# Floating Point Numbers

Prof. Usagi

# Price of a barrel of oil on April 20, 2020

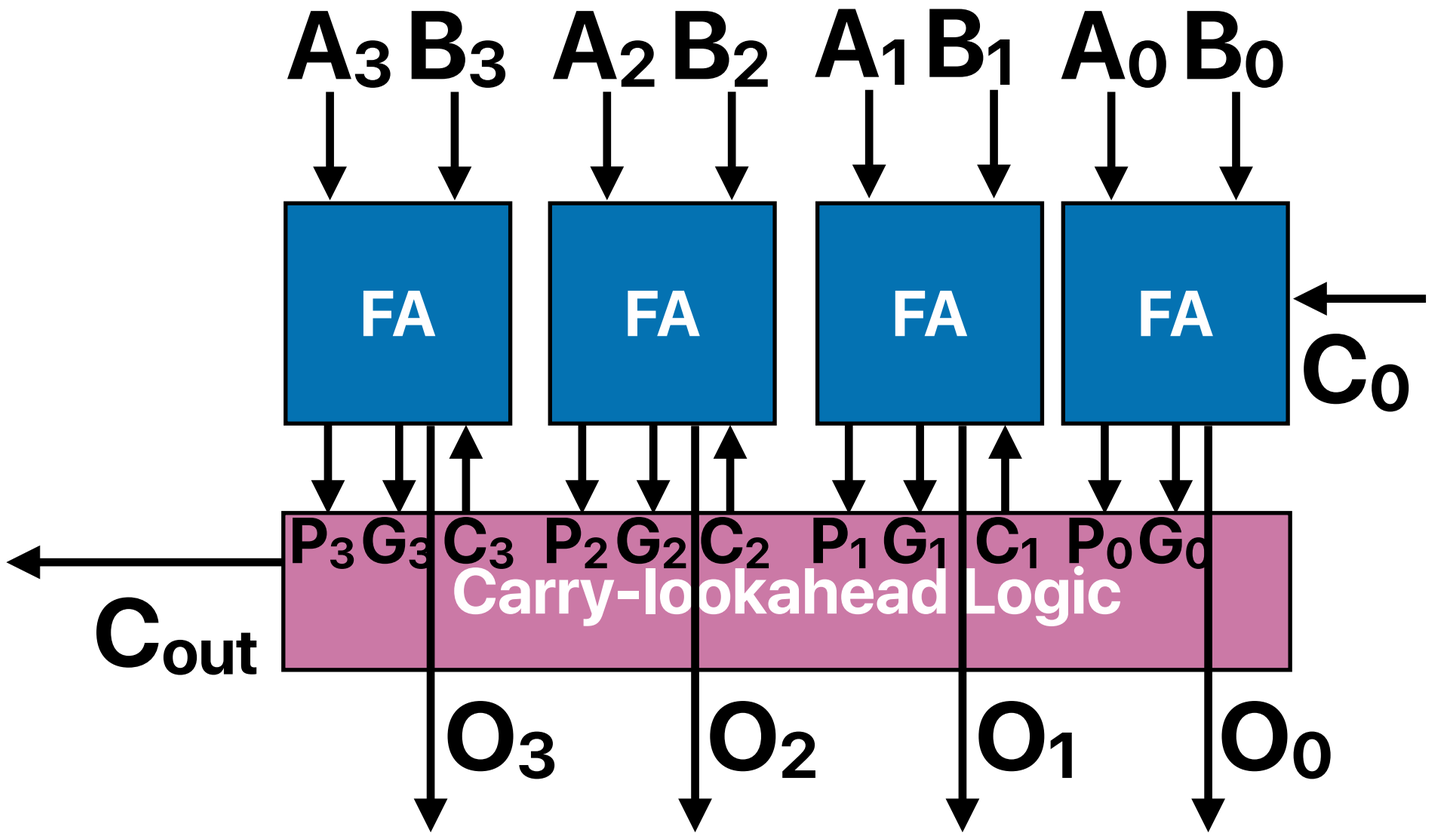
West Texas Intermediate crude futures prices for delivery in May


Source: Bloomberg

BUSINESS INSIDER

# Recap: CLA (cont.)

- All "G" and "P" are immediately available (only need to look over Ai and Bi), but "c" are not (except the c0).




$$G_i = A_i B_i$$

$$P_i = A_i \text{ XOR } B_i$$

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0)$$
$$= G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2$$
$$= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

$$C_4 = G_3 + P_3 C_3$$
$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$

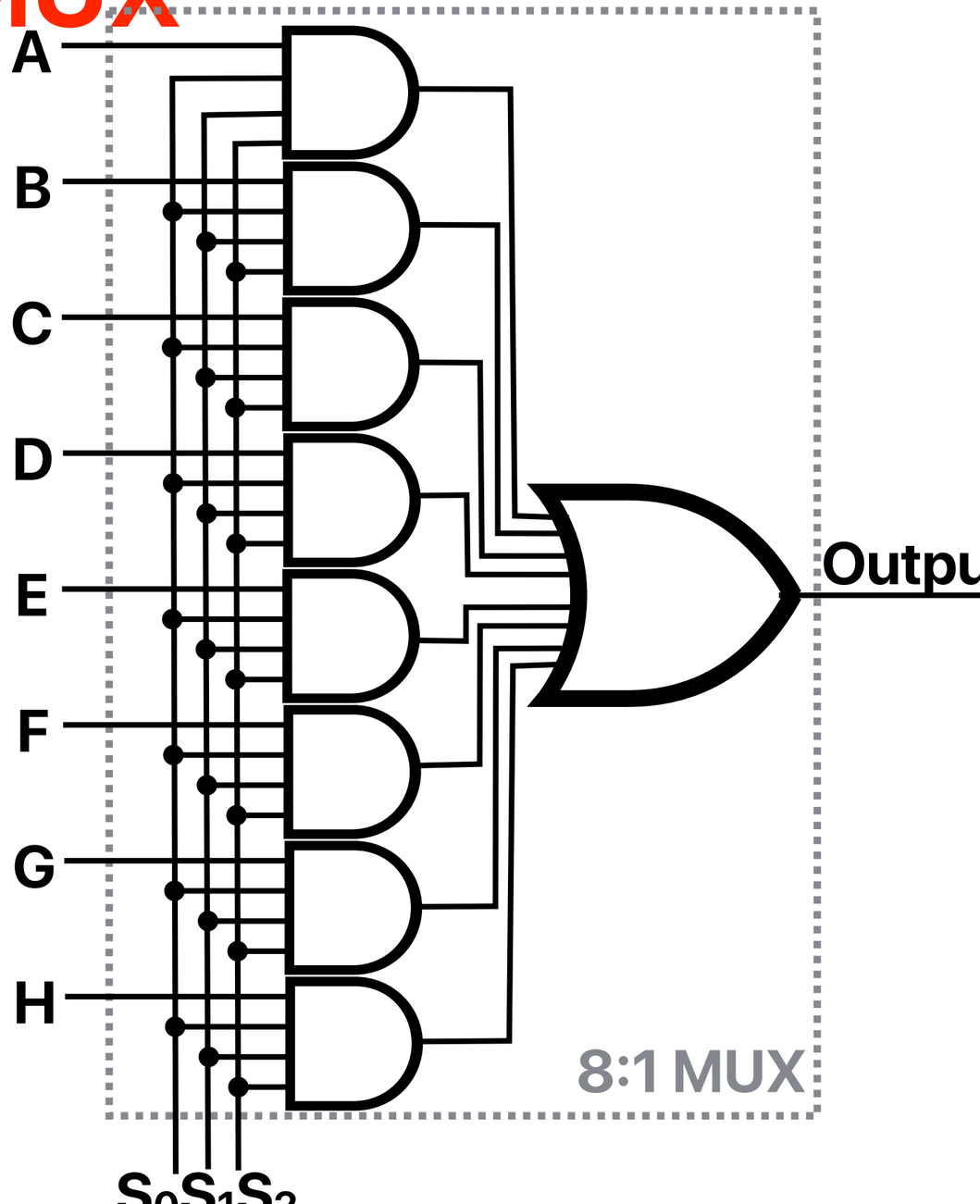
# Recap: CLA v.s. Carry-ripple

- Size:
  - 32-bit CLA with 4-bit CLAs — requires 8 of 4-bit CLA
    - Each requires 116 for the CLA 4*(4*6+8) for the A+B — 244 gates
    - 1952 transistors
  - 32-bit CRA
    - 1600 transistors **Win!**
- Delay
  - 32-bit CLA with 8 4-bit CLAs
    - 2 gates * 8 = 16 **Win!**
  - 32-bit CRA
    - 64 gates

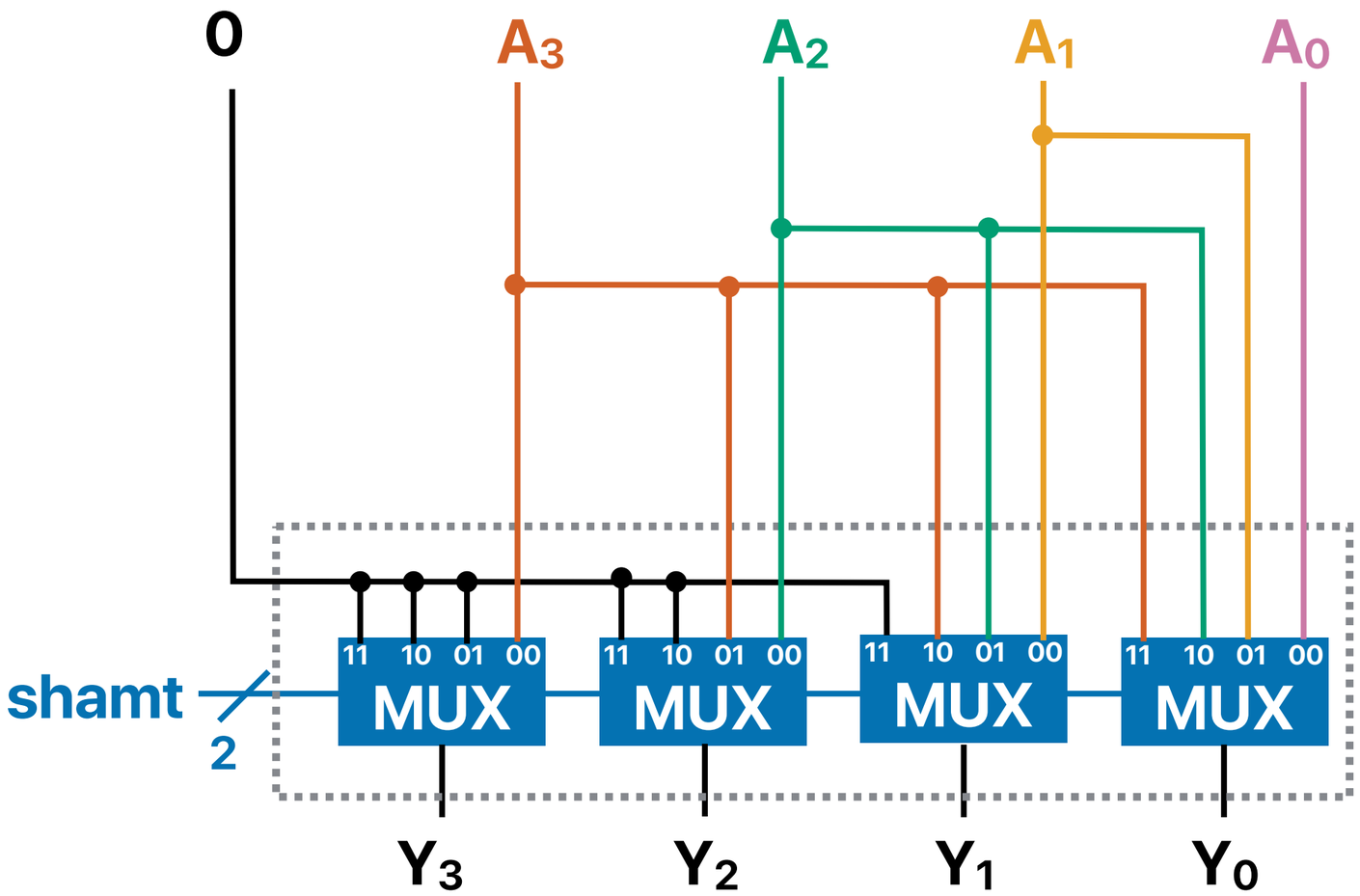
**Area-Delay Trade-off!**

# Recap: Gate delay of 8:1 MUX

- What's the estimated gate delay of an 8:1 MUX?
  - A. 1
  - B. 2
  - C. 4
  - D. 8
  - E. 16



# Recap: Shift "Right"




Example:
if S = 11
then
Y3 = 0
Y2 = 0
Y1 = 0
Y0 = A3

Example:
if S = 10
then
Y3 = 0
Y2 = 0
Y1 = A3
Y0 = A2

Example:
if S = 01
then
Y3 = 0
Y2 = A3
Y1 = A2
Y0 = A1

**The "chain" of multiplexers determines how many bits to shift**

**Based on the value of the selection input (shamt = shift amount)**

# Recap: What's after shift?

- Assume we have a data type that stores 8-bit unsigned integer (e.g., unsigned char in C). How many of the following C statements and their execution results are correct?

| | Statement | C = ? |
|---|---|---|
| I | `c = 3; c = c >> 2;` | **0** |
| II | `c = 255; c = c << 2;` | 252 |
| III | `c = 256; c = c >> 2;` | **0** |
| IV | `c = 128; c = c << 1;` | **0** |

A. 0

B. 1

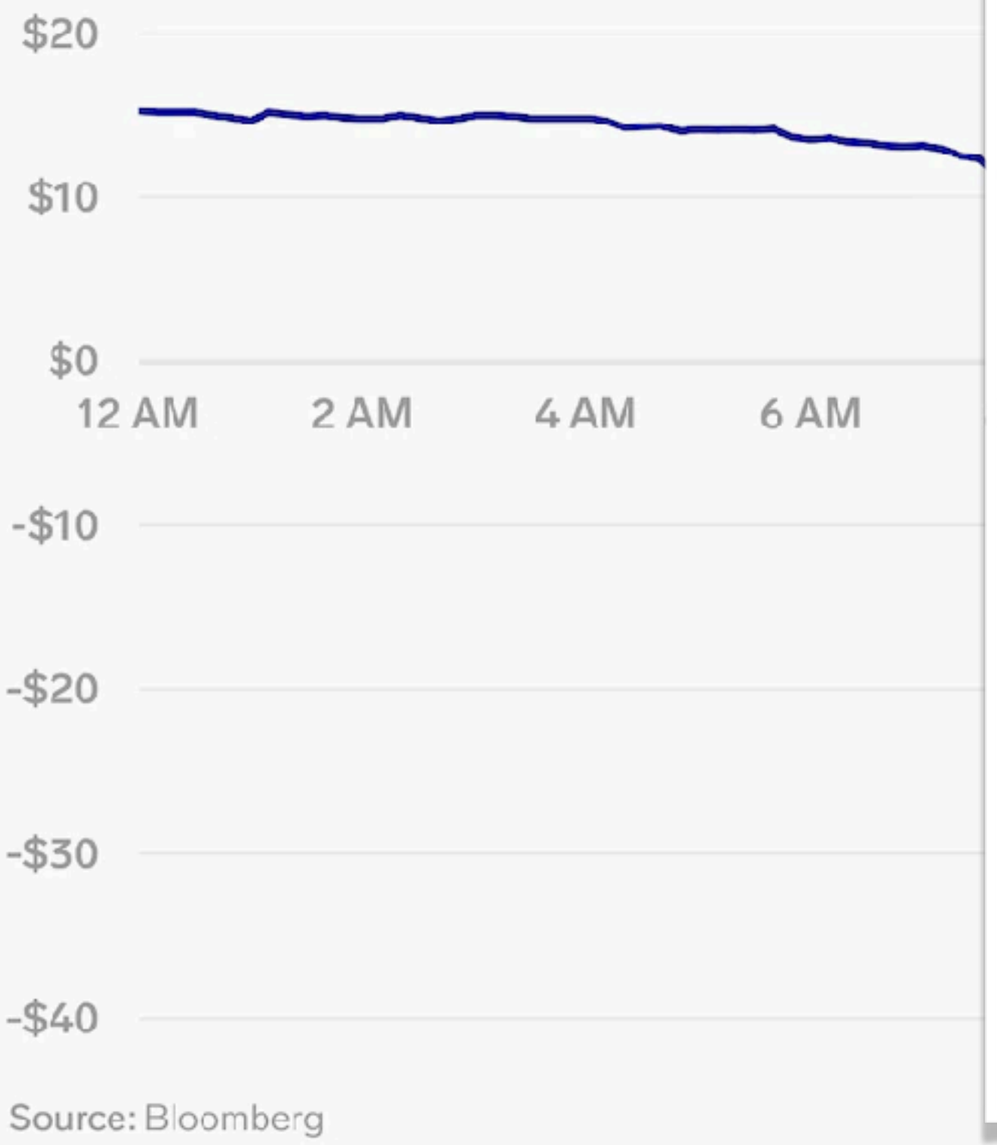
C. 2

D. 3

E. 4

# Price of a barrel of oil

West Texas Intermediate crude futures prices




Source: Bloomberg

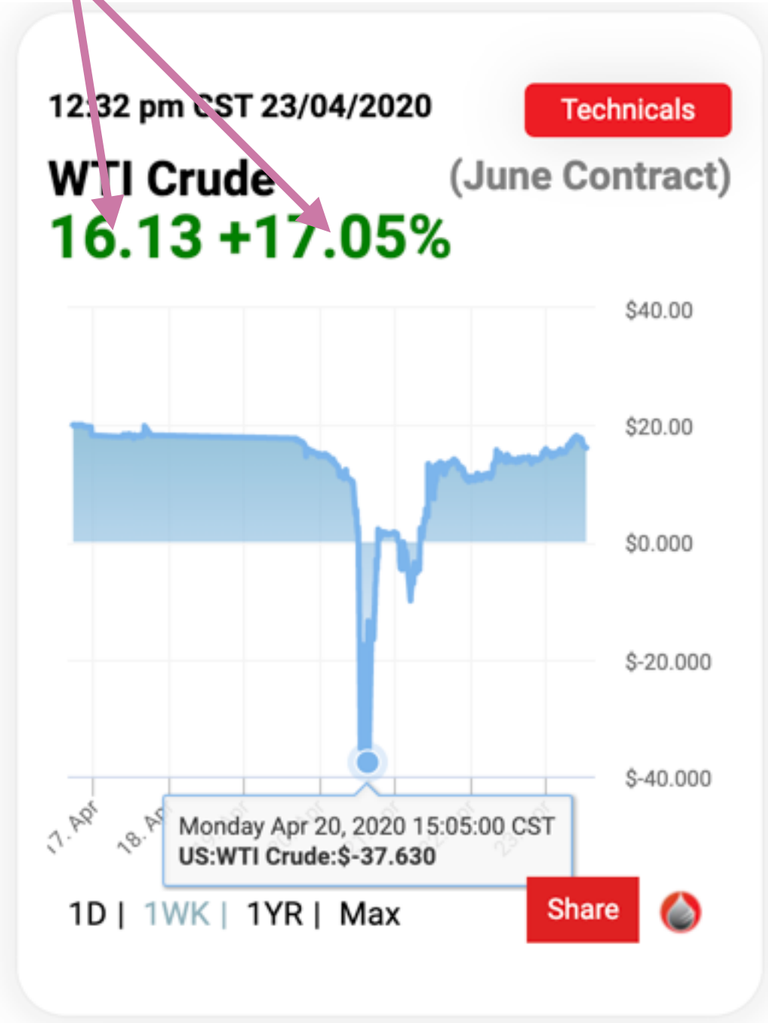
FILE PHOTO: U.S. President Donald Trump applauds Continental Resources CEO Harold Hamm during a tax reform event with workers at the Andeavor Refinery in Mandan, North Dakota, U.S., September 6, 2017. REUTERS/Jonathan Ernst

Continental's executive chairman Harold Hamm sent a letter dated Tuesday, April 21, to the U.S. Commodity Futures Trading Commission asking the regulator to probe whether "potential market manipulation, failed systems or computer programming failures" was behind Monday's price crash, which took U.S. oil futures into negative territory for the first time.

https://www.reuters.com/article/us-global-oil-cftc-hamm/oil-exec-and-trump-ally-hamm-seeks-us-probe-of-oil-price-crash-idUSKCN2242UO

# Outline

- Representing a number with a **decimal point**
- Floating point numbers
- Floating point hardware



Poll close in 1:30

# Will the loop end?

- Consider the following two C programs.

**X**

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    int i=0;
    while(i >= 0) i++;
    printf("We're done! %d\n", i);
    return 0;
}
```

**Y**

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    float i=0.0;
    while(i >= 0) i++;
    printf("We're done! %f\n",i);
    return 0;
}
```

Please identify the correct statement.

A. X will print "We're done" and finish, but Y will not.
B. X won't print "We're done" and won't finish, but Y will.
C. Both X and Y will print "We're done" and finish
D. Neither X nor Y will finish

# Will the loop end?

- Consider the following two C programs.

**X**

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    int i=0;
    while(i >= 0) i++;
    printf("We're done! %d\n", i);
    return 0;
}
```

**Y**

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    float i=0.0;
    while(i >= 0) i++;
    printf("We're done! %f\n",i);
    return 0;
}
```

**To know why — We need to figure out how "float" is handled in hardware!**

Please identify the correct statement.

A. X will print "We're done" and finish, but Y will not.

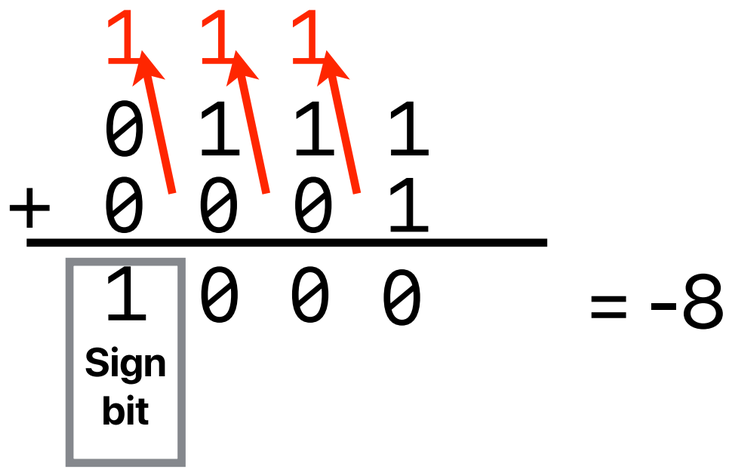
B. X won't print "We're done" and won't finish, but Y will.

C. Both X and Y will print "We're done" and finish

D. Neither X nor Y will finish

# Let's revisit the 4-bit binary adding

- 7 + 1 = ?

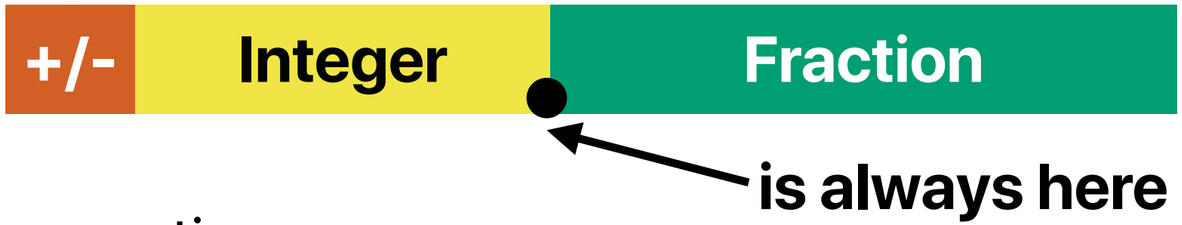



- If you add the largest integer with 1, the result will become the smallest integer.

# Representation of numbers with decimal points

# “Floating” v.s. “Fixed” point

- We want to express both a relational number’s “integer” and “fraction” parts
- Fixed point
  - One bit is used for representing positive or negative
  - Fixed number of bits is used for the integer part
  - Fixed number of bits is used for the fraction part
  - Therefore, the decimal point is **fixed**
- Floating point
  - One bit is used for representing positive or negative
  - A fixed number of bits is used for exponent
  - A fixed number of bits is used for fraction
  - Therefore, the decimal point is **floating — depending on the value of exponent**




Poll close in 1:30

# The advantage of floating/fixed point

- Regarding the pros of floating point and fixed point expressions, please identify the correct statement
  A. Fixed point can be express wider range of numbers than floating point numbers, but the hardware design is more complex
  B. Floating point can be express wider range of numbers than floating point numbers, but the hardware design is more complex
  C. Fixed point can be express wider range of numbers than floating point numbers, and the hardware design is simpler
  D. Floating point can be express wider range of numbers than floating point numbers, and the hardware design is simpler

# The advantage of floating/fixed point

- Regarding the pros of floating point and fixed point expressions, please identify the correct statement
  A. Fixed point can be express wider range of numbers than floating point numbers, but the hardware design is more complex
  B. Floating point can be express wider range of numbers than floating point numbers, but the hardware design is more complex
  C. Fixed point can be express wider range of numbers than floating point numbers, and the hardware design is simpler
  D. Floating point can be express wider range of numbers than floating point numbers, and the hardware design is simpler

# IEEE 32-bit floating point format

# IEEE 754 format


- Realign the number into 1.**F** * $2^{e}$
- Exponent stores **e** + 127
- Fraction only stores **F**

Poll close in 1:30

# IEEE 754 format


- Realign the number into 1.**F** * $2^{e}$
- Exponent stores **e** + 127
- Fraction only stores **F**
- Convert the following number
  1 1000 0010 0100 0000 0000 0000 0000 000
  A. - 1.010 * 2^130
  B. -10
  C. 10
  D. 1.010 * 2^130
  E. None of the above

# IEEE 754 format


- Realign the number into 1.**F** * $2^{e}$
- Exponent stores **e** + 127
- Fraction only stores **F**
- Convert the following number
  1 1000 0010 0100 0000 0000 0000 0000 000

A. - 1.010 * 2^130
B. -10
C. 10
D. 1.010 * 2^130
E. None of the above

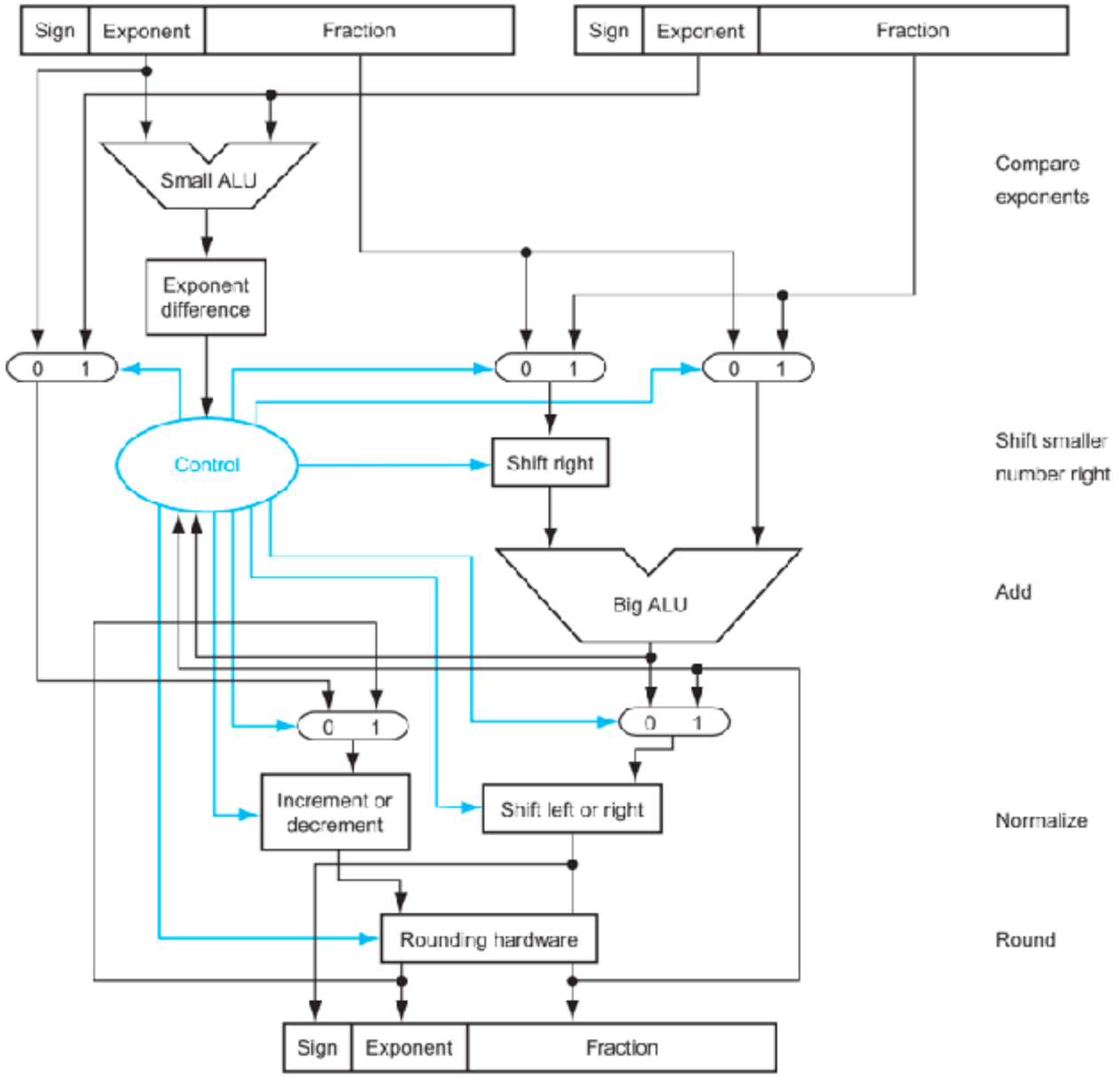

\- e = 130
-127 = 3

1.f = 1.01 = 1 + 0*$2^{-1}$ + 1* $2^{-2}$ = 1.25

1.25 * 2^3 = 10

# Floating point hardware

# Floating point adder



# Why — Will the loop end?

- Consider the following two C programs.

**X**

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    int i=0;
    while(i >= 0) i++;
    printf("We're done! %d\n", i);
    return 0;
}
```

**Y**

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    float i=0.0;
    while(i >= 0) i++;
    printf("We're done! %f\n",i);
    return 0;
}
```

**Because Floating Point Hardware Handles "sign", "exponent", "mantissa" separately**

Please identify the correct statement.

A. X will print "We're done" and finish, but Y will not.

B. X won't print "We're done" and won't finish, but Y will.

C. Both X and Y will print "We're done" and finish

D. Neither X nor Y will finish

Poll close in 1:30

# Comparing float and int

- Comparing 32-bit floating point (float) and 32-bit integer, which of the following statement is correct?
  A. An int can represent more different numbers than float, but the maximum number a float can express is larger than int
  B. A float can represent more different numbers than float, but the maximum number an int can express is larger than float
  C. A float can represent more different numbers than int and the maximum number in float is larger than int
  D. A int can represent more different numbers than float and the maximum number in int is larger than float
  E. None of the above is correct

# Maximum and minimum in float

**1111 1111 = NaN**


**254-127 =127** **1.1111 1111 1111 1111 1111 111**

**= 340282346638528859811704183484516925440**
**= 3.40282346639e+38**

max in int32 is 2^31-1 = 2147483647

**But, this also means that float cannot express all possible numbers between its max/min — lose of precisions**

# Demo — what's in c?

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    float a, b, c;
    a = 1280.245;
    b = 0.0004;
    c = a + b;
    printf("1280.245 + 0.0004 = %f\n",c);
    return 0;
}
```

# What's 0.0004 in IEEE 754?


| | after x2 | > 1? |
|---|---|---|
| 0.0004 | 0.0008 | 0 |
| 0.0008 | 0.0016 | 0 |
| 0.0016 | 0.0032 | 0 |
| 0.0032 | 0.0064 | 0 |
| 0.0064 | 0.0128 | 0 |
| 0.0128 | 0.0256 | 0 |
| 0.0256 | 0.0512 | 0 |
| 0.0512 | 0.1024 | 0 |
| 0.1024 | 0.2048 | 0 |
| 0.2048 | 0.4096 | 0 |
| 0.4096 | 0.8192 | 0 |
| 0.8192 | 1.6384 | 1 |
| 0.6384 | 1.2768 | 1 |
| 0.2768 | 0.5536 | 0 |
| 0.5536 | 1.1072 | 1 |
| 0.1072 | 0.2144 | 0 |
| 0.2144 | 0.4288 | 0 |
| 0.4288 | 0.8576 | 0 |
| 0.8576 | 1.7152 | 1 |
| 0.7152 | 1.4304 | 1 |

12

| | after x2 | > 1? |
|---|---|---|
| 0.4304 | 0.8608 | 0 |
| 0.8608 | 1.7216 | 1 |
| 0.7216 | 1.4432 | 1 |
| 0.4432 | 0.8864 | 0 |
| 0.8864 | 1.7728 | 1 |
| 0.7728 | 1.5456 | 1 |
| 0.5456 | 1.0912 | 1 |
| 0.0912 | 0.1824 | 0 |
| 0.1824 | 0.3648 | 0 |
| 0.3648 | 0.7296 | 0 |
| 0.7296 | 1.4592 | 1 |
| 0.4592 | 0.9184 | 0 |
| 0.9184 | 1.8368 | 1 |
| 0.8368 | 1.6736 | 1 |
| 0.6736 | 1.3472 | 1 |
| 0.3472 | 0.6944 | 0 |
| 0.6944 | 1.3888 | 1 |
| 0.3888 | 0.7776 | 0 |
| 0.7776 | 1.5552 | 1 |
| 0.5552 | 1.1104 | 1 |

-12 + 127 = 115 = 0b01110011

# Demo — Are we getting the same numbers?

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    float a, b, c;
    a = 1280.245;
    b = 0.0004;
    c = (a + b)*10.0;
    printf("(1280.245 + 0.0004)*10 = %f\n",c);
    c = a*10.0 + b*10.0;
    printf("1280.245*10 + 0.0004*10 = %f\n",c);
    return 0;
}
```

Poll close in 1:30

# Demo — Are we getting the same numbers?

- For the following code, please identify how many statements are correct
  - ① We will see the same output at X and Y
  - ② X will print — 12802.454
  - ③ Y will print — 12802.454
  - ④ Neither X nor Y will print the right result, but X is closer to the right answer
  - ⑤ Neither X nor Y will print the right result, but Y is closer to the right answer
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

```c
#include <stdio.h>

int main(int argc, char **argv) {
    float a, b, c;
    a = 1280.245;
    b = 0.0004;
    c = (a + b)*10.0;
    printf("%f\n",c); // X
    c = a*10.0 + b*10.0;
    printf("%f\n",c); // Y
    return 0;
}
```

# Demo — Are we getting the same numbers?

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    float a, b, c;
    a = 1280.245;
    b = 0.0004;
    c = (a + b)*10.0;
    printf("(1280.245 + 0.0004)*10 = %f\n",c);
    c = a*10.0 + b*10.0;
    printf("1280.245*10 + 0.0004*10 = %f\n",c);
    return 0;
}
```

**Commutative law is broken!!!**

# Are we getting the same numbers?

- For the following code, please identify how many statements are correct
  - ① We will see the same output at X and Y
  - ② X will print — 12802.454
  - ③ Y will print — 12802.454
  - ④ Neither X nor Y will print the right result, but X is closer to the right answer
  - ⑤ Neither X nor Y will print the right result, but Y is closer to the right answer

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

```c
#include <stdio.h>

int main(int argc, char **argv) {
    float a, b, c;
    a = 1280.245;
    b = 0.0004;
    c = (a + b)*10.0;
    printf("%f\n",c); // X
    c = a*10.0 + b*10.0;
    printf("%f\n",c); // Y
    return 0;
}
```

# What's 0.0004 in IEEE 754?


| | after x2 | > 1? |
|---|---|---|
| 0.0004 | 0.0008 | 0 |
| 0.0008 | 0.0016 | 0 |
| 0.0016 | 0.0032 | 0 |
| 0.0032 | 0.0064 | 0 |
| 0.0064 | 0.0128 | 0 |
| 0.0128 | 0.0256 | 0 |
| 0.0256 | 0.0512 | 0 |
| 0.0512 | 0.1024 | 0 |
| 0.1024 | 0.2048 | 0 |
| 0.2048 | 0.4096 | 0 |
| 0.4096 | 0.8192 | 0 |
| 0.8192 | 1.6384 | 1 |
| 0.6384 | 1.2768 | 1 |
| 0.2768 | 0.5536 | 0 |
| 0.5536 | 1.1072 | 1 |
| 0.1072 | 0.2144 | 0 |
| 0.2144 | 0.4288 | 0 |
| 0.4288 | 0.8576 | 0 |
| 0.8576 | 1.7152 | 1 |
| 0.7152 | 1.4304 | 1 |

12

| | after x2 | > 1? |
|---|---|---|
| 0.4304 | 0.8608 | 0 |
| 0.8608 | 1.7216 | 1 |
| 0.7216 | 1.4432 | 1 |
| 0.4432 | 0.8864 | 0 |
| 0.8864 | 1.7728 | 1 |
| 0.7728 | 1.5456 | 1 |
| 0.5456 | 1.0912 | 1 |
| 0.0912 | 0.1824 | 0 |
| 0.1824 | 0.3648 | 0 |
| 0.3648 | 0.7296 | 0 |
| 0.7296 | 1.4592 | 1 |
| 0.4592 | 0.9184 | 0 |
| 0.9184 | 1.8368 | 1 |
| 0.8368 | 1.6736 | 1 |
| 0.6736 | 1.3472 | 1 |
| 0.3472 | 0.6944 | 0 |
| 0.6944 | 1.3888 | 1 |
| 0.3888 | 0.7776 | 0 |
| 0.7776 | 1.5552 | 1 |
| 0.5552 | 1.1104 | 1 |

| | after x2 | > 1? |
|---|---|---|
| 0.1104 | 0.2208 | 0 |
| 0.2208 | 0.4416 | 0 |
| 0.4416 | 0.8832 | 0 |
| 0.8832 | 1.7664 | 1 |
| 0.7664 | 1.5328 | 1 |
| 0.5328 | 1.0656 | 1 |
| 0.0656 | 0.1312 | 0 |
| 0.1312 | 0.2624 | 0 |
| 0.2624 | 0.5248 | 0 |
| 0.5248 | 1.0496 | 1 |
| 0.0496 | 0.0992 | 0 |
| 0.0992 | 0.1984 | 0 |
| 0.1984 | 0.3968 | 0 |
| 0.3968 | 0.7936 | 0 |
| 0.7936 | 1.5872 | 1 |
| 0.5872 | 1.1744 | 1 |
| 0.1744 | 0.3488 | 0 |
| 0.3488 | 0.6976 | 0 |
| 0.6976 | 1.3952 | 1 |
| 0.3952 | 0.7904 | 0 |

# Special numbers in IEEE 754 float

| | | | |
|---|---|---|---|
| +0 | 0 | 0000 0000 | 0000 0000 0000 0000 0000 000 |
| -0 | 1 | 0000 0000 | 0000 0000 0000 0000 0000 000 |
| +Inf | 0 | 1111 1111 | 0000 0000 0000 0000 0000 000 |
| -Inf | 1 | 1111 1111 | 0000 0000 0000 0000 0000 000 |
| +NaN | 0 | 1111 1111 | xxxx xxxx xxxx xxxx xxxx xxx |
| -Nan | 1 | 1111 1111 | xxxx xxxx xxxx xxxx xxxx xxx |

# Comparing float and int

- Comparing 32-bit floating point (float) and 32-bit integer, which of the following statement is correct?

A. An int can represent more different numbers than float, but the maximum number a float can express is larger than int

B. A float can represent more different numbers than float, but the maximum number an int can express is larger than float

C. A float can represent more different numbers than int and the maximum number in float is larger than int

D. A int can represent more different numbers than float and the maximum number in int is larger than float

E. None of the above is correct

Poll close in 1:30

# Will the loop end? (one more run)

- Consider the following C program.

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    float i=1.0;
    while(i > 0) i++;
    printf("We're done! %f\n",i);
    return 0;
}
```

Please identify the correct statement.

A. The program will finish since i will end up to be +0
B. The program will finish since i will end up to be -0
C. The program will finish since i will end up to be something < 0
D. The program will not finish since i will always be a positive non-zero number.
E. The program will not finish but raise an exception since we will go to NaN first.

# Will the loop end? (one more run)

- Consider the following C program.

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    float i=1.0;
    while(i > 0) i++;
    printf("We're done! %f\n",i);
    return 0;
}
```

Please identify the correct statement.

A. The program will finish since i will end up to be +0
B. The program will finish since i will end up to be -0
C. The program will finish since i will end up to be something < 0
D. The program will not finish since i will always be a positive non-zero number.
E. The program will not finish but raise an exception since we will go to NaN first.

# Recap: Demo — Are we getting the same numbers?

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    float a, b, c;
    a = 1280.245;
    b = 0.0004;
    c = (a + b)*10.0;
    printf("(1280.245 + 0.0004)*10 = %f\n",c);
    c = a*10.0 + b*10.0;
    printf("1280.245*10 + 0.0004*10 = %f\n",c);
    return 0;
}
```

# Announcement

- Assignment 2 due **TONIGHT**
  - All challenge questions up to **3.5**
- Reading quiz 5 due 4/28 **BEFORE** the lecture
  - Under iLearn > reading quizzes
- Lab 3 due 4/30
  - Watch the video and read the instruction BEFORE your session
  - There are links on both course webpage and iLearn lab section
  - Submit through iLearn > Labs
- Midterm on 5/7 during the lecture time, access through iLearn — no late submission is allowed — make sure you will be able to take that at the time
- Check your grades in iLearn

Electrical
Computer Science
Engineering

120A

つづく