Coding schemes, data representation

ASCII

(ASCII) is a character-encoding scheme and it was the first character encoding standard. ASCII uses 8 bits to encode each character. ASCII has a total of 256 characters.

Advantages:

you save a lot of space

Disadvantages:

fewer bits give you a limited choice but

Unicode

Unicode is a standard which defines the internal text coding system in almost all operating systems used in computers at present. Unicode assigns each character a unique number, or code point. Unicode defines 2^{32} characters

Unicode uses a variable bit encoding program where you can choose between 32, 16, and 8-bit encodings.

Advantages:

huge number of characters

Disadvantages:

Takes a lot of space

Summary

1.ASCII uses an 8-bit encoding while Unicode uses a variable bit encoding.
2.Unicode is standardized while ASCII isn’t.
3.Unicode represents most written languages in the world while ASCII does not.
4.ASCII has its equivalent within Unicode.

Data types

Reserved Word	Data Type	Size	Range of Values
byte	Byte Length Integer	1 bytes	-2^{8} to $2^{7}-1$
short	Short Integer	2 bytes	-2^{16} to $2^{16}-1$
int	Integer	4 bytes	-2^{32} to $2^{31}-1$
long	Long Integer	8 bytes	-2^{64} to $2^{63}-1$
float	Single Precision	4 bytes	-2^{32} to $2^{31}-1$
double	Real number with double	8 bytes	-2^{64} to $2^{62}-1$
char	Character (16 bit unicode)	2 bytes	0 to 216 - 1
boolean	Has value true or false	A boolean value	true or false

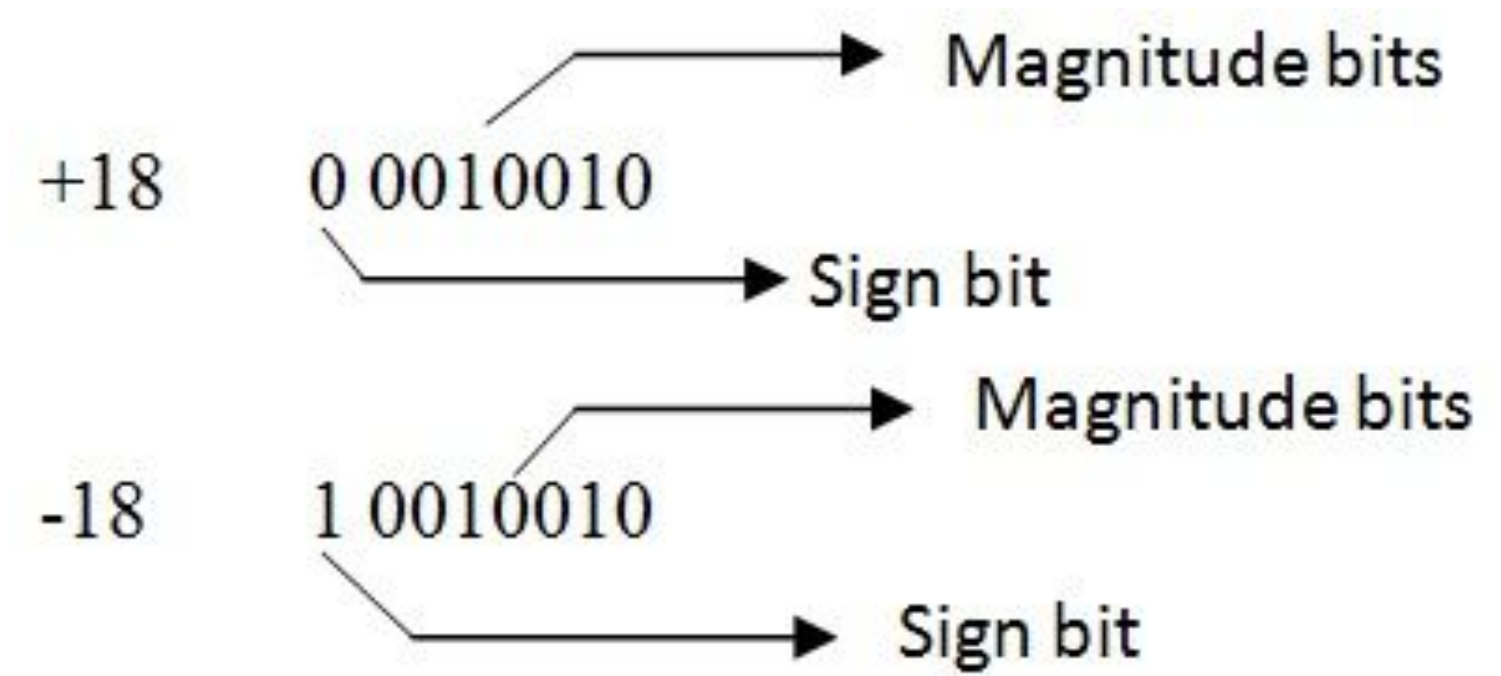
Why we use different types of data?

Data types are blocks or limited area confined for storing some specific item. Data type of int type can store integer value. In the same way there are many other data type double, float, char which can store large integer value, large decimal value, and character value.

Fixed Point and Floating Point Number Representations

Signed binary numbers

- **<u>0</u>**000 0101 (positive)
- **<u>1</u>**111 1011 (negative)



Method 1: converting twos complement to denary

To find the value of the negative number we must find and keep the right most 1 and all bits to its right, and then flip everything to its left. Here is an example:

```
1111 1011 note the number is negative
```

```
1111 1011 find the right most one

1111 1011
0000 0101 flip all the bits to its left
```

We can now work out the value of this new number which is:

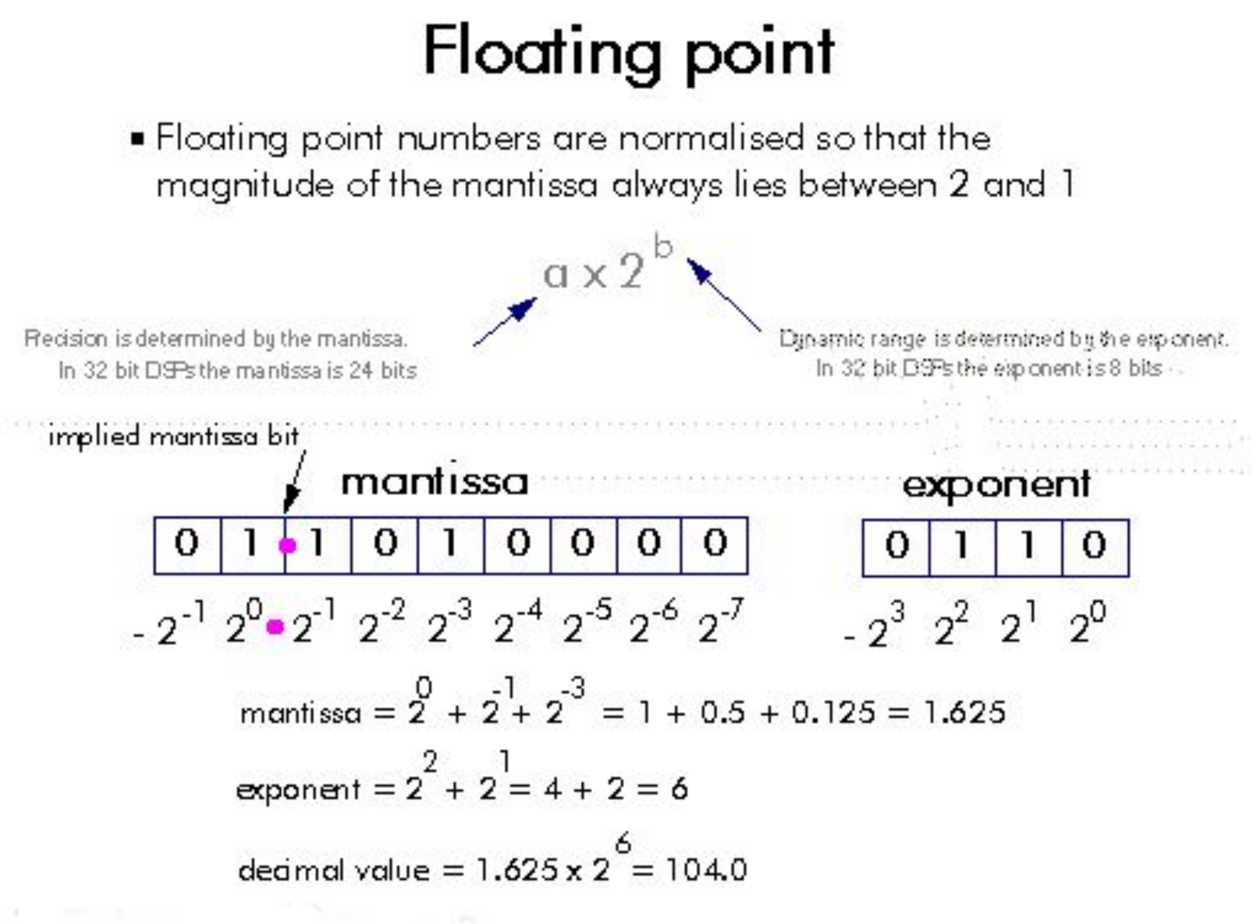
```
128  64  32  16   8   4   2   1
  0   0   0   0   0   1   0   1
                      4   +   1 = -5   (remember the sign you worked out earlier!)
```

Method 2: converting twos complement to denary

To find the value of the negative number we must take the MSB and apply a negative value to it. Then we can add all the heading values together

```
1111 1011 note the number is negative
-128  64  32  16   8   4   2   1
   1   1   1   1   1   0   1   1
-128 +64 +32 +16  +8      +2  +1 = -5
```

Fractional numbers using floating point



Exercise: Simple binary floating point

Work out the denary for the following, using 10 bits for the mantissa and 6 bits for the exponent:

0.001101000 000110

Answer: [Collapse

1. Sign: the mantissa starts with a zero, therefore it is a **positive** number.
2. Slide: work out the value of the exponent

```
000110 = +6
```

3. Bounce: we need to move the decimal point in the mantissa. In this case the exponent was **positive** so we need to move the decimal point 6 places to the right

```
0.001101000 -> 0001101.000
```

4. Flip: as the number isn't negative we don't need to do this
5. Swim: work out the value on the left hand side and right hand side of the decimal point

```
1+4+8 = +13 FINISHED!
```

```
1 011111010 000101
```

Answer:

[Collaps

1. Sign: the mantissa starts with a one, therefore it is a **negative** number.
2. Slide: work out the value of the exponent

```
000101 = +5
```

3. Bounce: we need to move the decimal point in the mantissa. In this case the exponent was **positive** so we need to move the decimal point 5 places to the right

```
1.011111010 -> 101111.1010
```

4. Flip: the mantissa is negative as noted in step one so we need to convert this number

```
101111.1010 -> 010000.0110
```

5. Swim: work out the value on the left hand side and right hand side of the decimal point

```
16+1/4+1/8 = -16.375 FINISHED!
```

Example: denary to binary floating point

If we are asked to convert the denary number 39.75 into binary floating point we first need to find out the binary equivalent:

```
128 64 32 16 8 4 2 1 . ½ ¼ ⅛
 0    0  1   0 0 1 1 1 . 1 1 0
```

How far do we need to move the binary point to the left so that the number is normlised?

```
0 0 . 1 0 0 1 1 1 1 1 0 (6 places to the left)
```

So to get our decimal point back to where it started, we need to move 6 places to the right. 6 now becomes your exponent.

```
0.100111110 | 000110
```

If you want to check your answer, convert the number above into decimal. You get 39.75!