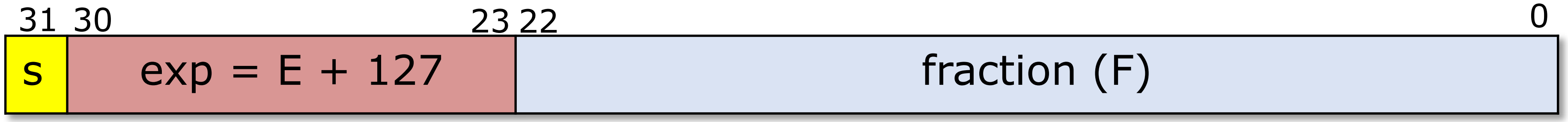
Floats (continued)
Intro to C programming

Lesson plan

- Rounding
- FP operations and caveats
- C programming: overview
- C programming: bitwise operators

IEEE Floating Point

Normalized Encoding:



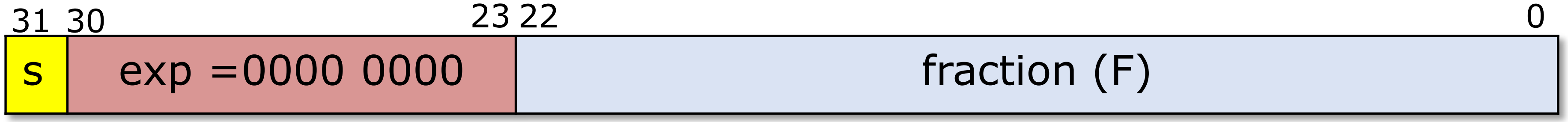
31 30 23 22 0

s | exp = E + 127 | fraction (F)

1 <= M < 2, M = (1.F $)_2$

$\underline{+}M * 2^E$

Denormalized Encoding:

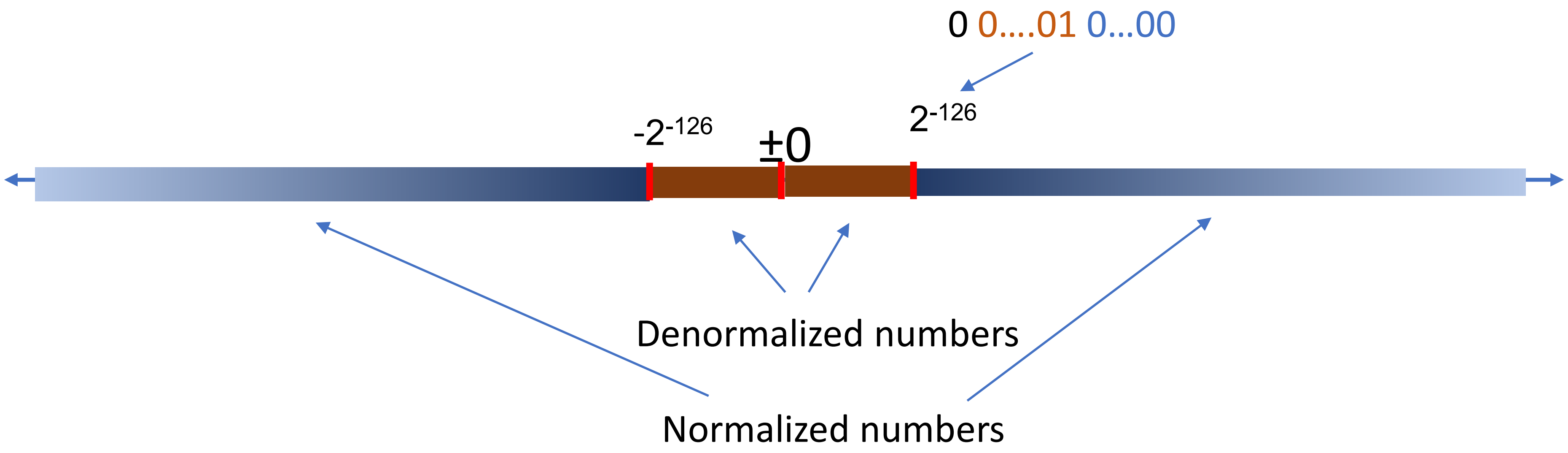


31 30 23 22 0

s | exp =0000 0000 | fraction (F)

E = 1 – bias = -126

0 <= M < 1, M = (0.F $)_2$



0 0....01 0...00

-2^{-126}

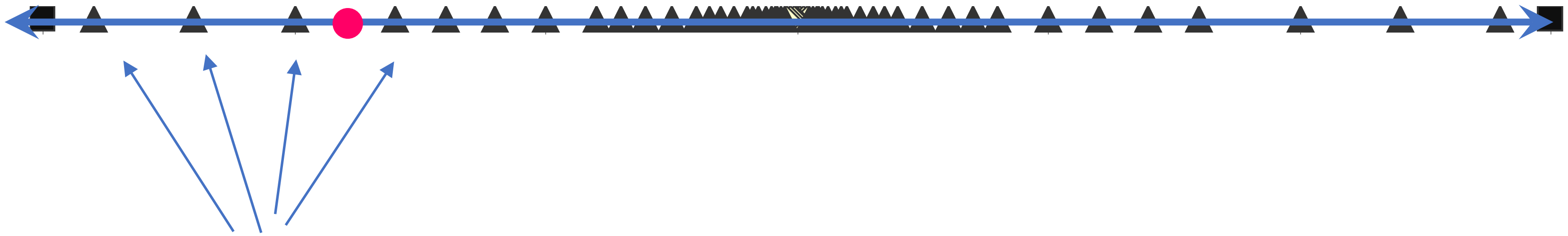
±0

2^{-126}

Denormalized numbers

Normalized numbers

FP: Rounding



Values that are represented precisely

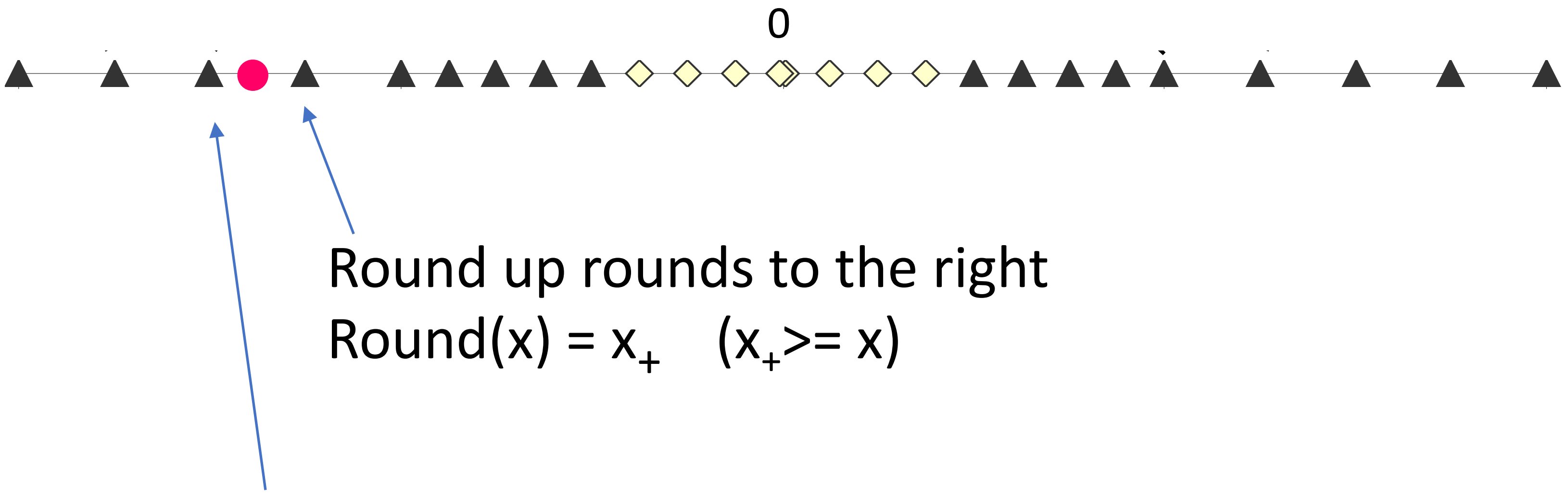
What if the result of computation is at ● ?

Rounding: Use the “closest” representable value *x’* for x.

4 modes:

- Round-down
- Round-up
- Round-toward-zero
- Round-to-nearest (Round-to-even in text book)

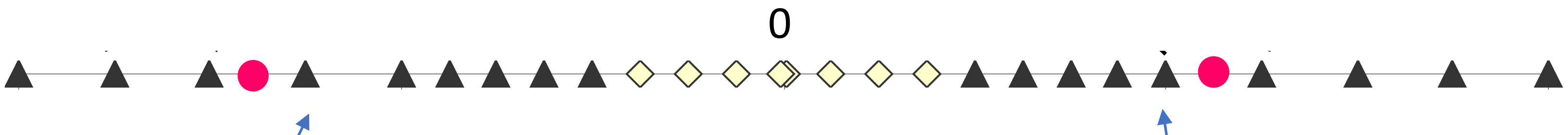
Round down vs. round up



Round up rounds to the right
Round(x) = x_+ (x_+>= x)

Round down rounds to the left
Round(x) = x_- (x_-<= x)

Round towards zero



Rounds to the right if x < 0

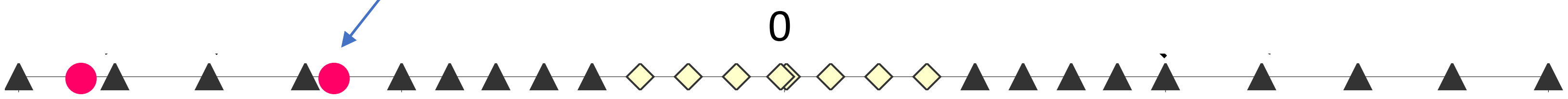
Round(x) = x_+ if $x < 0$

Rounds to the left if x >0

Round(x) = x_- if $x < 0$

Round to nearest; ties to even

Round to the left if x_- is nearer to x than x_+

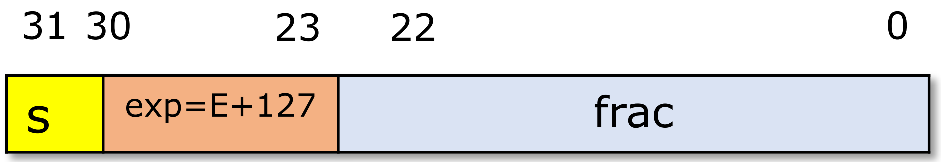


Round to the right if x_+ is nearer to x than x_-

In case of a tie, the one with its least significant bit equal to zero is chosen.

IEEE FP: single vs. double precision

single precision
(32 bits)



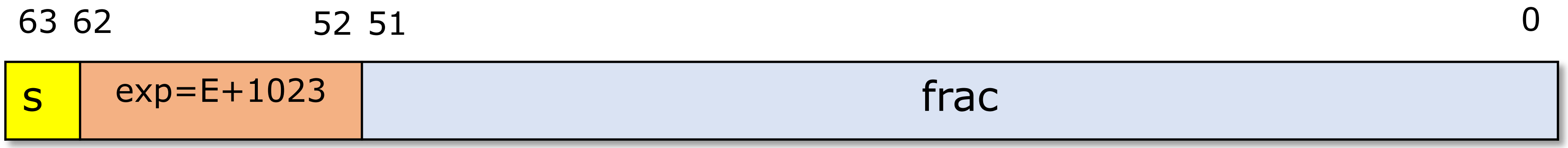
31 30 23 22 0

s | exp=E+127 | frac

C program:

```
float f = 0.1;
double d = 0.1;
```

double precision
(64 bits)



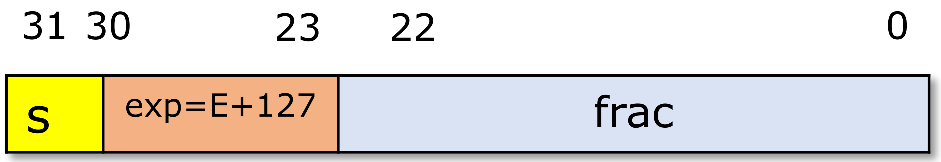
63 62 52 51 0

s | exp=E+1023 | frac

- What’s the highest precision? (aka intervals between two denormalized numbers?)
- What’s the largest positive FP?

IEEE FP: single vs. double precision

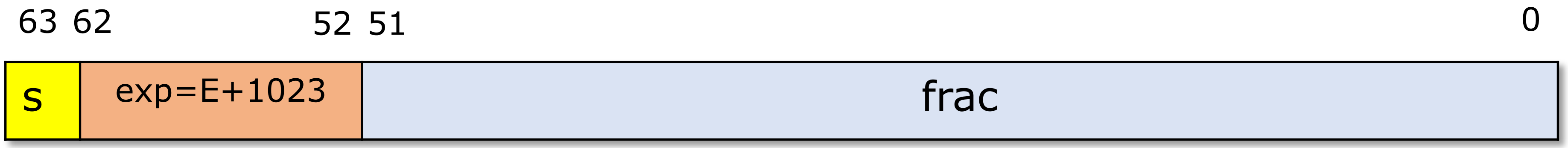
single precision
(32 bits)



C program:

```
float f = 0.1;
double d = 0.1;
```

double precision
(64 bits)



- What’s the highest precision? (aka intervals between two denormalized numbers?)
  - (single) 2^{-149} (double) 2^{-1045}
- What’s the largest positive FP?
  - (single) $\approx 2^{128}$ (double) $\approx 2^{1024}$

How does CPU know if data is FP or integer ?

4-byte data: 0x80000001

Interpret as signed int:

Interpret as IEEE single-precision FP:

- CPU has separate registers for FPs and integers.
- CPU uses different instructions for FPs and integer operations.

Floating point operations

- Addition, subtraction, multiplication, division etc.
- Invalid operations (resulting in NaN):
  - 0/0
  - sqrt(-1)
  - $\infty + \infty$
- Divide by zero: $x/0 \rightarrow \infty$
- Caveats:
  - Overflow: Outside the range
  - Underflow: 0 < result < smallest denormalized value
  - Inexact: due to rounding

Floating point addition

- Commutative? x+y == y+x?
- Associative? (x+y)+z = x + (y+z)?
  - Rounding:

```
(3.14+1e10)-1e10 = 0
 3.14+(1e10-1e10) = 3.14
```

  - Overflow
- Every number has an additive inverse?
  - Yes, by flipping the sign.

Floating point multiplication

- Commutative? x* y == y*x?
- Associative? (x*y)*z = x * (y*z)?
  - Overflow:

  `(1e20*1e20)*1e-20= inf, 1e20*(1e20*1e-20)= 1e20`
  - Rounding
- Distributive? (x+y)*z = x*z + y*z?
  - `1e20*(1e20-1e20)= 0.0, 1e20*1e20 – 1e20*1e20 = NaN`

FP precision decreases as value gets larger

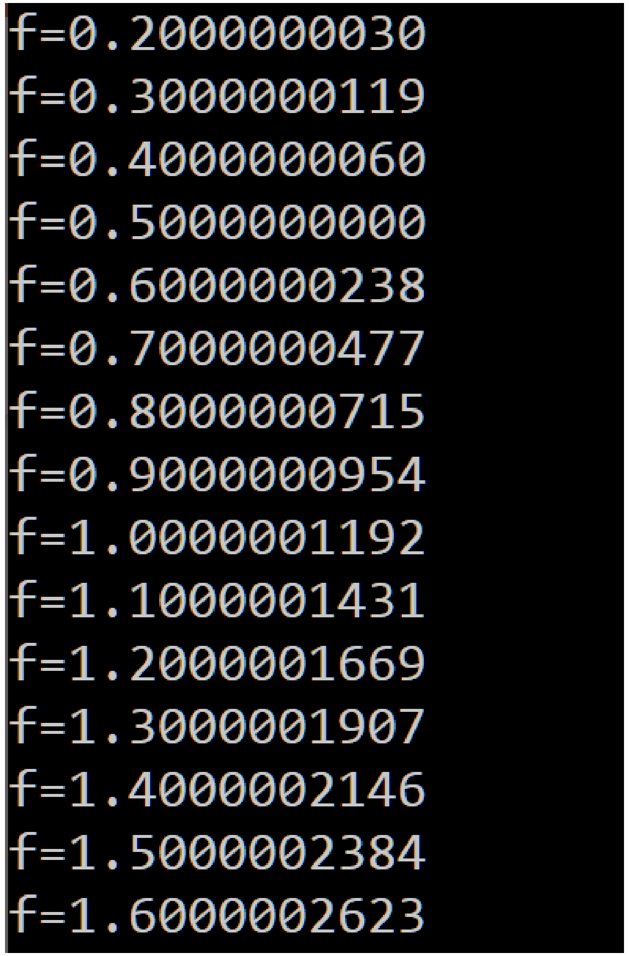
- Storing time in computer games as a FP?
- Precision diminishes as time gets bigger

FP value (decimal)	Time value	FP precision	Time precision
1	1 sec	1.19E-07	119 nanoseconds
100	~1.5 min	7.63E-06	7.63 microseconds
10 000	~3 hours	0.000977	.976 milliseconds
1000 000	~11 days	0.0625	62.5 milliseconds

Floating point trouble

- Comparing floats for equality is a bad idea!

```
float f = 0.1;
while (f != 1.0) {
  f += 0.1;
}
```



```
f=0.2000000030
f=0.3000000119
f=0.4000000060
f=0.5000000000
f=0.6000000238
f=0.7000000477
f=0.8000000715
f=0.9000000954
f=1.0000001192
f=1.1000001431
f=1.2000001669
f=1.3000001907
f=1.4000002146
f=1.5000002384
f=1.6000002623
```

Breakout time!

Breakout exercises

- In a shooter game, the accuracy of shooting another player 1200m away is:

$1200 = 2^{10}*(1.17)_{10}$ Precision: $2^{10}*2^{-23}=2^{-13}$

- Result of count?

```
int count = 0;
for (int f = 0; f <= 10; f += 1) {
  count++;
}
```

count=11

```
int count = 0;
for (float f = 0.0; f <= 1.0; f += 0.1) {
  count++;
}
```

count=10

Floating point summary

- FP format is based on normalized exponential notation
- Floating points are tricky
  - Precision diminishes as magnitude grows
  - overflow, rounding error
- Many real world disasters due to FP trickiness
  - Patriot Missile failed to intercept due to rounding error (1991)
  - Ariane 5 explosion due to overflow in converting from double to int (1996)