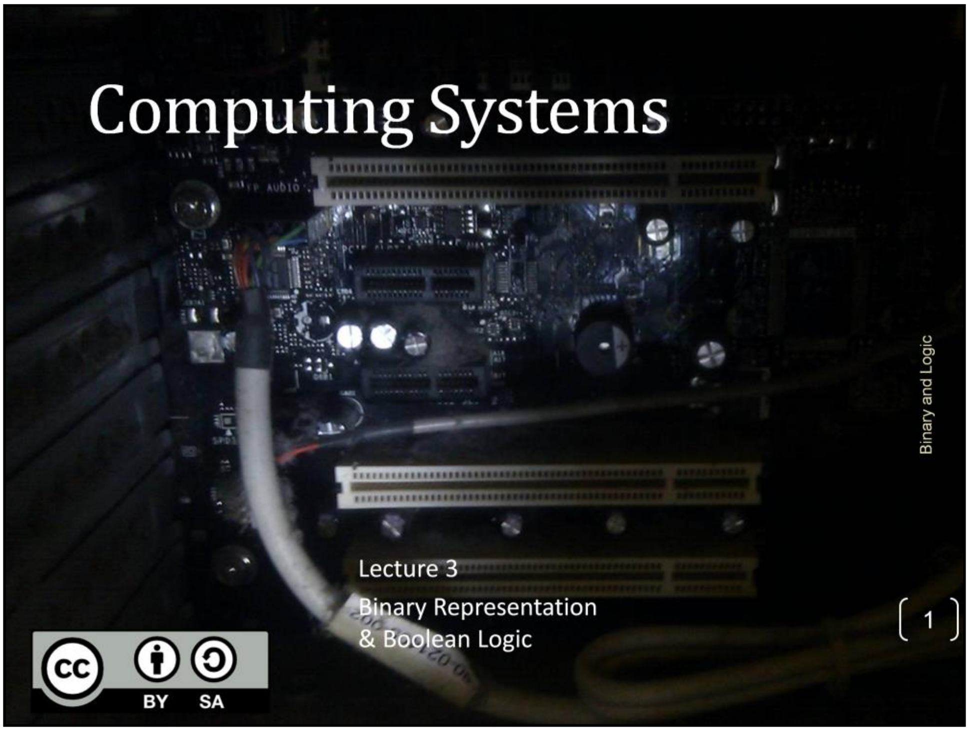
Computing Systems



Lecture 3

Binary Representation
& Boolean Logic

Binary and Logic

1

Zeroes and Ones (Again, again)

- Introduced binary representation
  - How a single transistor can form a simple electronic switch (one ***bit***)
  - How binary can be used to represent numbers and symbols
- Now to build on that...
  - Representing **negative** and **real** numbers
  - More **complex logical operations**
  - How computers **add** numbers

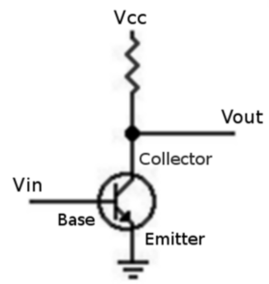
Logic

- Logic circuits are the building blocks of computers
- Boolean values are **True/False** values – equivalent to the **On/Off** or **1/0** values of binary
- Boolean operations take some number **true/false** inputs, and give a **true/false** output

A Simple Transistor Inverter

- Simple inverter circuit
- If Vin is ***on***, Vout will be ***off***
- Vin ***off***, Vout ***on***
- A ***NOT*** gate

Input	Output
0	1
1	0



A simple transistor invertor or NOT gate. A transistor whereby a signal at the base switches the transistor on / off. Input signal is ON there is a connection between V_{out} and the emitter the emitter is at the low voltage value therefore V_{out} will have the low value. When Input signal is OFF there is no connection to the emitter, therefore the V_{out} will be based on the V_{cc} input value, the HIGH input voltage. The presence of the resistor (the wiggly line) makes this circuit work.

This is a NOT gate.

The truth table is shown above.

Transistors to Logic Gates

- Using different circuits of transistors & resistors, a range of basic *logic gates* can be built up
- Inverter is a **NOT** gate
  - 0 input becomes 1 output and vice versa
- Other gates take two (or more) inputs
  - E.g. **AND** gate produces 1 output only if ***both*** inputs are **1**

- NOT 
- AND 

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

A	B	A·B
0	0	0
0	1	0
1	0	0
1	1	1

Small circle is used to indicate that the output is inverted.

Not A – **A** with a **bar** above it.

<u>AND Gate</u>

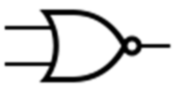
A.B – means **A and B**

Two inputs and 1 output

- OR

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

- NOR



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

OR → A+B

NOR → A+B with a bar above.

Similar symbol but has a circle indication NOT.

- NAND

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

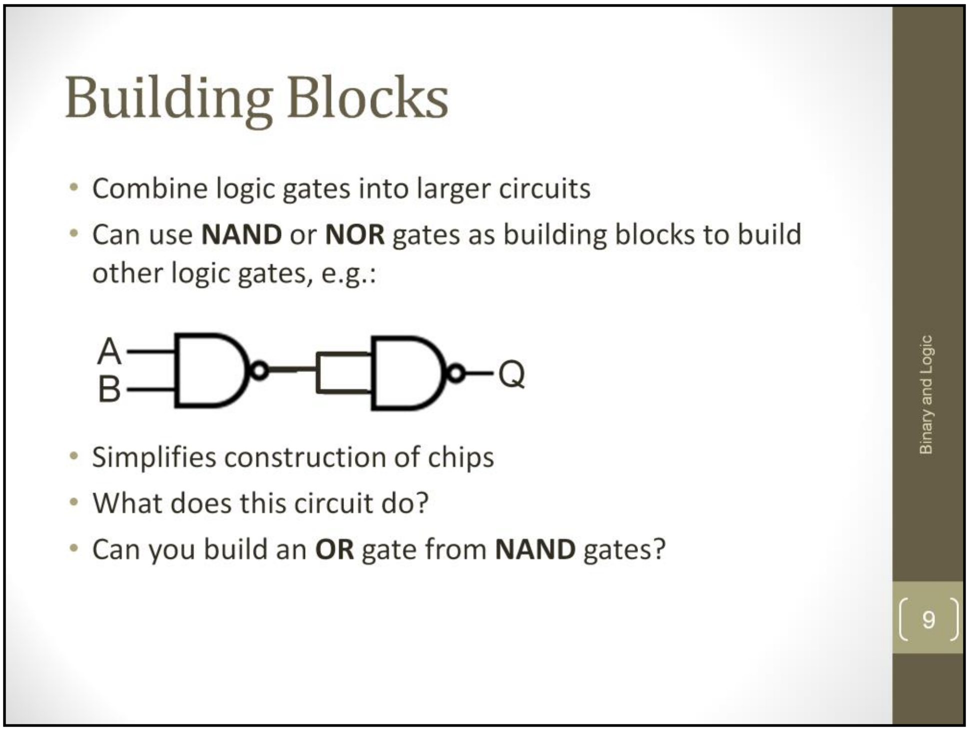
- XOR (Exclusive OR)

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

NAND → NOT AND

Opposite of the **AND** gate

XOR (sometimes known as **EOR**) → output is 1 when **A or B** is 1 but **not** when **BOTH** are 1.



Use a TRUTH table

NAND gate.

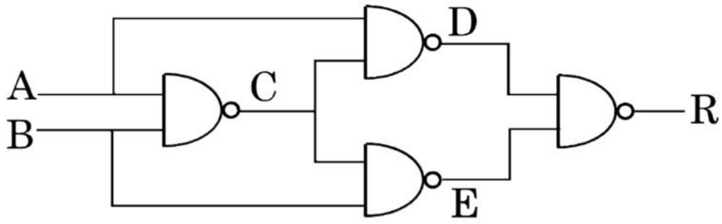
Output a **zero** if **A and B** are BOTH **1,** all other **outputs** will be **1.**

The **output** will then be fed to **BOTH** inputs at the next stage. E.g. if the values are both **zero** the output will be **1**, if both are **1** the output will be **zero**. If both **INPUTS** have the same value, this is acting as an **invertor / NOT** gate.

It creates a **AND** circuit from **TWO NAND** logic gates.

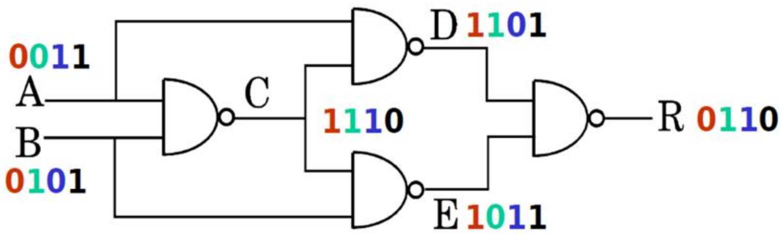
Week 3 Slide 10 quick quiz

The diagram below shows NAND gates used to build another logic gate.
Which gate does it build?
Join ‘Room **642124**’ in the ‘**Socrative**’ app and give your answer.



Week 3 Slide 10 quick quiz solution

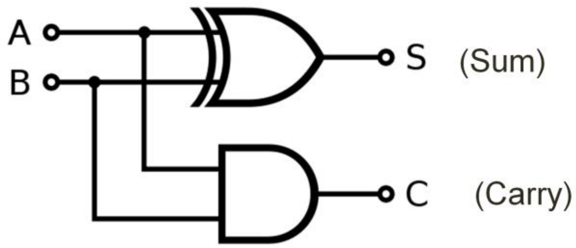
- Construct an input / output truth table and verify that it is the same as the gate in your answer.



Binary and Logic

11

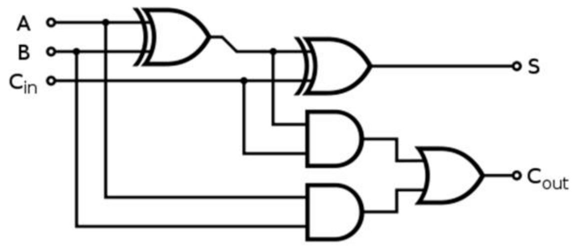
Half Adder



A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Half Adder, Binary, ONE BIT.

Full Adder



A	B	C_{in}	C_{out}	S	C_{in}	C_{out}	S
0	0	0	0	0	1	0	1
0	1	0	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	0	1	1	1

Binary and Logic

13

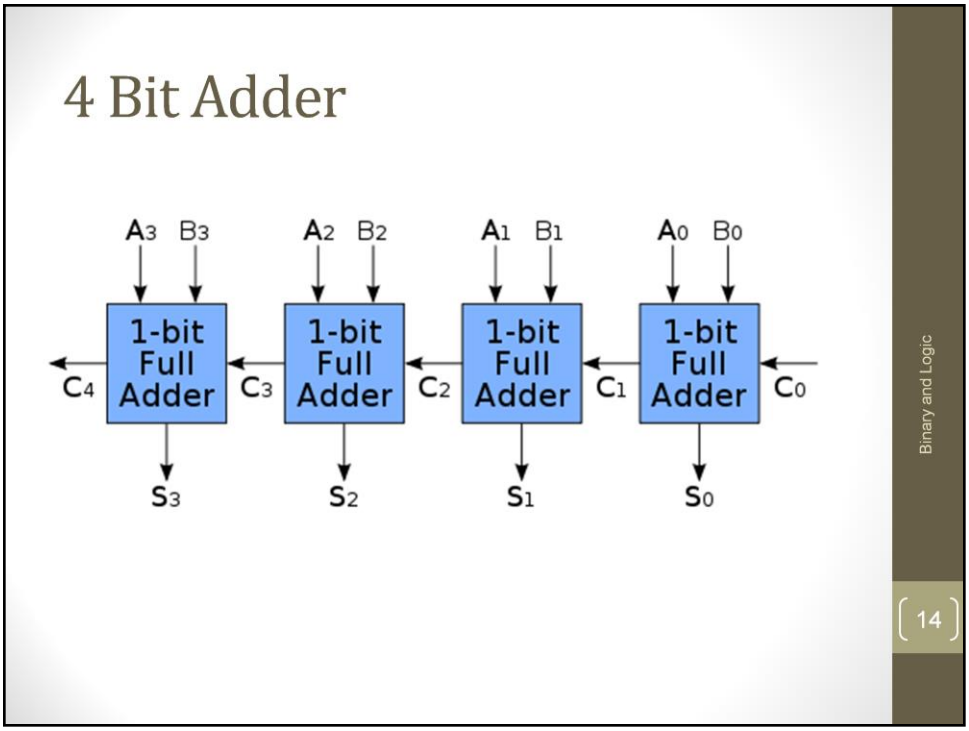
Full Adder – 3 INPUTs

A, B and C the carry over from the previous column.

Step 1:

Add A, B and C_{in} of zero to give C_{out} and Sum

This will add TWO bits



Least significant bit on the right.

Eights, Four, Twos, Units

We can add values up to 15.

By chaining them together we can carry over the extra bits to form a calculation.

Numbers: Basic Terms

- **Integers**: The set of whole numbers, e.g.
  ... -3, -2, -1, 0, 1, 2, 3, ...
- **Real** numbers: Numbers which may have a fractional element, e.g.
  -3.2, 0.01, 3.14,...
  - **Repeating** (**recurring**) numbers: Real number where the fractional element contains infinitely repeating digits, e.g. 1/3 = 0.33333...
- **Irrational Numbers**: Numbers with infinitely long non-repeating sequences, eg PI
  π = 3.14159265358979323846264 33...

Binary and Logic

15

*Π – Ratio that r*elates the radius of a circle to the circumference. A never ending non repeating sequence.

Unsigned Representations

- Programming languages such as **C** or **C++** allow developers to use ***unsigned*** integers
- **Positive only** numbers
- Range **0** to **2n-1**
  - where **n** is the number of **bits** used to represent the number
  - **n** typically equals **8 (byte), 16, 32 or 64**
- Negative numbers?

Week 3 Slide 17 quick quiz

- What range can an ***unsigned*** 8 bit number represent?
- Join ‘Room 642124’ in the ‘Socrative’ app and give your answer.

Binary and Logic

17

Range 0 to 2^n-1

N has 8 bits → range 0 to 2^n-1 → 0 to 255

Therefore an unsigned 8 bit number can represent 0 to 255.

Sign and Magnitude

- Simple approach: Use the first bit of any number to indicate whether it is positive or negative
- First bit is sign (+ or -) bit, remainder is the magnitude (size) of the number
- E.g. with 4 bits:

  0011 = +3, 1011 = -3

  0000 = +0, 1000 = -0

range for 8 bit sign & magnitude → -127 to +127

2^7 -1 → 128-1=127 therefore range is -127 to +127

Week 3 Slide 19 quick quiz

- What is the range for 8 bit ***sign & magnitude***?
- Join ‘Room 642124’ in the ‘Socrative’ app and give your answer.

Binary and Logic

19

range for 8 bit sign & magnitude → -127 to +127

2^7 -1 → 128-1=127 therefore range is -127 to +127

Problems?

- Simple arithmetic becomes complicated with sign & magnitude representations
- E.g. one plus minus one:

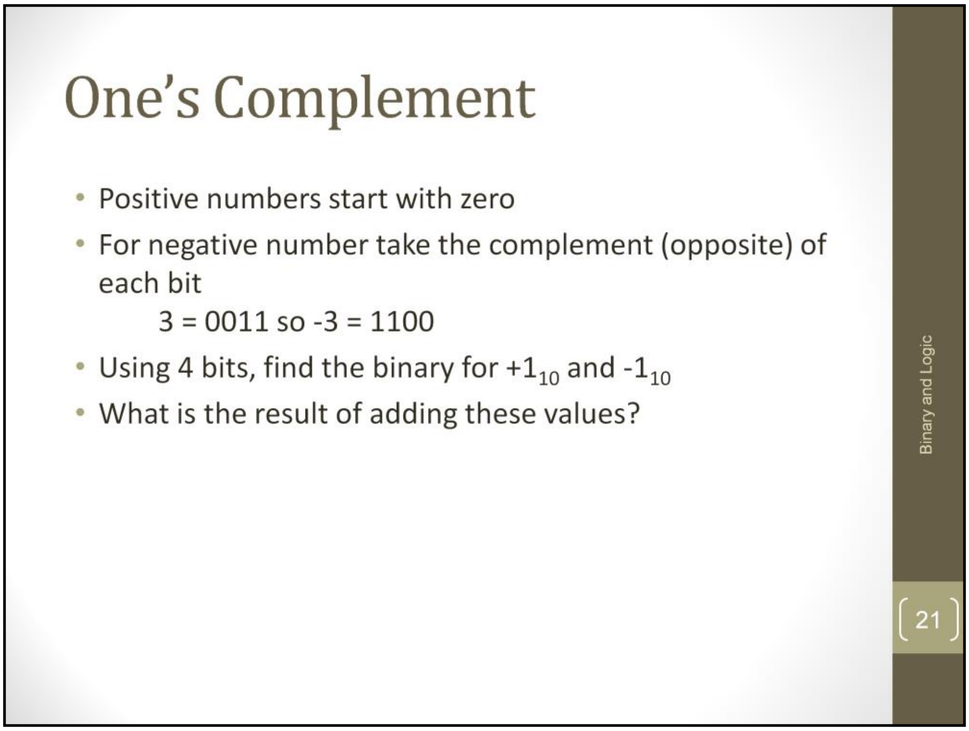
  $1_{10} + (-1_{10})$

  $0001 + 1001 = 1010 = -2_{10}$
- Two representations for zero
  - 0000 and 1000
  - Extra logic required to test when two numbers are equal, in event both are zero

Cannot easily feed in numbers to a Full Adder Circuit and get a correct answer. As the Binary addition doesn’t reflect what is actually happening with the numbers they are supposed to represent. i.e.

$1_{10} + (-1_{10})$
$0001 + 1001 = 1010 = -2_{10}$

Does not work with binary.



Complement for each bit take the opposite.

What is the result of adding these values?

$+1_{10}$ and -1_{10} → 0001 + 1110 = zero 1111

Does give correct result, but we still have two values for zero. We need to add the carry value back in.

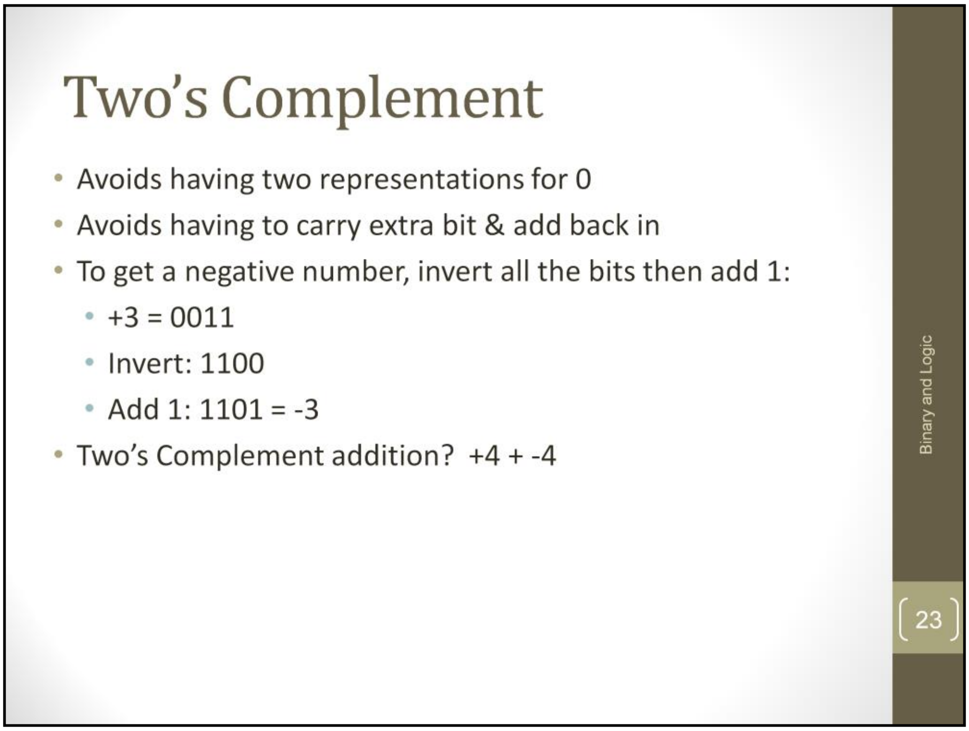
One's Complement

- Still two values for zero: 0000 & 1111
- But addition works as it should...
  ...almost. Need to carry and add back in if the result overflows:

	Binary	Decimal	
	1110	-1	
+	0010	+2	
1	0000	0	Incorrect
		+1	Add carry
0	0001	1	Correct!

Binary and Logic

22



Two's Complement addition? +4 + -4

Leave as an exercise

Two's Complement

- Only one zero: 0000
- 4bit:

0111 = 7	**0000 = 0**	**1001 = -7**	**1000 = -8**

- Easy way to find the negative of a number:

	Example 1	Example 2
1. Starting from the right, find the first '1'	010100**1**	0101**1**00
2. Invert all of the bits to the left of that one	**101011**1	**1010**100

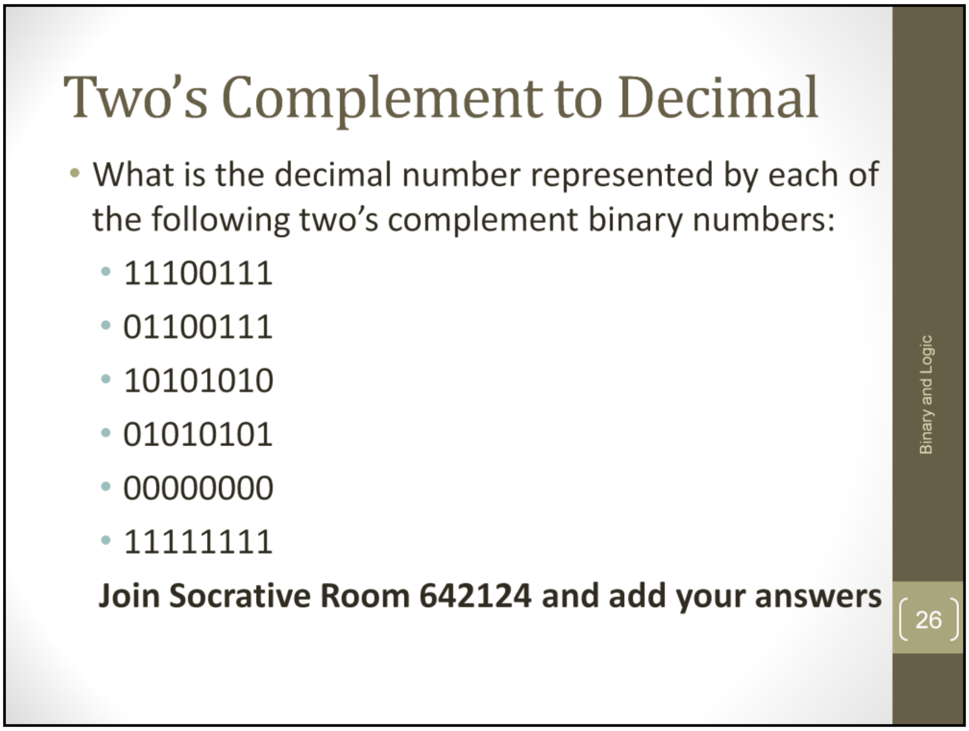
Can get an extra negative value → -8

Two's complement 11100111 to decimal

- Leading 1, so a negative number
- Negate to find the magnitude

1. Starting from the right, find the first '1'	1110011**1**
2. Invert all of the bits to the left of that one	**0001100**1
3. Convert to decimal	16 + 8 + 1 = 25

- Solution: -25



If the leading value is a zero then it is a positive number, if it is a 1 then it is a negative number.

For negative numbers: Start from right, find the first 1, invert all the bits to the left of that 1. Convert to decimal.

Two's Complement to Decimal solution

128 + 64 + 32 + 16 + 8 + 4 + 2 +1

- What is the decimal number represented by each of the following two's complement binary numbers:
  - 11100111 → **00011001 → 16 + 8 + 1 → - 25**
  - 01100111→ **64 + 32 + 4 + 2 + 1 → + 103**
  - 10101010 → **01010110 → 64 + 16 + 4 + 2 → -86**
  - 01010101 → **64 + 16 + 4 + 1 → 85**
  - 00000000 → **0**
  - 11111111 →**00000001 → - 1**

Fractions: Fixed Point Binary

- Decimal point is set at fixed position
  - E.g. With 16 bits, could have 12 bits then point then remaining 4 bits
- Significance of digits after the point:
  $2^{-1}, 2^{-2}, 2^{-3}\ldots 2^{-n}$ (1/2, 1/4, 1/8, ...)
- What are the *largest* and *smallest* values that can be stored in an unsigned 16 bit (12.4) representation?
- $2^{12} = 4096$, $2^{-4} = 1/2^4 = 0.0625$

Binary and Logic

28

Note: largest number in unsigned 12.4 is actually 4096.9375 (2^12 + (1-0.0625))

Fixed Point Challenge

Decimal	Binary
0.5	0.1
0.25	0.01
0.125	0.001
0.0625	0.0001
0.03125	0.00001
0.015625	0.000001
0.0078125	0.0000001
0.00390625	0.00000001

With 8 bits after the point, binary for:

a) 0.75_{10} =
b) 0.3125_{10}=
c) 0.1_{10}=

Add your answers to 'Socrative Room 642124'

Answers are: a) 0.11, b) 0.0101 and c) with 8 bits after the point, the closest we can get is 0.00011001_2 = 0.09765625 (0.0625 + 0.03125 + 0.00390625) –not all fractional numbers can be represented – sometime we have to approximate. Some numbers easily written in decimal cannot be represented with binary.

Floating Point Numbers

- **Fixed point** representations place quite *small limits* on
  - the *size* of the *largest numbers* that can be represented
  - the *size* of the *smallest fraction* that can be represented
- **Floating point** numbers allow computers to represent both *very large* and *very small* numbers
  - but still have *issues* representing some numbers

Binary and Logic

30

We are going to use a number to represent where the point is.

Might be an idea to take a break here... The next few slides get heavy going!

Decimal Floating Point Numbers

- Can represent numbers using a floating point system, in decimal this might be:

Number		Mantissa		Exponent
-293.87	=	-0.29387	x	10^3
0.0000983	=	0.983	x	10^{-4}

- Significant digits are the ***mantissa*** or ***significand***
- Multiplied by 10 to the power of some ***exponent*** here the exponents are 3 and -4

Binary and Logic

31

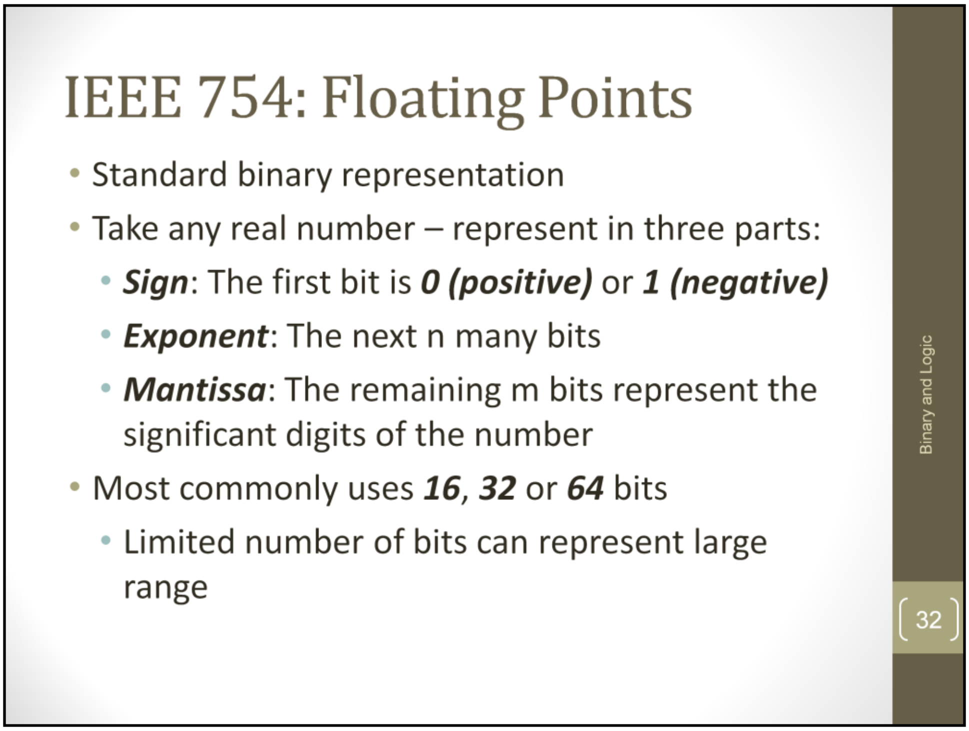
Significand → the significant digits.

Mantissa * Exponent gives us our number.

We are interested in the exponent, the 3 and -4 are what we are interested in. We can now represent very large and small numbers using only the significant digits.

The 3 → move the point 3 places to the right.

The -4 → move the point 4 places to the left.



IEEE 754 → standard

Relatively limited number of bits can represent a very large range of numbers.

32 bit IEEE Floating Point

- A ***float*** variable in C/C++

Sign	Exponent	Mantissa	Bias	Precision
1 bit	8 bits	23 bits	127	24 bit

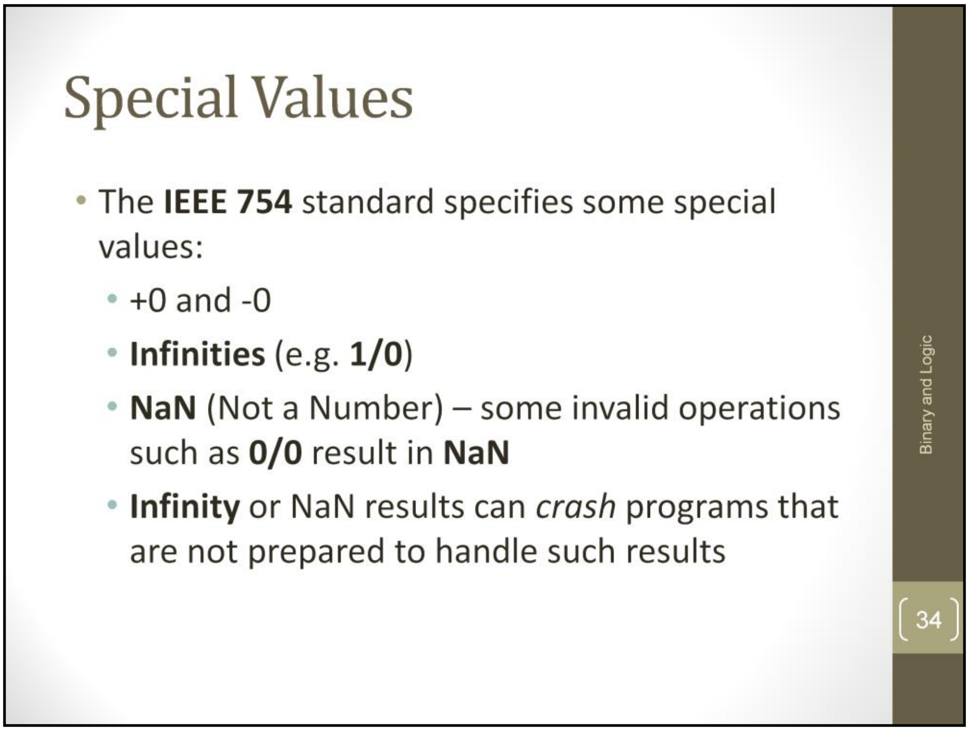
- 8 bit exponent is ***unsigned***, but has bias value added – to give a range of **-127...+128**
  - Allows for very small or very large numbers

For computer representation – use binary number.

1 Bit for sign(number positive or negative), 8 bits for exponent(very large or very small number, how much do we want to shift the decimal point), and 23 bits for Mantissa(significant digits of the number).

Exponent itself is unsigned. How do we get a negative value for the exponent.

Use a Bias value to get a negative value for the exponent, subtract the bias from the exponent value to get a negative range. If the exponent is zero, it means to the power of -127 and if the exponent value is all 1s it is 255 – 127(bias) = 128.



Depending on timing, this is also a good point to take a break... Next slide is the start of the section on logic

Further Reading

- See *Module References* section on Blackboard
  - Lots of links
- PCH, 2.1 – 2.3, 4.1 – 4.3, 4.7 – 4.10
  - Remainder of Chapters 2 & 4 for interest only

Next Week

- ***Required** Reading*
- HCW Part 3
  - Overview (p82-93)
  - Chapter 7 (p94-103)
- Additional Reading
  - ECS Chapters 6 & 10

Image Credits:

Title Image and Transistor Inverter circuit CC-BY-SA Daniel Livingstone, 2011 (Inverter circuit drawn using Klunky)

Logic Gates, Half Adder and Full Adder circuits all public domain images, sourced from Wikipedia and Wikimedia Commons

4 Bit Adder, CC-BY-SA en:User:Cburnett, http://en.wikipedia.org/wiki/File:4-bit_ripple_carry_adder.svg