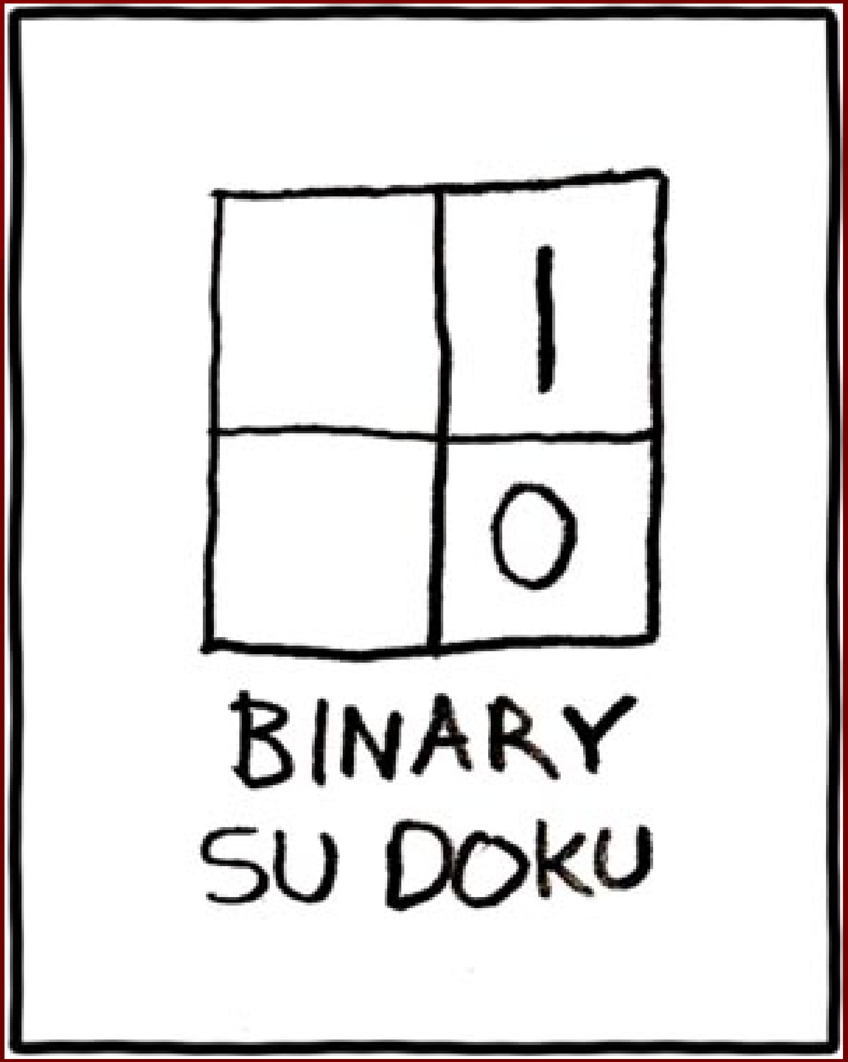
CS140:
Backgrounder: Binary Math
Addition, Subtraction,
Negative and Real Numbers

John Magee

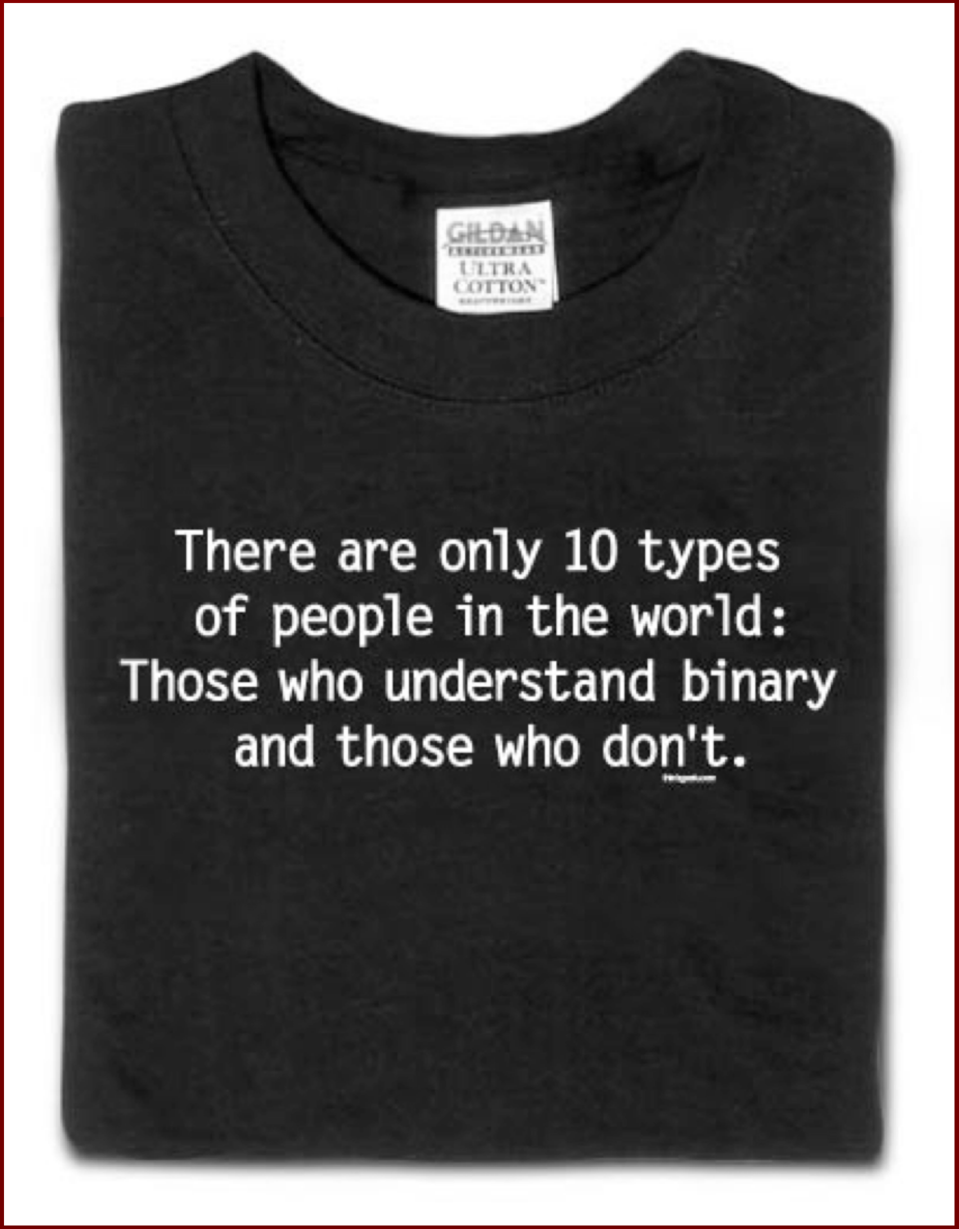
25 January 2017

Some images courtesy Wikimedia Commons

Some slides credit Aaron Stevens



– Source: xkcd.org


GILDAN
ULTRA
COTTON
There are only 10 types
of people in the world:
Those who understand binary
and those who don't.

Overview/Questions

– Oh yeah, binary numbers.
– How can we do arithmetic with binary numbers?
– What about negative numbers?
– And real numbers?

Binary Representations

Recall: a single bit can be either a 0 or a 1

What if you need to represent more than 2 choices?

1 Bit	2 Bits
0	00
1	01
	10
	11

Binary Combinations

1 Bit	2 Bits	3 Bits	4 Bits	5 Bits
0	00	000	0000	00000
1	01	001	0001	00001
	10	010	0010	00010
	11	011	0011	00011
		100	0100	00100
		101	0101	00101
		110	0110	00110
		111	0111	00111
			1000	01000
			1001	01001
			1010	01010
			1011	01011
			1100	01100
			1101	01101
			1110	01110
			1111	01111
				10000
				10001
				10010
				10011
				10100
				10101
				10110
				10111
				11000
				11001
				11010
				11011
				11100
				11101
				11110
				11111

Binary Combinations

What happens every time you increase the number of bits by one?

How many combinations can ***n*** bits *represent?*

1 bits → 2 combinations, e.g. range (0,1)

2 bits → 4 combinations , e.g. range (0,3)

3 bits → 8 combinations, , e.g. range (0,7)

4 bits → 16 combinations, , e.g. range (0,15)

... and so forth

n bits can represent $\mathbf{2}^{n}$ possible combinations

How could we represent really big numbers?

Arithmatic

Let’s remember 3rd grade...

How did you do addition and subtraction?

Recall: Add with Carry

Recall how to add 2 multi-digit decimal numbers:

Example: adding 35 + 77

5 + 7 is 12, or 2 with a carry of 1

Carry Values

```
 1 1
   3 5
 + 7 7
 -----
 1 1 2
```

Addend Values

Sum

Remember: work right to left.

Binary: Add with Carry

Remember that there are only 2 digits in binary, 0 and 1

1 + 1 is 0 with a carry of 1

Carry Values

```
  1   1 1 1 1 1
    1 0 1 0 1 1 1
  + 1 0 0 1 0 1 1
  ---------------
  1 0 1 0 0 0 1 0
```

Operands

Sum

Hint: work right to left.

Binary Subtract with Borrow

Remember borrowing? **Apply that concept here:**


```
   1 2
    2 0 2
 1 0 1 0 1 1 1
-  1 1 1 0 1 1
 -------------
 0 0 1 1 1 0 0
```

(check: 87 – 59 = 28)

Hint: work right to left.

Integer Numbers

Natural Numbers

Zero and any number obtained by repeatedly adding one to it.

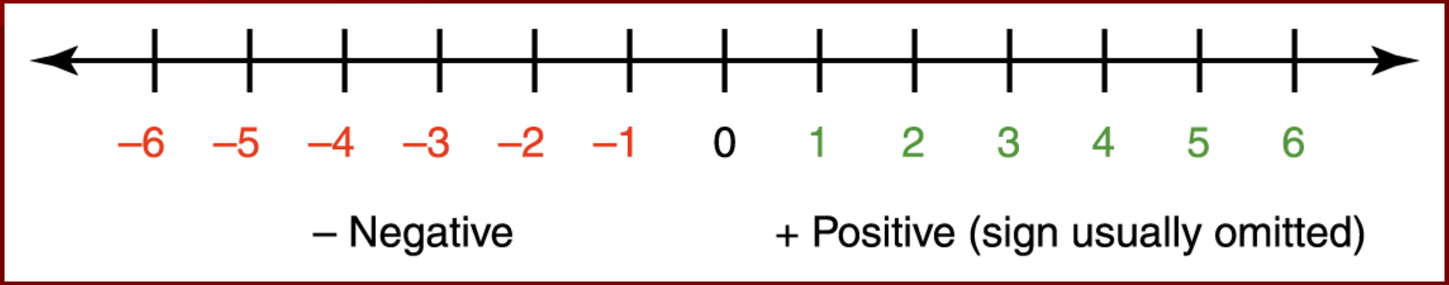
Examples: 100, 0, 45645, 32

Negative Numbers

A value less than 0, with a – sign

Examples: -24, -1, -45645, -32

Representing Negative Values



Signed-magnitude numbers

The sign represents the ordering, and the digits represent the magnitude of the number.

Representing Negative Values

There is a problem with the sign-magnitude representation:

There is a plus zero and minus zero, which causes unnecessary complexity.

Solution:

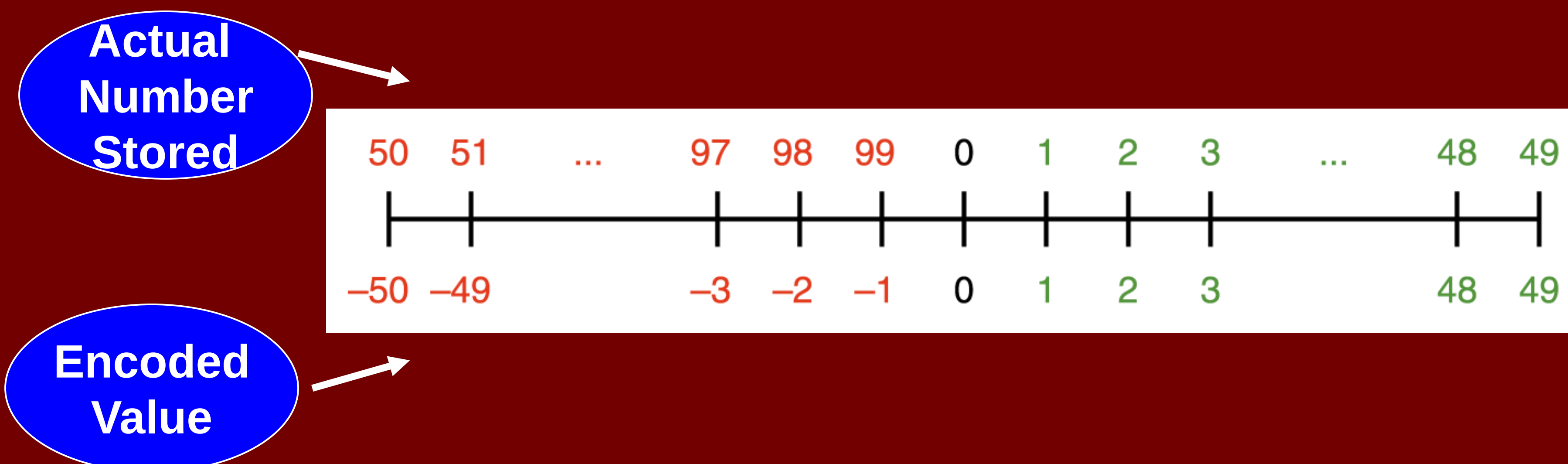
Store all numbers as natural integer values, with half of them representing negative numbers

Representing Negative Values

An example using two decimal digits,

let 0 through 49 represent 0 through 49

let 50 through 99 represent -50 through -1



This representation scheme is called the ten's complement.

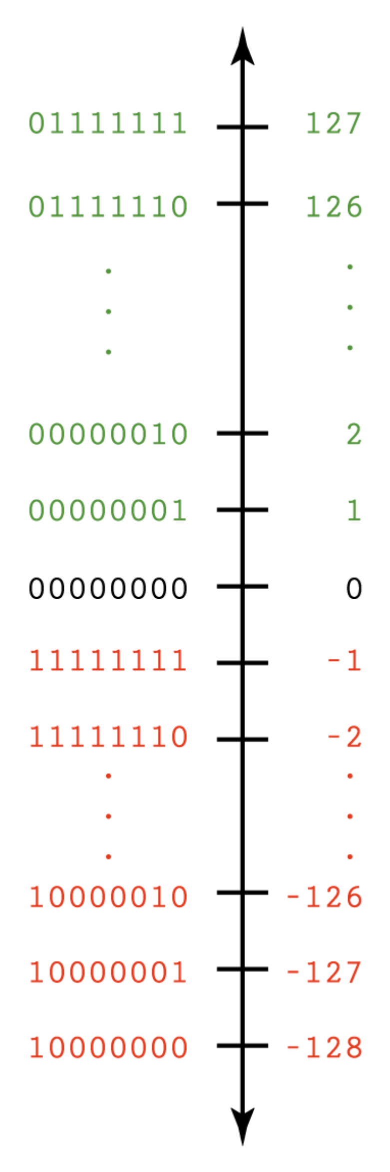
Representing Negative Values

Two's Complement
Using binary numbers, we call this the *two's complement*.

All numbers are stored as positive binary numbers.

Half are *encoded* to be interpreted as negative.

What do you observe about the left-most bit?



Two’s Complement Numbers

How To Calculate a Two's Complement Number

First, find the equivalent binary number.
If the decimal number was positive: you're done.
If the decimal number was negative: invert all the bits,
and add 1 (with carries as needed).

Examples

25 decimal is 00011001 binary. It’s positive, so all done.
-25 decimal:

Begin with binary:	00011001	
Invert all the bits to get:	11100110	
Add 1 to get:	11100111	(decimal value -25)

Two's Complement Arithmetic

With 2s complement, we can use addition instead of subtraction -- much easier!

```
 123          01111011
-25          +11100111
----------------------
 98           01100010
```

(last bit carried out is ignored)

Number Overflow

If each value is stored using eight bits, consider adding 127 to 3:

```
    01111111
  + 00000011
  ----------
    10000010
```

How do we interpret the value 10000010?

Adding these two positive integers yields a negative integer. This is called overflow.

Number Overflow

We interpret the value `10000010` as decimal number -126. How did that happen?

– The left-most bit is 1, so we know it is negative.
– The most negative signed 8-bit number is binary `10000000`, which is -128 (-2^7) in decimal.
– Add binary `00000010` (2) to get `10000010`, which is -126 decimal.

What does one do about overflow?

Representing Real Numbers

Real numbers

A number with a whole part and a fractional part

104.32, 0.999999, 357.0, and 3.14159

For decimal numbers, positions to the right of the decimal point are the tenths position: 10^{-1}, 10^{-2} , 10^{-3} …

Floating Point Numbers

A real value in base 10 can be defined by the following formula:

$$\text{sign} * \text{mantissa} * 10^{\text{exp}}$$

The mantissa (or precision) is a decimal number.

The representation is called **floating point** because the number of digits of precision is fixed but the decimal point "floats."

Example: 12345.67 can be expressed as 1.234567 X 10^4

Binary Floating Point Numbers

Same rules apply in binary as in decimal.

Decimal point is actually the **radix point**.

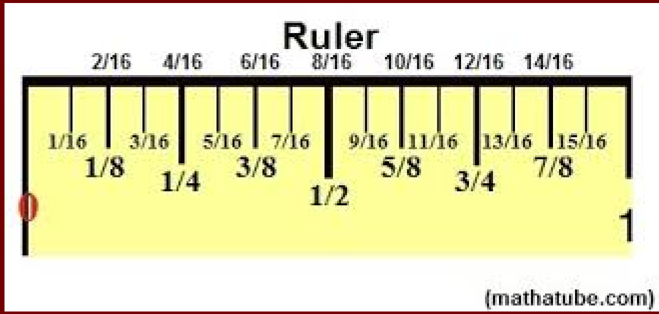
Positions to the right of the radix point in binary are

2^{-1} (one half),

2^{-2} (one quarter),

2^{-3} (one eighth)

...



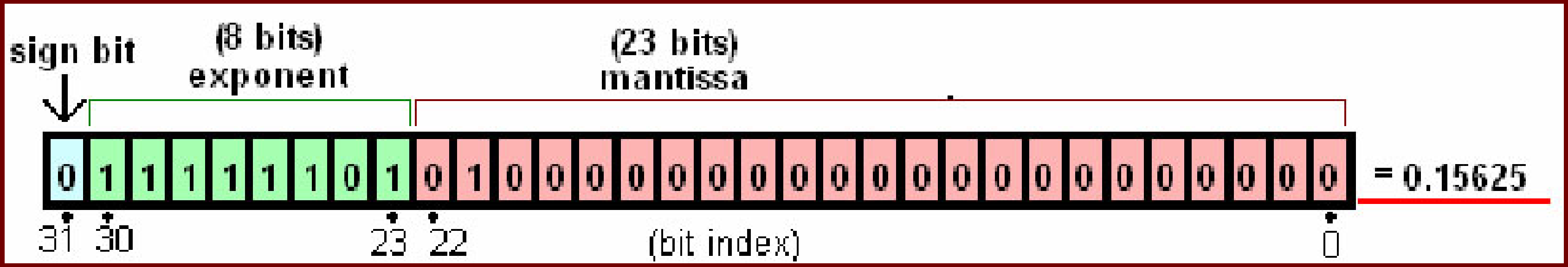
Binary Floating Point Numbers

IEEE 754

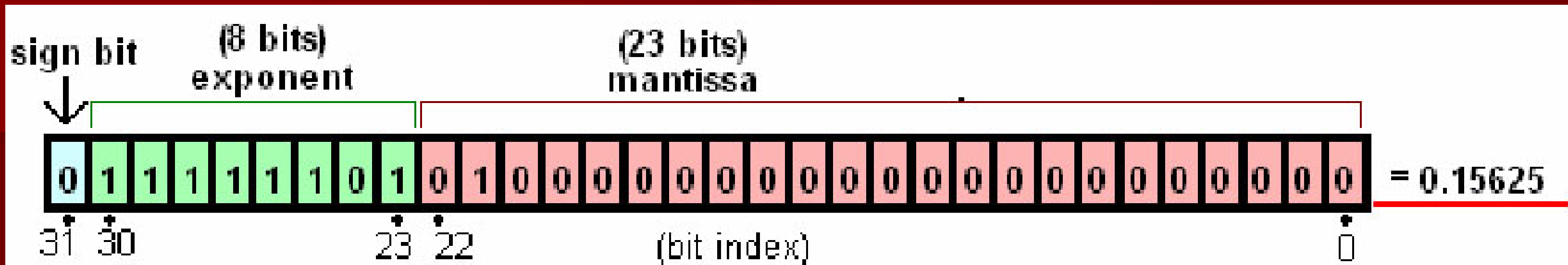
A standard for representation of binary floating point numbers, as expressed by the formula:

sign * mantissa * 2^{exp}

A single-precision binary floating-point number is stored in a 32-bit word:



Binary Floating Point Numbers



Decoding the example shown above:

- the sign is 0 → this is a positive number
- the exponent is 11111101 → −3 (decimal)
- the mantissa is 1.01 (binary) → 1.25 (decimal)
- The represented number is therefore

$$+1.25 \times 2^{-3} = +0.15625 \text{ in decimal.}$$

Binary Floating Point Numbers

32-bit floating point numbers ...

– The smallest non-zero positive, and largest non-zero negative numbers are:

$\pm 2^{-149} \approx \pm 1.4012985 \times 10^{-45}$

– The largest finite positive and smallest finite negative numbers are:

$\pm(2^{128} - 2^{104}) \approx \pm 3.4028235 \times 10^{38}$

Binary Floating Point Numbers

IEEE 754 also defines a double-precision 64-bit binary floating point number.

– You can read it for yourself.

MORE IMPORTANT

– IEEE754 binary floating point numbers are only *approximations* of decimal floating point numbers.

– Not all decimal floating point numbers can be exactly represented in IEEE 754. Example:

$$0.10 \approx \frac{0}{2}+\frac{0}{4}+\frac{0}{8}+\frac{1}{16}+\frac{1}{32}+\frac{0}{64}+\frac{0}{128}+\frac{1}{256}+\frac{1}{512}+\frac{0}{1024}+\frac{0}{2048}+\frac{1}{4096}+\cdots$$

Take-Away Points

– Binary Addition
– Two's Complement
– IEEE754: Binary Floating Point Numbers