Numbers and Representation

Bit = 1 or 0 Bytes = 8 bits

We use bits to represent numbers in computer in base 2 (binary).

Unsigned Integers

- no negative numbers
- only integers
- don't need to know size of bits
- in C, unsigned int is typically: 64 bits, or 8 bytes

Binary: base 2 Decimal: base 10 Hexadecimal: base 16 (includes digits a, …, f) $\mathrm{A}_{16} = 10_{10}, \mathrm{B}_{16} = 11_{10}, ..., \mathrm{F}_{16} = 15_{10}$

Converting from binary to decimal:

Number the digits from right to left $0...n$ Be careful! Starts at 2^0, **not** 2^1.

e.g. Convert 1101_2 to decimal:

$$\begin{aligned}1101_2 = 1*2^3&+\\1*2^2&+\\0*2^1&+\\1*2^0&\\=13&\end{aligned}$$

Hexadecimal is the same, but use powers of 16 instead of 2.

Converting from decimal to binary:

Use method 1 for smaller numbers (< 128), and method 2 for larger numbers.

Method 1: Subtract the largest powers of 2, until 0 or 1.

e.g. Convert 19_{10} to binary:

$$\begin{aligned}&19 - 2^4 = 3\\&3 - 2^1 = 1\\&1 = 2^0\\&\\&19_{10} = 10011_2\end{aligned}$$

Method 2: Divide repeated by 2, until reaching 0 or 1. Then take the remainders from each division (the last division you do is the digit farthest to the left).

e.g. Convert 19_{10} *to binary:*

$$\begin{aligned}19/2 &= 9,\ r = 1\\ 9/2 &= 4,\ r = 1\\ 4/2 &= 2,\ r = 0\\ 2/2 &= 1,\ r = 0\\ 1/2 &= 0,\ r = 1\end{aligned}$$

$$19_{10} = 10011_2$$

Fast conversion from binary to hexadecimal

Group the binary number into groups of 4 digits, starting from the right side (smallest digits). Add additional zeros to the left side if it does not line up to make a final group of 4. Then convert each group of 4 digits into a hexadecimal digit. Notice how in binary, 4 digits represent 1 hexadecimal digit (thanks to 16 being a power of 2).

e.g. Convert 11011101010_2 *to hexadecimal:*

$$11011101010_2 \rightarrow 110, 1110, 1010$$

$$\begin{aligned}&110_2 = 0110_2 = 6\\ &1110_2 = 14_{10} = \mathrm{E}_{16}\\ &1010_2 = 10_{10} = \mathrm{A}_{16}\end{aligned}$$

$$11011101010_2 = 6\mathrm{EA}_{16}$$

Helpful for checking unsigned conversions:

https://www.binaryhexconverter.com/binary-to-decimal-converter https://www.binaryhexconverter.com/hex-to-decimal-converter

Signed Integers

- represents negatives as well as positives
- we use "two's complement" to represent
- we can only represent half of the positive numbers of unsigned integers (but the other half is negatives)

- *we have to know the number of bits*

Converting from signed binary integer to decimal

First, look at the most significant bit (the leftmost). Make sure that you add any zeros to the left side if the number is not of the length of the needed bits. If it starts with 1, it is negative. Otherwise, if it starts with 0, it is a positive number and conversion is same as a unsigned integer.

If it is negative: 1. Flip the bits in the number (0 to 1, 1 to 0) 2. Add 1 to the resulting flipped binary number 3. Convert as normal 4. Don't forget to add on the negative!

e.g. Convert 4-bit signed int 1001_2 *to decimal:* We can see that 1001 is negative, because the first digit is 1.

Flip the bits: $1001 \to 0110$ Add 1 (in binary): $0110_2 + 1_2 = 0111_2$ Convert: $0111_2 = 7_{10}$ Add the sign: -7

e.g. Convert 4-bit signed int 11_2 *to decimal:* 11 is positive, because in 4-bits it is 0011 with first digit 0. Convert as normal.

$$11_2 \to 0011_2 0011_2 = 3_{10}$$

Converting from signed decimal to signed binary integer

Similar to the above. We first check if the base 10 number is negative or positive. If positive, do a normal conversion. If negative, we need to know how many bits there are in the binary representation.

If negative: 1. We convert the number as if it is positive to binary. 2. We add any zeros to the left hand side of the binary number to get to the number of bits in the representation. 3. Then we flip the bits and add 1 to get the *two's complement*, which is the signed binary representation.

e.g. Convert -7_{10} to binary: Convert: $7_{10} = 0111_2$ Flip the bits: $0111 \to 1000$ Add 1 (in binary): $1000_2 + 1_2 = 1001_2$ This is the two's complement form: 1001

Helpful for checking signed conversions:

http://www.exploringbinary.com/twos-complement-converter/

Floating Point

Similar to scientific notation, but in binary.

3 components: 1. sign bit (1 bit) 2. exponent bits (depends) 3. mantisa / fraction bits (depends)

Also important: - bias value (typically about half of the max exponent number) - normalized: implicit 1._ added in front of fraction bits - denormalized: when exponent bits are all 0, implicit 1._ becomes implicit 0._

The exponent bits are treated like a positive binary integer, but we subtract the *bias* to get the true exponent value.

sign = $-$ if 1, $+$ if 0 exponent = exponent bits $-$ bias fraction = 1.fraction bits number = sign $2^{\text{exponent}} *$ fraction

When denormalized, (all exponent bits are 0): fraction = 0.fraction bits

IEEE Floating Point Reference:

https://www.h-schmidt.net/FloatConverter/IEEE754.html