Byte-sized maths

At GCSE, bytes are probably the largest binary numbers you are going to need to be able to work with. Typically, exam questions might ask you to add a couple of bytes, to convert a couple of decimal numbers into bytes and then add them, and possibly to convert back to decimal. (There's another little catch they're likely to throw in, but we'll leave that for now – we should be managing break-sized snacks by now, but lets focus on our nibbles and bytes before we start making a meal of things.)

Presenting your work

For reasons that will make even more sense in the next section or two, we generally represent our byte as its two nibble parts. It helps us to manage what we're doing a bit more easily, and makes life a bit easier still, later on.

It also helps to be able to show what number system we are using. For example, if I just write 1010, you have no real way of knowing whether I mean "one thousand and ten" in decimal, or "ten" in binary.

So … we use a little trick called a **subscript**.

It works like this:

If I write 1010_{10} that means I am talking about "one thousand and ten" in decimal.
BUT
If I write 1010_2 then that signifies a binary value of "ten".

Since we can have other **number bases** besides binary and decimal, it helps to identify 192_{10} to be really clear that we are talking about "one hundred and ninety two" in good old human-friendly decimal, and not some other value in a different base.

Representing a byte

Our byte, broken into its two nibbles, looks something like this:

128	64	32	16
2^7	2^6	2^5	2^4

8	4	2	1
2^3	2^2	2^1	2^0

I'm sure you're familiar with the idea of 2^2 as 2x2, or "2 squared" and 2^3 as "2 cubed" or 2x2x2. If we use the words "Two to the power of two" and "two to the power of three" to describe these, then it helps to explain:

2^1 (or "two to the power of 1") is simply 2, not multiplied by anything.

2^0 (or anything to the power of zero) is always 1. Strange little bit of maths, but even 0^0 is 1. So, that's our units column.

Exercises

Example

Represent 15_{10} and 31_{10} as binary bytes, then add them together in binary.

OK – so 15_{10} is fairly easy, we know that's as much as we can get into our smallest nibble; and we just fill any "spare" bits in the left-hand nibble with zeros:

128	64	32	16
0	**0**	**0**	**0**

8	4	2	1
1	**1**	**1**	**1**

Now we put in our 31_{10} [you can always go back to the presentation from Topic 1 for a reminder, if you need it…]

128	64	32	16
0	0	0	0
0	**0**	**0**	**1**

8	4	2	1
1	1	1	1
1	**1**	**1**	**1**

Next we add, remembering to use "carry" digits where we need them (I've put mine in red to be clear) – with the answer row in **bold**:

128	64	32	16
0	0	0	0
0	0	0	1
0	**0**	**1**	**0**
		1	1

8	4	2	1
1	1	1	1
1	1	1	1
1	**1**	**1**	**0**
1	1	1	

Double check (all in decimal!): 32 + 8 + 4 + 2 = 46; 15+31=46

Questions:

Remember: you can always refer back to earlier Topics for reminders.
In questions 0001 and 0010 please also convert your answer back to decimal, to make sure your working is correct.

Q. 0001

Using whole bytes, work out 109_{10} x 2_{10}

[Hint: you can do this simply by adding!]

128	64	32	16

8	4	2	1

Q 0010

Using whole bytes, convert 58_{10} and 53_{10} to binary and add them.

128	64	32	16

8	4	2	1

Q0011

In a dance competition,
Arun scores 0101 0011_2 points, while Bianca scores 0110 1101_2
Enter these scores, find their combined score, and convert all three values back to decimal

128	64	32	16

8	4	2	1

Q0100

In a sports competition, two teams are always vying for the top place on the league table.

Yesterday, their scores looked like this:
Jaguars: $0011\ 0101_2$
Stags: $0011\ 1000_2$

In today's matches, they scored:
Jaguars: 12_{10}
Stags: 9_{10}

After converting the decimal values to binary, work out what each teams current score is, and say which team is now at the top of the table.

128	64	32	16

8	4	2	1

128	64	32	16

8	4	2	1

Justify this by converting all values back to decimal to double-check.