2.1

Binary arithmetic

Syllabus

Binary numbers and strings,
arithmetic modulo powers of 2

2.2

The set of *natural numbers*, $\mathbb{N}$, consists of the numbers

$$1, 2, 3, \ldots .$$

The set of *integers* contains 0 and the negative numbers too:

$$\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$$

So the natural numbers are the same as the positive integers.

2.3

Computers think of the natural numbers differently from us:

- We use symbols 0, 1, . . . , 9,
  but computers use just the two symbols 0, 1
  (i.e. we use *decimal* numbers, computers use *binary*).
- In a computer, all numbers have a fixed length, e.g. 8 symbols.
  (In fact computers sometimes use ‘short’ numbers of one length and ‘long’ numbers of another length, depending on how they are programmed.)

2.4

Decimal	Binary	Fixed-length binary
1	1	0001
2	10	0010
3	11	0011
4	100	0100
5	101	0101
6	110	0110
7	111	0111
8	1000	1000
⋮	⋮	⋮

2.5

At 16 the binary numbers of length 4 run out and we have to go back to 0000, which represents 0:

Decimal	Binary	Fixed-length binary
15	1111	1111
16	10000	0000
17	10001	0001
⋮	⋮	⋮

So 0, 16, 32, 48 etc. all have the same representation in binary numbers of length 4.

2.6

Notice the powers of 2:

Decimal	Binary
$1 = 2^0$	1
$2 = 2^1$	10
$4 = 2^2$	100
$8 = 2^3$	1000
$16 = 2^4$	10000
$32 = 2^5$	100000
$64 = 2^6$	1000000
$128 = 2^7$	10000000
⋮	⋮

2.7

So the binary numbers of fixed length n come back to 0 first at 2^n, which is 1 followed by n 0s.

In binary notation with fixed length n, we identify each number k with

$$k + 2^n,\ k + 2 \times 2^n,\ k + 3 \times 2^n \text{ etc. etc..}$$

We describe this situation by saying that the numbers in binary notation with fixed length n are *modulo* 2^n.

Except where we say otherwise, we shall assume the numbers that we discuss are not modulo anything.

2.8

Addition of binary numbers is the same as addition of decimal numbers,
except that we carry at 2 (i.e. binary 10) instead of at 10,
and we use the addition table for binary digits:

		+
1	1	10
1	0	1
0	1	1
0	0	0

2.9

Example

```
 110011
+ 11101
 ’’’’’
-------
1010000
```

The rest of these notes won’t show the carries
(they are awkward to print).

2.10

Advice: Adding three or more binary numbers at once is dangerous,
because we may have to carry into two or more columns at once, and the result is confusing.
It’s best to add several binary numbers one at a time:

2.11

$$\begin{array}{r} 1111 \\ +\ 111 \\ \hline 10110 \\ +\ 1110 \\ \hline 100100 \end{array}$$

2.12

Multiplication of binary numbers

Multiplication of binary digits is easy:

		$\times$
1	1	1
1	0	0
0	1	0
0	0	0

Same as the table for $\wedge$!

2.13

Multiplication of binary numbers is essentially the same as multiplication of decimals.

For example to multiply 1101 by 1011,
we express 1011 as

$$1 + 10 + 1000,$$

we multiply 1101 by each of these in turn,
and we add up the results.

Notice that putting a 0 at the end of a binary number
is the same as multiplying it by 2.

2.14

To study this multiplication, we express it as follows:

1101	
$\times$ 1011	
1101	(since 1011 ends in 1)
11010	(since 101 ends in 1)
(110100)	(since 10 ends in 0)
1101000	(since 1 ends in 1)
10001111	

Normally one leaves out the parts in brackets.

2.15

Compare with the same calculation in decimal numbers:

13	
$\times$ 11	
13	(since 11 is odd)
26	(since $5 = (11-1)/2$ is odd)
(52)	(since $2 = (5-1)/2$ is even)
104	(since $1 = 2/2$ is odd)
143	

2.16

Binary multiplication using decimal numbers, as above,
is still used in some parts of Russia and is known as
Russian peasant multiplication. It is quite efficient.
Normally we would write the calculation just as

2.17

13	
× 11	
13	11
26	5
(52)	2
104	1
143	

2.18

The idea of Russian peasant multiplication is that
we can use only the operations of binary arithmetic
even when we write the numbers in decimal notation.

This idea is very useful, because it gives us a way of
translating from decimal notation to binary, or vice versa.

To convert binary m to a decimal number, work out $1 \times m$
using decimal numbers on the left and binary on the right.

To convert decimal n to a binary number, work out $1 \times n$
using binary numbers on the left and decimal on the right.

2.19

Example: We convert 1101101 to decimal notation.

1	1101101
(2)	110110
4	11011
8	1101
(16)	110
32	11
64	1
109	

2.20

Example: We convert 291 to binary notation.

1	291
10	145
(100)	72
(1000)	36
(10000)	18
100000	9
(1000000)	4
(10000000)	2
100000000	1
100100011	

2.21

To see how to **subtract** binary numbers,
we need to see first how to subtract decimal numbers.
(You may have learned another method
that doesn't adapt smoothly to binary numbers.)

How do we subtract 11 from 100 and get 89?

2.22

Example

$$\begin{array}{lccc} & A & B & C \\ (i) & 1 & 0 & 0 \\ (ii) - & & 1 & 1 \\ \hline \end{array}$$

(The labels A, B, C, (i), (ii) are so that we can
talk about the rows and columns.
They are not part of the calculation.)

2.23

We try to take 1 away from 0 in column C.
We can't do it, so we borrow 10 from column B.
To mark this, we write a small 1 so that the 0 turns into 10:

$$\begin{array}{lccc} & A & B & C \\ (i) & 1 & 0 & {}^{1}0 \\ (ii) - & & 1 & 1 \\ \hline \end{array}$$

2.24

Now we can take 1 from 10 in column C and get 9:

$$\begin{array}{lccc} & A & B & C \\ (i) & 1 & 0 & {}^{1}0 \\ (ii) - & & 1 & 1 \\ \hline & & & 9 \end{array}$$

2.25

Next we go to column B.
We must take two numbers away from the 0 in row (i):

- we must take away the 1 in row (ii),
- we must take away 1 to pay for the 10
  that we put in column C.

So we must take away $1 + 1 = 2$ from 0.

2.26

Again we borrow 10 from column A:

$$\begin{array}{rccc} & A & B & C \\ (i) & 1 & {}^{1}0 & {}^{1}0 \\ (ii) - & & 1 & 1 \\ \hline & & & 9 \end{array}$$

2.27

Taking away the 2 from 10 gives 8:

$$\begin{array}{rccc} & A & B & C \\ (i) & 1 & {}^{1}0 & {}^{1}0 \\ (ii) - & & 1 & 1 \\ \hline & & 8 & 9 \end{array}$$

2.28

Finally we look at column A.
We must take away from the 1 in row (i):

- 0 from row (ii),
- 1 to pay for the 10 borrowed in column B.

So we take 0 + 1 = 1 away from 1
This gives the answer 089, i.e. 89.

2.29

The same calculation works in binary,
except that the 10 in row (i) of columns B and C
means binary 10, i.e. 2,
and then $2 - 2 = 0$ instead of 8:

$$\begin{array}{lccc} & A & B & C \\ (i) & 1 & {}^{1}0 & {}^{1}0 \\ (ii) - & & 1 & 1 \\ \hline & 0 & 0 & 1 \end{array}$$

2.30

Example:

$$\begin{array}{r} 1011100 \\ -\ 1001111 \\ \hline 0001101 \end{array}$$

2.31

When we subtract a larger number from a smaller number,
we get a negative answer.
So as in decimals, we subtract the smaller from the larger
and put a $-$ sign at the front.

For example

$$1001111 - 1011100 = -1101$$

by the previous calculation.

2.32

When we calculate with binary numbers of length n,
subtracting any number from a string of n 1's is very easy:

$$\begin{array}{r} \texttt{11111111} \\ \texttt{01001101} \\ \hline \texttt{10110010} \end{array}$$

The bottom line is the opposite of the second.
We say the bottom number is the *ones complement*
of the second number.

2.33

A binary number of length n is never changed by adding

$$10\ldots0 \quad (n \text{ 0's})$$

But

$$\begin{aligned} 100 &= 11+1, \\ 1000 &= 111+1, \\ 10000 &= 1111+1 \end{aligned}$$

and so on.

So for example, calculating modulo 2^5, we can always add $11111+1$ without changing the answer.

2.34

Example. We calculate $1001111-1011100$ modulo 2^7.
First note that 1011100 is bigger than 1001111,
so the answer will be negative unless we add 2^7.

$$\begin{aligned} 1001111-1011100 &= 1001111+(1111111-1011100)+1 \\ &= 1001111+0100011+1 \\ &= 1110010+1 \\ &= 1110011 \end{aligned}$$

2.35

In the next example we subtract a larger number
from a smaller number modulo 2^6.

$$\begin{aligned} 101001-110000 &= 101001+(111111-110000)+1 \\ &= 101001+001111+1 \\ &= 111000+1 \\ &= 111001 \end{aligned}$$