Chapter 1: Binary representation

- Anything can be represented as bits [0 or 1], where high voltage wire = 1 (true) and 0 (false) for low voltage.
 - Since each bit has either 2 values, and there are N bits total, the total number of possible unique combinations of 1s and 0s for an N-bit number is 2 x 2 x 2 ... N times, which is 2N.
 - For the first bit, it can be either 1 or 0; for the second bit, it can also be either 1 or
 And so on for all N bits.
 - For example, if N=3, then there are 3 bits.
 - Number Systems

Decimal - Base10	Binary - Base2	Octal - Base8	Hexadecimal - Base16
[0, 1, 29]	0 and 1	[0, 1, 27]	[09] [A[10]F[15]]

• 1.1 Base Conversions

- o Decimal to Binary conversion
 - Put n^2 on the right and stop at n
 - !! Or Successive Division \rightarrow divide each n % 2 \rightarrow go from MSB to LSB
- Binary to Decimal conversion
 - From right to left, multiple binary to 2ⁿ then add the 1 bits.
 - You can ignore the 0 bits
- Binary to Hex conversion
 - Left pad the number with 0's to make 4 bit groups
 - Convert each group its appropriate hex representation
- Hex to Binary Conversion
 - Use the hex \rightarrow binary chart. Expand each digit to its binary representation
 - Drop any leading zeros
- Useful table

ful les	20	0
	21	2
	22	4
	23	8
	24	16
	25	32
	26	64
	27	128
	28	256
	29	512
	210	1024

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

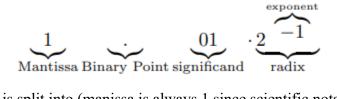
• 1.2 Numeric representation

- So in binary, the way numbers are represented as unsigned or signed affects the range of values possible and how overflow/underflow is handled.
- o Computer programs need to account for finite storage and prevent overflow.
 - 1. Unsigned integers
 - Decimal integers directly converted to binary. Can't be negative.
 - Eg. $5 \rightarrow 101$
 - 2. Signed integers
 - Also decimal integers converted to binary. Extra bits for if pos/neg.
 - Eg. $5 \to 0101$ and $-5 \to 1101$
- Binary math follows a decimal math algorithm although computers have a finite number of bits for each number. If the result exceeds allocated bits, leftmost bits are lost (*overflow*).
 - Eg. Unsigned: 11111111 = 255 (max positive)
 - Eg. Signed: 01111111 = 127 (max positive), 10000000 = -128 (min negative)
- o 1.2.1 Sign and Magnitude
 - The leftmost bit is the sign bit. If the sign bit is 1 = negative number. If sign bit is 0 = positive number
 - The remaining bits represent the magnitude or abs value of the number
 - Eg. 01010110 = +86 and 11011010 = -154
 - Downside is that sign and magnitude representation does not handle overflow well because it wraps incorrectly to the minimum neg. number.
- o 1.2.2 One's Complement
 - To fix bit wrapping, to form a negative number, we flip each bit.
 - To represent a positive number, the binary bits are simply the normal binary representation of the number.
 - Eg. positive 5 is 0101 and negative 5 is 1010 (flip each bit)
- o 1.2.3 Two's Complement
 - To convert a decimal \rightarrow two's complement binary:
 - If positive, convert to normal binary
 - If negative, convert positive version to binary by a) inverting each bit and b) adding 1 to the result

$$\circ$$
 Eg. 5 \rightarrow 0101 || -5 \rightarrow 0101 \rightarrow invert: 1010 \rightarrow +1: 1011

- 1.2.4 Bias Encoding
 - Is another way to represent signed numbers in binary. It works by adding a bias value to the unsigned binary representation. So if we take an unsigned number and add the bias -8, we get the signed representation.
 - Eg. 0101 (unsigned 5) + bias -8 = 1011 (-3 signed)

- 1.3 Floating Point Representation
 - \circ To represent decimal \rightarrow binary, we use floating point representation.
 - Any number in scientific notation has the following components:
 - Mantissa: the number in front of the point
 - Sginifcand: the digits after the point
 - Radix: the base of the number
 - Exponent: how many times the point should be shifted to recover the og #



- Number is split into (manissa is always 1 since scientific notation, so forget it):
 - Sign (1 bit): positive or negative
 - Exponent (8 bits): power of 2 (-127 bias)
 - Significand (23 bits): digits after binary point 2
- We can convert a floating point number back to decimal by:

$$n = (-1)^s (1 + significand) \cdot 2^{exponent-127}$$

n = decimal number

sign = 0 for positive, 1 for negative

significand = fraction bits

exponent = exponent bits

127 =exponent bias

- A few things to notice:
 - If the exponent is larger than 8 bits, overflow will occur because we only have 8 bits for the exponent.
 - If a negative exponent is more than 8 bits, underflow occurs for the same reason.
 - There are two zero values positive 0 and negative 0.
- Certain sequence in floating point representation are designated to be specific numbers:

Exponent	Significand	Object
0	0	0
0	nonzero	denorm
1-254	anything	± #
255	0	$\pm \infty$
255	nonzero	NaN

O Denormalized numbers: where we don't have implicit 1 as the mantissa which allows very small numbers and acts as if the exponent is -126.