Section 4 Lecture 22 – Binary, Octal and Hexadecimal

This is a supplementary set of notes, summarising the basic ideas of counting in different number systems. It is provided for your benefit – you do not need it to follow the cryptography to come, but I hope it will be of benefit!

In our normal way of counting and representing numbers, we use what is known as the *decimal* number system. You've known this and used this for years: you have a certain number of ones, a certain number of tens, a certain number of hundreds, a certain number of thousands, and so on. So, for example, you read the number "7356" as "seven thousand, three hundred, and fifty-six" where "fifty-six" is really short for "five tens and six ones".

Remember that we write, for example, 10^3 to mean $10 \times 10 \times 10$. Look at the following table:

$10 = 10^1$
$100 = 10 \times 10 = 10^2$
$1000 = 10 \times 10 \times 10 = 10^3$
$10000 = 10 \times 10 \times 10 \times 10 = 10^4$
$100000 = 10 \times 10 \times 10 \times 10 \times 10 = 10^5$
….etc

So when we talk about tens, hundreds, thousands, and so on, we could talk about powers of 10 instead. What about the ones?

Following the pattern in the chart, we "ought to have" $1 = 10^0$. So we define 10^0 to be 1 so that our pattern follows nicely.

Now, we'll write these powers of 10 in the following table. The first row lists the powers of 10, and the second row lists the actual value corresponding to these powers. The reason for the third row being blank will become clear shortly.

10^6	10^5	10^4	10^3	10^2	10^1	10^0
1000000	100000	10000	1000	100	10	1

Note that we started with 10^0 (the smallest value) on the *right-hand side*. You can imagine how we could continue this table further and further to the left if we wanted to, with columns for 10^7, 10^8, etc.

Now take any integer, let's say 53084. Write this in the third row, digit by digit, so the last digit finishes in the last column (corresponding to the 10^0), like this:

10^6	10^5	10^4	10^3	10^2	10^1	10^0
1000000	100000	10000	1000	100	10	1
		5	**3**	**0**	**8**	**4**

So, the number 53084 corresponds to 5 "ten thousands", 3 thousands, no hundreds, 8 tens and 4 ones – just like you've always used numbers.
Formally, you can say that

$53084 = (5 \times 10000) + (3 \times 1000) + (0 \times 100) + (8 \times 10) + (4 \times 1)$

$= 50000 + 3000 + 0 + 80 + 4$

Remember there's nothing complex here – it's just what you've done for years when you think about numbers.

Binary numbers

In our counting system we use 10s. Why do we use 10s? It's natural for us as humans to count in 10s – we have 10 fingers for a start. But what about a computer? A computer doesn't count in 10s. Effectively, it works on millions of switches which can either be on or off. Alternatively, you can think of it as millions of "bits", each of which either takes the value 0 or 1. A typical computer number might look like:

01001001001010010000101001010101010010010101000010101000101010101001

There's no 2s, 3s, 4s, etc in a computer number, there are only two possible digits, either 0 or 1. Hence the computer is using a counting system of 2s. This is called a ***binary*** number system.

What we want to do is convert numbers in our decimal number system to our computer's binary system, and vice versa. If we again define $2^0 = 1$, then we can write a similar list to before:

$1 = 2^0$
$2 = 2^1$
$4 = 2 \times 2 = 2^2$
$8 = 2 \times 2 \times 2 = 2^3$
$16 = 2 \times 2 \times 2 \times 2 = 2^4$
$32 = 2 \times 2 \times 2 \times 2 \times 2 = 2^5$
…..etc

and again we can write a similar table to before (using a few more columns this time, again you can extend it with more columns to the left if required)

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
256	128	64	32	16	8	4	2	1

Whereas before we expressed a number as so many ones, tens, hundreds, thousands, etc, now we are going to express it as a number of ones, twos, fours, eights, sixteens, etc.

Converting binary to decimal

Let's see an example of how it works by converting a binary number to a decimal number.

Take the binary number 1011011 (remember it only has 0s and 1s as possible digits) and write it into our table just as before, so it finishes in the last column.

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
256	128	64	32	16	8	4	2	1
		1	**0**	**1**	**1**	**0**	**1**	**1**

So, you can read this as we've got 1 "lot of 64", 0 "lots of 32", 1 "lot of 16", etc.

Doing the multiplying again, like did before, our number is equivalent to:

$(1 \times 64) + (0 \times 32) + (1 \times 16) + (1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1)$

$= 64 + 0 + 16 + 8 + 0 + 2 + 1$

$= 91$

So our binary number 1011011 is equivalent to the decimal number 91.

Note that here, our answer is the same as just adding up 64 + 16 + 8 + 2 + 1, that is adding up all those numbers in the second row with a 1 underneath them. This will always work for binary numbers.

Converting decimal to binary

How do we convert the other way round, from a decimal number to a binary number? There are various ways to do, but I will give a basic rule to follow:

- Work out the biggest power of 2 that you need.
- Work your way down through the powers down to 1, deciding at each stage whether you need 0 or 1 of that power by seeing what's left for you to make.

- Read off your answer.

Let's do this with the example 173.

First of all, you can't need any 256s or any bigger powers of 2, they are too big, we only want 173. So the first power must be $128 = 2^7$. So draw our table going up to 2^7, and fill in that we need 1 of the 2^7 power:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1							

Now, we want 173 and we've got 128 so far, so we need to make another $173 – 128 = 45$.

Hence, we can't have any 64s, or we'd have too much. So fill in a 0 under the 64:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	**0**						

But we can have a 32 (in fact, we must have – always if we can have a number, we *must* have it). So write a 1 under the 32.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	**0**	**1**					

Now we need to make a further $45 – 32 = 13$. So we can't have a 16, but we must have an 8.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	**0**	**1**	**0**	**1**			

Now we need to make $13 – 8 = 5$. So we must have a 4.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	**0**	**1**	**0**	**1**	**1**		

Now we need to make $5 – 4 = 1$. So we can't have a 2 but we must have a 1.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	**0**	**1**	**0**	**1**	**1**	**0**	**1**

Therefore we have the binary answer 10101101. You can check this is right:

128 + 32 + 8 + 4 + 1 = 173.

Manipulating binary numbers

You can add up and multiply binary numbers in the same way as you do with "normal" numbers, using long addition and long multiplication. *(Subtraction and division are also possible but calculations may require the binary representation of negative and fractional numbers, which I won't cover here, but will mention in the class).*

Adding binary numbers

Just like with normal long addition, you sometimes have to "carry" numbers. For example, try to add 3492 and 4745 using long addition:

```
   3  4  9  2
+  4  7  4  5
-------------
```

You start by adding up 2 and 5 to get 7:

```
   3  4  9  2
+  4  7  4  5
-------------
            7
```

Next you add up 9 and 4. This makes 13, which you can't fit in one space, so you write down 3 and "carry" 1 over to the next column.

```
   3  4  9  2
+  4  7  4  5
-------------
      3  7
   1
```

Now add up 4, 7 and the "carried" 1 to get 12. Again you can't write this down, so write down 2 and "carry" the 1 over to the next column

```
   3  4  9  2
+  4  7  4  5
-------------
   2  3  7
1  1
```

Finally add 3, 4, and the carried 1, and you get 8:

	3	4	9	2
+	4	7	4	5
	8	2	3	7
	1	1		

and you do indeed get the correct answer of 8237.

Binary is actually much simpler, since you only have two possible digits to contend with. The only difficult case is when you have to add together 1 + 1. Remember that this number (2 in decimal) is represented in binary by 10. This means that when you add together 1 and 1, you write down 0 and "carry" the 1, just like with decimals. To summarise:

- Add together 0 and 0, you write 0
- Add together 0 and 1, you write 1
- Add together 1 and 0, you write 1
- Add together 1 and 1, you write 0, and "carry" 1

To illustrate, let's try to add together 10110 and 10011.

	1	0	1	1	0
+	1	0	0	1	1

Firstly we add together 0 and 1. This makes 1.

	1	0	1	1	0
+	1	0	0	1	1
					1

Next, add together 1 and 1. This makes 0, and carry 1 to the next column:

	1	0	1	1	0
+	1	0	0	1	1
				0	1
			1		

Now add together the 1 and the 0, and the carried 1. This is effectively just adding 1 and 1, and so we write down 0 and carry 1:

	1	0	1	1	0
+	1	0	0	1	1
			0	0	1
		1	*1*		

Now add 0, 0, and 1, and this gives 1:

	1	0	1	1	0
+	1	0	0	1	1
		1	0	0	1
		1	*1*		

Finally add 1 and 1 to get 0, and carry 1:

	1	0	1	1	0
+	1	0	0	1	1
	0	1	0	0	1
1		*1*	*1*		

and so filling in the last carried digit we get:

	1	0	1	1	0
+	1	0	0	1	1
1	0	1	0	0	1
1		*1*	*1*		

and a final answer of 101001. You can verify that this is correct – in decimals, 10110 corresponds to 22 and 10011 to 19, which add up to 22 + 19 = 41, which is 101001 in binary as expected.

Multiplying binary numbers

Multiplying can also be done by "long multiplication" as with "normal" decimal numbers. However, it's even easier than with decimals, as there is never any carrying to do when multiplying. All you need to remember is:

- 0 multiplied by 0 is 0
- 0 multiplied by 1 is 0
- 1 multiplied by 0 is 0
- 1 multiplied by 1 is 1

You can remember this as "0 multiplied by anything gives 0, and 1 multiplied by anything gives the same number" which is true just as it is for "normal" numbers

This is a pretty easy set of times tables to learn!

Let's look at the following example, where we multiply 10110 by 101. Note that in decimals, these numbers represent 22 and 5 so we expect the answer 22 × 5 = 110 which is 1101110.

1 0 1 1 0

×			1	0	1

The last digit in the second number (101) is 1. So what we do (just as in normal long multiplication is multiply the whole of the first number (10110) by 1. That means multiply every symbol by 1, which just gives us 10110 again:

	1	**0**	**1**	**1**	**0**
×			1	0	**1**
	1	**0**	**1**	**1**	**0**

Now multiply the first number by the second-to-last digit, which is 0. Multiplying every symbol by 0 will just give us 0, so we get 00000. You don't have to write this down but it's probably easier to do so. The important point is to write it "shifted left" by 1 (whether or not you choose to add a zero at the end is up to you), so it looks like this:

	1	**0**	**1**	**1**	**0**
×			1	**0**	1
	1	0	1	1	0
0	**0**	**0**	**0**	**0**	

Carry on in the same way with the final digit, which is 1. Multiplying the first number by 1 you get 10110 which you again write down "shifted left":

		1	**0**	**1**	**1**	**0**
	×			**1**	0	1
		1	0	1	1	0
	0	0	0	0	0	
1	**0**	**1**	**1**	**0**		

Now finally, just as with long multiplication with "normal" numbers, you add up the rows we created. The first column is just a 0 with nothing to add to it, so write down 0:

		1	0	1	1	0
	×			1	0	1
		1	0	1	1	0
	0	0	0	0	0	
1	0	1	1	0		
						0

The next column is 1 plus 0, which makes 1:

		1	0	1	1	0
	×			1	0	1
		1	0	1	1	0
	0	0	0	0	0	

1	0	1	1	0		
					1	0

Now we need to add together the three numbers 1, 0 and 0. This makes 1 in binary:

		1	0	1	1	0
	×			1	0	1
		1	0	1	1	0
	0	0	0	0	0	
1	0	1	1	0		
				1	1	0

The next column, add together 0, 0 and 1 and you get 1:

		1	0	1	1	0
	×			1	0	1
		1	0	1	1	0
	0	0	0	0	0	
1	0	1	1	0		
			1	1	1	0

Next column, you add together 1, 0 and 1, which is the same as 1 + 1, which if you remember is 0, carry 1 (remember that 1 + 1 = 2 which is 10 in binary)

		1	0	1	1	0
	×			1	0	1
		1	0	1	1	0
	0	0	0	0	0	
1	0	1	1	0		
		0	1	1	1	0
	1					

Almost there now...the next column needs 0, 0 and the carried 1 to add up, which makes 1:

		1	0	1	1	0
	×			1	0	1
		1	0	1	1	0
	0	0	0	0	0	
1	0	1	1	0		
	1	0	1	1	1	0
	1					

Finally just a 1 left over in the final column, and we get the answer 1101110 we expected.

		1	0	1	1	0
	×			1	0	1
		1	0	1	1	0
	0	0	0	0	0	
1	0	1	1	0		
1	1	0	1	1	1	0
	1					

Note: If the numbers get longer, and we have several rows, then there is the potential for more carrying needed to be done and you may have to carry across more than one column – again, I'll discuss this further in the lecture.

The decimal system uses powers of 10 and is said to be to *base 10*, similarly the binary system uses powers of 2 and so is said to be to base 2. But why restrict ourselves to 2 and 10? The techniques we used can be used for any numbers, not just 2 and 10. We'll look at two other systems (two other bases) which are commonly used in computing.

Octal number system

You might know that computers work in bits and bytes – a bit is a single binary digit (so either 0 or 1), and a byte is eight bits. Since the number 8 occurs here (eight bits in a byte) then another system often used is to use base 8 – the *octal number system*.

With base 8, there are eight digits available to us – 0, 1, 2, 3, 4, 5, 6, 7.

The important thing to bear in mind is that we aren't going to do anything different to what we did before, we'll just use 8s rather than 2s or 10s. So follow exactly the same procedure.

Our basic table looks like this (going up to a sensible amount):

8^6	8^5	8^4	8^3	8^2	8^1	8^0
262144	32768	4096	512	64	8	1

So, what is the octal number 31702 in decimal? **Follow exactly our usual procedure.** Write the number down so it finishes at the right-hand side.

8^6	8^5	8^4	8^3	8^2	8^1	8^0
262144	32768	4096	512	64	8	1

		3	**1**	**7**	**0**	**2**

Now read off our answer doing the multiplying in each column:

$4096 \times 3 + 512 \times 1 + 64 \times 7 + 8 \times 0 + 1 \times 2 = 13250$

So 31702 in octal is the same as 13250 in decimal.

Next, what is the decimal number 1799 in octal? **Follow exactly our usual procedure.**

The first power of 8 that we need is 512 (4096 is too big). How many 512s do we need? Since $3 \times 512 = 1536$ (which is OK) and $4 \times 512 = 2048$ (which is too big), it's clear we need to have three 512s.

8^3	8^2	8^1	8^0
512	64	8	1
3			

We are left to make $1799 - 1536 = 263$.

So how many 64s do we need? $4 \times 64 = 256$ which is OK, but $5 \times 64 = 320$ which is too big. So we must have four 64s.

8^3	8^2	8^1	8^0
512	64	8	1
3	**4**		

Now we are left to make $263 - 256 = 7$. So we don't need any 8s at all, and we need seven 1s.

8^3	8^2	8^1	8^0
512	64	8	1
3	**4**	**0**	**7**

Hence the equivalent octal number to the decimal 1799 is 3407.

Hexadecimal number system

Hopefully you are getting the idea that we can do this same technique for any base at all. Another commonly used one in computing (especially in coding and cryptography) is *hexadecimal* (sometimes abbreviated to *hex*) which is base 16. This one has an additional problem to overcome.

With base 16, we have 16 possible digits available to use. But we can't take 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15. If we did, what would the number 1015 mean? Do we mean 10, then 15? Or 1, then 0, then 1, then 5? Or 1, then 0, then 15? It's ambiguous. To get over this, we use letters instead for the numbers corresponding to 10, 11, 12, 13, 14, 15. The following table shows the numbers 0–15 and the hexadecimal symbol corresponding to them:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

So the numbers 0-9 are as normal, then A represents 10, B represents 11, and so on.

Having done this, we proceed exactly as normal. Our basic table looks like this (going up to a sensible amount:

16^5	16^4	16^3	16^2	16^1	16^0
1048576	65536	4096	256	16	1

So for example, what is the hexadecimal number 6F1A in decimal?

Writing it so it finishes at the right-hand side just as normal, we get:

16^5	16^4	16^3	16^2	16^1	16^0
1048576	65536	4096	256	16	1
		6	**F**	**1**	**A**

And so it corresponds to $4096 \times 6 + 256 \times F + 16 \times 1 + 1 \times A$.

Now, remembering that F corresponds to 15 and A corresponds to 10, this is the same as

$4096 \times 6 + 256 \times 15 + 16 \times 1 + 1 \times 10 = 28442$.

And so the decimal equivalent of 6F1A is 28442.

We go the other way round just as before. For example, what is the hexadecimal equivalent of 13013?

We must start at 4096 (65536 is too big) and $3 \times 4096 = 12288$, which is OK, but $4 \times 4096 = 16384$ which is too big. So we need three 4096s.

16^3	16^2	16^1	16^0
4096	256	16	1
3			

We are left to make 13013 – 12288 = 725. How many 256s? Well, 2 × 256 = 512 which is OK, but 3 × 256 = 768 which is too big. So we need two 256s:

16^3	16^2	16^1	16^0
4096	256	16	1
3	**2**		

We are left to make 725 – 512 = 213. How many 16s? Well, 13 × 16 = 208 which is OK, but 14 × 16 = 224 which is too big. So we need thirteen 16s. Remembering that we write 13 as D, we have

16^3	16^2	16^1	16^0
4096	256	16	1
3	**2**	**D**	

Finally we have to make 213 – 208 = 5, and so we need five 1s.

16^3	16^2	16^1	16^0
4096	256	16	1
3	**2**	**D**	**5**

Hence the hexadecimal equivalent of 13013 is 32D5.

Remember – whatever base you have to work in, just follow exactly the same technique to convert numbers to and from decimal

If you have to convert a number from one base to another, say from binary to hexadecimal, it is possible to do it directly but since we have the techniques above to dealing with converting to and from decimal, it's probably easier just to convert your number first into decimal, and then convert that decimal into your new base.

So to convert from binary to hexadecimal you will probably find it easiest to do *Binary → Decimal → Hexadecimal*, and so on for other bases.

Manipulating numbers in other bases

You can add up and multiply numbers in other bases using long addition and long multiplication, just as you know how to do with decimal numbers, and which we then did with binary numbers.

To illustrate, we will again consider only addition and multiplication (again because subtraction and division require negatives and fractions) and we will consider only hexadecimal, but hopefully you'll see how the principle can apply to any base.

Addition in hexadecimal

To add up in hexadecimal, you add up columns in just the same way as with normal long addition, remembering to "carry" whenever needed. To "carry" a number in hexadecimal:

- Add up the numbers involved. If it is less than 16 there is no carrying to do.
- If it is more than 16, decide how many times 16 goes into the number. This is your "carried" figure.
- The remainder is the figure you write down in the solution.

This is *exactly the same* as you do when adding up numbers in "normal" numbers, base 10. If you had a sum of 6 + 7 to deal with, you have the answer 13. 10 goes into 13 once, so you "carry" 1, and have 3 left over that you write in your solution

$$\begin{array}{rr} & 6 \\ + & 7 \\ \hline & 3 \\ \hline 1 & \end{array}$$

If you had to do 8 + 6 + 9, this is 23, so you "carry" 2 (10 goes into 23 twice) and write down 3 (the remainder):

$$\begin{array}{rr} & 8 \\ & 6 \\ + & 9 \\ \hline & 3 \\ \hline 2 & \end{array}$$

You are used to doing this with "normal" numbers – you do exactly the same with hexadecimal and any other base.

So, let's add up A3E and 2D9 which are numbers in hexadecimal. Note that in decimal, these numbers are 2622 and 729 so we expect the answer 3351, which is D17 in hexadecimal. Start by writing them as normal:

$$\begin{array}{rrrr} & A & 3 & E \\ + & 2 & D & 9 \\ \hline & & & \\ \hline \end{array}$$

So, proceed exactly as normal. Add together "E" and "9". Remembering that "E" corresponds to 14, this gives us the answer 23. This is too big. 16 goes into 23 one time with remainder 7, so we write down 7 and "carry" 1.

	A	3	E
+	2	D	9
			7
		1	

Now the next column. Add together 3, D, and the carried 1. This is 3 + 13 + 1 = 17. Again too big – 16 goes into 17 once, with a remainder of 1, so we write down 1 and "carry" the 1 over:

	A	3	E
+	2	D	9
		1	7
	1	1	

Finally, A + 2 + 1 is the same as 10 +2 + 1 = 13, which corresponds to "D". So we get:

	A	3	E
+	2	D	9
	D	1	7
	1	1	

and we have the answer D17 which is what we expected.

Multiplication in hexadecimal

Multiplication in hexadecimal follows exactly the same rules regarding carrying. Because we are dealing with base 16, the numbers can get reasonably big and there can be quite a lot of "carrying" happening. All the examples you will see in this course will be very small but hopefully you can see how the concept generalises.

We will multiply the hexadecimal number 3D7 by the hexadecimal number 42. In decimal, this corresponds to multiplying 983 by 66 and so we expect the answer 64878 in decimal, which is FD6E in hexadecimal.

So, let's go…. We start by multiplying the first number by 2 (the last digit of the second number). Firstly 2 × 7 is 14, which in hexadecimal is E:

	3	D	**7**
×		4	**2**
			E

Next, we do 2 × D, which is 2 × 13 = 26. This is too big, so we need to carry. Since 16 goes into 26 once, with remainder 10 (which in hexadecimal is A), we write down A and "carry" 1:

	3	**D**	7
×		4	**2**
		A	E
	1		

Now multiply 2 × 3 and you get 6, together with the carried 1 you get 7. Write this down:

	3	D	7
×		4	**2**
	7	A	E
	1		

That has completed the multiplying by 2. Now we multiply by 4, and "shift to the left" just like with normal long multiplication:

Start by doing 4 × 7 = 28. This is too big. 16 goes into 28 once, with remainder 12, which is C in hexadecimal. So write down C and "carry" the 1:

	3	D	**7**
×		**4**	2
	7	A	E
	1		
		C	
	1		

Next, we do 4 × D and add the 1 we "carried". In decimal this is 4 × 13 + 1 = 53. Again too big so we'll have to do some carrying. 16 goes into 53 three times with remainder 5. So we write down 5 and "carry" 3:

	3	**D**	7
×		**4**	2
	7	A	E
	1		
	5	C	
3	*1*		

Almost done now – all that's left is to multiply 4 by 3. This makes 12, together with the carried 3, you get 15, which is F in hexadecimal.:

		3	D	7
	×		**4**	2
		7	A	E
		1		
	F	5	C	
	3	*1*		

Now add up like normal (we can ignore the carried numbers now, they only helped with the multiplication). The E at the end is just an E:

		3	D	7
	×		4	2
		7	A	E
	F	5	C	
				E

Looking at the next column, A and C represent 10 and 12 in decimals. Adding them up gives 22, which is too big – we end up with an answer of 6 and "carry" 1 (since 16 goes into 22 once with remainder 6):

		3	D	7
	×		4	2
		7	A	E
	F	5	C	
			6	E
		1		

Now, add 7 and 5, together with the carried 1, and you get 13. In hexadecimal this is D:

		3	D	7
	×		4	2
		7	A	E
	F	5	C	
		D	6	E
		1		

Finally there is an F left over, and we get the answer FD6E we expected:

	3	D	7

×		4	2
	7	A	E
F	5	C	
F	D	6	E
	1		

This can seem difficult and complicated on a first reading. But if you can recognise that are doing exactly the same as you have always done with long addition and multiplication, then it makes sense. Just remember that we are counting out of 16 instead of the more 'usual' 10.

Summary

Everything that you know how to do (and have probably known for years since you were at primary school) with "normal" decimal numbers also works for any base.

With other bases, all we are doing is counting out of something other than ten.

In essence, what's so special about ten? It's just a number. All the techniques work when you count out of 2 (binary), 8 (octal), 16 (hexadecimal) and anything else.

Nothing is special about 10!