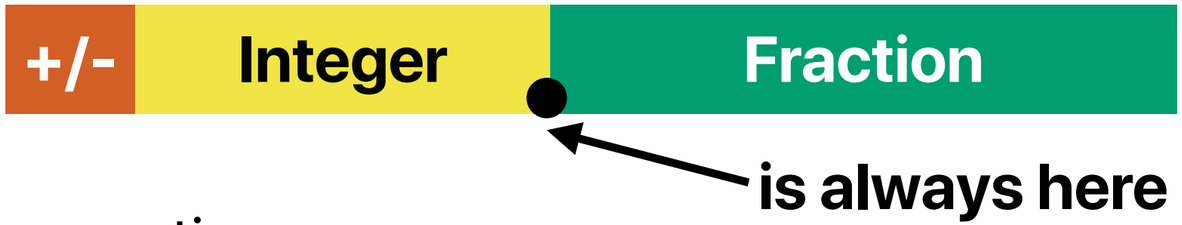
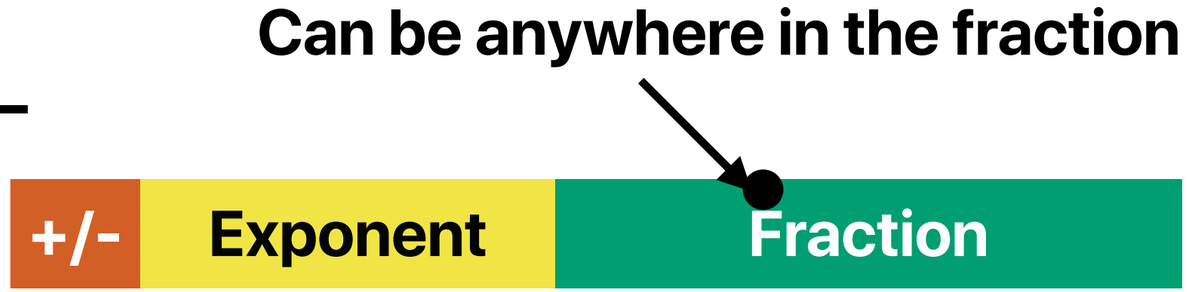
Floating Point (cont.) & Sequential Circuits

Prof. Usagi

Recap: “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**





Recap: What's 0.0004 in IEEE 754?

0	0	1	1	1	0	0	1	1	1	0	1	0	0	0	1	1	0	1	1	0	1	1	1	0	0	0	1	1	1	0	1

	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

Recap: Special numbers in IEEE 754 float

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.

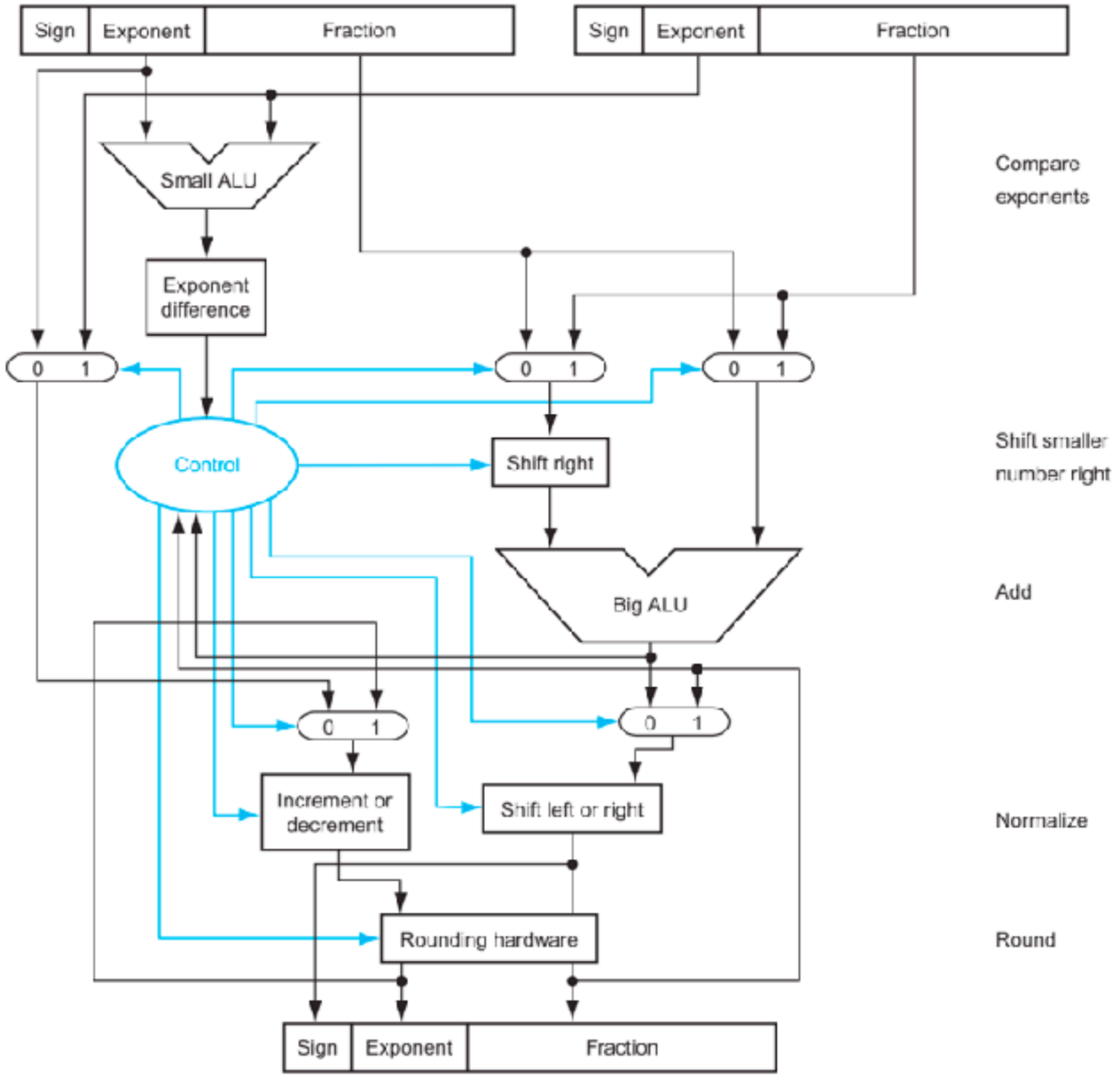
A. X will print "We're done" and finish, but Y will not.

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

Floating point adder



Recap: 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.

Poll close in 1:30

Why stuck at 16777216?

- 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;
}
```

Why i stuck at 16777216.000?

A. It's a special number in IEEE 754 standard that an adder will treat it differently
B. It's a special number like +Inf/-Inf or +NaN/-NaN with special meaning in the IEEE 754 standard
C. It's just the maximum integer that IEEE 754 standard can represent
D. It's nothing special, but just happened to be the case that 16777216.0+1.0 will produce 16777216.0
E. It's nothing special, but just happened to be the case that 16777216.0 add anything will become 16777216.0

Why stuck at 16777216?

- 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;
}
```

Why i stuck at 16777216.000?

A. It's a special number in IEEE 754 standard that an adder will treat it differently
B. It's a special number like +Inf/-Inf or +NaN/-NaN with special meaning in the IEEE 754 standard
C. It's just the maximum integer that IEEE 754 standard can represent
D. It's nothing special, but just happened to be the case that 16777216.0+1.0 will produce 16777216.0
E. It's nothing special, but just happened to be the case that 16777216.0 add anything will become 16777216.0

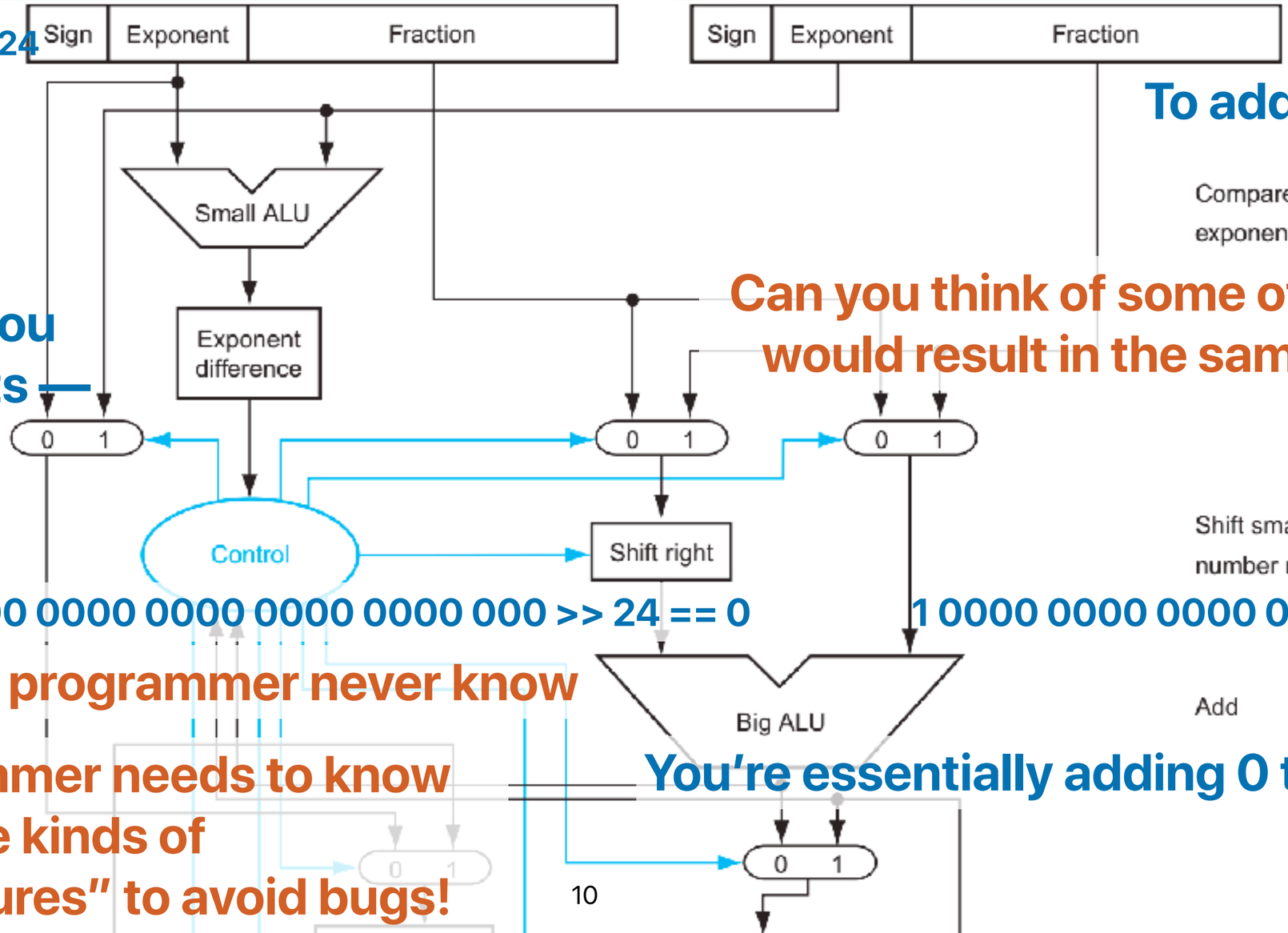
What's 16777216 special about?

0 | 10010111 | 0000 0000 0000 0000 0000 000

16777216 = 1.0 * 2^{24}

0 | 0111 1111 | 0000 0000 0000 0000 0000 000

To add 1.0 = 1.0 *2^0



to this number, you have to shift 24 bits —

Can you think of some other numbers would result in the same situation?

1 0000 0000 0000 0000 0000 000 >> 24 == 0

1 0000 0000 0000 0000 0000 000

— even worse — programmer never know

You're essentially adding 0 to 16777216

A good programmer needs to know these kinds of "hardware features" to avoid bugs!

Poll close in 1:30

Will the loop end? (last run)

- Consider the following C program.

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    float i=1.0;
    while(i > 0) i+=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 something < 0
C. The program will not finish since i will always be a positive non-zero number.
D. The program will not finish since i will end up staying at some special FP32 presentation
E. The program will not finish but raise an exception since we will go to NaN first.

Will the loop end? (last run)

- Consider the following C program.

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    float i=1.0;
    while(i > 0) i+=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 something < 0
C. The program will not finish since i will always be a positive non-zero number.
D. The program will not finish since i will end up staying at some special FP32 presentation
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;
}
```

Outline

- More on floating points, data types
- Introduction to sequential circuit
- Finite State Machines

More on floating points

Other floating point formats

Floating Point Bit Depth	Largest value	Smallest value	Decimal digits of precision
64-bit double	1.80×10^{308}	2.23×10^{-308}	~ 16
32-bit Float	3.4028237×10^{38}	1.175494×10^{-38}	~ 7

Revisit: Demo — Are we getting the same numbers?

```c
#include <stdio.h>

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

Poll close in 1:30

Will the loop end? — if we use double

- Consider the following C program.

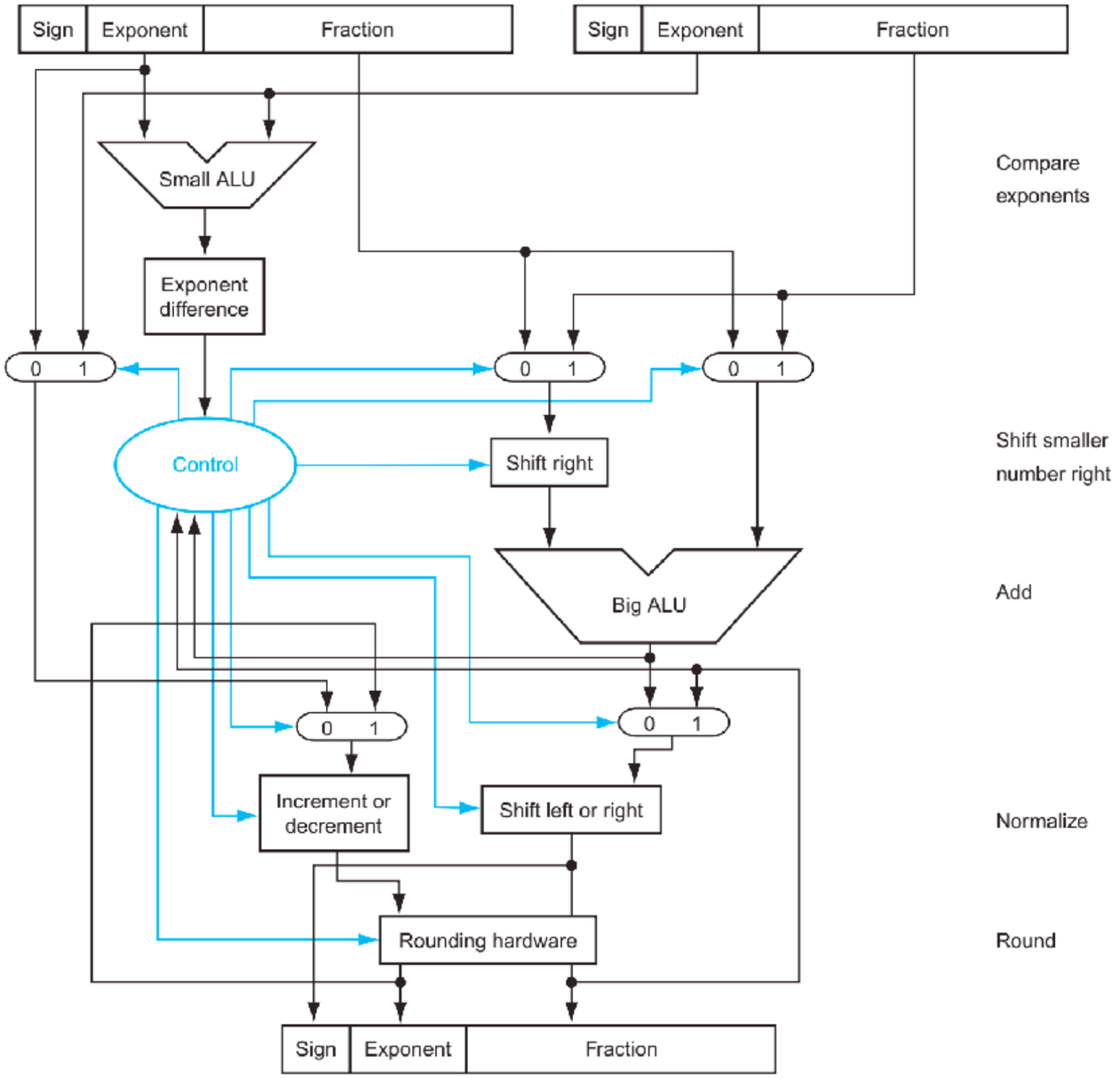
```c
#include <stdio.h>

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

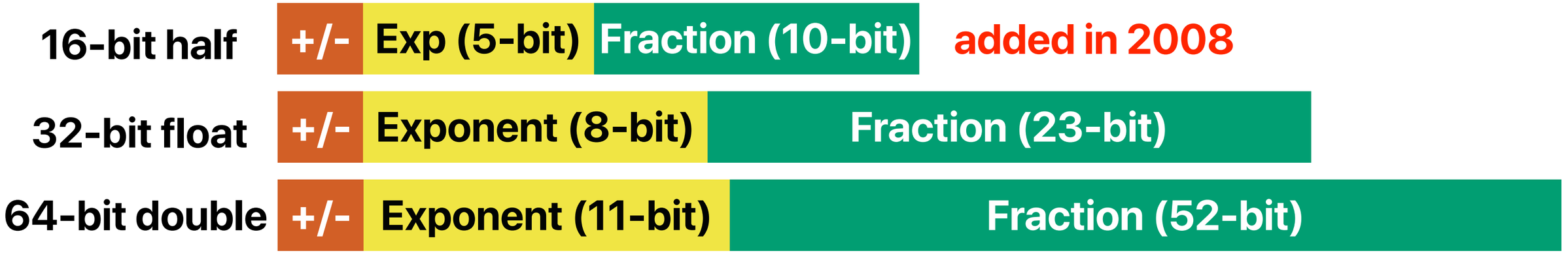
Please identify the correct statement.

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 since i will reach +inf and stay at +inf.

Recap: Floating point adder



Other floating point formats



Floating Point Bit Depth	Largest value	Smallest value	Decimal digits of precision
64-bit double	1.80×10^{308}	2.23×10^{-308}	~ 16
32-bit Float	3.4028237×10^{38}	1.175494×10^{-38}	~ 7
16-bit Float	6.55×10^{4}	6.10×10^{-5}	~ 3

Poll close in 1:30

Why float-16?

- Recent processors/GPUs/accelerators start to provide native support less-precise data types (16-bit floating point). How many of the following are possible reasons explaining such a move?
  - ① Using less-precise data types allows the same number of transistors to deliver more results simultaneously
  - ② Using less-precise data types + numerical methods can reach the same precision with shorter latency
  - ③ Using less-precise data types allows the hardware to support more applications
  - ④ Using less-precise data types helps to lower the hardware cost in delivering the same computation throughput

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

Why float-16?

- Recent processors/GPUs/accelerators start to provide native support less-precise data types (16-bit floating point). How many of the following are possible reasons explaining such a move?
  - ① Using less-precise data types allows the same number of transistors to deliver more results simultaneously
  - ② Using less-precise data types + numerical methods can reach the same precision with shorter latency
  - ③ Using less-precise data types allows the hardware to support more applications
  - ④ Using less-precise data types helps to lower the hardware cost in delivering the same computation throughput

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

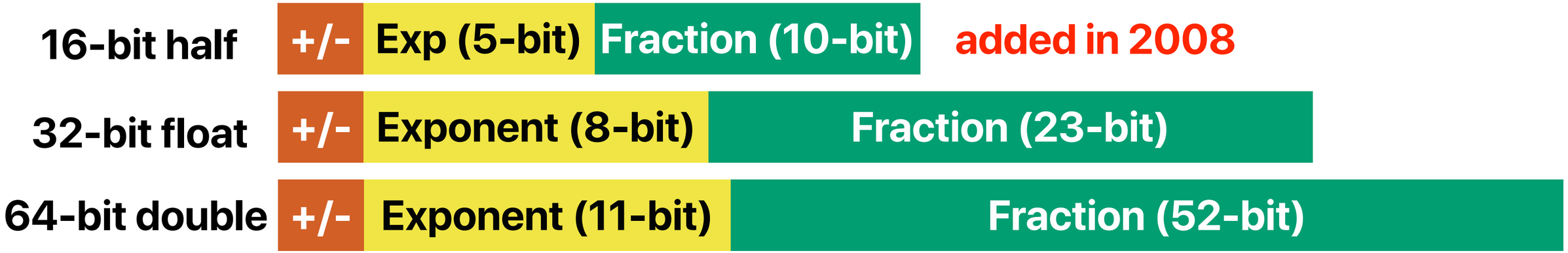
Can you tell the difference?

Higher resolution

But we all can tell they are our mascots!

How about this?

Other floating point formats



- Not all applications require “high precision”
- Deep neural networks are surprisingly error tolerable

Why float-16?

- Recent processors/GPUs/accelerators start to provide native support less-precise data types (16-bit floating point). How many of the following are possible reasons explaining such a move?
  - ① Using less-precise data types allows the same number of transistors to deliver more results simultaneously
  - ② Using less-precise data types + numerical methods can reach the same precision with shorter latency
  - ③ Using less-precise data types allows the hardware to support more applications
  - ④ Using less-precise data types helps to lower the hardware cost in delivering the same computation throughput
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

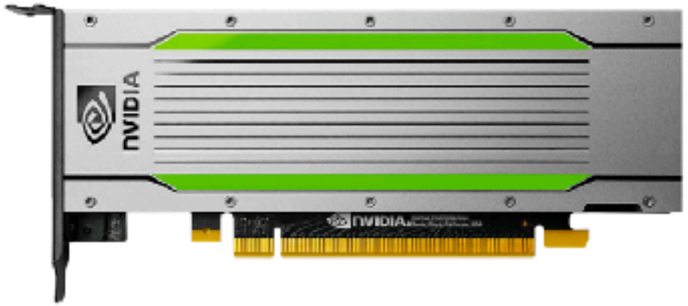
Mixed-precision

Double Precision Results

GPU	Tesla T4	Tesla V100	Tesla P100
Max Flops (GFLOPS)	253.38	7072.86	4736.76
Fast Fourier Transform (GFLOPS)	132.60	1148.75	756.29
Matrix Multiplication (GFLOPS)	249.57	5920.01	4256.08
Molecular Dynamics (GFLOPS)	105.26	908.62	402.96
S3D (GFLOPS)	59.97	227.85	161.54

Single Precision Results

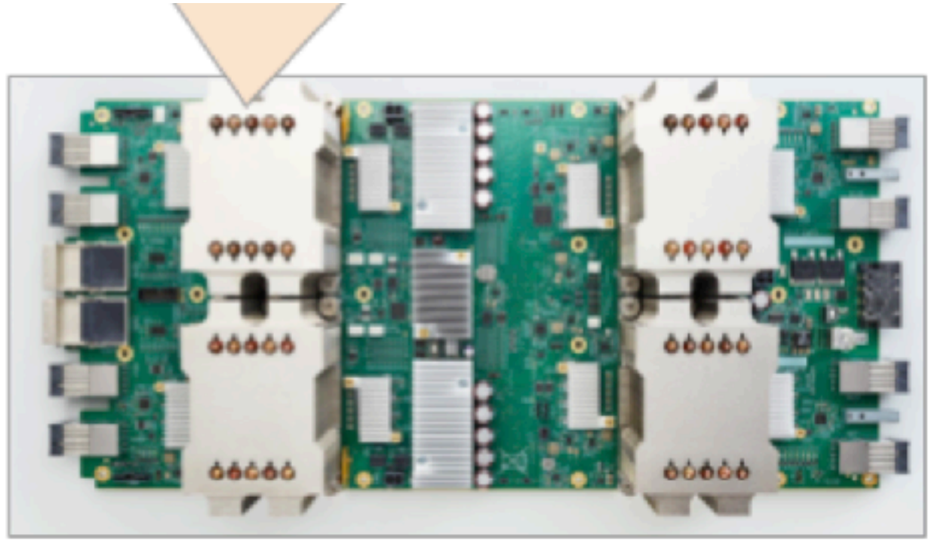
GPU	Tesla T4	Tesla V100	Tesla P100
Max Flops (GFLOPS)	8073.26	14016.50	9322.46
Fast Fourier Transform (GFLOPS)	660.05	2301.32	1510.49
Matrix Multiplication (GFLOPS)	3290.94	13480.40	8793.33
Molecular Dynamics (GFLOPS)	572.91	997.61	480.02
S3D (GFLOPS)	99.42	434.78	295.20



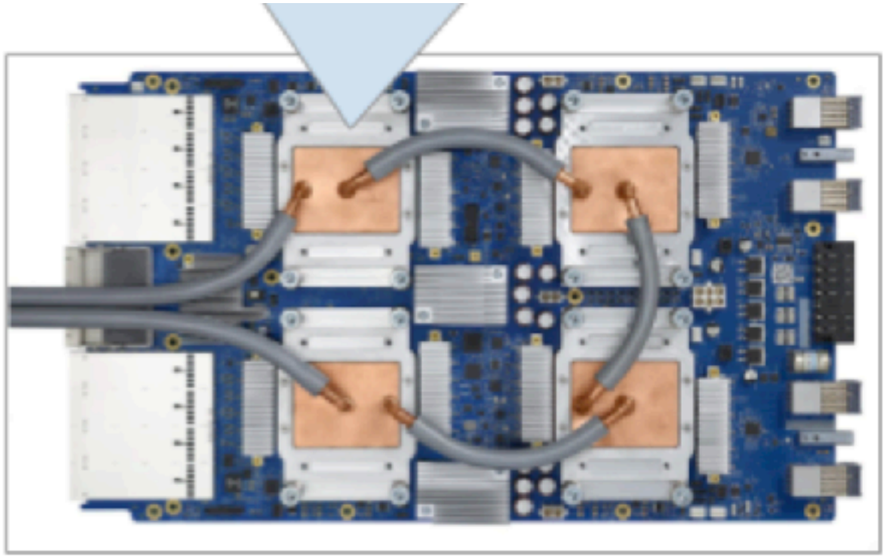
SPECIFICATIONS

GPU Architecture	**NVIDIA Turing**
NVIDIA Turing Tensor Cores	**320**
NVIDIA CUDA® Cores	**2,560**
Single-Precision	**8.1 TFLOPS**
Mixed-Precision (FP16/FP32)	**65 TFLOPS**
INT8	**130 TOPS**
INT4	**260 TOPS**
GPU Memory	**16 GB GDDR6 300 GB/sec**
ECC	**Yes**
Interconnect Bandwidth	**32 GB/sec**
System Interface	**x16 PCIe Gen3**
Form Factor	**Low-Profile PCIe**
Thermal Solution	**Passive**
Compute APIs	**CUDA, NVIDIA TensorRT™, ONNX**

Google's Tensor Processing Units



TPU v2 - 4 chips, 2 cores per chip



TPU v3 - 4 chips, 2 cores per chip

Each TPU core has scalar, vector, and matrix units (MXU). The MXU provides the bulk of the compute power in a TPU chip. Each MXU is capable of performing 16K multiply-accumulate operations in each cycle. While the MXU inputs and outputs are 32-bit floating point values, the MXU performs multiplies at reduced bfloat16 precision. Bfloat16 is a 16-bit floating point representation that provides better training and model accuracy than the IEEE half-precision representation.

https://cloud.google.com/tpu/docs/system-architecture

EdgeTPU

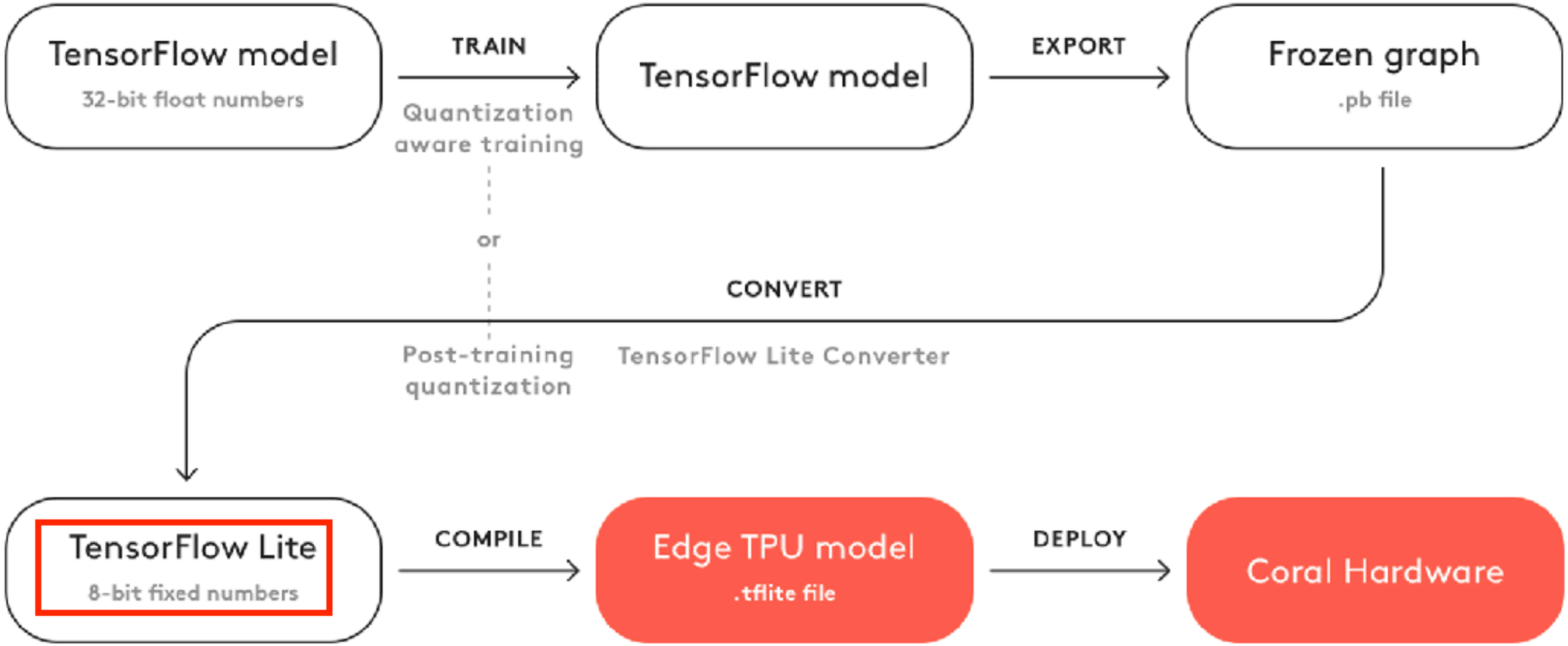
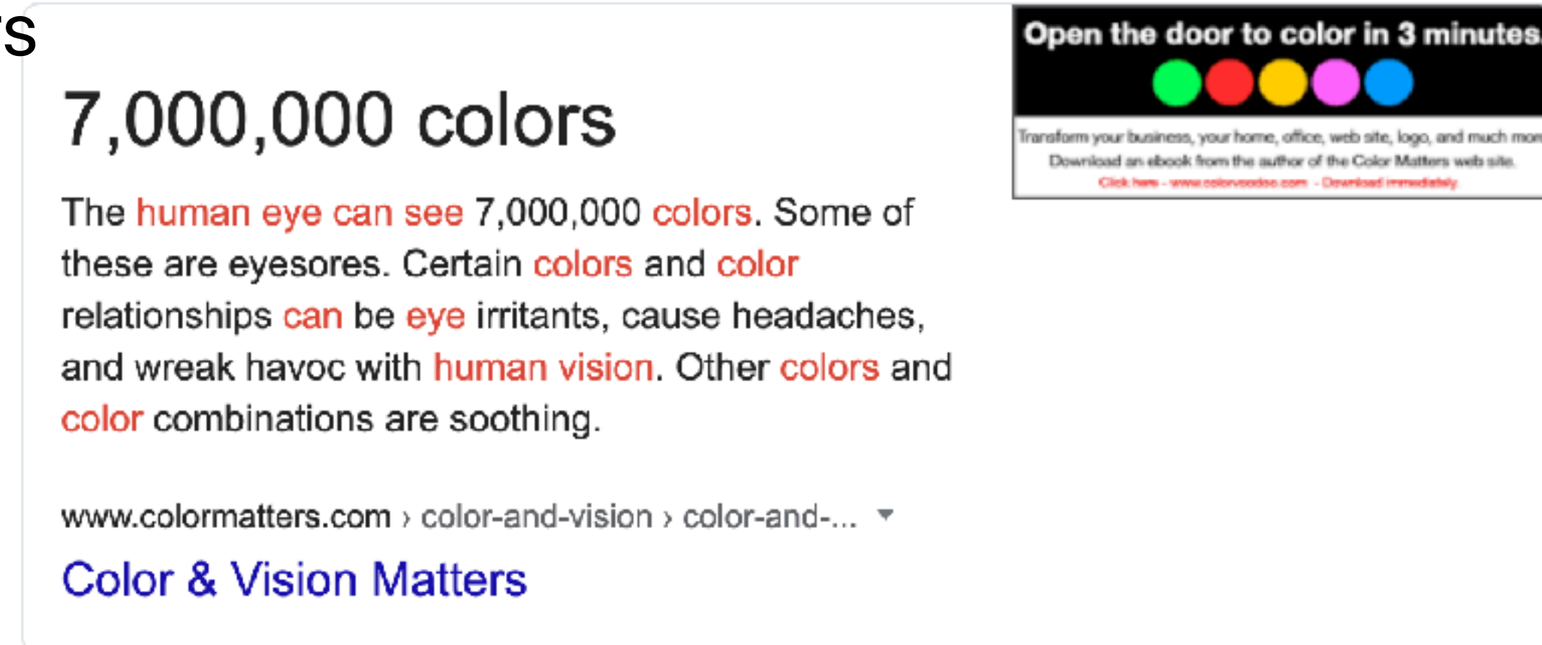


Figure 1. The basic workflow to create a model for the Edge TPU

Poll close in 1:30

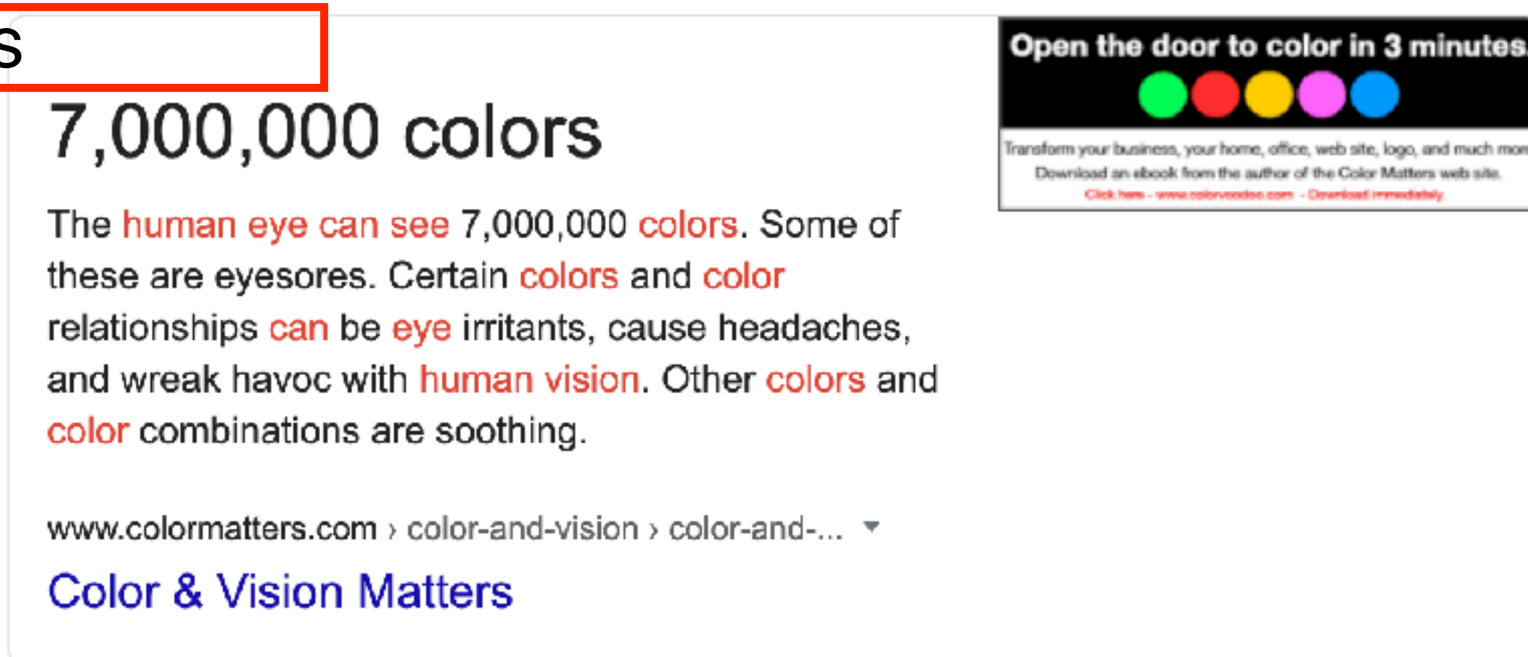
How to efficiently express colors?

- Since human eyes can only see 7 million colors, which of the following would be the most efficient way to express the color of a pixel?
  A. 8-bit character
  B. 16-bit short integer
  C. Three 8-bit characters
  D. 32-bit integer
  E. 32-bit float



How to efficiently express colors?

- Since human eyes can only see 7 million colors, which of the following would be the most efficient way to express the color of a pixel?
  A. 8-bit character
  B. 16-bit short integer
  C. Three 8-bit characters
  D. 32-bit integer
  E. 32-bit float



Summary of what we have seen so far

- Transistors
- Boolean algebra
- Basic gates
- Logic functions and truth tables
- Canonical forms (SOP and POS)
- Two-level logic minimization
- Kmaps
- Decoders
- Multiplexers (behavior and how to implement logic functions with them)
- Adders, subtractors, and other ALU components
- All above are “combinational circuits”!

Introduction on Sequential Circuits

Recap: Combinational v.s. sequential logic

- Combinational logic
  - The output is a pure function of its current inputs
  - The output doesn’t change regardless how many times the logic is triggered — Idempotent
- Sequential logic
  - The output depends on current inputs, previous inputs, their history

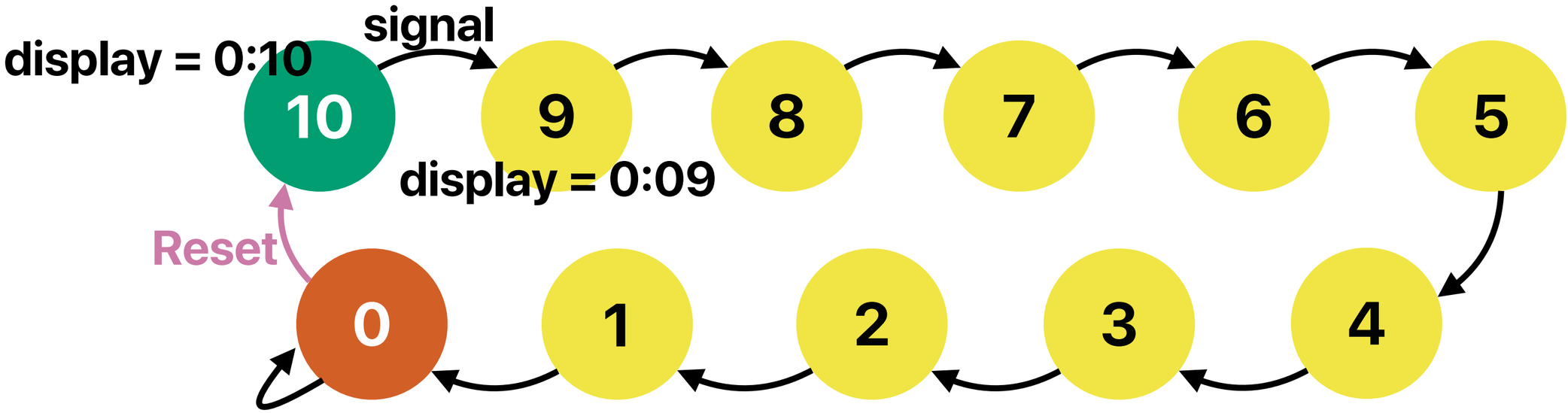
Sequential circuit has memory!

Recap: Theory behind each

- A **Combinational logic** is the implementation of a **Boolean Algebra** function with only Boolean Variables as their inputs
- A **Sequential logic** is the implementation of a **Finite-State Machine**

Count-down Timer

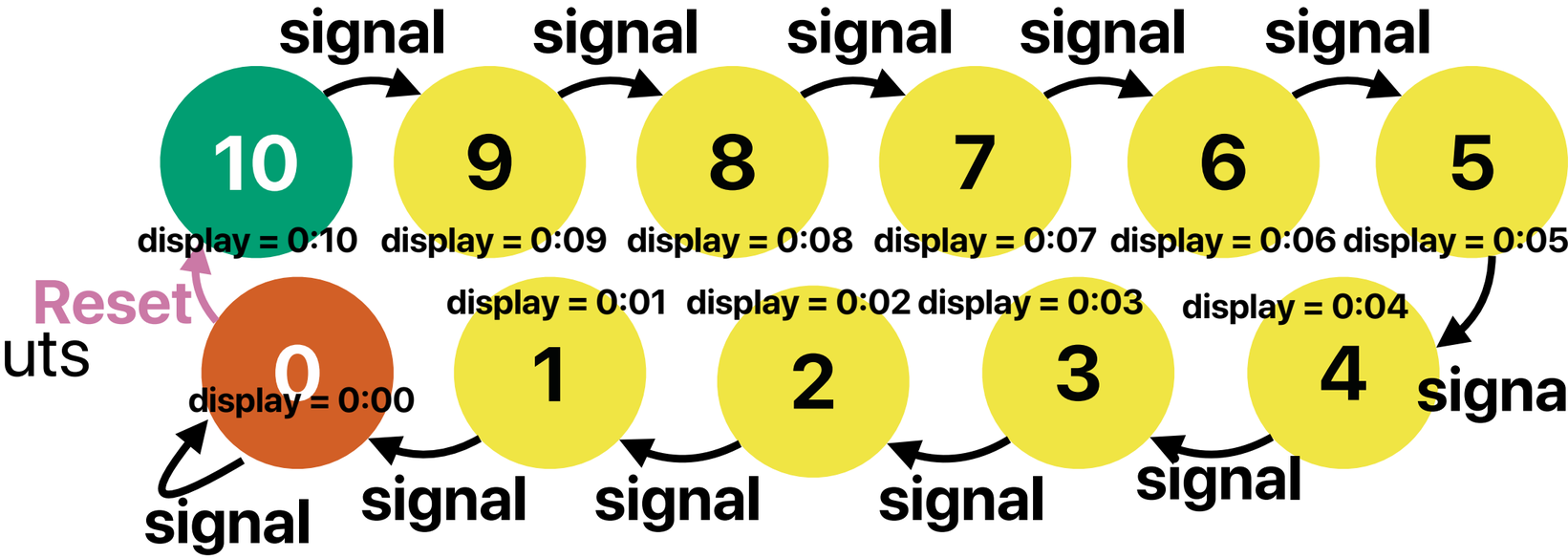
- What do we need to implement this timer?
  - Set an initial value/"state" of the timer
  - "Signal" the design every second
  - The design changes its "state" every time we received the signal until we reaches "0" — the final state



Finite-State Machines

Finite State Machines

- FSM consists of
  - Set of states
  - Set of inputs, set of outputs
  - Initial state
  - Set of transitions
    - Only one can be true at a time
- FSM representations:
  - State diagram
  - State table



Current State	Next State Signal	
	0	1
10	10	9
9	9	8
8	8	7
7	7	6
6	6	5
5	5	4
4	4	3
3	3	2
2	2	1
1	1	0
0	0	0

Poll close in 1:30

Life on Mars

- Mars rover has a binary input x. When it receives the input sequence x(t-2, t) = 001 from its life detection sensors, it means that the it has detected life on Mars and the output y(t) = 1, otherwise y(t) = 0 (no life on Mars).
- This pattern recognizer should have
  A. One state because it has one output
  B. One state because it has one input
  C. Two states because the input can be 0 or 1
  D. More than two states because ….
  E. None of the above

Life on Mars

- Mars rover has a binary input x. When it receives the input sequence x(t-2, t) = 001 from its life detection sensors, it means that the it has detected life on Mars and the output y(t) = 1, otherwise y(t) = 0 (no life on Mars).
- This pattern recognizer should have
  A. One state because it has one output
  B. One state because it has one input
  C. Two states because the input can be 0 or 1
  D. More than two states because ….
  E. None of the above

Announcement

- 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
  - Covers: Chapter 1, Chapter 2, Chapter 3.1 — 3.12, Chapter 3.15 & 3.16, Chapter 4.1—4.9
  - Midterm review next Tuesday will reveal more information (e.g., review on key concepts, test format, slides of a sample midterm)
- Lab 4 is up — due after final (5/12). Would rely on the content from today & Thursday's lecture
- Check your grades in iLearn

Electrical
Computer Science
Engineering

120A

つづく