

Foundations of Computer Science

Comp109

University of Liverpool

Boris Konev

konev@liverpool.ac.uk

<http://www.csc.liv.ac.uk/~konev/COMP109>

Part 5. Propositional Logic, digital circuits & computer arithmetic

Comp109 Foundations of Computer Science

- [Discrete Mathematics and Its Applications](#), K.H. Rosen, Sections 1.1–1.3.
- [Discrete Mathematics with Applications](#), S. Epp, Chapter 2.

- The language of propositional logic
- Semantics: interpretations and truth tables
- Semantic consequence
- Logical equivalence
- Logic and digital circuits
- Computer representation of numbers & computer arithmetic

Logic is concerned with

- the truth and falsity of statements;
- the question: *when does a statement follow from a set of statements?*

Propositional logic

Propositions

A **proposition** is a statement that can be true or false.
(but not both in the same time!)

- Logic is easy;
- I eat toast;
- $2 + 3 = 5$;
- $2 \cdot 2 = 5$.
- $4 + 5$;
- What is the capital of UK?

- Logic is not easy;
- Logic is easy or I eat toast;

Compound propositions

- More complex propositions formed using **logical connectives** (also called **Boolean connectives**)
- Basic logical connectives:
 1. \neg : negation (read "not")
 2. \wedge : conjunction (read "and"),
 3. \vee : disjunction (read "or")
 4. \Rightarrow : implication (read "if...then")
 5. \Leftrightarrow : equivalence (read "if, and only if,")
- Propositions formed using these logical connectives called **compound propositions**; otherwise **atomic propositions**
- A propositional formula is either an atomic or compound proposition

An *interpretation* I is a function which assigns to any atomic proposition p_i a *truth value*

$$I(p_i) \in \{0, 1\}.$$

- If $I(p_i) = 1$, then p_i is called *true* under the interpretation I .
- If $I(p_i) = 0$, then p_i is called *false* under the interpretation I .

Given an assignment I we can compute the truth value of compound formulas step by step using so-called *truth tables*.

Negation

The negation $\neg P$ of a formula P

It is not the case that P

Truth table:

P	$\neg P$
1	
0	

Conjunction

The conjunction ($P \wedge Q$) of P and Q .

both P and Q are true

Truth table:

P	Q	$(P \wedge Q)$
1	1	
1	0	
0	1	
0	0	

Disjunction

The disjunction ($P \vee Q$) of P and Q

at least one of P and Q is true

Truth table:

P	Q	$(P \vee Q)$
1	1	
1	0	
0	1	
0	0	

Equivalence

The equivalence ($P \Leftrightarrow Q$) of P and Q

P and Q take the same truth value

Truth table:

P	Q	$(P \Leftrightarrow Q)$
1	1	
1	0	
0	1	
0	0	

Implication

The implication ($P \Rightarrow Q$) of P and Q

if P then Q

Truth table:

P	Q	$(P \Rightarrow Q)$
1	1	
1	0	
0	1	
0	0	

So, given an interpretation I , we can compute the truth value of any formula P under I .

- If $I(P) = 1$, then P is called **true** under the interpretation I .
- If $I(P) = 0$, then P is called **false** under the interpretation I .

Example

List the Interpretations I such that $P = ((p_1 \vee \neg p_2) \wedge p_3)$ is true under I .

p_1	p_2	p_3	$\neg p_2$	$(p_1 \vee \neg p_2)$	$P = ((p_1 \vee \neg p_2) \wedge p_3)$

For values see <http://www.csc.liv.ac.uk/~konev/COMP109/lecturelog.html>

Logical puzzles

- An island has two kinds of inhabitants, knights, who always tell the truth, and knaves, who always lie.
- You go to the island and meet A and B.
 - A says “B is a knight.”
 - B says “The two of us are of opposite types.”
- What are A and B?

p : “A is a knight”; and q : “B is a knight”

- Options for A.

- p is true

$$p \Rightarrow q$$

- p is false

$$\neg p \Rightarrow \neg q$$

- Options for B.

- q is true

$$q \Rightarrow \neg p$$

- q is false

$$\neg q \Rightarrow \neg p$$

Truth table

p	q	$\neg p$	$\neg q$	$p \Rightarrow q$	$\neg p \Rightarrow \neg q$	$q \Rightarrow \neg p$	$\neg q \Rightarrow \neg p$
0	0						
0	1						
1	0						
1	1						

Definition Suppose Γ is a finite set of formulas and P is a formula. Then P follows from Γ (“is a semantic consequence of Γ ”) if the following implication holds for every interpretation I :

$$\text{If } I(Q) = 1 \text{ for all } Q \in \Gamma, \text{ then } I(P) = 1.$$

This is denoted by

$$\Gamma \models P.$$

Example

Show $\{(p_1 \wedge p_2)\} \models (p_1 \vee p_2)$.

p_1	p_2	$(p_0 \wedge p_1)$	$(p_0 \vee p_1)$
1	1		
1	0		
0	1		
0	0		

Example

Show $\{p_1\} \not\models p_2$.

p_1	p_2
1	1
1	0
0	1
0	0

Example

Show $\{p_1\} \models (p_1 \vee p_2)$.

p_1	p_2	$(p_1 \vee p_2)$
1	1	
1	0	
0	1	
0	0	

- *Modus Ponens*

Direct proof corresponds to the following semantic consequence

$$\{P, (P \Rightarrow Q)\} \models Q;$$

- *Reductio ad absurdum*

Proof by contradiction corresponds to

$$\{(\neg P \Rightarrow \perp)\} \models P,$$

where \perp is a **special proposition**, which is false under every interpretation.

- *Modus Tollens*

Another look at proof by contradiction

$$\{(P \Rightarrow Q), \neg Q\} \models \neg P$$

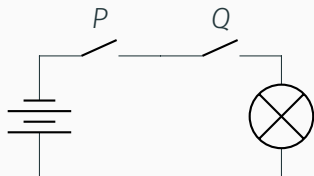
- Case analysis

$$\{(P \Rightarrow Q), (R \Rightarrow Q), (P \vee R)\} \models Q$$

- We have studied proofs as carefully reasoned arguments to convince a sceptical listener that a given statement is true.
 - “Social” proofs
- **Proof theory** is a branch of mathematical logic dealing with proofs as mathematical objects
 - Strings of symbols
 - Rules for manipulation
 - Mathematics becomes a ‘game’ played with strings of symbols
 - Can be read and interpreted by computer

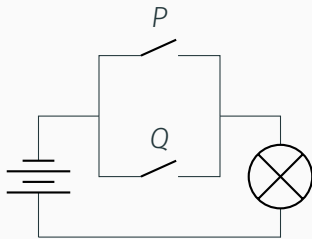
Application: Digital logic circuits

Logic and electric circuits



“in series”

<i>P</i>	<i>Q</i>	light
<i>closed</i>	<i>closed</i>	on
<i>closed</i>	<i>open</i>	off
<i>open</i>	<i>closed</i>	off
<i>open</i>	<i>open</i>	off



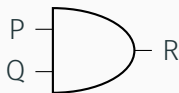
“in parallel”

<i>Q</i>	<i>Q</i>	light
<i>closed</i>	<i>closed</i>	on
<i>closed</i>	<i>open</i>	on
<i>open</i>	<i>closed</i>	on
<i>open</i>	<i>open</i>	off

Modern computers use **logic gates**

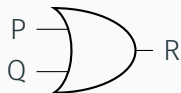
Basic logic gates

AND gate



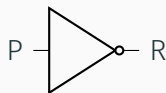
P	Q	R
1	1	1
1	0	0
0	1	0
0	0	0

OR gate



P	Q	R
1	1	1
1	0	1
0	1	1
0	0	0

NOT gate



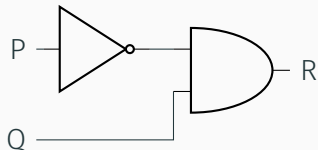
P	R
1	0
0	1

- Never combine two input wires.
- A single input wire can be split partway and used as input for two separate gates.
- An output wire can be used as input.
- *No output of a gate can eventually feed back into that gate.*

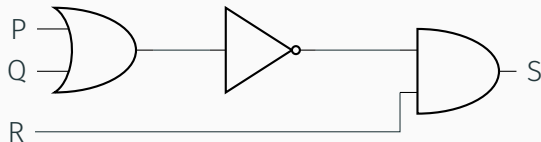
Determining output for a given circuit

Boolean expression

Input signals: $P = 0$ and $Q = 1$



Input signals: $P = 1$, $Q = 1$ and $R = 1$

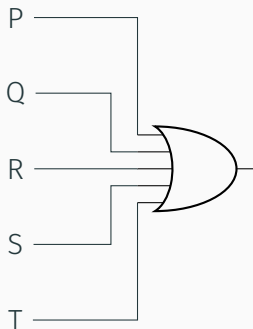
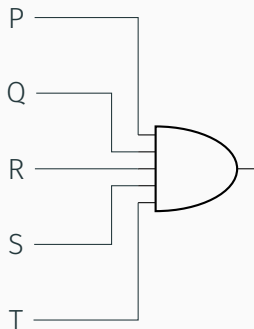


Constructing circuits for Boolean expressions

- $(\neg P \wedge Q) \vee \neg Q$

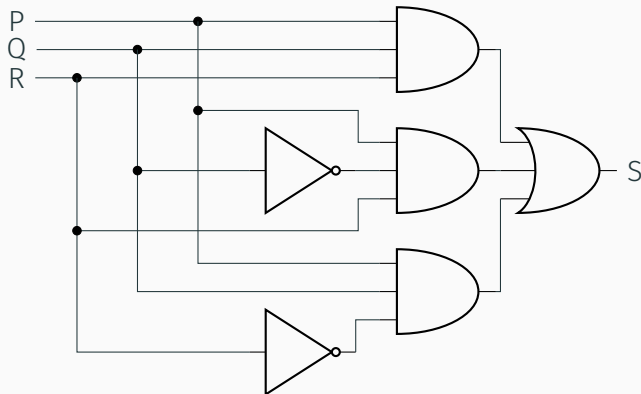
- $((P \wedge Q) \wedge (R \wedge S)) \wedge T$

Multi-input AND and OR gates



Designing a circuit for a given input/output table

Input			Output
P	Q	R	S
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0



$$(P \wedge Q \wedge R) \vee (P \wedge \neg Q \wedge R) \vee (P \wedge Q \wedge \neg R)$$

disjunctive normal form (DNF)

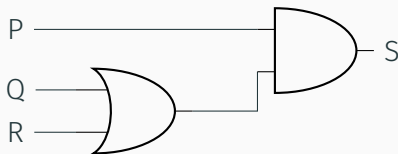
Another example

Input			Output
P	Q	R	S
1	1	1	1
1	1	0	0
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

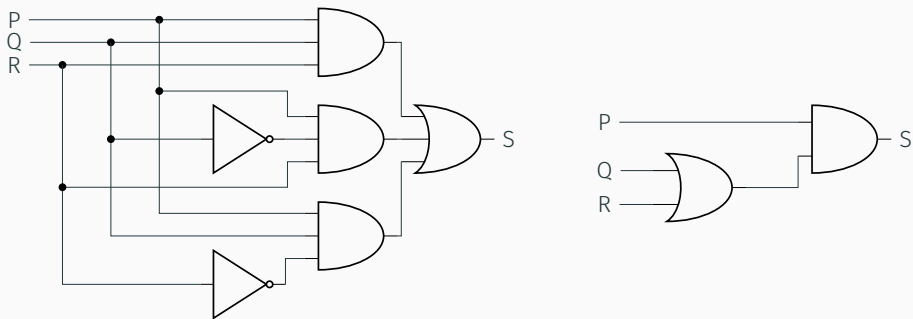
Reopen the first case

Input			Output
P	Q	R	S
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

$$P \wedge (Q \vee R)$$



Circuit equivalence



- Two digital circuits are **equivalent** if they produce the same output given the same inputs.

Definition Two formulas P and Q are called **equivalent** if they have the same truth value under every possible interpretation. In other words, P and Q are equivalent if $I(P) = I(Q)$ for every interpretation I . This is denoted by

$$P \equiv Q.$$

$$(P \wedge Q \wedge R) \vee (R \wedge \neg Q \wedge R) \vee (P \wedge Q \wedge \neg R) \equiv P \wedge (Q \vee R)$$

Theorem The relation \equiv is an equivalence relation on \mathcal{P} .

Proof

- \equiv is reflexive, since, trivially, $I(P) = I(P)$ for every interpretation I .
- \equiv is transitive, since $P \equiv Q$ and $Q \equiv R$ implies $P \equiv R$.
- \equiv is symmetric, since $P \equiv Q$ implies $Q \equiv P$.

Exercises:

- $(P \Rightarrow Q) \equiv (\neg P \vee Q)$
- $\neg(P \Rightarrow Q) \equiv (P \wedge \neg Q)$
- $(P \Leftrightarrow Q) \equiv ((P \Rightarrow Q) \wedge (Q \Rightarrow P))$
- $(P \Leftrightarrow Q) \equiv (\neg P \Leftrightarrow \neg Q)$
- $(P \wedge (P \vee Q)) \equiv P$
- $\neg(P \vee (\neg P \wedge Q)) \equiv (\neg P \wedge \neg Q)$

The following equivalences can be checked by truth tables:

■ Associative laws:

$$(P \vee (Q \vee R)) \equiv ((P \vee Q) \vee R),$$

$$(P \wedge (Q \wedge R)) \equiv ((P \wedge Q) \wedge R);$$

■ Commutative laws:

$$(P \vee Q) \equiv (Q \vee P), \quad (P \wedge Q) \equiv (Q \wedge P);$$

■ Identity laws:

$$(P \vee \perp) \equiv P, \quad (P \vee \top) \equiv \top, \quad (P \wedge \top) \equiv P, \quad (P \wedge \perp) \equiv \perp;$$

■ Distributive laws:

$$(P \wedge (Q \vee R)) \equiv ((P \wedge Q) \vee (P \wedge R))$$

$$(P \vee (Q \wedge R)) \equiv ((P \vee Q) \wedge (P \vee R));$$

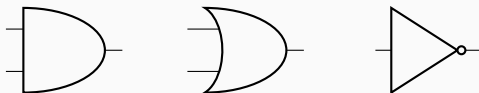
■ Complement laws:

$$P \vee \neg P \equiv \top, \neg \top \equiv \perp, \neg \neg P \equiv P, P \wedge \neg P \equiv \perp, \neg \perp \equiv \top;$$

■ De Morgan's laws:

$$\neg(P \vee Q) \equiv (\neg P \wedge \neg Q), \neg(P \wedge Q) \equiv (\neg P \vee \neg Q).$$

- A function $F : \{0, 1\}^k \rightarrow \{0, 1\}$, where $k \in \mathbb{Z}^+$ is **arity** of F , is called a **Boolean function**
- Any Boolean function can be expressed as a combination of \wedge , \vee , \neg



- Do we need all three types of gates?

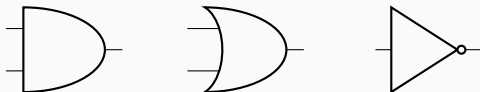
Boolean functions of arity 2

P	Q								
1	1	0	1	0	1	0	1	0	1
1	0	0	0	1	1	0	0	1	1
0	1	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0	0	0

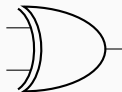
P	Q								
1	1	0	1	0	1	0	1	0	1
1	0	0	0	1	1	0	0	1	1
0	1	0	0	0	0	1	1	1	1
0	0	1	1	1	1	1	1	1	1

Logic gates

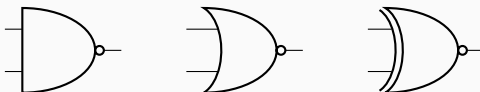
■ AND, OR, NOT



■ XOR



■ NAND, NOR, XNOR



Universality of NAND and NOR

- NAND (AKA Sheffer stroke)
- and NOR (AKA Pierce arrow)

$$P \mid Q = \neg(P \wedge Q)$$

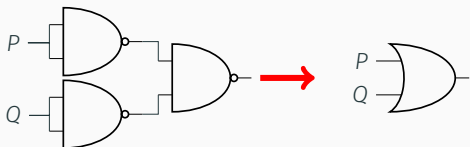
$$P \downarrow Q = \neg(P \vee Q)$$

are universal:

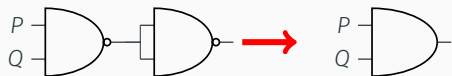
$$\neg P \equiv P \mid P$$



$$P \vee Q \equiv (P \mid P) \mid (Q \mid Q)$$



$$P \wedge Q \equiv (P \mid Q) \mid (P \mid Q)$$



Application: Number systems and circuits for addition

Positional system: multiply each digit by its place value

- Decimal notation:

$$4268_{10} = 4 \cdot 10^3 + 2 \cdot 10^2 + 6 \cdot 10^1 + 8 \cdot 10^0$$

- Binary notation

$$\begin{aligned} 1100\ 0111_2 &= 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 128 + 64 + 0 + 0 + 0 + 4 + 2 + 1 = 199_{10} \end{aligned}$$

*Here indices 10 and 2 are used to highlight the **base** of the number system*

Convert decimal numbers to binaries: divide by 2

Rule: divide repeatedly by 2, writing down the remainder from each stage from **right to left**.

Example:	$533/2 = 266$	remainder = 1
	$266/2 = 133$	remainder = 0
	$133/2 = 66$	remainder = 1
	$66/2 = 33$	remainder = 0
	$33/2 = 16$	remainder = 1
	$16/2 = 8$	remainder = 0
	$8/2 = 4$	remainder = 0
	$4/2 = 2$	remainder = 0
	$2/2 = 1$	remainder = 0
	$1/2 = 0$	remainder = 1

$$533_{10} = 1000010101_2$$

- If you know powers of 2, continually subtract largest power value from the number

$$\begin{aligned}123_{10} &= 64 + (123 - 64) = 64 + 59 \\ &= 64 + 32 + (59 - 32) = 64 + 32 + 27 \\ &= 64 + 32 + 16 + (27 - 16) = 64 + 32 + 16 + 11 \\ &= 64 + 32 + 16 + 8 + (11 - 8) = 64 + 32 + 16 + 8 + 3 = \\ &= 64 + 32 + 16 + 8 + 2 + (3 - 2) \\ &= 64 + 32 + 16 + 8 + 2 + 1 = \\ &= 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = \\ &= 1111011_2\end{aligned}$$

Binary addition

$$0_2 + 0_2 = 0_2$$

$$0_2 + 1_2 = 1_2$$

$$1_2 + 0_2 = 1_2$$

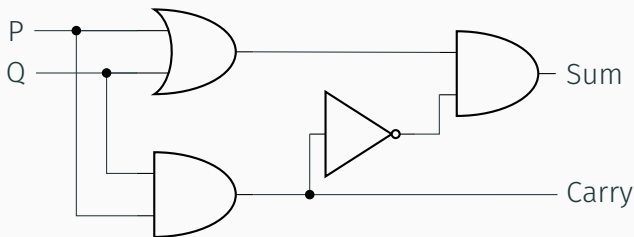
$$1_2 + 1_2 = 10_2$$

$$\begin{array}{r} \\ \\ + \\ \hline 1 \end{array}$$

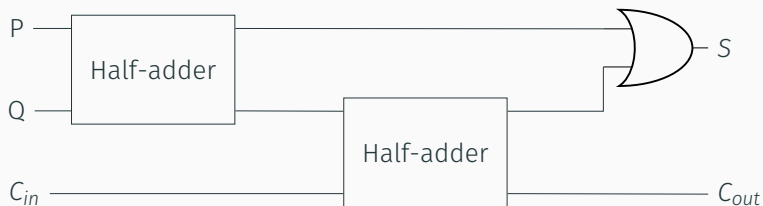
Half-adder

$$\begin{array}{r} \\ \\ + \\ \hline 1 \end{array}$$

P	Q	Carry	Sum
1	1	1	0
1	0	0	1
0	1	0	1
0	0	0	0

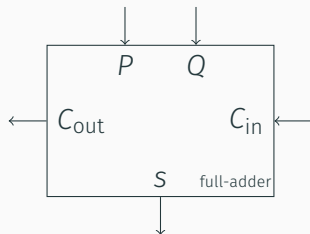
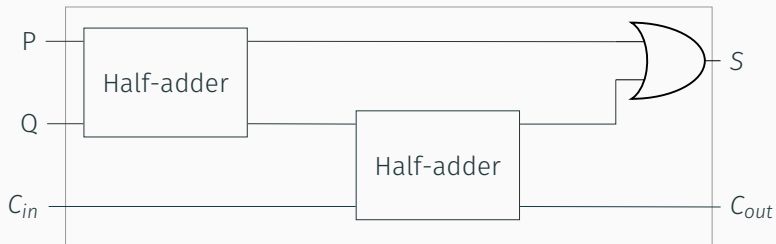


Full-adder



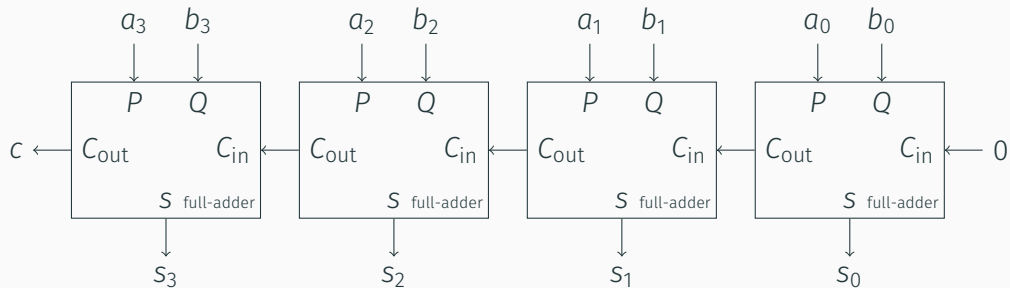
P	Q	C_{in}	C_{out}	S
1	1	1	1	1
1	1	0	1	0
1	0	1	1	0
1	0	0	0	1
0	1	1	1	0
0	1	0	0	1
0	0	1	0	1
0	0	0	0	0

'Black box' notation



4-bit adder

$$\begin{array}{r} \\ + \\ \hline c \\ S_3 \\ S_2 \\ S_1 \\ S_0 \end{array}$$



Computer representation of negative integers

- Typically a fixed number of bits is used to represent integers: 8, 16, 32 or 64 bits
 - **Unsigned** integer can take all space available
- Signed integers

- **Leading sign**

$$0\ 000\ 0001_2 = 1_{10}$$

$$1\ 000\ 0001_2 = -1_{10}$$

but then

$$1\ 000\ 0000_2 = -0_{10} \text{ (?!)}$$

- **Two's complement:**
given a positive integer a , the **two's complement of a relative to a fixed bit length n** is the binary representation of

$$2^n - a.$$

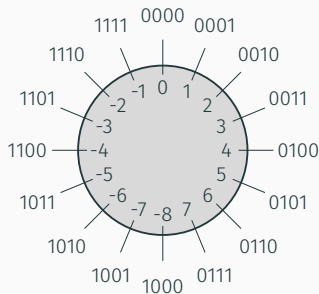
Example: 4-bit two's complement (n=4)

- $a = 1$, two's complement: $2^4 - 1 = 15 = 1111_2 = -1$
- $a = 2$, two's complement: $2^4 - 2 = 14 = 1110_2 = -2$
- $a = 3$, two's complement: $2^4 - 3 = 13 = 1101_2 = -3$
- ...
- $a = 8$, two's complement: $2^4 - 8 = 8 = 1000_2 = -8$

Properties

- Positive numbers start with 0, negative numbers start with 1
- 0 is always represented as a string of zeros
- -1 is always represented as a string of ones

Example: 4-bits



- The number range is split unevenly between +ve and -ve numbers
- The range of numbers we can represent in n bits is -2^{n-1} to $2^{n-1} - 1$

- Easy for computers
- Example: $2+3$

$$\begin{array}{r} 0010 \\ + 0011 \\ \hline 0101 \end{array}$$

- A carry that goes off the end can often be ignored
- Example: $-1 + -3$

$$\begin{array}{r} 1111 \\ + 1101 \\ \hline 11100 \end{array}$$

- Treat as an addition by negating second operand
- Example: $4 - 3 = 4 + (-3)$

$$\begin{array}{r} \\ + \\ \hline \\ \end{array}$$

- Example: $4 + 7$

$$\begin{array}{r} 0\ 1\ 0\ 0 \\ +\ 0\ 1\ 0\ 1 \\ \hline 1\ 0\ 0\ 1 \end{array}$$

- The correct result 9 is too big to fit into 4-bit representation
- Testing for overflow:
If both inputs to an addition have the same sign, and the output sign is different, overflow has occurred
 - Overflow cannot occur if inputs have opposite sign.

Two's complement and bit negation

Example $n = 4$

- $2^4 - a = ((2^4 - 1) - a) + 1$.
- The binary representation of $(2^4 - 1)$ is 1111_2
- Subtracting a 4-bit number a from 1111_2 just switches all the 0's in a to 1's and all the 1's to 0's.

For example,

$$\begin{array}{r} 1\ 1\ 1\ 1 \\ -\ 1\ 0\ 0\ 1 \\ \hline 0\ 1\ 1\ 0 \end{array}$$

- So, to compute the two's complement of a , flip the bits and add 1.

Example

- Find the 8-bit two's complement of 19.

- Conversely, observe that

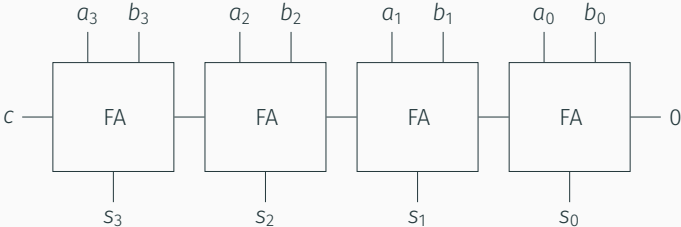
$$2^n - (2^n - a) = a$$

so to find the decimal representation of the integer with a given two's complement

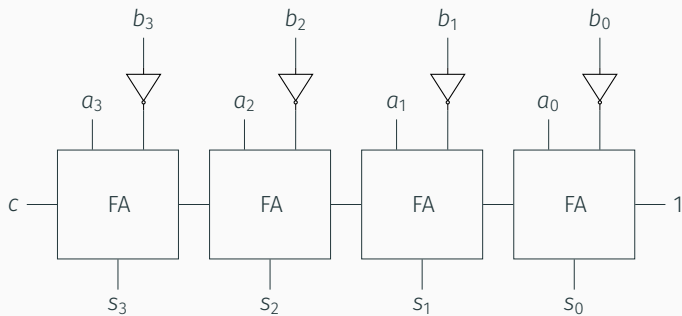
- Find the two's complement of the given two's complement
- Write the decimal equivalent of the result

Example: Which number is represented by 1010 1001?

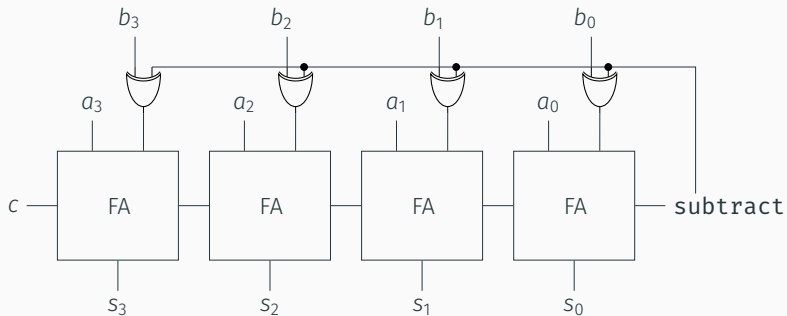
Recall: 4-bit adder



- Implementing $a + b$ as the sum of a and two's complement of b



4-bit adder / subtractor



■ When **subtract** is 0: 

■ When **subtract** is 1: 

Integer types in high-level languages

E.g. Java has the following integer data types, using 2's complement:

byte	8-bit	-128 to $+127$
short	16-bit	$-32\,768$ to $+32\,767$
int	32-bit	$-2\,147\,483\,648$ to $+2\,147\,483\,647$
long	64-bit	-2^{63} to $+2^{63} - 1$

- It is not always possible to express numbers in integer form.
- Real, or floating point numbers are used in the computer when:
 - the number to be expressed is outside of the integer range of the computer, like

$$3.6 \times 10^{40} \text{ or } 1.6 \times 10^{-19}$$

- or, when the number contains a decimal fraction, like

123.456

Scientific notation (AKA standard form)

The number is written in two parts:

- Just the digits (with the decimal point placed after the first digit), followed by
- $\times 10$ to a power that puts the decimal point where it should be (i.e. it shows how many places to move the decimal point).

$$123.456 = 1.23456 \times 10^2$$

In this example, 123.456 is written as 1.23456×10^2 because

$$123.456 = 1.23456 \times 100 = 1.23456 \times 10^2$$

Likewise, fractions can be represented base 2.

$$\begin{aligned}10.01_2 &= 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} \\ &= 1 \times 2 + 0 + 0 + 1 \times 0.25 \\ &= 1.25_{10}\end{aligned}$$

Scientific representation: $10.01_2 = 1.001 \times 2^1$

Note: in binary, for any non-zero number the leading digit is always 1

Computer representation

To represent a number in scientific notation:

- The sign of the number.
- The magnitude of the number, known as the **mantissa** or **significand**
- The sign of the exponent
- The magnitude of the **exponent**

Example: eight characters

S E E M M M M M

- **S** is the sign of the number
- **EE** are two characters encoding the exponent
 - both sign and magnitude
- **MMMMM** are five characters for the mantissa

- IEEE standard for **floating-point arithmetic**
- Implemented in many hardware units
- Stipulates computer representation of numbers
- For binary:
 - 16 bit **half precision** numbers: 5 for exponent, 11 for mantissa
 - 32 bit **single precision** numbers: 8 for exponent, 24 for mantissa
 - 64 bit **double precision** numbers: 11 for exponent 53 for mantissa
 - 128 bit **quadruple precision** numbers: 15 for exponent 113 for mantissa
 - 256 bit **octuple precision** numbers: 19 for exponent 237 for mantissa