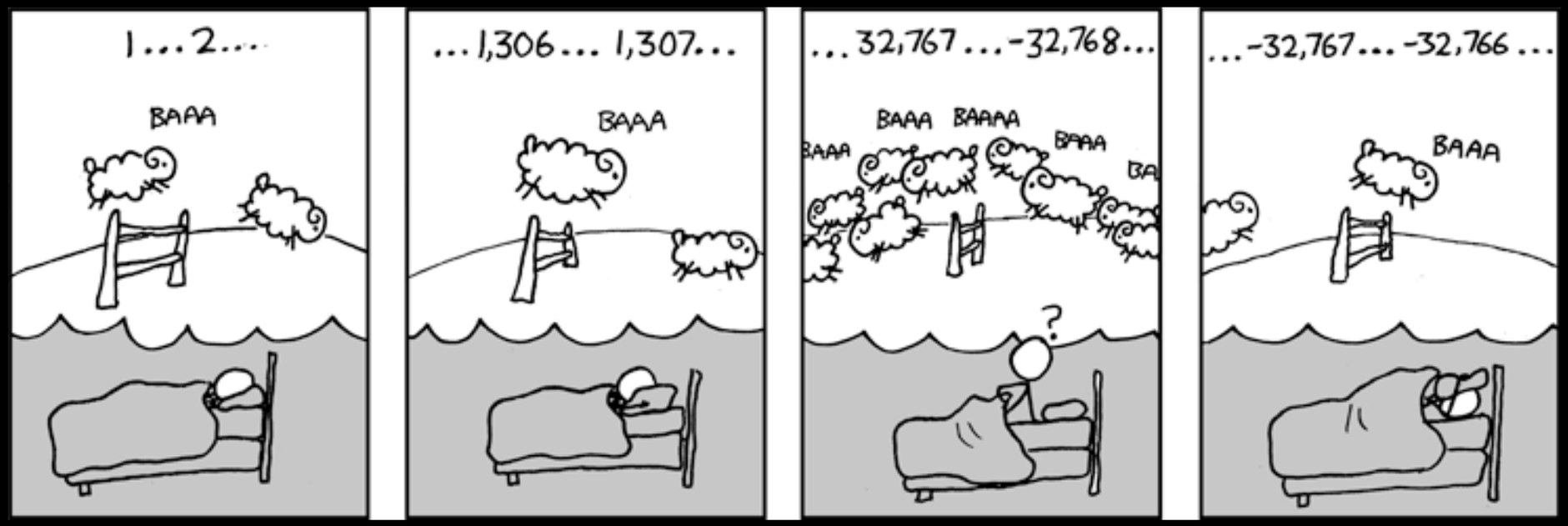
Week 2



- Data types and Input/Output
- Basic Arithmetic and Flow Control

What do we want to do?

- We want to give C numbers and do calculations on them
- Check whether a number is prime
- Check the price per square cm for pizza
- Calculate mortgage rate
- Find out how high your blood alcohol level is after some beers
- Find out how many seconds you have to work to buy a beer

What do we need?

- Some way to store numbers
- Do some quick maths on them
- Print the result

Storing numbers in C

- Different datatypes for different uses
- You want to store an Integer or Real Number?

- C knows three fundamental types:
  Integer (0,255, -24, 8, …)
  Float (2.5, 3.9, -215.6,…)
  Character (‚a‘, ‚L‘, ‚;‘, ‚m‘)

What does an Integer represent?

- A Number from: $\mathbb{Z}$ or $\mathbb{N}_0$
- What range?
- How is it stored in memory?

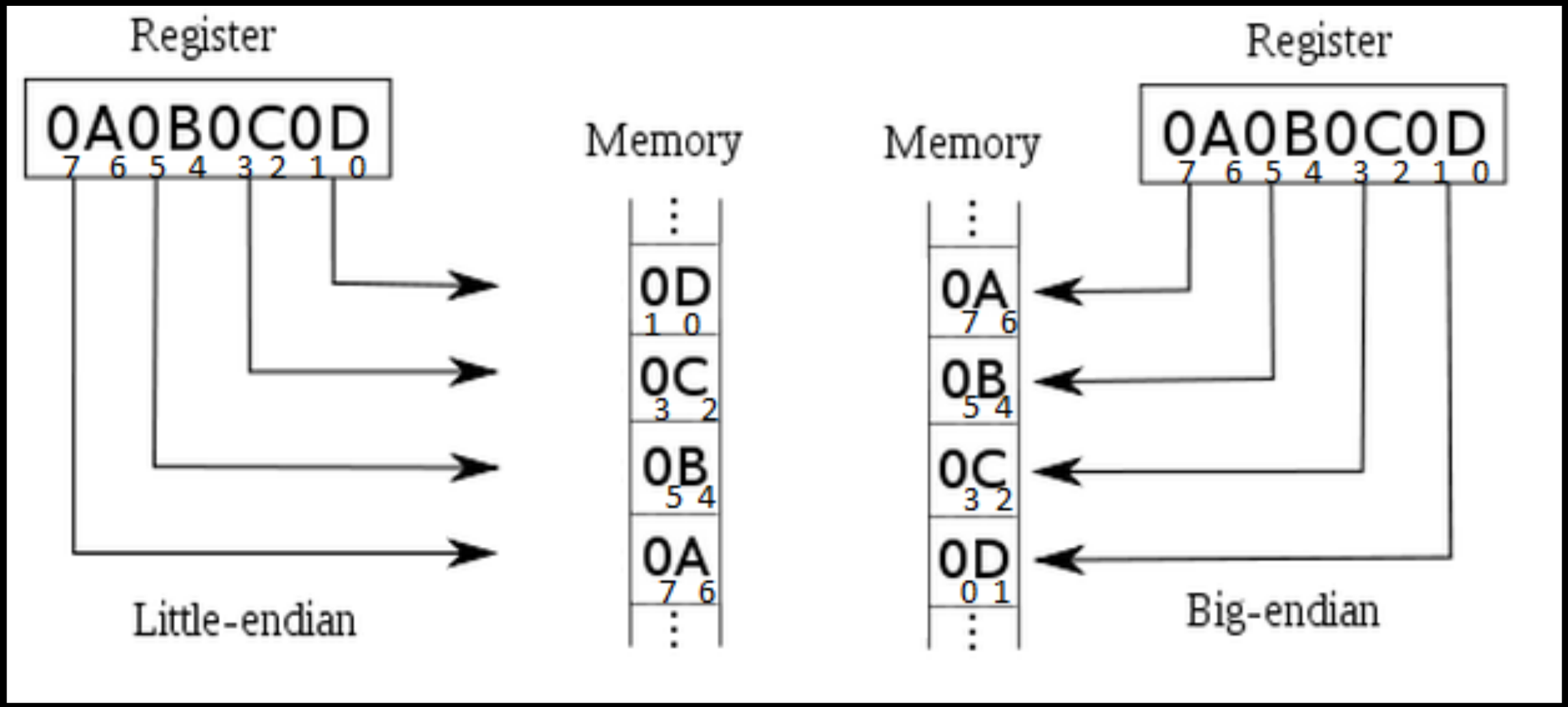
Which numbers?

- Use signed or unsigned to specify whether it is positive only or positive and negative

What range?

- The size of an int is specified as follows: At least 2 bytes
- Usually, but not always 4 bytes
- -32,768 to 32,767 or -2,147,483,648 to 2,147,483,647

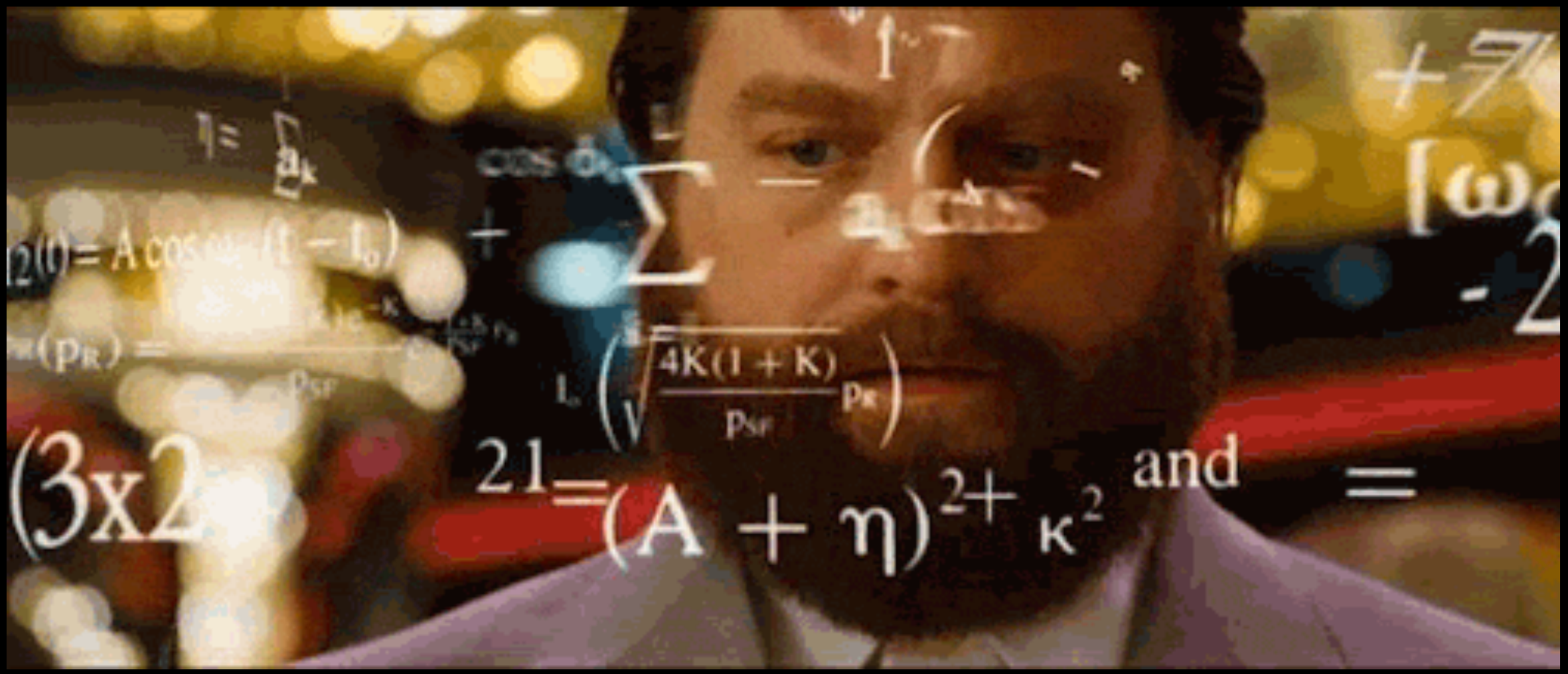
How is it aligned in memory?



What if I need other sizes?

- short: at least 2 bytes
- long: at least 4 bytes
- long and short can be used before types
- E.g. if you need a unsigned int with at least 8 bytes of memory: simply use a unsigned long long int

Trying to use correct types like



How to deal with other platforms?

Careful with int

- Int is fine for most use cases
- Int is guaranteed to hold at least 2 Bytes
- If your calculations assume 4 Bytes, your code is wrong on some systems

Rule 1:

Friends don’t let friends use standard integer data types for all problems

Introducing:

<stdint.h>

<stdint.h>

- Header file
- Gives you access to types like
  uint8_t
  int16_t
  uint64_t
- It won’t matter where your code runs, it will always have the same size
- But may be slower than int

Example!

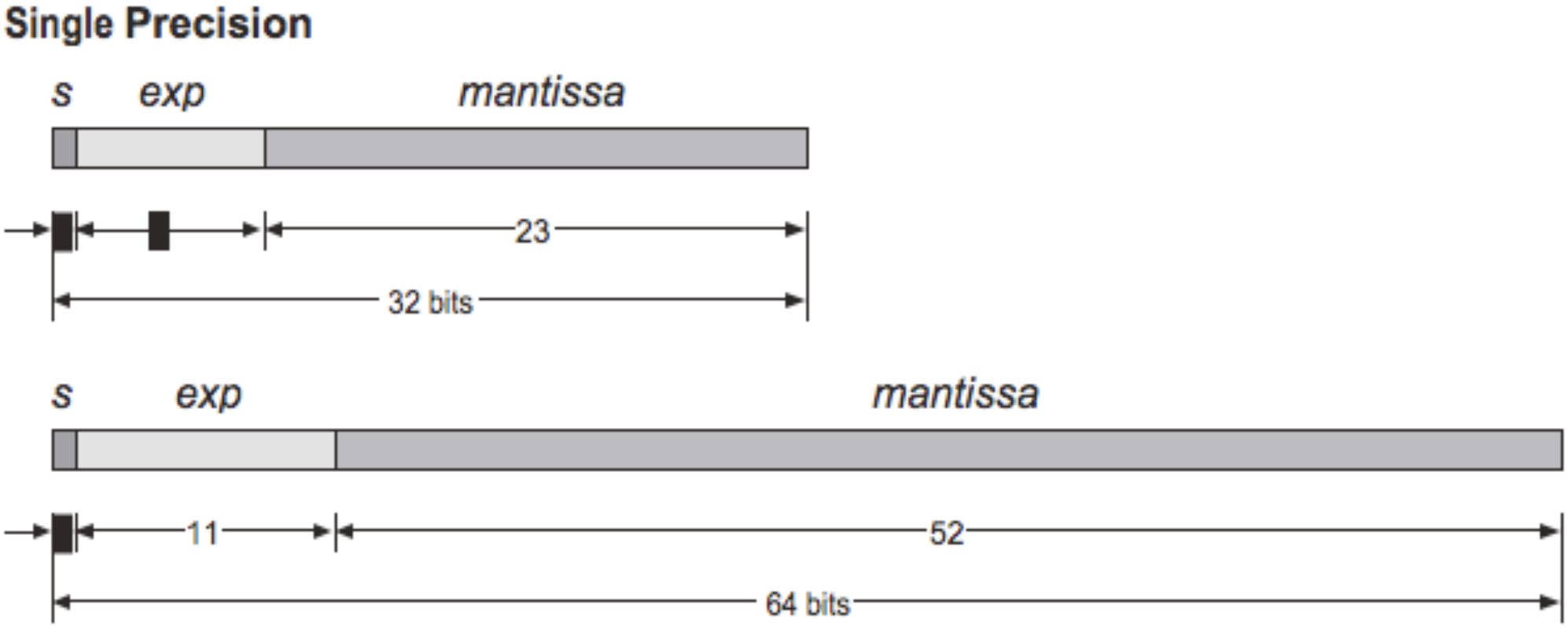
```c
#include <stdint.h>
#include <stdio.h>

int main(){
    uint8_t n_12 = 12;
    uint8_t n_250 = 250;
    uint8_t n_tooMuch = n_12 + n_250;
    printf("%u\n", n_tooMuch);
    return 0;
}
```

Where is undefined behavior?

Float

- Computer are not able to properly represent real numbers
- Sign, Exponent, Mantissa
- (+-)1.(Mantissa) * 2^Exponent
- Most real numbers are approximations on the pc

Float vs Double



Why does this never stop?

```c
int main(){
    double d = 0.0;
    while (d != 100.0){
      d += 0.1;
      printf("%f\n", d);
    }

    return 0;
}
```

Rule 2:

Never compare floating point numbers with == or !=

char

- Size: smallest addressable unit
- In most cases 1 Byte
- char can contain an ASCII symbol
- Standard does not specify whether char is signed or unsigned by default

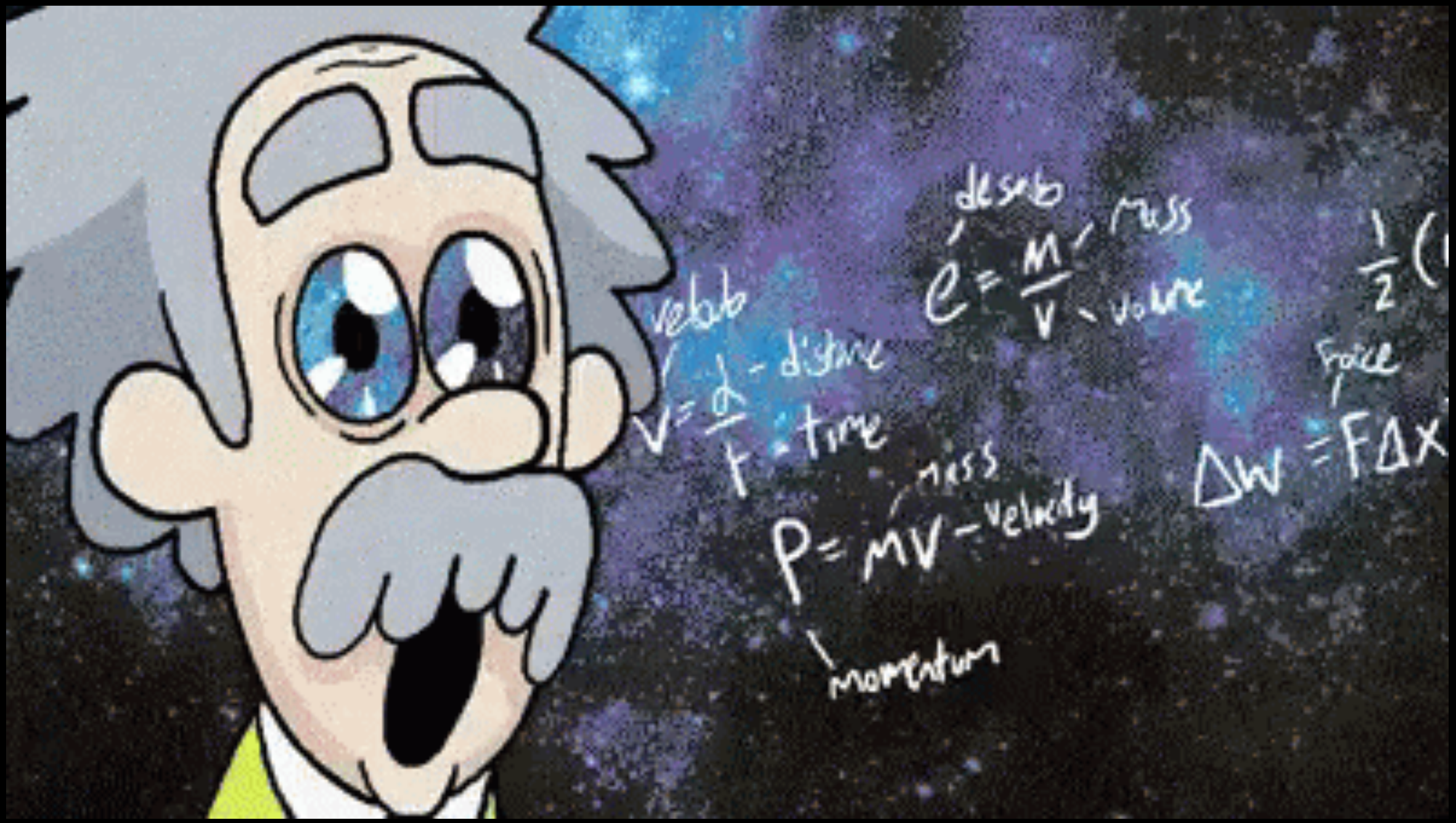
```
char c = 'b';
```

Prepare to be mind blown

- How does a char look like in memory?
- Exactly like a number
- The following is valid C

```c
#include <stdio.h>

int main(){
    char c = 'b';
    printf("%c\n",c);
    c++; // c can be incremented
    printf("%c\n",c);
    c = 67; // ascii code of 'C'
    printf("%c\n",c);
    c = c + 5; // move 5 letters forward
    printf("%c\n",c);
    if('a' > 'A'){ // 'a' == 97, 'A' == 65
        printf("%c\n",'a');
    }
    return 0;
}
```

MATH


density
$\rho = \frac{M}{V}$ - mass / volume
velocity
$v = \frac{d}{t}$ - distance / time
$P = MV$ - momentum
mass
velocity
$\frac{1}{2}$(
$\Delta W = F \Delta x$
Force

Operators in C

- +, -, *, /, %, = are well known
- ++, — could be known
- <<, >>, &, | , ~, ^, obscure black magic

Bitwise arithmetic

- Operations on binary representation of numbers
- Really fast
- Can improve speed if used correctly
- Might break your code if wrong

What the *curseword* is bitwise arithmetic?

- Normal arithmetic: Operate on numbers
  5 + 12 = 17

- Bitwise arithmetic: Iterate through bits of numbers and compare bitwise

0000 0101 & 0000 0111

&	7	6	5	4	3	2	1	0
5	0	0	0	0	0	1	0	1
6	0	0	0	0	0	1	1	0
4	0	0	0	0	0	1	0	0

| and &

- Bitwise OR and AND
- 10 == 0b 0000 1010
  5 == 0b 0000 0101
  5|10 == 0b 0000 1111
  5&10 ==. 0b 0000 0000

<< and >>

- << and >> shift the binary representation of a number by some amount of digits

- 5 == 0b00000101
  5 >> 1 == 0b00000010 == 2 (Divide by 2^1)
  5 >> 2 == 0b00000001 == 1 (Divide by 2^2)
  5 << 1 == 0b00001010 == 10 (Multiply by 2^1)
  5 << 2 == 0b00010100 == 20 (Multiply by 2^2)
  5 << 3 == 0b00101000 == 40 (Multiply by 2^3)
  5 << 4 == 0b01010000 == 80 (Multiply by 2^4)

What do we want to do?

- Check whether a number is prime
- Check the price per square cm for pizza
- Calculate mortgage rate
- Find out how high your blood alcohol level is after some beers
- Find out how many seconds you have to work to buy a beer

Bitwise FUN

- Print out the binary representation of an integer number
- Print out the binary representation of a floating point number
- Print all uint16_t numbers, which contain exactly 3 ‚1‘ in their binary representation
- Do bitwise arithmetic on signed and unsigned numbers mixed
- Find the biggest floating point number your pc can represent
- Find the biggest int number your pc can represent