Floating Point Numbers

Floating point numbers model a tiny finite subset of reals

- almost all real values have no exact representation, e.g. 1/3
- numbers close to zero have higher precision (more accurate)

C has two floating point types

- `float` ... typically 32-bit (lower precision, narrower range)
- `double` ... typically 64-bit (higher precision, wider range)
- `long double` ... typically 128-bits (but maybe only 80 bits used)

Literal floating point values: `3.14159`, `1.0/3`, `1.0e-9`

```
printf("%10.4lf", (double)2.718281828459);
// displays 2.7183
printf("%20.20lf", (double)4.0/7);
//displays 0.57142857142857139685
```

Floating Point Numbers

IEEE 754 standard ...

- C floats almost always IEEE 754 single precision (binary32)
- C double almost always IEEE 754 double precision (binary64)
- C long double might be IEEE 754 (binary128)
- scientific notation with *fraction* F and *exponent* E
- numbers have form $F \times 2^E$, where both F and E can be -ve
- `INFINITY` = representation for ∞ and $-\infty$ (e.g. 1.0/0)
- `NAN` = representation for invalid value (e.g. sqrt(-1.0))

Fraction part is *normalised* (i.e. 1.2345×10^2 rather than 123.45)
In binary, exponent is represented relative to a bias value B

- if the unsigned exponent value is e, the actual value is $e - B$

Floating Point Numbers

Example of normalising the fraction part in binary:

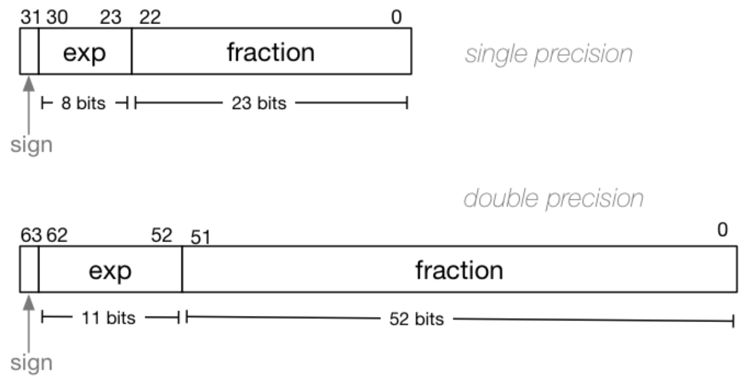
- `1010.1011` is normalized as 1.0101011×2^{011}
- `1010.`*`1011`* $= 10 + 11/16 = 10.6875$
- `1.0101011` $\times 2^{011} = (1 + 43/128) * 2^3 = 1.3359375 * 8 = 10.6875$

The normalised fraction part always has `1` before the decimal point.
Example of determining the exponent in binary:

- assume an 8-bit exponent, then bias $B = 2^{8-1} - 1 = 127$
- valid bit patterns for exponent 00000001 .. 11111110
- exponent values -126 .. 127

Floating Point Numbers

Internal structure of floating point values



More complex representation than `int` because 1.*dddededd*

IEEE-754 Single Precision example

```
$ ./explain_floating_point_representation -0.125
-0.125 is represented as IEEE-754 single-precision
by these bits: 10111110000000000000000000000000
sign | exponent | fraction
   1 | 01111100 | 00000000000000000000000
sign bit = 1
sign = -
raw exponent    = 01111100 binary
                = 124 decimal
actual exponent = 124 - exponent_bias
                = 124 - 127
                = -3
number = -1.00000000000000000000000 binary * 2**-3
       = -1 decimal * 2**-3
       = -1 * 0.125
       = -0.125
```

IEEE-754 Single Precision example

```
$ ./explain_floating_point_representation 150.75
150.75 is represented in IEEE-754 single-precision
by these bits: 01000011000101101100000000000000
sign | exponent | fraction
   0 | 10000110 | 00101101100000000000000
sign bit = 0
sign = +
raw exponent    = 10000110 binary
                = 134 decimal
actual exponent = 134 - exponent_bias
                = 134 - 127
                = 7
number = +1.00101101100000000000000 binary * 2**7
       = 1.17773 decimal * 2**7
       = 1.17773 * 128
       = 150.75
```

IEEE-754 Single Precision example

```
$ ./explain_floating_point_representation inf
inf is represented as IEEE-754 single-precision
by these bits: 01111111100000000000000000000000
sign | exponent | fraction
   0 | 11111111 | 00000000000000000000000
sign bit = 0
sign = +
raw exponent    = 11111111 binary
                = 255 decimal
number = +inf
```

IEEE-754 Single Precision example

```
$ ./explain_floating_point_representation NaN
NaN is represented as IEEE-754 single-precision
by these bits: 01111111110000000000000000000000
sign | exponent | fraction
   0 | 11111111 | 10000000000000000000000
sign bit = 0
sign = +
raw exponent    = 11111111 binary
                = 255 decimal
number = NaN
```

Exercise: Floating point $\rightarrow$ Decimal

Convert the following floating point numbers to decimal.
Assume that they are in IEEE 754 single-precision format.

```
0 10000000 11000000000000000000000

1 01111110 10000000000000000000000
```