Review: Programming in C

CS 3410: Computer System Organization and Programming

[K. Bala, A. Bracy, E. Sirer, Z. Susag, H. Weatherspoon, D. Garcia, and K. Laeufer]

Adding your own Types: typedef and struct

- typedef allows you to define new names for existing types:
  - `typedef uint8_t BYTE;`
- struct allows you to define a structured group of variables

```
typedef enum cardsuit{DIAMONDS, SPADES, HEARTS, CLUBS} suit_t;
typedef struct cardstruct {
    int rank;
    suit_t suit;
} card_t;

card_t card;
card.rank = 1;
card.suit = SPADES;
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}

int main() {
 greet("3410");
}
```

Compile

```
$ rv gcc lib1.c
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}

int main() {
 greet("3410");
}
```

Compile

```
$ rv gcc lib1.c
```

Execute

```
$ rv qemu ./a.out
    Hello, 3410!
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Compile

```
$ rv gcc lib2.c
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Compile

```
$ rv gcc lib2.c
```

```
lib2.c:3:2: error: implicit declaration of function ‘greet’
   3 | greet("3410");
     | ^~~~~
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Compile

```
$ rv gcc lib2.c
```

```
lib2.c:3:2
function ‘g
   3 | greet("341
     | ^~~~~
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
void greet(const char* name);
int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Compile

```
$ rv gcc lib3.c
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
void greet(const char* name);
int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Compile

```
$ rv gcc lib3.c
```

Execute

```
$ rv qemu ./a.out
```

```
Hello, 3410!
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
void greet(const char* name);
int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Exact name does not matter!

Compile

```
$ rv gcc lib3.c
```

Execute

```
$ rv qemu ./a.out
    Hello, 3410!
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
void greet(const char* name);
int main() {
 greet("3410");
}
```

← Function declaration.

↑ Exact name does not matter!

```c
void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

← Function definition.

Header file: greet.h

```
void greet(const char* name);
```

Function Declarations, Headers, Libraries

```c
#include <stdio.h>
#include "greet.h"


int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Separating files: greet.c

```
#include <stdio.h>
#include "greet.h"




void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Separating files: main.c

```
#include <stdio.h>
#include "greet.h"


int main() {
 greet("3410");
}
```

Compile **.c** files together

Compile

```
$ rv gcc main.c greet.c
```

Execute

```
$ rv qemu ./a.out
```

Floating Point Numbers

CS 3410: Computer System Organization and Programming

[K. Bala, A. Bracy, E. Sirer, Z. Susag, H. Weatherspoon, D. Garcia, and K. Laeufer]

Review: Binary numbers

- What is 24_{10} in binary?
- What is -24_{10} in binary?
- What is 0b100001 in decimal?
- What is 0b110010 in decimal?

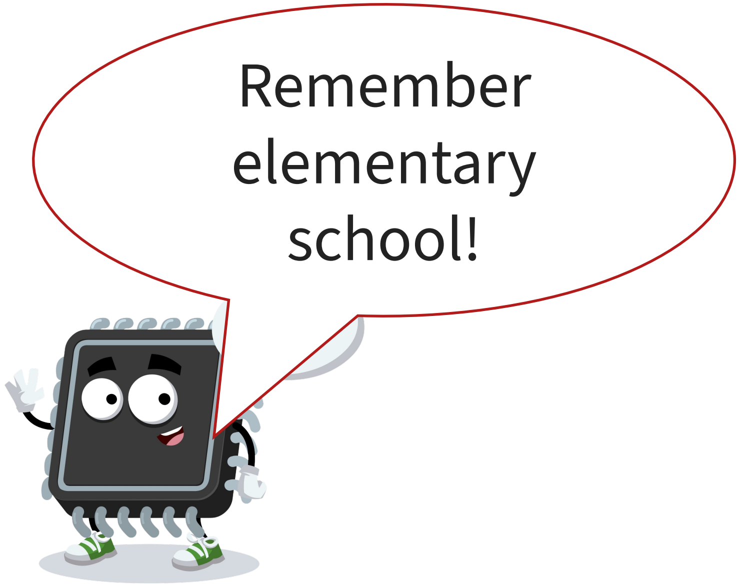
PollEv.com/cs3410

Important: Correction for last lecture

$\sim 011000_2 + 1 =$

$$
\begin{array}{rr}
-\ 24_{10} & 101000_2 \\
+\ 17_{10} & +\ 010001_2 \\
\hline
-\ 7_{10} & 111001_2
\end{array}
$$

$\sim 111001_2 + 1 = 111_2 = 7_{10}$



How to represent fractional numbers in binary?

$$637_{10} = 6 \times 10^2 + 3 \times 10^1 + 7 \times 10^0$$

$$101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 4_{10} + 0_{10} + 1_{10}$$
$$= 5_{10}$$

How to represent fractional numbers in binary?

$$63.7_{10} = 6 \times 10^1 + 3 \times 10^0 + 7 \times 10^{-1}$$

$$\begin{aligned} 101_2 &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 4_{10} + 0_{10} + 1_{10} \\ &= 5_{10} \end{aligned}$$

How to represent fractional numbers in binary?

$$6.37_{10} = 6 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2}$$

$$\begin{aligned} 101_2 &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 4_{10} + 0_{10} + 1_{10} \\ &= 5_{10} \end{aligned}$$

How to represent fractional numbers in binary?

$$6.37_{10} = 6 \times 10^{0} + 3 \times 10^{-1} + 7 \times 10^{-2}$$

$$10.1_{2} = 1 \times 2^{1} + 0 \times 2^{0} + 1 \times 2^{-1}$$

$$= 2_{10} + 0_{10} + 1/2_{10}$$

$$= 2.5_{10}$$

How to represent fractional numbers in binary?

$$6.37_{10} = 6 \times 10^{0} + 3 \times 10^{-1} + 7 \times 10^{-2}$$

$$10.1_{2} = 1 \times 2^{1} + 0 \times 2^{0} + 1 \times 2^{-1}$$
$$= 2_{10} + 0_{10} + 1/2_{10}$$
$$= 2.5_{10}$$

Fixed Point Format

How to represent fractional numbers in binary?

$$6.37_{10} = 6 \times 10^{0} + 3 \times 10^{-1} + 7 \times 10^{-2}$$

$$10.1_{2} = 1 \times 2^{1} + 0 \times 2^{0} + 1 \times 2^{-1}$$

$$= 2_{10} + 0_{10} + 1/2_{10}$$

$$= 2.5_{10}$$

Fixed Point Format

Number of bits: n = ??

How to represent fractional numbers in binary?

$$6.37_{10} = 6 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2}$$

$$10.1_2 = 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1}$$
$$= 2_{10} + 0_{10} + 1/2_{10}$$
$$= 2.5_{10}$$

Fixed Point Format

Number of bits: n = 3

Cornell Bowers CIS Computer Science

How to represent fractional numbers in binary?

$$6.37_{10} = 6 \times 10^{0} + 3 \times 10^{-1} + 7 \times 10^{-2}$$

$$10.1_{2} = 1 \times 2^{1} + 0 \times 2^{0} + 1 \times 2^{-1}$$

$$= 2_{10} + 0_{10} + 1/2_{10}$$

$$= 2.5_{10}$$

Fixed Point Format

Number of bits: n = 3

Exponent: E = ??

How to represent fractional numbers in binary?

$$6.37_{10} = 6 \times 10^{0} + 3 \times 10^{-1} + 7 \times 10^{-2}$$

$$10.1_{2} = 1 \times 2^{1} + 0 \times 2^{0} + 1 \times 2^{-1}$$
$$= 2_{10} + 0_{10} + 1/2_{10}$$
$$= 2.5_{10}$$

Fixed Point Format

Number of bits: $n = 3$

Exponent: $E = -1$

How to represent fractional numbers in binary?

$$6.37_{10} = 6 \times 10^{0} + 3 \times 10^{-1} + 7 \times 10^{-2}$$

$$1.01_{2} = 1 \times 2^{0} + 0 \times 2^{-1} + 1 \times 2^{-2}$$
$$= 1_{10} + 0_{10} + 1/4_{10}$$
$$= 1.25_{10}$$

Fixed Point Format

Number of bits: n = 3

Exponent: E = -2

How to represent fractional numbers in binary?

$$6.37_{10} = 6 \times 10^{0} + 3 \times 10^{-1} + 7 \times 10^{-2}$$

$$1.01_{2} = 1 \times 2^{2-\underline{2}} + 0 \times 2^{1-\underline{2}} + 1 \times 2^{0-2}$$

$$= \quad 1_{10} \quad + \quad 0_{10} + \quad 1/4_{10}$$

$$= \quad 1.25_{10}$$

– Fixed Point Format

Number of bits: $n = 3$

Exponent: $E = -\underline{2}$

How to represent fractional numbers in binary?

$6.37_{10} = 6 \times 10^{0} + 3 \times 10^{-1} + 7 \times 10^{-2}$

$0.101_2 = 1 \times 2^{2-\underline{3}} + 0 \times 2^{1-\underline{3}} + 1 \times 2^{0-\underline{3}}$

$= \quad 1/2_{10} \quad + \quad 0_{10} \quad + \quad 1/8_{10}$

$= \quad 0.625_{10}$

Fixed Point Format

Number of bits: $n = 3$

Exponent: $E = -\underline{3}$

How to represent fractional numbers in binary?

$$6.37_{10} = 6 \times 10^{0} + 3 \times 10^{-1} + 7 \times 10^{-2}$$

$$0.101_2 = 1 \times 2^{2-\underline{3}} + 0 \times 2^{1-\underline{3}} + 1 \times 2^{0-\underline{3}}$$

$$= \quad 1/2_{10} \quad + \quad 0_{10} \quad + \quad 1/8_{10}$$

$$= \quad 0.625_{10}$$

Fixed Point Format

Number of bits: n = 3

Exponent: E = -$\underline{3}$

If `i` is the integer value of the bits, then the represented value is:

$$i \times 2^{E}$$

How to represent fractional numbers in binary?

$6.37_{10} = 6 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2}$

$0.101_2 = 1 \times 2^{2-3} + 0 \times 2^{1-3} + 1 \times 2^{0-3}$

$= 5_{10} \times 2^{-3}$

$= 5 / 8$

$= 0.625$

Fixed Point Format

Number of bits: n = 3

Exponent: E = -3

If `i` is the integer value of the bits, then the represented value is:

$$i \times 2^E$$

Fixed Point Poll

- What is the decimal represented by 0b1001 with n=4, e=-1?
- What is the decimal represented by 0b1001 with n=4, e=2?
- What is the decimal represented by 0b1010 with n=4, e=-3?

PollEv.com/cs3410

How do you program with fixed point numbers?

- Need to statically define fixed point format for inputs and outputs of all calculations

How do you program with fixed point numbers?

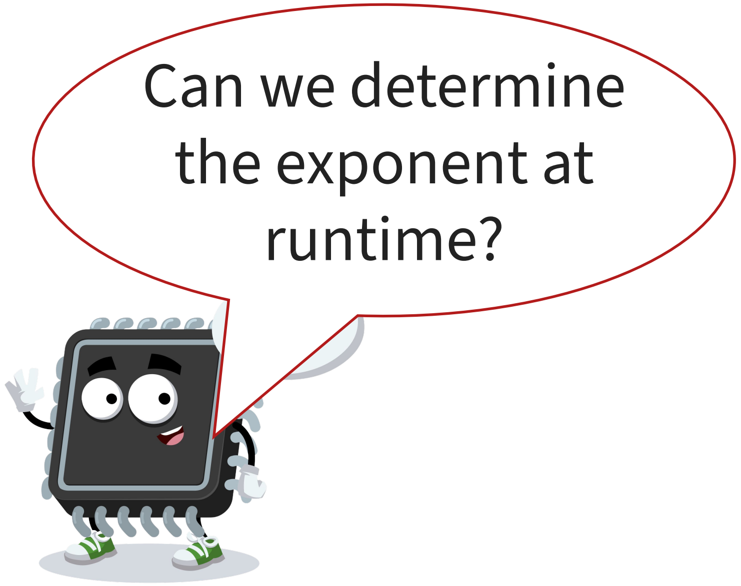
- Need to statically define fixed point format for inputs and outputs of all calculations
- **Problem #1:** input range might be unknown.

How do you program with fixed point numbers?

- Need to statically define fixed point format for inputs and outputs of all calculations
- **Problem #1:** input range might be unknown.
- **Problem #2:** need to keep track of output range.

How do you program with fixed point numbers?

- Need to statically define fixed point format for inputs and outputs of all calculations
- **Problem #1:** input range might be unknown.
- **Problem #2:** need to keep track of output range.



From Fixed- to Floating-Point Numbers

$$0.101_2 = 1 \times 2^{2-3} + 0 \times 2^{1-3} + 1 \times 2^{0-3}$$
$$= 1/2_{10} + 0_{10} + 1/8_{10}$$
$$= 0.625_{10}$$

Fixed Point Format

Number of bits: $n = 3$

Exponent: $E = -3$

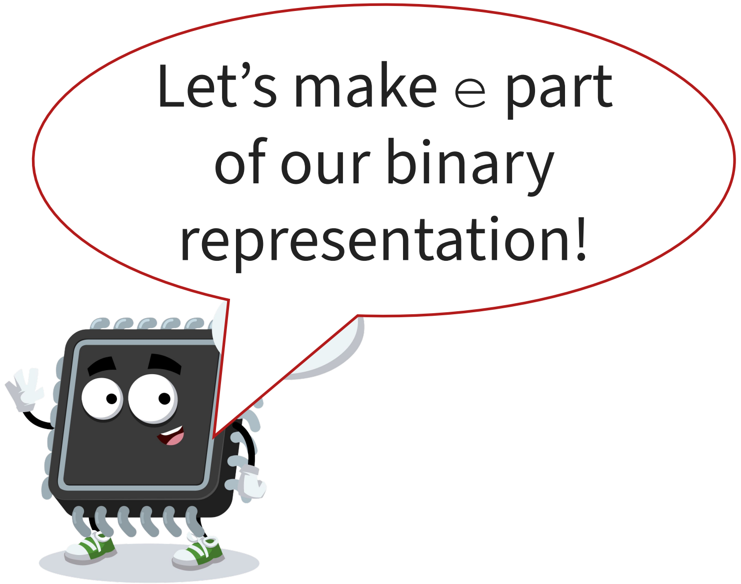
From Fixed- to Floating-Point Numbers

$$0.101_2 = 1 \times 2^{2-3} + 0 \times 2^{1-3} + 1 \times 2^{0-3}$$
$$= 1/2_{10} + 0_{10} + 1/8_{10}$$
$$= 0.625_{10}$$

Fixed Point Format

Number of bits: n = 3

Exponent: E = -3



From Fixed- to Floating-Point Numbers

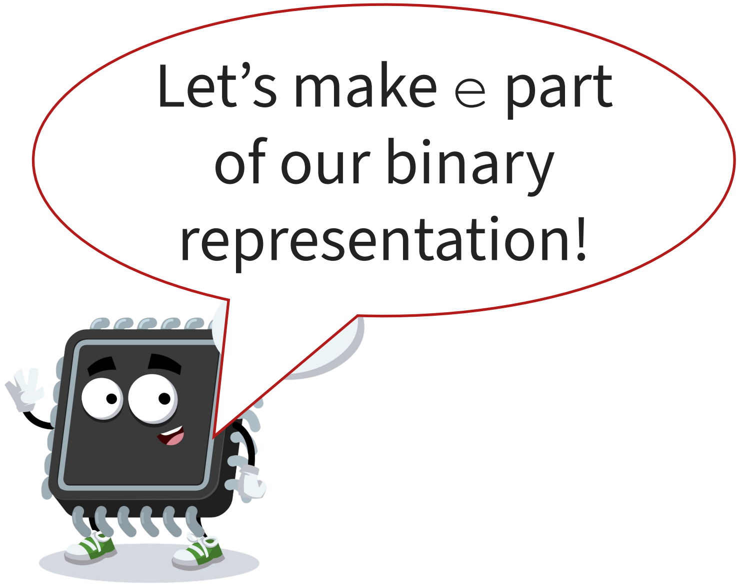
$0.101_2 = 1.01_2 \times 2^{-1}$

Fixed Point Format

Number of bits: $n = 3$

Exponent: $E = -1$

- Normalize: leading 1 in front of decimal point



From Fixed- to Floating-Point Numbers

$0.101_2 = 1.01_2 \times 2^{-1}$

101000

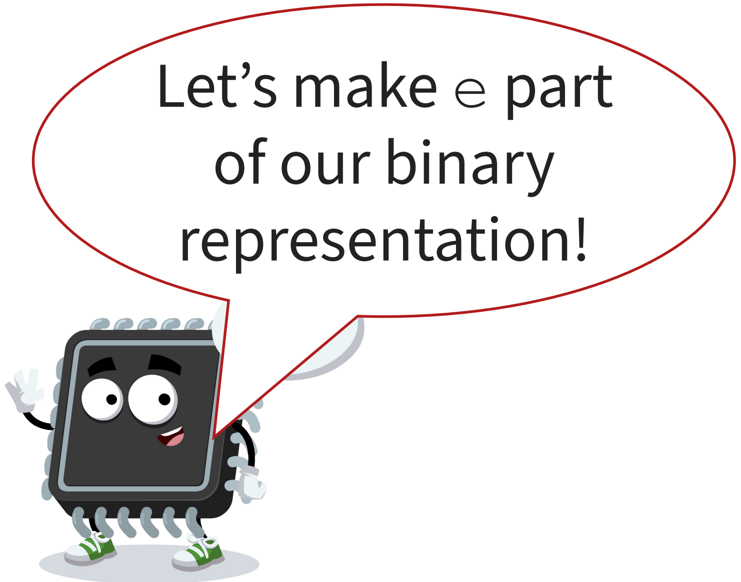
Significand

Fixed Point Format

Number of bits: n = 3

Exponent: E = -1

- Normalize: leading 1 in front of decimal point



From Fixed- to Floating-Point Numbers

$0.101_2 = 1.01_2 \times 2^{-1}$

Fixed Point Format

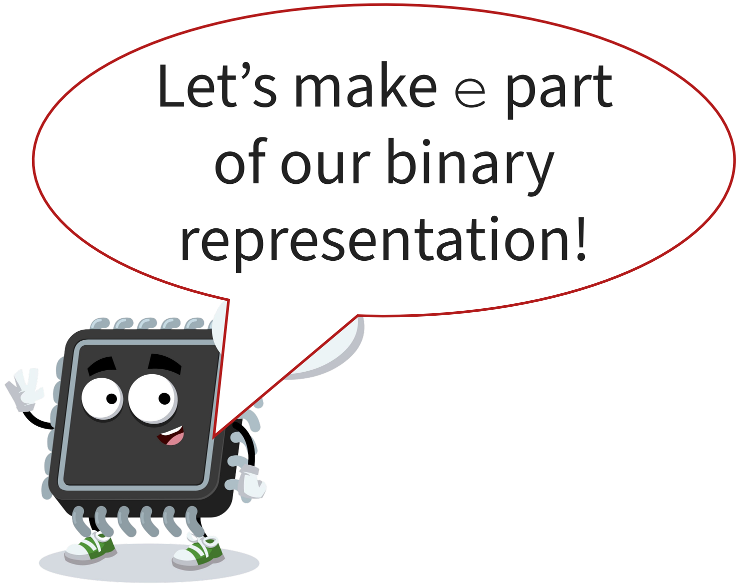
Number of bits: n = 3

Exponent: E = -1

<u>101000</u>

Significand (6-bit)

- Normalize: leading 1 in front of decimal point



From Fixed- to Floating-Point Numbers

$0.101_2 = 1.01_2 \times 2^{-1}$

101000

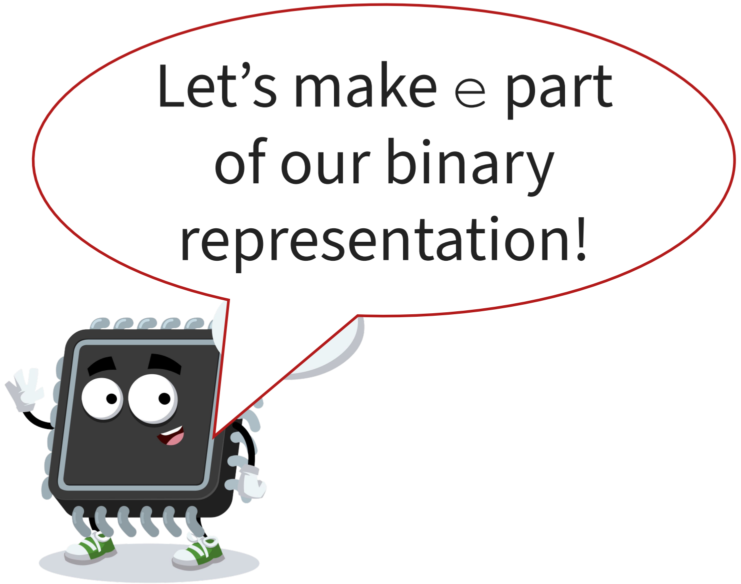
Significand (6-bit)

Fixed Point Format

Number of bits: n = 3

Exponent: E = -1

- Normalize: leading 1 in front of decimal point
- Encode exponent as biased number:
  e - 7 = -1 => e = 6



From Fixed- to Floating-Point Numbers

$0.101_2 = 1.01_2 \times 2^{-1}$

Fixed Point Format

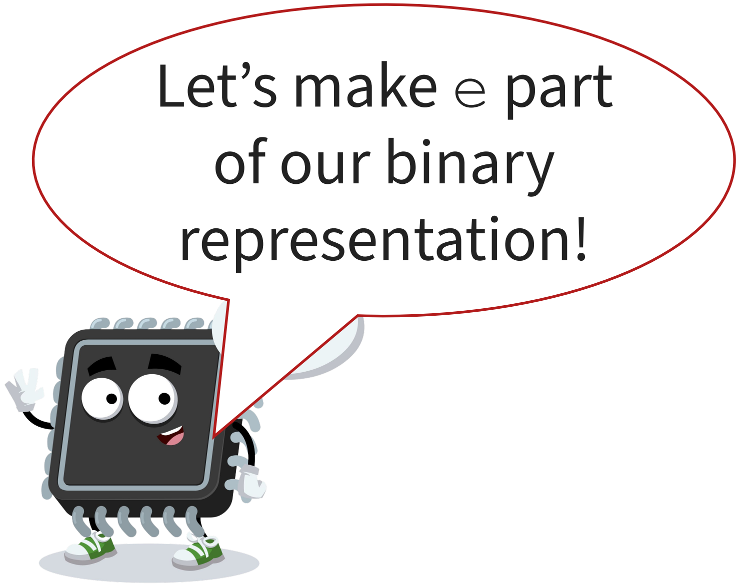
Number of bits: n = 3

Exponent: E = -1

101000

Significand (6-bit)

- Normalize: leading 1 in front of decimal point
- Encode exponent as biased number:
  $e - 7 = -1 \Rightarrow e = 6 = 0110_2$



From Fixed- to Floating-Point Numbers

$0.101_2 = 1.01_2 \times 2^{-1}$

`0110 101000`

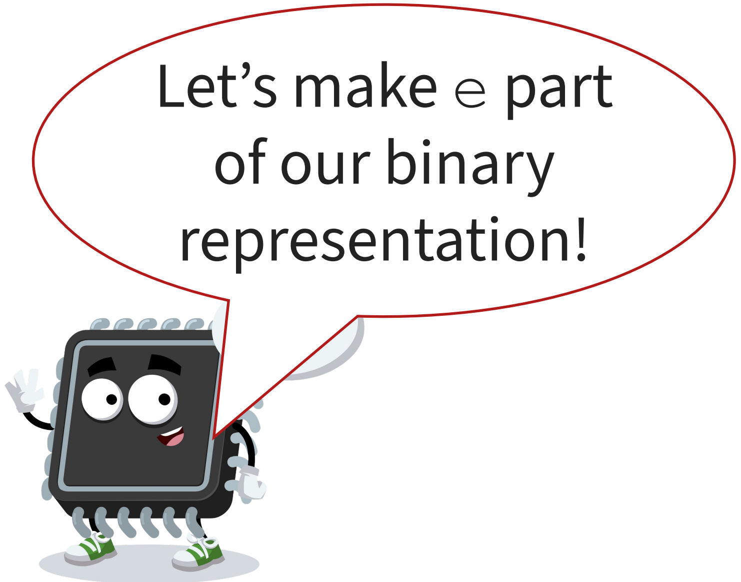
Exponent (4-bit) Significand (6-bit)

Fixed Point Format

Number of bits: n = 3

Exponent: E = -1

- Normalize: leading 1 in front of decimal point
- Encode exponent as biased number:
  $e - 7 = -1 \Rightarrow e = 6 = 0110_2$



From Fixed- to Floating-Point Numbers

$0.101_2 = 1.01_2 \times 2^{-1}$

Fixed Point Format

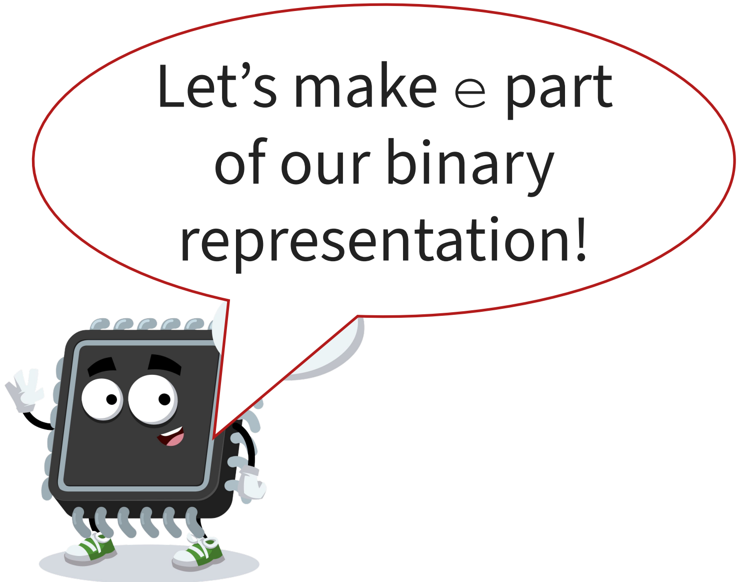
Number of bits: n = 3

Exponent: E = -1

0110 **101000**

Exponent (4-bit) Significand (6-bit)

- Normalize: leading 1 in front of decimal point
- Encode exponent as biased number:
  $e - 7 = -1 \Rightarrow e = 6 = 0110_2$
- Bias is chosen based on bits for the exponent.
  Normally $B = 2^{\#\text{ of exponent bits} - 1} - 1$



From Fixed- to Floating-Point Numbers

$0.101_2 = 1.01_2 \times 2^{-1}$

0110 101000

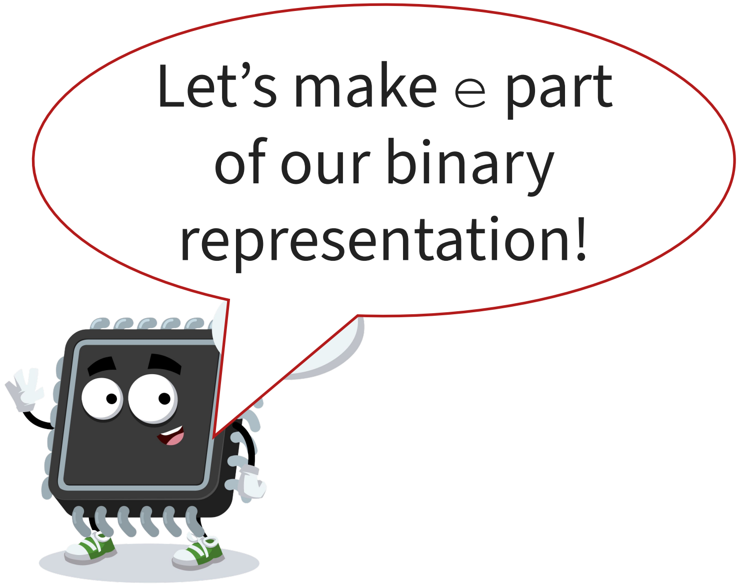
Exponent (4-bit) Significand (6-bit)

Fixed Point Format

Number of bits: n = 3

Exponent: E = -1

- Normalize: leading 1 in front of decimal point
- Encode exponent as biased number: $e - 7 = -1 \Rightarrow e = 6 = 0110_2$
- Sign is encoded as a single bit.



From Fixed- to Floating-Point Numbers

$0.101_2 = 1.01_2 \times 2^{-1}$

Sign (1-bit)

`0 0110 101000`

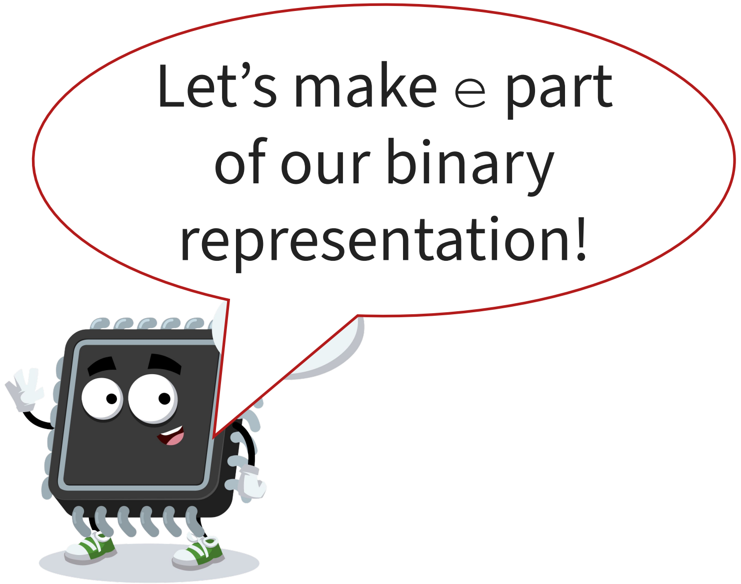
Exponent (4-bit) Significand (6-bit)

Fixed Point Format

Number of bits: n = 3

Exponent: E = -1

- Normalize: leading 1 in front of decimal point
- Encode exponent as biased number:
  $e - 7 = -1 \Rightarrow e = 6 = 0110_2$
- Sign is encoded as a single bit.



Floating-Point Numbers

$0.101_2 = 1.01_2 \times 2^{-1}$

Sign (1-bit)

`0 0110 101000`

Exponent (4-bit) Significand (6-bit)

- Normalize: leading 1 in front of decimal point
- Encode exponent as biased number:
  $e - 7 = -1 \Rightarrow e = 6 = 0110_2$
- Sign is encoded as a single bit.

Floating-Point Numbers

$$0.101_2 = 1.01_2 \text{ x } 2^{-1} = (-1)^{\mathbf{s}} \text{ x } \mathbf{g_5.g_4g_3g_2g_1g_0} \text{ x } 2^{\mathbf{e} - 7}$$

Sign **s**

```
0 0110 101000
```

Exponent **e**

Significand **g**

- Normalize: leading 1 in front of decimal point
- Encode exponent as biased number:
  $e - 7 = -1 \Rightarrow e = 6 = 0110_2$
- Sign is encoded as a single bit.

Floating-Point Numbers

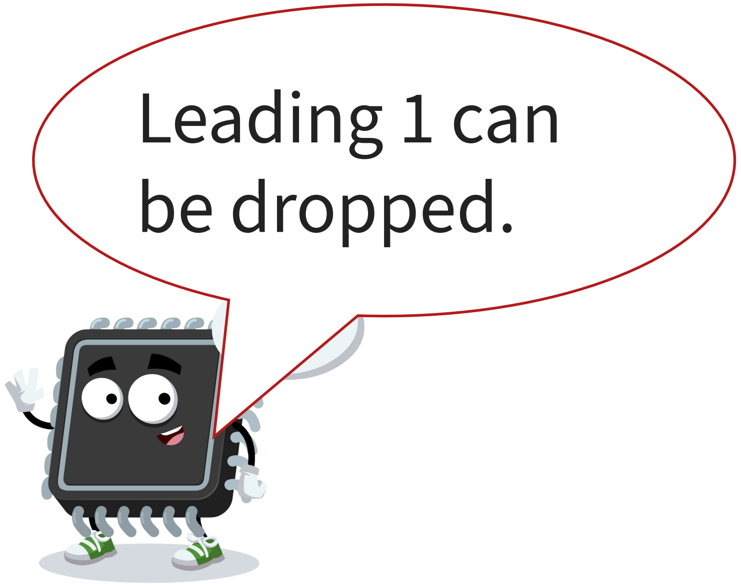
$$0.101_2 = \boxed{1}.01_2 \times 2^{-1} = (-1)^{\mathbf{s}} \times \mathbf{g_5.g_4g_3g_2g_1g_0} \times 2^{\mathbf{e}-7}$$

Sign **s**

```
0 0110 101000
```

Exponent **e**   Significand **g**

- Normalize: leading 1 in front of decimal point
- Encode exponent as biased number:
  $e - 7 = -1 \Rightarrow e = 6 = 0110_2$
- Sign is encoded as a single bit.



Floating-Point Numbers

$$0.101_2 = \boxed{1}.01_2 \times 2^{-1} = (-1)^{\mathbf{s}} \times 1.\mathbf{g_5 g_4 g_3 g_2 g_1 g_0} \times 2^{\mathbf{e}-7}$$

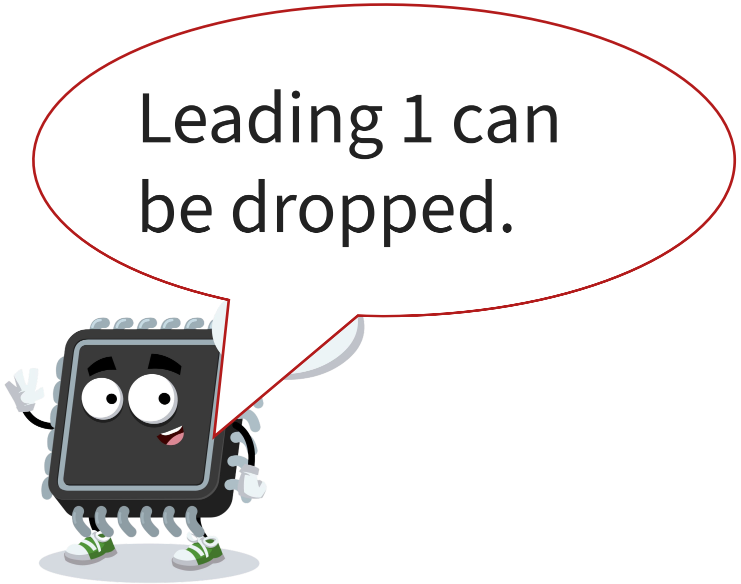
Sign **s**

```
0 0110 010000
```

Exponent **e**

Significand **g**

- Normalize: leading 1 in front of decimal point
- Encode exponent as biased number: $e - 7 = -1 \Rightarrow e = 6 = 0110_2$
- Sign is encoded as a single bit.



Converting to Floating Point

$$8.25_{10} = 1000.01_2$$
$$= (-1)^0 \text{ x } 1000.01_2$$
$$= (-1)^0 \text{ x } 1.00001_2 \text{ x } 2^3$$
$$= (-1)^0 \text{ x } 1.00001_2 \text{ x } 2^{10-7}$$

Sign **s**

? ? ? ? ? ? ? ? ? ? ?

Exponent **e**

Significand **g**

Converting to Floating Point

$$8.25_{10} = 1000.01_2$$
$$= (-1)^0 \times 1000.01_2$$
$$= (-1)^0 \times 1.00001_2 \times 2^3$$
$$= (-1)^{\boxed{0}} \times 1.00001_2 \times 2^{10-7}$$

Sign **s**

`0 ???? ??????`

Exponent **e**

Significand **g**

Converting to Floating Point

$$8.25_{10} = 1000.01_2$$
$$= (-1)^0 \times 1000.01_2$$
$$= (-1)^0 \times 1.00001_2 \times 2^3$$
$$= (-1)^{\boxed{0}} \times 1.00001_2 \times 2^{\boxed{10}-7}$$

Sign **s**

`0 1010 ??????`

Exponent **e**

Significand **g**

Converting to Floating Point

$$8.25_{10} = 1000.01_2$$
$$= (-1)^0 \times 1000.01_2$$
$$= (-1)^0 \times 1.00001_2 \times 2^3$$
$$= (-1)^0 \times 1.00001_2 \times 2^{10-7}$$

exponent bias B

Sign **s**

`0 1010 ??????`

Exponent **e**

Significand **g**

Converting to Floating Point

$$8.25_{10} = 1000.01_2$$
$$= (-1)^0 \times 1000.01_2$$
$$= (-1)^0 \times 1.00001_2 \times 2^3$$
$$= (-1)^{\boxed{0}} \times \boxed{1.00001}_2 \times 2^{\boxed{10}-7}$$

exponent bias B

Sign **s**

`0 1010 ??????`

Exponent **e**

Significand **g**

Converting to Floating Point

$$8.25_{10} = 1000.01_2$$
$$= (-1)^0 \times 1000.01_2$$
$$= (-1)^0 \times 1.00001_2 \times 2^3$$
$$= (-1)^{\boxed{0}} \times 1.\boxed{00001}_2 \times 2^{\boxed{10}-7}$$

exponent bias B

Sign **s**

`0 1010 00001?`

Exponent **e**

Significand **g**

Converting to Floating Point

$$8.25_{10} = 1000.01_2$$
$$= (-1)^0 \times 1000.01_2$$
$$= (-1)^0 \times 1.00001_2 \times 2^3$$
$$= (-1)^{\boxed{0}} \times 1.\boxed{00001}_2 \times 2^{\boxed{10}-7}$$

exponent bias B

Sign **s**

`0 1010 000010`

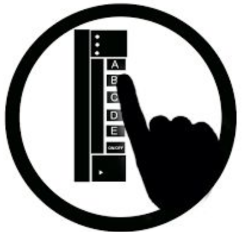
Exponent **e**

Significand **g**

Floating Point Poll

- Encode -5.125 in our floating point format (4-bit exponent, 6-bit significant).

PollEv.com/cs3410

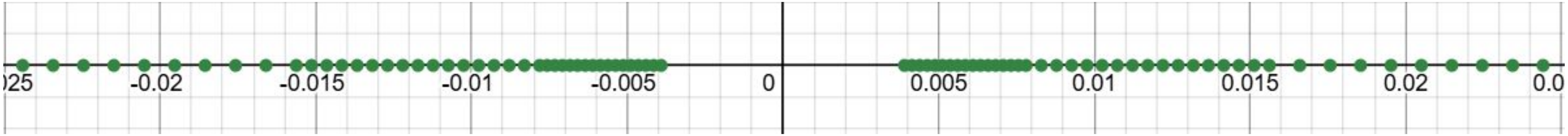


Standard floating point formats

- float: 32-bit, “single precision”
  - 1-bit sign, 8-bit exponent, 23-bit significand
- double: 64-bit, “double precision”
  - 1-bit sign
  - 11-bit exponent
  - 54-bit significand
- Half-precision: 16-bit, “half precision"
  - 1-bit sign
  - 5-bit exponent
  - 10-bit significand
- bfloat, 16-bit, “brain floating point”
  - Invented for machine learning (ML): Deep learning needs more range, but less precision ok
  - 1-bit sign
  - 8-bit exponent
  - 7-bit significand

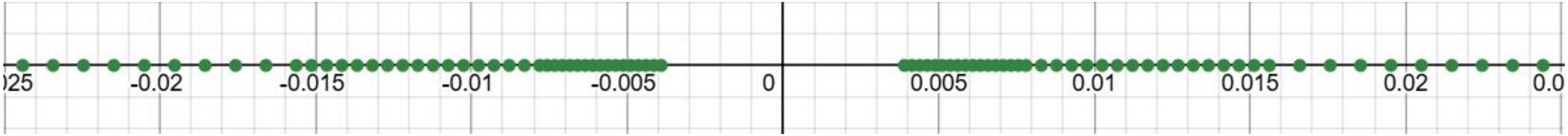
Subnormals

Representable numbers around 0 if we require a leading 1 in front of the decimal point. (with the 32-bit float format)



Subnormals

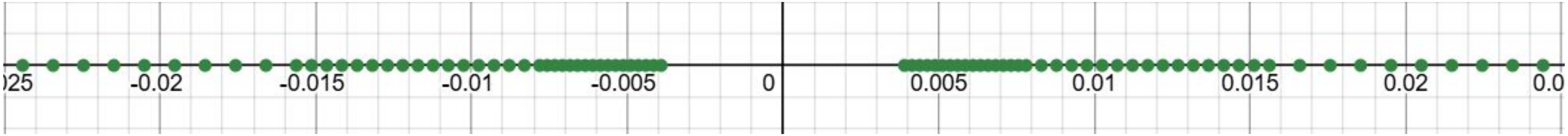
Representable numbers around 0 if we require a leading 1 in front of the decimal point. (with the 32-bit float format)



The smallest non-negative number is 2^{-127}. Cannot represent 0.

Subnormals

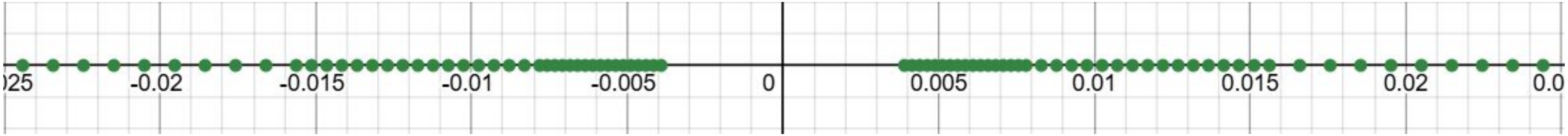
Representable numbers around 0 if we require a leading 1 in front of the decimal point. (with the 32-bit float format)



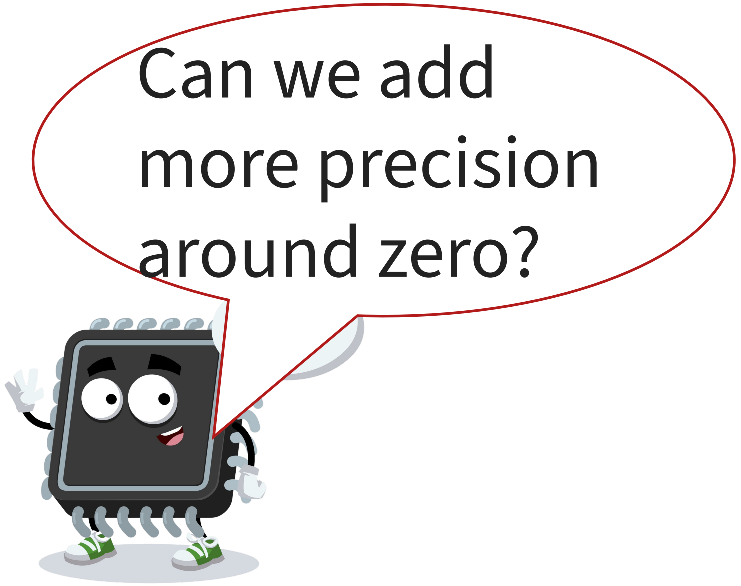
The smallest non-negative number is 2^{-127}. Cannot represent 0. **Underflow** results in a very incorrect result.

Subnormals

Representable numbers around 0 if we require a leading 1 in front of the decimal point. (with the 32-bit float format)

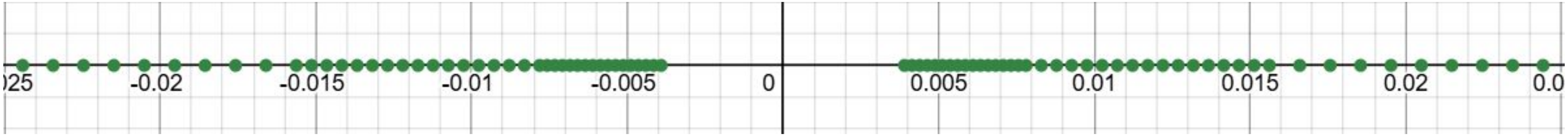


The smallest non-negative number is 2^{-127}.
Cannot represent 0.
Underflow results in a very incorrect result.

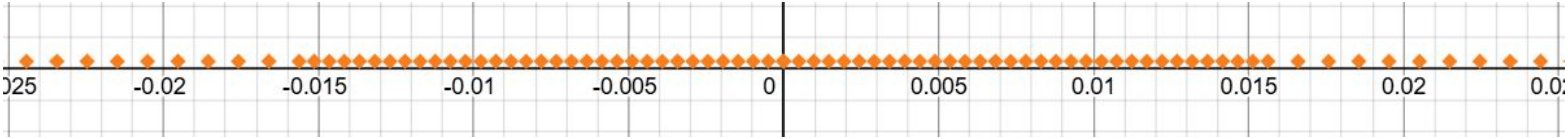


Subnormals

Representable numbers around 0 if we require a leading 1 in front of the decimal point. (with the 32-bit float format)

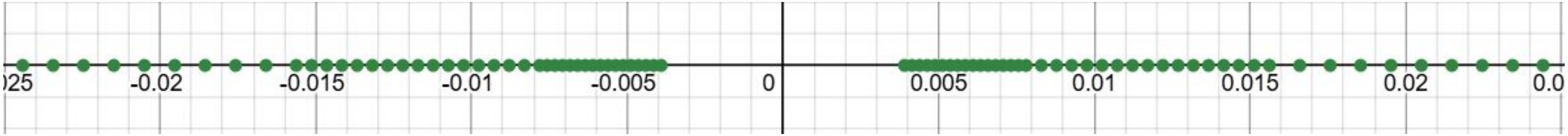


When e = 0, we use a leading 0 instead of a leading 1 → loses precision more gradually.

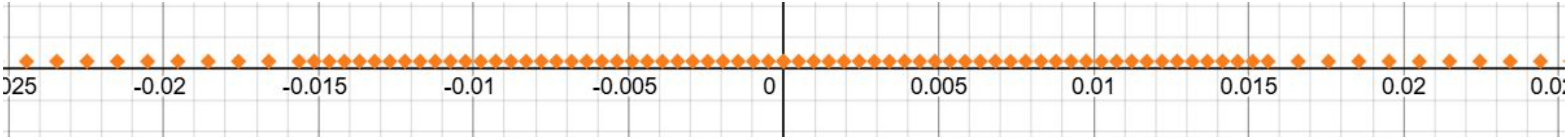


Subnormals

Representable numbers around 0 if we require a leading 1 in front of the decimal point. (with the 32-bit float format)

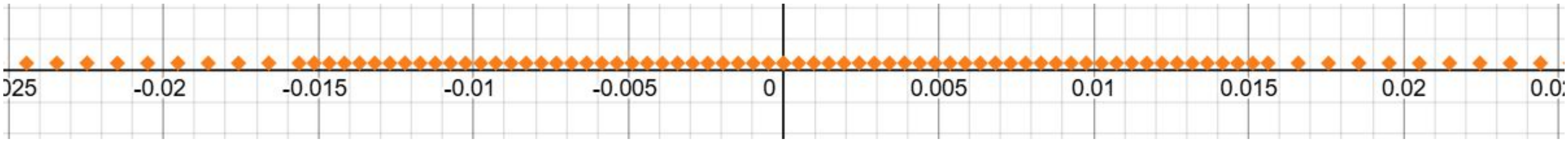


When e = 0, we use a leading 0 instead of a leading 1 → loses precision more gradually.



Subnormals

- e = 0: $(-1)^{\mathbf{s}} \times 0.\mathbf{g} \times 2^{-6}$
- e > 0: $(-1)^{\mathbf{s}} \times 1.\mathbf{g} \times 2^{\mathbf{e}-7}$
- Thus, to represent 0, we set e = 0 and g = 0



Infinity and NaN

- e = all ones

Infinity and NaN

- e = all ones
- for our representation with 4-bit exponent:
  e = 0b1111

Infinity and NaN

- e = all ones
- for our representation with 4-bit exponent: e = 0b1111
- +/- infinity: e = 0b1111 and g = 0

Infinity and NaN

- e = all ones
- for our representation with 4-bit exponent: e = 0b1111
- +/- infinity: e = 0b1111 and g = 0
- Not a Number (NaN): e = 0b1111 and g =/= 0

Infinity and NaN

- e = all ones
- for our representation with 4-bit exponent:
  e = 0b1111
- +/- infinity: e = 0b1111 and g = 0
- Not a Number (NaN): e = 0b1111 and g =/= 0
- Dividing zero by zero is NaN, but dividing other numbers by zero is infinity!

Guidelines

- Floating-point numbers are ***not*** real numbers
  - Expect to accumulate some error when using floats
- Never use floating-point numbers to represent currency
  - When people say $123.45, they want that exact number of cents, not $123.40000152.
  - Use an integer number of cents: i.e., a fixed-point representation with a fixed decimal point
- Be suspicious of equality, f1 == f2
  - E.g. try (0.1 + 0.2) == 0.3 ?
  - Consider using an "error tolerance" in comparisons, like abs(f1 - f2) < epsilon.
- Floating-point arithmetic is not free
  - It is slower and more energy than integer or fixed-point arithmetic
  - The flexibility is expensive since the complexity requires more complex for the hardware
  - As a result, a lot of applications such as ML convert (quantize) models to a fixed-point representation so they can run efficientl.

Floating Point Error Analysis

Take CS4210 or CS4220