Learning Aims: Binary arithmetic

- Use sign and magnitude to represent negative numbers in binary
- Use two's complement to represent negative numbers in binary
- Add and subtract binary integers
- Represent fractions in fixed point binary

Signed vs unsigned binary

Signed binary number can be positive or negative vs **unsigned binary number** can **only be positive**.

If a binary number is signed, then the first bit (the most significant bit, the leftmost) will be a 1 if it is a negative number, or a 0 if it is positive, e.g. 10010111 will be a negative number, whilst 01011011 will be positive.

There are a number of different ways to represent a negative number**:**

- **Sign and magnitude**
- **Two's complement**

In an examination you will be told what form the binary is in, whether it is signed or unsigned, or if it is in two's compliment or sign and magnitude. If you are not told anything, then treat it as an unsigned binary number.

Sign and magnitude

- Sign and magnitude is the simplest way of representing negative numbers in binary.
- The most significant bit (MSB) is simply used to represent the sign: 1 means the number is negative, 0 means it is positive.
- So to represent −4 in 8-bit binary you would write out the 4 in binary, 00000100, then change the MSB to a 1, creating 10000100.
- In another example, −2 would be 10000010 in the sign and magnitude notation.
- -103 would be:

Column Value	Sign bit	64	32	16	8	4	2	1
Binary number	1	1	1	0	0	1	1	1

Sign and magnitude

- This notation isn’t used very often as using the MSB as a sign bit means that the largest number that can be represented using 8 bits is 127, much less than 255.
- It also makes it harder to do calculations as different bits mean different things; some represent numbers, others represent signs.
- Also, their value of zero is represented twice as both positive and negative 0.
- MSB notation can represent numbers in the range −127 to 127.

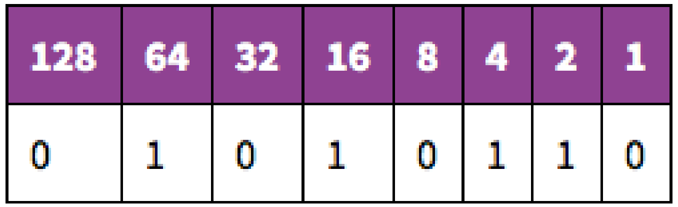
Two’s complement

- Two’s complement is a much more useful way of representing binary numbers as it allows you to use negative numbers without reducing the range of numbers you can use (being able to represent −128 to 127).
- This method makes the MSB a negative value.

- The easiest way to show a negative number using two’s complement is to write it out as a positive number using the usual binary method. Then, starting at the right-hand side, leave every bit up to and including the first 1 alone, but invert all the bits after this.

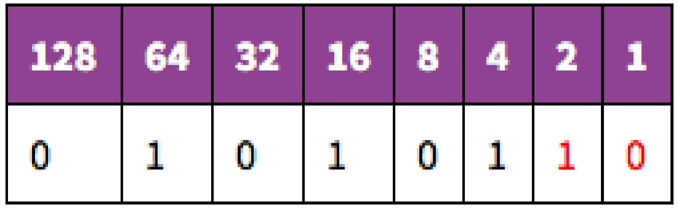
Column Value	-128	64	32	16	8	4	2	1

• For example, to show −86 in binary, first work out 86. 86 is 01010110 because 64 16 4 2 equals 86:



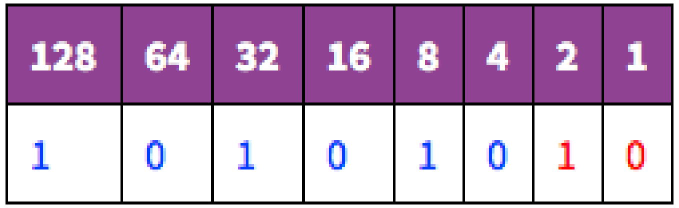
128	64	32	16	8	4	2	1
0	1	0	1	0	1	1	0

• Then start at the right-hand side and leave everything alone up to and including the first 1:



128	64	32	16	8	4	2	1
0	1	0	1	0	1	1	0

• Finally invert everything after this, so all the 1s become 0s and vice versa:



128	64	32	16	8	4	2	1
1	0	1	0	1	0	1	0

Column Value	-128	64	32	16	8	4	2	1
Binary number	1	0	1	0	1	0	1	0

Two’s complement

- Another way
- To store -103 we record -128 + 25 or

Column Value	-128	64	32	16	8	4	2	1
Binary number	1	0	0	1	1	0	0	1

Representing Positive number using 2's complement

32	16	8	4	2	1
0	1	0	0	1	0

MSB

IF the MSB is a 0 then it will be a positive number

16 + 2 = 18

-32	16	8	4	2	1
1	1	0	0	1	0

MSB

IF the MSB is a 1 then it will be a negative number

-32 + 18 = -14

Adding integers in binary

- In binary when we add 0s and 1s we have the following possible outcomes.

	Carry	Sum
0 + 0	0	0
0 + 1	0	1
1 + 0	0	1
1 + 1	1	0
1 + 1 + 1	1	1

Example

00001011 + 10011011

	Carry	Sum
0 + 0	0	0
0 + 1	0	1
1 + 0	0	1
1 + 1	1	0
1 + 1 + 1	1	1

	128	64	32	16	8	4	2	1	Denary Check
	0	0	0	0	1	0	1	1	11
+	1	0	0	1	1	0	1	1	155
=	1	0	1	0	0	1	1	0	166
			1	1		1	1		

Adding using two's complement numbers

- Adding two's complement values is the same process as adding standard binary integers, but adding two large numbers together does throw up an interesting phenomenon.
- The two large numbers when added together are too large to store in the 8-bit two's complement integer and the value **overflows** the available bits, creating a negative number.
- If the calculation were to result in a number that was too small to represent, then this would be called **underflow**.

Adding together 2 two’s complement numbers

Add together these 2 two’s complement numbers:

0010 1010 and 1101 0110

Ignoring the overflow, what result do you get?

The result is 1 0000 0000, which is –256. Ignoring the overflow gives 0000 0000.

Why do you think this is the case?

These numbers represent +42 and –42. They add to zero. It does this with an overflow bit, which is not stored in the result.

Binary subtraction using two’s complement

- Subtracting two’s complement numbers is a relatively straightforward process.
- We convert the number to be subtracted into a negative two’s complement number and add them together.
- If you get an overflow when subtracting you lose the 1 value. This will still give the correct positive two’s complement value in the 8-bits.
- You can use one’s complement to change the 0s to 1s and 1s to 0s in a binary number when subtracting.

Example

Binary subtraction is best done by using the negative two's complement number and then adding the second number. For example denary 17-14 would be:

73 – 58 or 01001001 - 00111010

	-128	64	32	16	8	4	2	1	Denary	Notes
	0	0	1	1	1	0	1	0	58	Work out + 58
	1	1	0	0	0	1	1	0	-58	Flip after first 1
+	0	1	0	0	1	0	0	1	73	Add 73
(1)	0	0	0	0	1	1	1	1	15	Overflow (1) is lost

Key points

- When adding two binary numbers, we only have a small range of possibilities
- When we add two relatively large values, it is possible there is a carry in the MSB, leading to overflow, that is, the number is too big to store in the space allocated.
- When adding two two's complement values, this overflow can carry into the MSB and turn a positive value into a negative value.
- To subtract one two's complement number from another, simply take the two's complement form of the number to be subtracted and add.

Did you know?

The digital electronics that makes a computer work doesn’t know (or care) if 10001000 is 2s complement or not.

It could just as easily represent the positive number 136 (128 + 8).

The computer processes the binary data (for addition, subtraction etc…) in exactly the same way.

It’s down to the software rather than the hardware to remember if the binary data is a signed (2s complement) number or an unsigned (positive) number.

This can sometimes lead to some pretty catastrophic software failures when programmers aren’t aware of how computers handle negative numbers.