3
R 2.1

Logic gates

We have seen that all integers can be represented in terms of series of bits (binary digits) and that these can be manipulated so as to carry out integer arithmetic.

For example :

Instruction:
$3 \times 9 =$

→ Represent 3 via binary 0011
Represent 9 via binary 1001

↓

Multiply

```
  0011
  1001
 -----
  0011
 0000
0000
0011
-------
 11011
```

→ Output = 11011

↳ Convert to decimal.
16+8+2+1 = 27

Hidden here are certain fundamental binary arithmetic operations on single bits. For example the operations

1) modular addition $a \oplus b$ occurs in binary addition
2) single bit multiplication ab " " " multiplication

What we ask is:

> "Is there a small set of basic binary operations which can be used repeatedly so as to implement any arithmetic operation involving binary representations of numbers?"

There are various reasons for asking this. Two are:

1) if we can reduce any arithmetic operation or function to a set of basic operations then we can quantify the difficulty of implementing the function in terms of the number of basic operations.

2) if we can reduce any arithmetic operation or function to a set of basic operations then we only need to be able to construct physical hardware to implement each basic operation to eventually implement the function...

This will be the idea behind logic gates. These are basic operations from which all others can be constructed.

Single bit gates

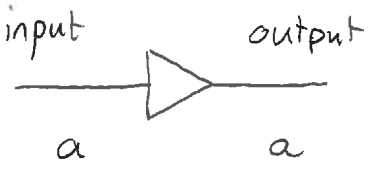
Consider a single bit, a and recall that this can take on two values, $a=0$ or $a=1$.

There are two possible single bit operations:

1) Identity

Input a	Output $=a$
0	0
1	1

Then we represent this via a "circuit" element



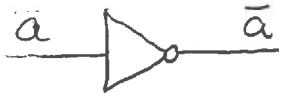
direction of time / evolution or information processing.

2) NOT operation

The NOT operation inverts the value of the bit. It takes input a and produces output denoted $\bar{a}$.

Input a	Output $\bar{a}$
0	1
1	0

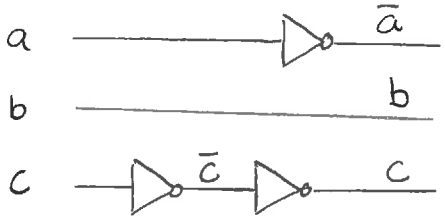
The circuit element for this is :



"NOT gate"

Ordinarily we do not use the identity operation. We could now start to create circuits. Here each line represents a bit and the diagram is read from left to right as time passes.

For example:



since $\bar{\bar{c}} = c$

information processing →

Two bit gates

We need operations that involve two bits. The following are two standard operations

1) AND operation

This takes two bits as input and produces a single bit as output

Input : a, b → Output: a AND b ≡ $a \wedge b$

The output is establish according to:

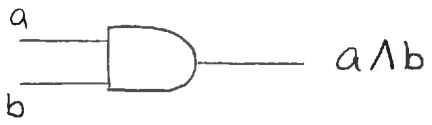
a	b	$a \wedge b$
0	0	1
0	1	0
1	0	0
1	1	1

" Truth table"

Note that this gives single digit multiplication

$ab = a \wedge b$

The gate for this is:



2) OR operation

This also takes two bits as input and produces a single bit as output:

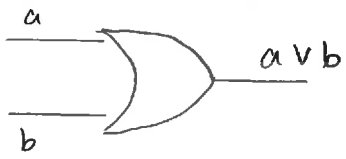
Input: a, b → Output: "a OR b"

$a \vee b$

The output is established according to the following truth table

a	b	$a \vee b$
0	0	0
0	1	1
1	0	1
1	1	1

This bears some resemblance to addition although not completely. The relevant gate is denoted



Exercise For all single bit inputs show that

a) $a \vee \bar{a} = 1$

b) $a \wedge \bar{a} = 0$

c) $a \vee b = a \oplus b \oplus (ab)$

Answer a)

Input a	Input $\bar{a}$	Output $a \vee \bar{a}$
0	1	1
1	0	1

So $a \vee \bar{a} = 1$

b)

Input a	Input $\bar{a}$	Output $a \wedge \bar{a}$
0	1	0
1	0	0

So $a \wedge \bar{a} = 0$

c)

a	b	ab	$a \oplus b \oplus ab$	$a \vee b$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	1

match

So $a \oplus b \oplus ab = a \vee b$

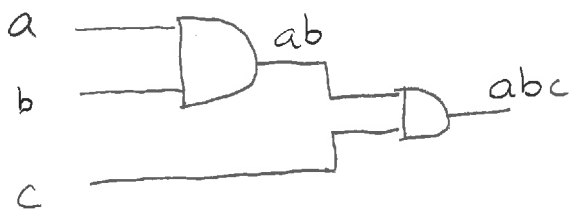
Exercise Use AND, OR and NOT gates to construct a circuit that does:

$$a,b,c \rightarrow abc$$

Answer. This is

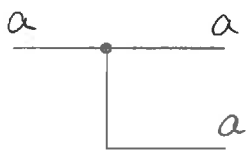
$$a,b,c \rightarrow (ab)\,c$$

(a AND b) AND c



Copy Operation

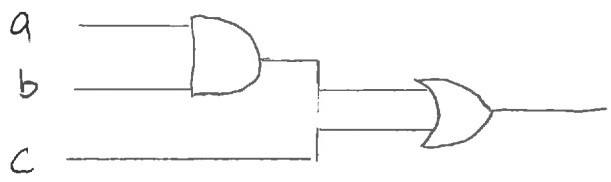
One more operation that is allowed with classical bits is to copy a bit value. In circuits this is denoted as



Such copying often appears in classical circuits

Binary / logic algebra / Boolean algebra

We can clearly perform successive binary operations or binary gates. For example



This corresponds to

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

It turns out that these can be manipulated algebraically. One can show that:

1) $(a \wedge b) \wedge c = a \wedge (b \wedge c)$

2) $(a \vee b) \vee c = (a \vee b) \vee c$

3) $(a \vee b) \wedge c = (a \wedge c) \vee (b \wedge c)$

4) $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

Various other identities can be established. Important cases are

De Morgan's Laws

$$\overline{a \wedge b} = \bar{a} \vee \bar{b}$$

$$\overline{a \vee b} = \bar{a} \wedge \bar{b}$$

Exercise: By considering all possible single bits show that

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

b) $\overline{a \vee b} = \bar{a} \wedge \bar{b}$

Answer: a)

a	b	c	a ∨ b	(a ∨ b) ∧ c	a ∧ c	b ∧ c	(a ∧ c) ∨ (b ∧ c)
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	0	1	1
1	0	0	1	0	0	0	0
1	0	1	1	1	1	0	1
1	1	0	1	0	0	0	0
1	1	1	1	1	1	1	1

LHS

RHS

b)

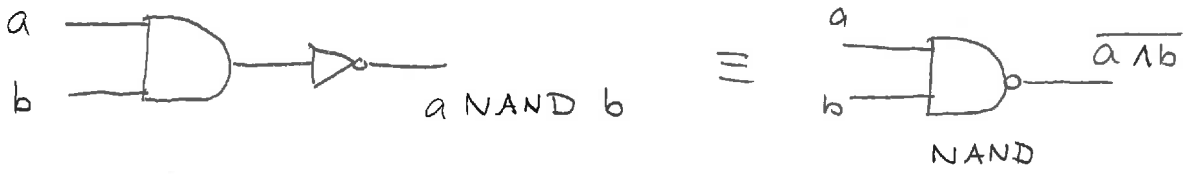
a	b	a ∨ b	$\overline{a \vee b}$	$\bar{a}$	$\bar{b}$	$\bar{a} \wedge \bar{b}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

match

Finally there are some special gates that are useful:

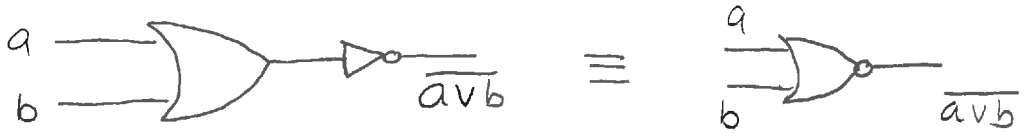
1) NAND maps $a, b \to \overline{a \wedge b}$. So

"Not AND"



2) NOR maps $a, b \to \overline{a \vee b}$

"Not OR"



3) XOR maps $a, b \to a \oplus b$

We can see that

> AND and XOR are sufficient to construct any binary arithmetic operation since they allow for modular addition and binary multiplication

Furthermore one can show (HW):

$$a \oplus b = (\bar{a} \wedge b) \vee (a \wedge \bar{b})$$

XOR

Thus we get:

> Any binary arithmetic operation can be accomplished with combinations of AND, OR and NOT

Thus we say that the three gates AND, OR and NOT are universal – they are all that is needed for any binary arithmetic.

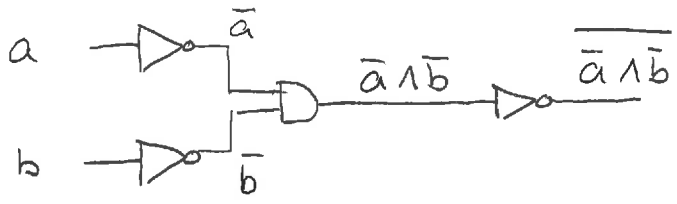
We can further reduce this set by using De Morgan's Laws.

Exercise: Use De Morgan's laws to provide a circuit for the OR gate in terms of AND and NOT gates.

Answer: De Morgan $\Rightarrow$ $\overline{a \vee b} = \bar{a} \wedge \bar{b}$,

$\Rightarrow$ $a \vee b = \overline{\bar{a} \wedge \bar{b}}$

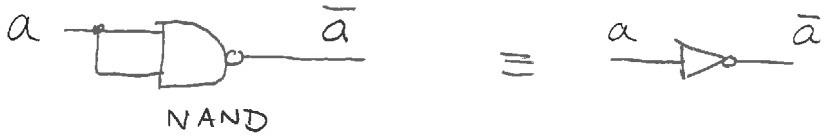
So



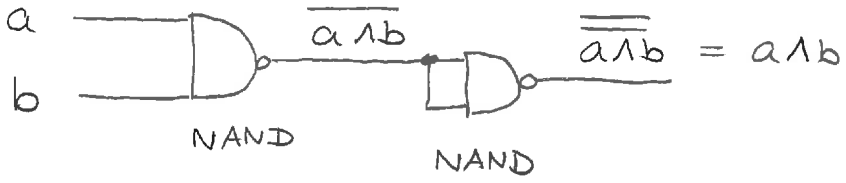
Thus we get:

> Any binary arithmetic operation can be done using combinations of AND and NOT gates. These two are universal.

Finally we can reduce the gates even more by noting that the NAND can construct both NOT and AND. Specifically



This gives NOT. And AND is constructed from:



So we get

The NAND gate is universal for binary arithmetic

All classical (deterministic) computing can be reduced to binary arithmetic. So we get:

1) Any computation can be realized by a sequence of AND and NOT gates (plus copying).
   - or NAND gates
2) The difficulty of executing any computation can be assessed by counting the number of AND and NOT gates needed to do it.