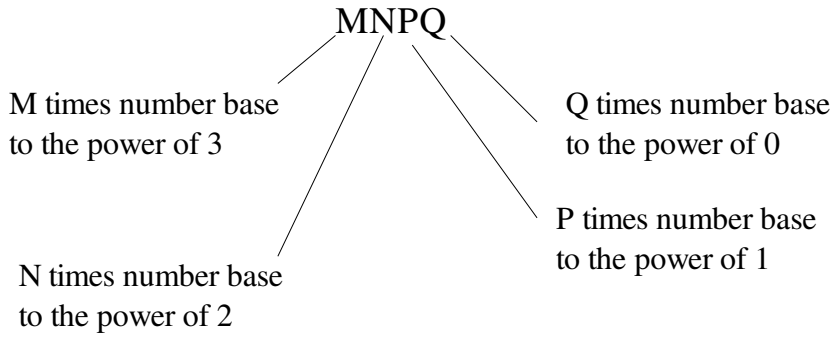
Number systems.
We normally count in the decimal number system; the digits 0→9 being the full set of numbers we have to work with. Other number systems are possible however. The binary number system for example has only two allowed digits: 0 and 1. Counting in the binary system proceeds as follows:

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

and so on.
Why bother with such a strange number system? Well the answer lies in the nature of digital electronics. A 3V[1] signal is equivalent to a binary '1' while a 0V signal is equivalent to a binary '0' (more exotic systems exist but these are not of importance to this module). Digital electronic devices transmit sequences of pulses (3V, 0V etc.) between each other to transfer data. In many cases, the devices use several wires to transmit data and so increase the speed with which information may be transferred. The devices could be thought of as transferring binary numbers between one another therefore.

In general, for any number base we can say that each digit represents a quantity as shown below (for a 4 digit number).



(remember, any number to the power of zero is 1).

1 Different systems can use different voltage levels. Older microprocessors commonly used 5V while more more modern one use level of less than 1V to save power.

Thus the decimal number 1234 represents

4×10^0 i.e. 4 units
+ 3×10^1 i.e. 3 tens
+ 2×10^2 i.e. 2 hundreds
+ 1×10^3 i.e. 1 thousand

The binary number 1010 represents

0×2^0 = no units
+ 1×2^1 = 1 two
+ 0×2^2 = no four's
+ 1×2^3 = 1 eight.

which of course is 10.

Converting from decimal to binary is accomplished by successive division by 2 as follows:
Find the binary representation of 14 decimal.

2 into 14
= 7 Plus a remainder of **0**
2 into 7
= 3 Plus a remainder of **1**
2 into 3
= 1 Plus a remainder of **1**
2 into 1
= 0 Plus a remainder of **1**

The result (you stop when the division produces a result of 0) is
1110 binary
(i.e. reading from the bottom up).

In computing, two number systems (apart from decimal) are of particular importance. These are the binary number system and the hexadecimal number system. The hexadecimal number system is based on the number 16. The reason for this apparently odd choice of number base is that one hexadecimal digit can be used to represent all possible states of four binary digits thus significantly shortening the business of writing down numbers. The following table shows equivalents for some binary, hexadecimal and decimal number systems.

Binary	Hexadecimal	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Note that letters are used to represent hex digits beyond 9 and that there are 16 digits in all. The highest hex digit is F. The number which follows this is hex 10, often written as \$10 or 10_{16}. Hex numbers follow the general format as described above; thus \$4C9A represents

A (decimal 10) x 16^0 = 10 decimal
\+ 9 (decimal 9) x 16^1 = 144 decimal
\+ C (decimal 12) x 16^2 = 3072 decimal
\+ 4 (decimal 4) x 16^3 = 16384 decimal

or 19610 decimal

Bits, Bytes, Nibbles and Words.
A **bit** is a single **B**inary Dig**it**, i.e. possible values either a 1 or a 0.
A **byte** is a collection of 8 bits: range of possible values from 0 to 255
A **nibble** is half a byte (four bits): range of possible values from 0 to 15.
A **word** is usually treated as two bytes: range of possible values from 0 to 65535

Exercises:
1) What is the range of decimal numbers that can be represented by 8 binary digits?
2) What is the range of hexadecimal numbers that can be represented by 8 binary digits?
3) Perform the following calculations:

10011100_2	AD21
$+00001100_2$	+1230

Finite word length

Consider the following binary number sequence:

0,1,10,11,100,101…

As you can see, the number of bits required to represent the numbers grows. Now, suppose you only have enough room to store 8 bits (a byte). The sequence would begin:

```
0000 0000
0000 0001
0000 0010
     :
     :
1111 1111
```

What happens when you add one again? Well, you will end up with 0000 0000 which of course is wrong. This sort of error is called an overflow error and it can happen very easily and lead to some disastrous consequences. Imagine a submarine which contained a counter such as this to measure its depth. *"…253 feet, 254 feet 255 feet, 0 feet. Oh - we're on the surface, wonder what the weather is like outside. Lets take a look…"* As you will see later, microprocessors contain special alert systems to help the programmer identify and trap errors of this nature before they can cause harm.

Negative numbers.

Consider the following sequence:

```
0000 0011
0000 0010
0000 0001
0000 0000
????  ????
```

What happens when you take 1 from 0000 0000? Well, in theory you will end up with an infinite number of 1's. In practice the number of 1's is normally constrained to the number of bits in the microprocessor's internal working register(s). So, in an 8 bit system, the following results:

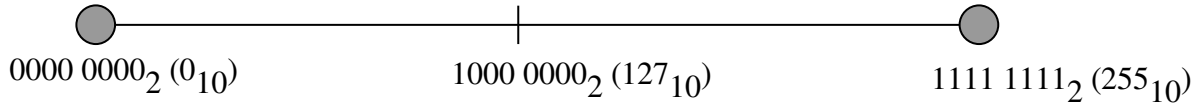
```
0000 0000
-         1
=======
1111 1111
```

How can this be? The binary for -1 is the same as that for 255 it seems. How can you perform any useful mathematics with a number system like this? Well, ***you can***. The way to go about it is that when planning your program, you decide, *in advance*, whether you are going to deal with positive numbers only (without the possibility of a negative result - ***ever***), or, you may decide that there is a need for negative number representation in your program and decide to program for this possibility. This choice is very important because it means that you must choose between two incompatible number systems where a mix-up between them could mean disaster. Both number systems are as follows (for 8 bit systems)

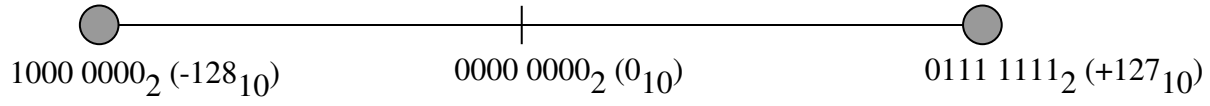
Binary Code	Equivalent Unsigned No.	Equivalent Signed No.
1111 1111	255	-1
0111 1111	127	+127
1000 0000	128	-128
0000 0000	0	0

The unsigned number line looks like:

Unsigned



Signed



The above method of representing negative binary numbers is known as the two's compliment number system. To obtain the two's compliment (binary negative) of a number, the procedure is as follows:

Convert all 1's to 0's and all 0's to 1's. This is known as the 1's compliment or often just as the compliment.
Add 1.

This procedure is symmetrical so performing the procedure on a particular number twice will leave you with the original number. Example, Find the 2's compliment of the decimal number 5 in 8 bits and verify that it is indeed equivalent to -5 by adding it to +5.

$5_{10} = 0000\ 1001_2$

The 1's compliment is:

$1111\ 0110_2$

Adding 1 we get.

$1111\ 0111_2$

Now add to $+5_{10}$.

0000 1001
1111 0111
========
0000 0000

One feature of negative binary numbers that should become obvious is that ***the most significant bit (MSB) (the left-most one) is always a '1'.***

Note: we have constrained the answer to 8 bits (the bit that carries out into the 9th location is lost)
Exercises:
Find the 8 bit. 2's compliment of the following and verify the answer by addition with positive value.

$12_{10}, 45_{10}, 127_{10}$

What is the 16 bit 2's compliment of 6?