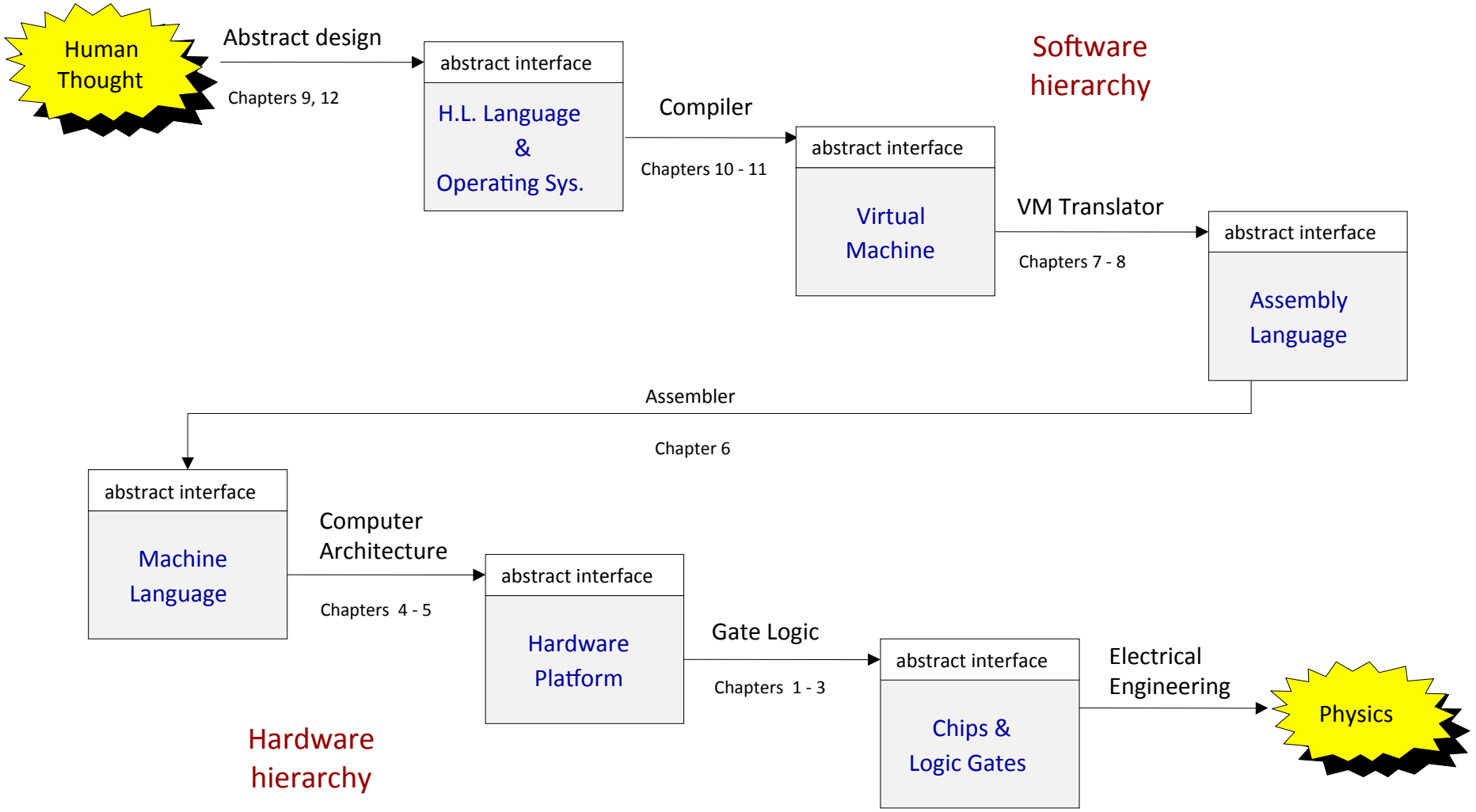
School of Computer Science

COMP SCI 2000 Computer Systems
Lecture 3

The whole system



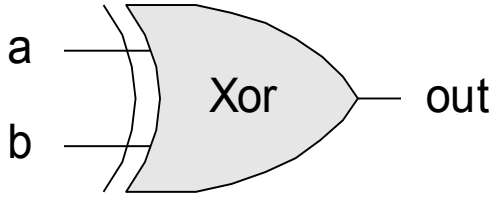
(Abstraction–implementation paradigm)

Review: What is gate logic?

- Our hardware is an inter-connected set of chips.
- Chips are built of simpler chips, down to the simplest structure of all – the elementary logic gate.
- Logic gates are hardware implementations of Boolean functions. This allows us to represent logical statements in computer form.
- Every chip and gate has:
  - An interface: Telling us what it does
  - An implementation: Telling us how it does it.
- Worksheet Question 1:
  - In 1-2 sentences write why interfaces are important.

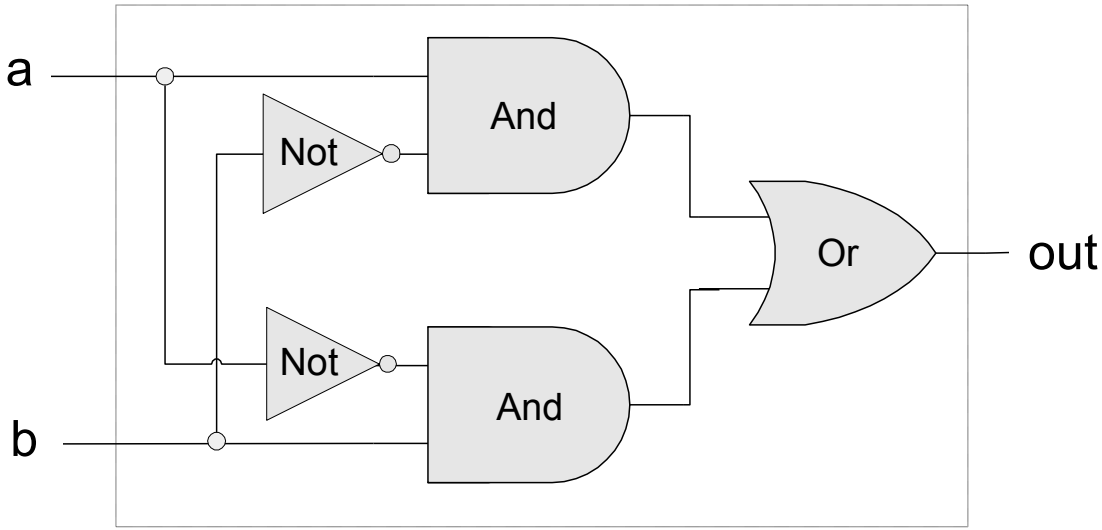
Review: Example

Interface



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

Implementation



All Boolean functions of 2 variables

Function	x	**0**	**0**	**1**	**1**
	y	**0**	**1**	**0**	**1**
Constant 0	0	0	0	0	0
And	$x \cdot y$	0	0	0	1
x And Not y	$x \cdot \overline{y}$	0	0	1	0
x	x	0	0	1	1
Not x And y	$\overline{x} \cdot y$	0	1	0	0
y	y	0	1	0	1
Xor	$x \cdot \overline{y} + \overline{x} \cdot y$	0	1	1	0
Or	$x + y$	0	1	1	1
Nor	$\overline{x+y}$	1	0	0	0
Equivalence	$x \cdot y + \overline{x} \cdot \overline{y}$	1	0	0	1
Not y	$\overline{y}$	1	0	1	0
If y then x	$x + \overline{y}$	1	0	1	1
Not x	$\overline{x}$	1	1	0	0
If x then y	$\overline{x} + y$	1	1	0	1
Nand	$\overline{x \cdot y}$	1	1	1	0
Constant 1	1	1	1	1	1

Do Lecture 3 worksheet question 2.

What can we build from NAND?

- What is NAND?

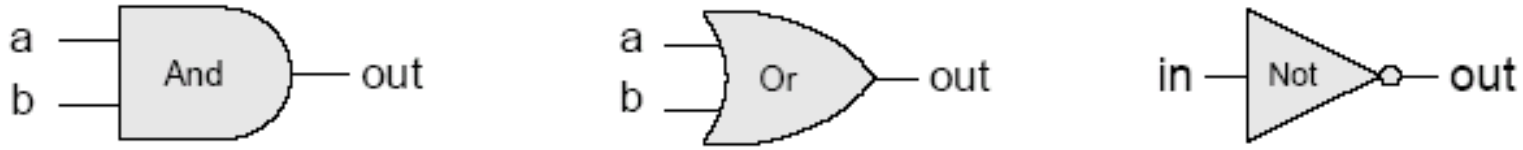
What can we build from NAND?

- What is NAND?
  - NOT AND(a,b)
- Think about how could we build:
  - NOT from NAND?
  - TRUE if we have NAND and FALSE?
  - AND from NAND?
  - OR from NAND?
  - XOR from NAND?

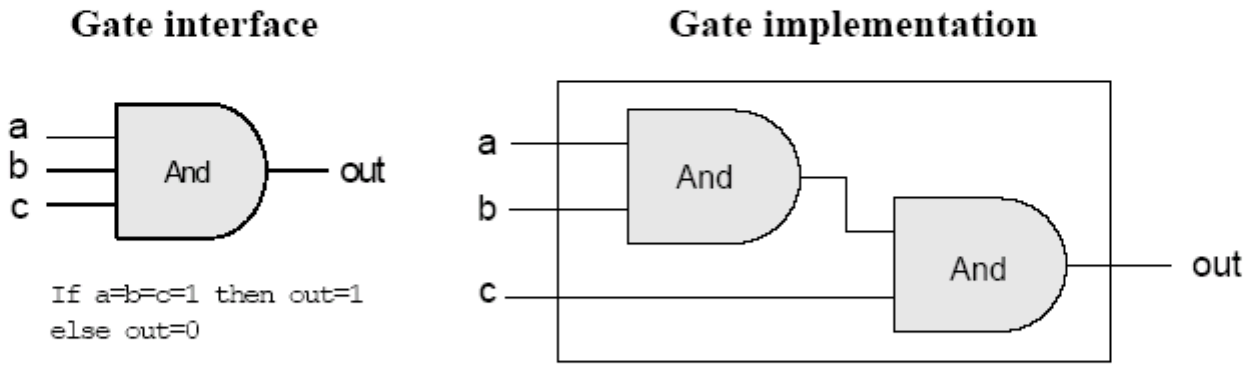
Review: Gate logic

- Gate logic – a gate architecture designed to implement a Boolean function

- Elementary gates:



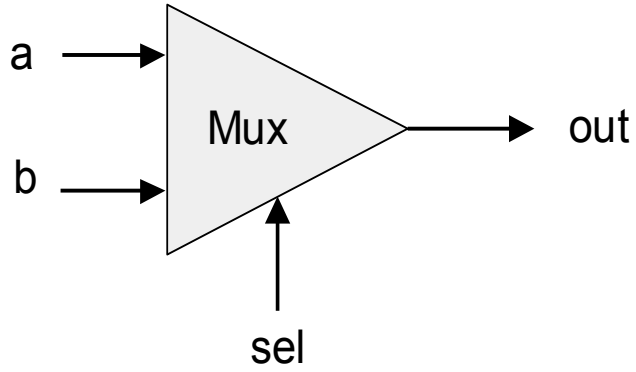
- Composite gates:



- Important distinction: Interface (*what*) VS implementation (*how*).

Project 1: Multiplexer

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

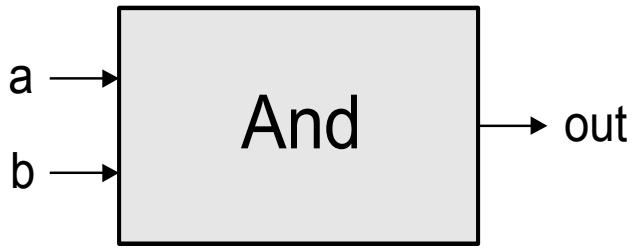


sel	out
0	a
1	b

Proposed Implementation: based on Not, And, Or gates.

What does this *do*? What is a multiplexer?

Example: Building an **And** gate



And.cmp

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

Contract:

When running your **.hdl** on our **.tst**, your **.out** should be the same as our **.cmp.**

And.hdl

```
CHIP And
{    IN  a, b;
     OUT out;
     // implementation missing
}
```

And.tst

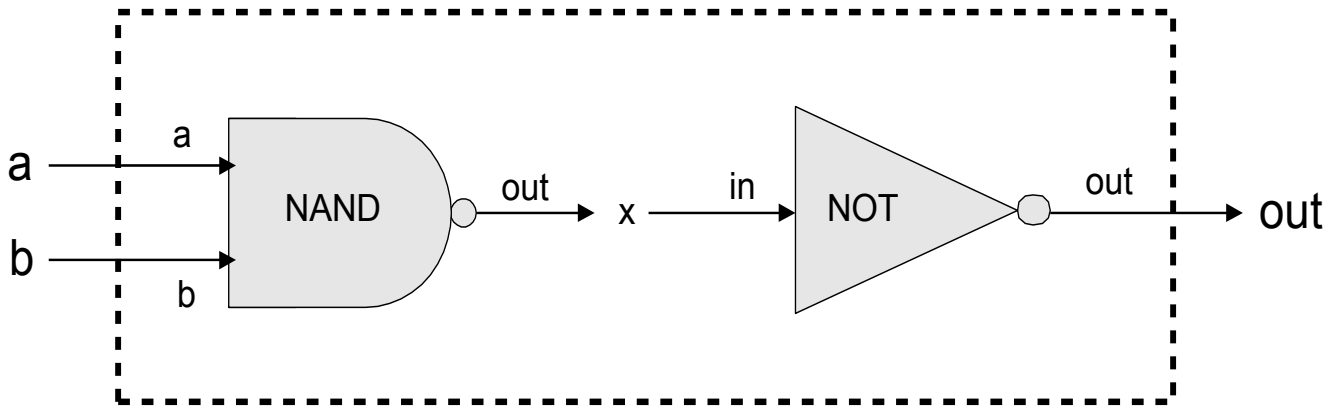
```
load And.hdl,
output-file And.out,
compare-to And.cmp,
output-list a b out;
set a 0,set b 0,eval,output;
set a 0,set b 1,eval,output;
set a 1,set b 0,eval,output;
set a 1, set b 1, eval, output;
```

What can we build from NAND?

- What is NAND?
  - NOT AND(a,b)
- How could we build:
  - NOT from NAND?
  - TRUE if we have NAND and FALSE?
  - AND from NAND? (Spoiler Alert!!!!)
  - OR from NAND?
  - XOR from NAND?

Building an **And** gate

Implementation: And(a,b) = Not(Nand(a,b))

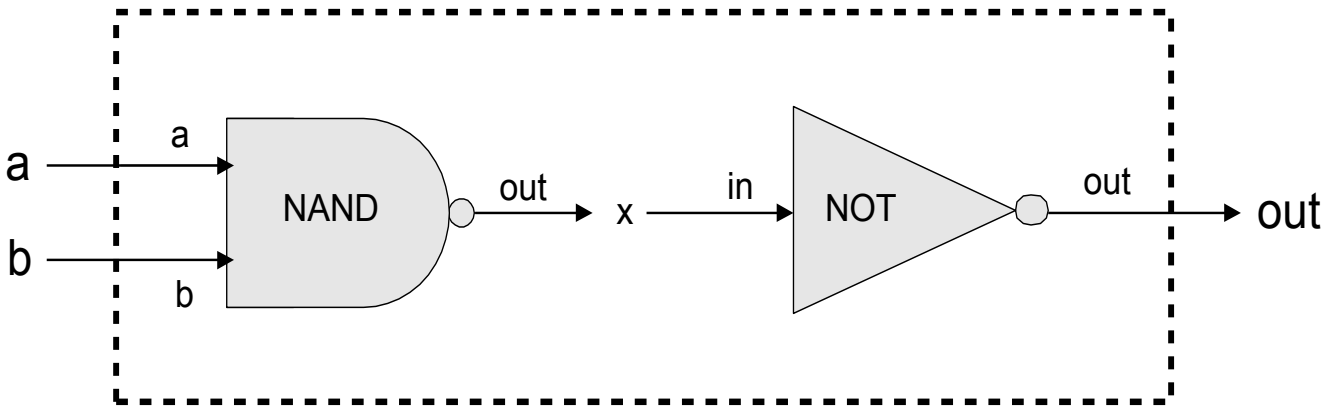


And.hdl

```
CHIP And
{    IN  a, b;
     OUT out;
    // What goes in here?
}
```

Building an **And** gate

Implementation: And(a,b) = Not(Nand(a,b))



And.hdl

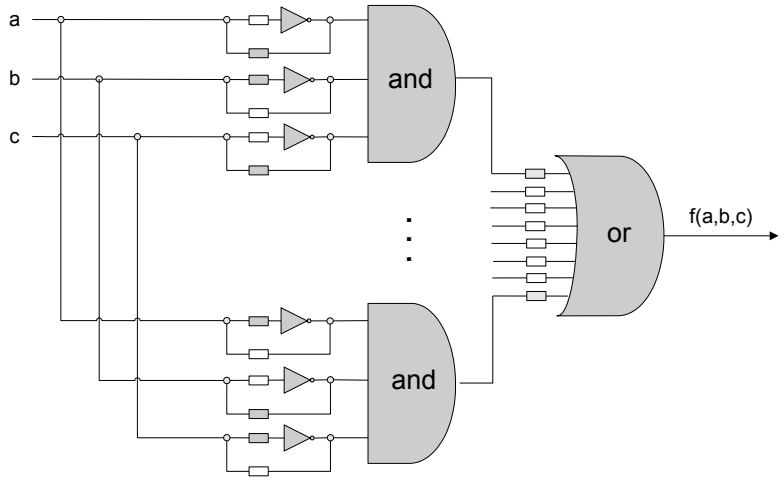
```
CHIP And
{   IN  a, b;
    OUT out;
    Nand(a = a,
         b = b,
         out = x);
    Not(in = x, out = out)
}
```

Hardware Simulator – Demo!

Try worksheet question 3

Quick stop!

- Each Boolean function has a canonical representation
- The canonical representation is expressed in terms of And, Not, Or
- And, Not, Or can be expressed in terms of Nand alone
- Every Boolean function can be realized by a standard circuit consisting of Nand gates only
- Mass production
- Universal building blocks, unique topology
- Gates, neurons, atoms, …



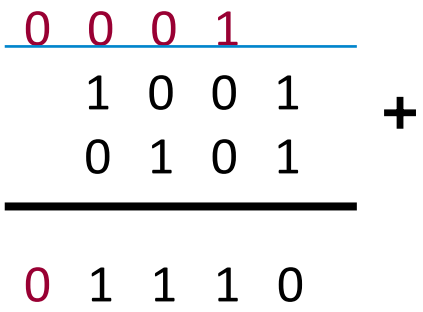
Counting systems

quantity	decimal	binary	3-bit register
	0	0	000
*	1	1	001
**	2	10	010
***	3	11	011
****	4	100	100
*****	5	101	101
******	6	110	110
*******	7	111	111
********	8	1000	overflow
*********	9	1001	overflow
**********	10	1010	overflow

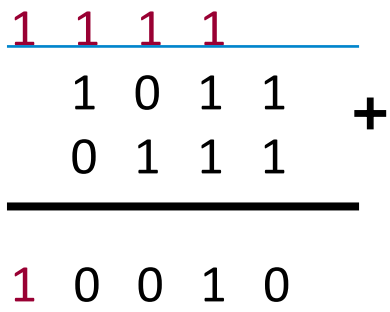
Answer worksheet question 4

Binary Addition

Assuming a 4-bit system:



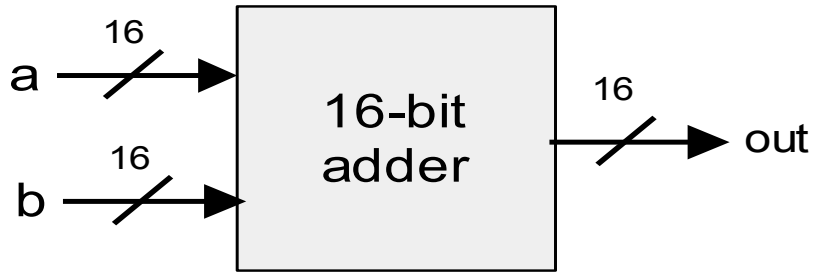
no overflow



overflow

- Algorithm: exactly the same as in decimal addition
- Overflow (MSB carry) has to be dealt with.
- How do we represent negative numbers? (Group work worksheet question 5)

Building an Adder chip

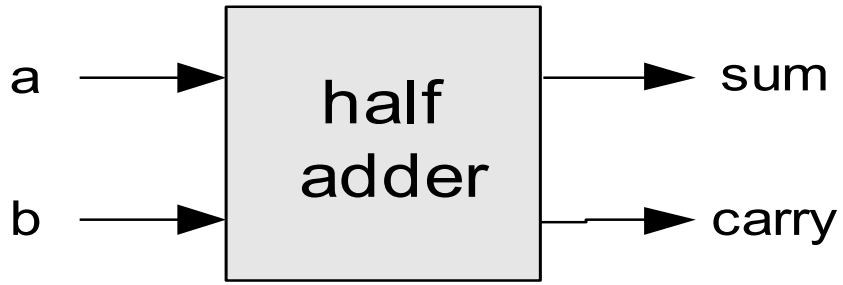


- Adder: a chip designed to add two integers

- Proposed implementation:
  - Half adder: designed to add 2 bits
  - Full adder: designed to add 3 bits
  - Adder: designed to add two *n*-bit numbers.

Half adder (designed to add 2 bits)

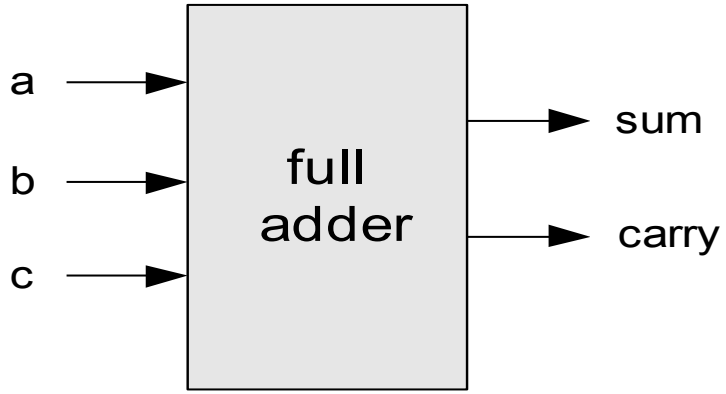
a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Implementation: based on two gates that you've seen before.
What do you think they are? (Group discussion: Worksheet question 6)

Full adder (designed to add 3 bits)

a	b	c	sum	carry
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



Implementation: can be based on half-adder gates. How? (exercise for after the lecture).

Perspective

- Combinational logic
- Our adder design is very basic: no parallelism
- It pays to optimize adders (but we won't do that here).
- Where is the seat of more advanced math operations?
  a typical hardware/software tradeoff.

Summary

- You can construct many gates from NAND – this is just one example of how gates are built up.
- By understanding arithmetic, we can combine gates to add two numbers, then combine half-adders to add larger numbers.

Next week

- There is a lecture on Monday!
- There is a tutorial next week.
- Make sure you get familiar with the hardware simulator!
- You should read “Chapter 3” from the forums and keep working on your Assignment 1.
  - Remember there is a milestone due!
- Any questions? Ask on the forum or right now!