Binary Number System

Base 10 digits: 0 1 2 3 4 5 6 7 8 9

Base 2 digits: 0 1

Recall that in base 10, the digits of a number are just coefficients of powers of the base (10):

$$417 = 4 * 10^2 + 1 * 10^1 + 7 * 10^0$$

Similarly, in base 2, the digits of a number are just coefficients of powers of the base (2):

$$1011 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0$$

Any real number can be represented in any base; humans generally use base 10, but computer hardware generally uses base 2 representation.

Converting from Base 10 to Base 2

The base 10 system is also known as the decimal system; base 2 is referred to as binary.

How can we convert an integer from decimal to binary? Here is a simple algorithm:

```
While N > 0 Do
    Write  N % 2       // remainder when N is divided by 2
    N  <--   N / 2     // divide N by 2
Endwhile
```

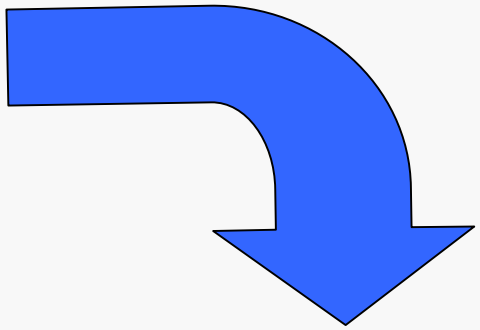
Note that the remainder will always be either 0 or 1, a binary digit or bit. The resulting sequence of bits is the binary representation of the integer N.

See the next slide for an example...

Converting Integers

Find the binary representation of the decimal integer N = 23:

Integer	Remainder
23	
11	1
5	1
2	1
1	0
0	1



So the decimal integer N = 23 is represented in base 2 as: 10111

Let's check that:

$$10111 = 1*2^4 + 1*2^2 + 1*2^1 + 1*2^0$$
$$= 16 + 4 + 2 + 1$$
$$= 23$$

Note: we just shifted from base 2 to base 10

Some Quick Tables

Here are some handy facts:

Power of 2	Base 2	Base 10
2^{-4}	.0001	.0625
2^{-3}	.001	.125
2^{-2}	.01	.25
2^{-1}	.1	.5
2^{0}	1	1
2^{1}	10	2
2^{2}	100	4
2^{3}	1000	8
2^{4}	10000	16
2^{5}	100000	32
2^{6}	1000000	64
2^{7}	10000000	128
2^{8}	100000000	256

N	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Converting Fractions from Base 10 to Base 2

How can we convert a fraction from decimal to binary? Here is a simple algorithm:

```
While F != 0 Do
    Multiply F by 2
    Record the "carry" across the decimal point
    F  <--   the fractional part
Endwhile
```

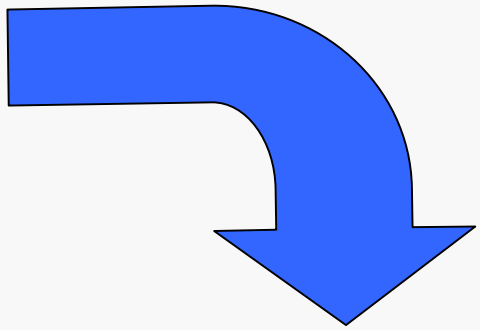
Note that the carry will always be either 0 or 1, a binary digit or bit. The resulting sequence of bits is the binary representation of the fraction F.

See the next slide for an example...

Converting Fractions

Find the binary representation of the decimal fraction F = 0.3125:

Carry	Fraction
	.3125
0	.6250
1	.2500
0	.5000
1	.0000



So the decimal fraction F = 0.3125 is represented in base 2 as: .0101

Let's check that:

$.0101 = 1*2^{-2} + 1*2^{-4}$

$= .25 + .0625$

$= .3125$

Note: we just shifted from base 2 to base 10

Converting a General Decimal Number

A general decimal number, like 43.375, would be converted to binary by converting the integer and fractional parts separately (as shown earlier) and combining the results.

The decimal integer 43 would be represented in binary as:

101011

Integer	Remainder
43	

The fractional part .375 would be represented in binary as:

.011

Carry	Fraction
	.375

So 43.375 would be represented in binary as: 101011.011

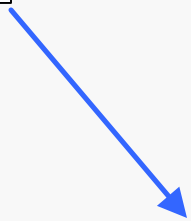
Binary Representation and Precision

Of course, many decimal fractions have infinite decimal expansions. The same is true of binary representation, but there are some (perhaps) surprising differences.

Consider the decimal fraction 0.1; how would this be represented in binary?

Carry	Fraction
	.1
0	.2
0	.4
0	.8
1	.6
1	.2
0	.4
0	.8
1	.6
1	.2
??	??

Clearly, this pattern will now repeat forever



So 0.1 would be represented in binary as:

0.0 0011 0011 0011 0011….

Float Representation

So 0.1 would be represented exactly in binary as: 0.0 0011 0011 0011 0011….

What would actually be stored in hardware? Suppose that we have a `float` variable:

```
float x = 0.1;
```

A `float` is stored in scientific form (but in binary):

`0.0 0011 0011 0011 0011….  =(1.1 0011 0011 0011….) * 2^-100`

That's -4 in binary

The exponent is stored using 7 bits and the fractional part is stored using 23 bits (with two bits used for the signs). We cheat and don't store the first '1', so 0.1 would be stored as:

```
-0000100   +.1 0011 0011 0011 0011 0011 00
```

Exponent

Mantissa (fractional part)

Storage Error

So 0.1 would be stored in hardware as:

```
-0000100   +.1 0011 0011 0011 0011 0011 00
```

Converting that to decimal, you have the value: `0.099999940395355`

That's fairly close, but not quite equal to, 0.1. This is called storage error.

This is typical. Most real numbers cannot be stored exactly as `floats` or even as `doubles`. Using a `double` will improve accuracy since a `double` stores 53 bits for the mantissa, but there will still be some inevitable storage error.