COMPUTER SYSTEMS AND ORGANIZATION

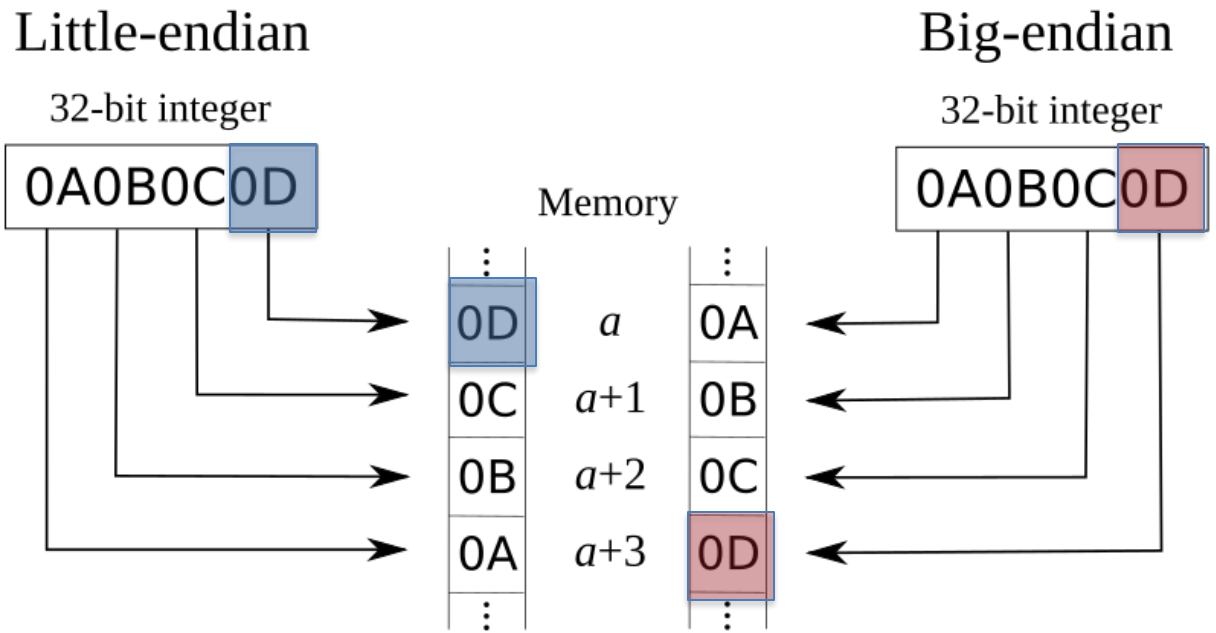
Muxes

Daniel Graham

UNIVERSITY of VIRGINIA | ENGINEERING

REVIEW

ENDIANNESS



EXAM QUESTION FALL 2018

Question 13 [2 pt]: If the 32-bit number `0x12345678` is stored in **little-endian** at address `0x20`, what is the value of the byte at address `0x22`? Answer in hexadecimal.

Answer:

Question 14 [2 pt]: If you read the bytes [`fe`, `dc`] as an unsigned **big-endian** 16-bit number, what is that number? Answer in hexadecimal.

Answer:

EXAM QUESTION FALL 2018

Question 13 [2 pt]: If the 32-bit number `0x12345678` is stored in **little-endian** at address `0x20`, what is the value of the byte at address `0x22`? Answer in hexadecimal.

Answer: 0x34

Question 14 [2 pt]: If you read the bytes [`fe`, `dc`] as an unsigned **big-endian** 16-bit number, what is that number? Answer in hexadecimal.

Answer: 0xfedc

EXAM QUESTIONS FALL 2019

Suppose an array of two 32-bit values (`[0xabcdef01, 0x7645231]`) is stored at address `0x200`. What byte is stored at address `0x204`? Answer in hexadecimal.

Question 13 [2 pt]: (see above) Assume little-endian storage.

Answer:

Question 14 [1 pt]: (see above) Assume big-endian storage.

Answer:

Remember to draw a picture of how arrays are stored in memory.

(Remember to group second number into bytes)

EXAM QUESTIONS FALL 2019

Suppose an array of two 32-bit values (`[0xabcdef01, 0x7645231]`) is stored at address `0x200`. What byte is stored at address `0x204`? Answer in hexadecimal.

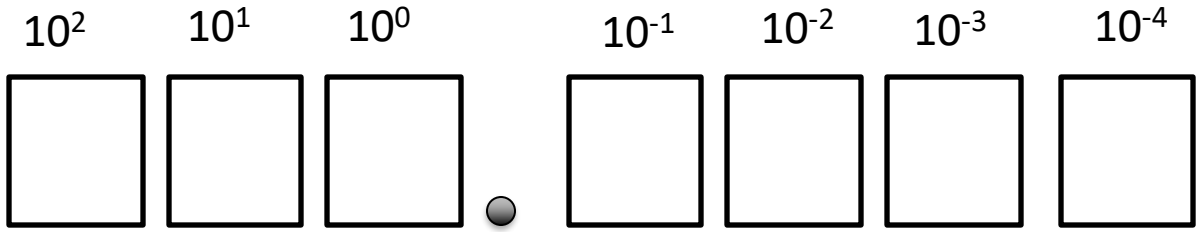
Question 13 [2 pt]: (see above) Assume little-endian storage.

Answer: 0x31

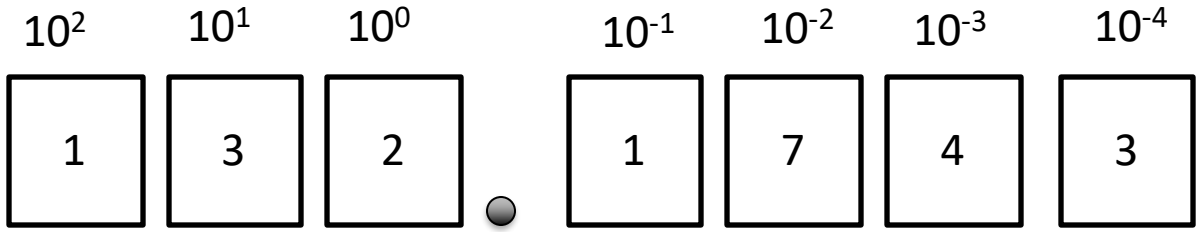
Question 14 [1 pt]: (see above) Assume big-endian storage.

Answer: 0x07

FLOATING POINT BASE 10

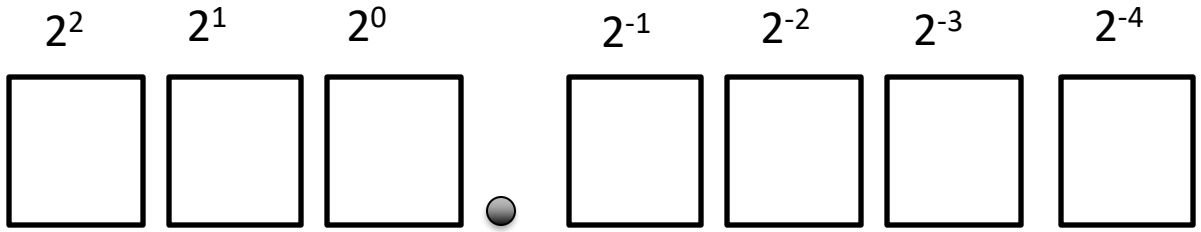


FLOATING POINT BASE 10

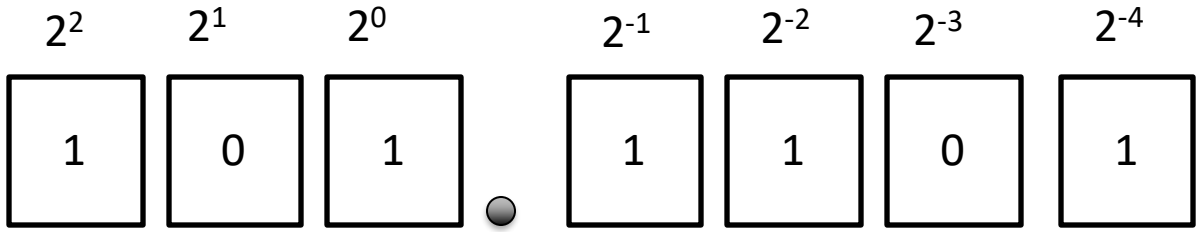


$1 \times 10^2 + 3 \times 10^1 + 2 \times 10^0 + 1 \times 1/10 + 7 \times 1/100 + 4 \times 1/1000 + 3 \times 1/10000 = 132.1743$

FLOATING POINT BASE 2



FLOATING POINT BASE 2



$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$

$1 \times 1/2 + 1 \times 1/4 + 0 \times 1/8 + 1 \times 1/16 = 13/16$

$5\ 13/16 = 5+0.8125 = 5.8125$

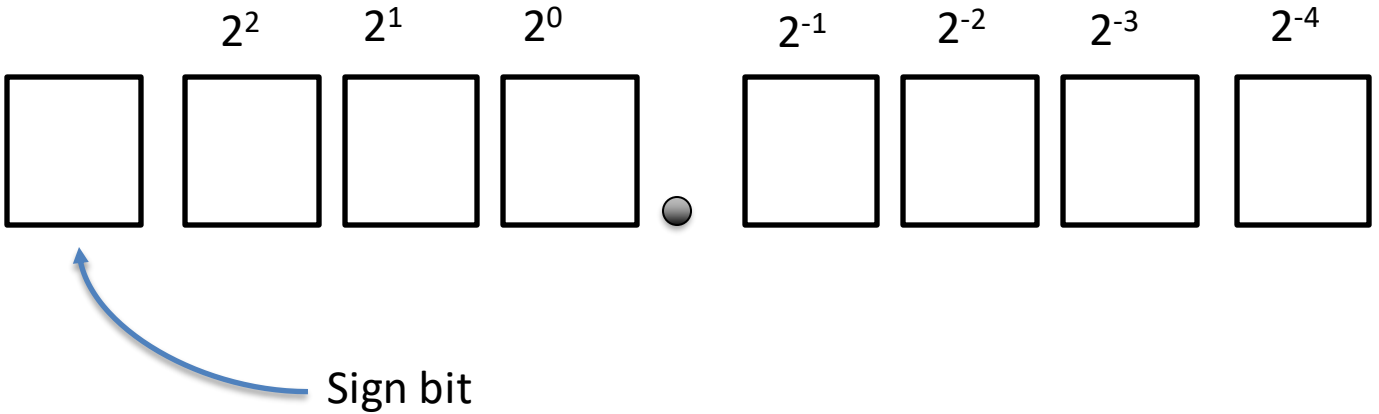
NOW WE JUST NEED THE EXPONENT

Now we need the exponent so we can float the point.

$74.0 * 10^2$
$740.0 * 10^1$
$7400.0 * 10^1$

$-74.0 * 10^2$
$- 740.0 * 10^1$
$- 7400.0 * 10^1$

FLOATING POINT BASE 2 (NEGATIVE NUMBERS)

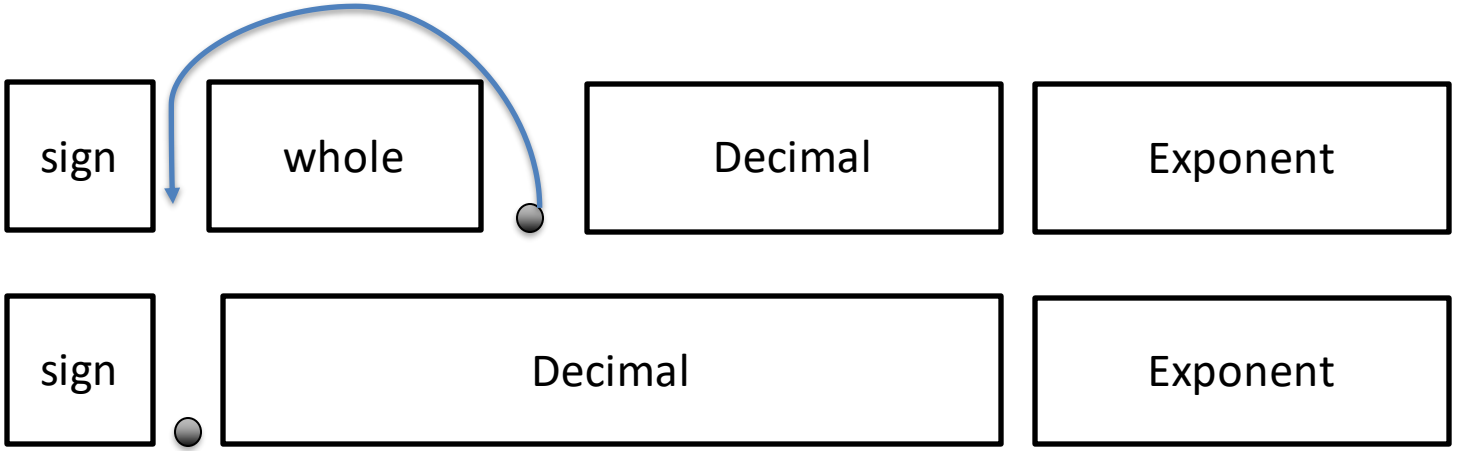


NOW WE JUST NEED THE EXPONENT

number = sign(whole + decimal) x $2^{exponent}$

1 11.111 x 2^{E}

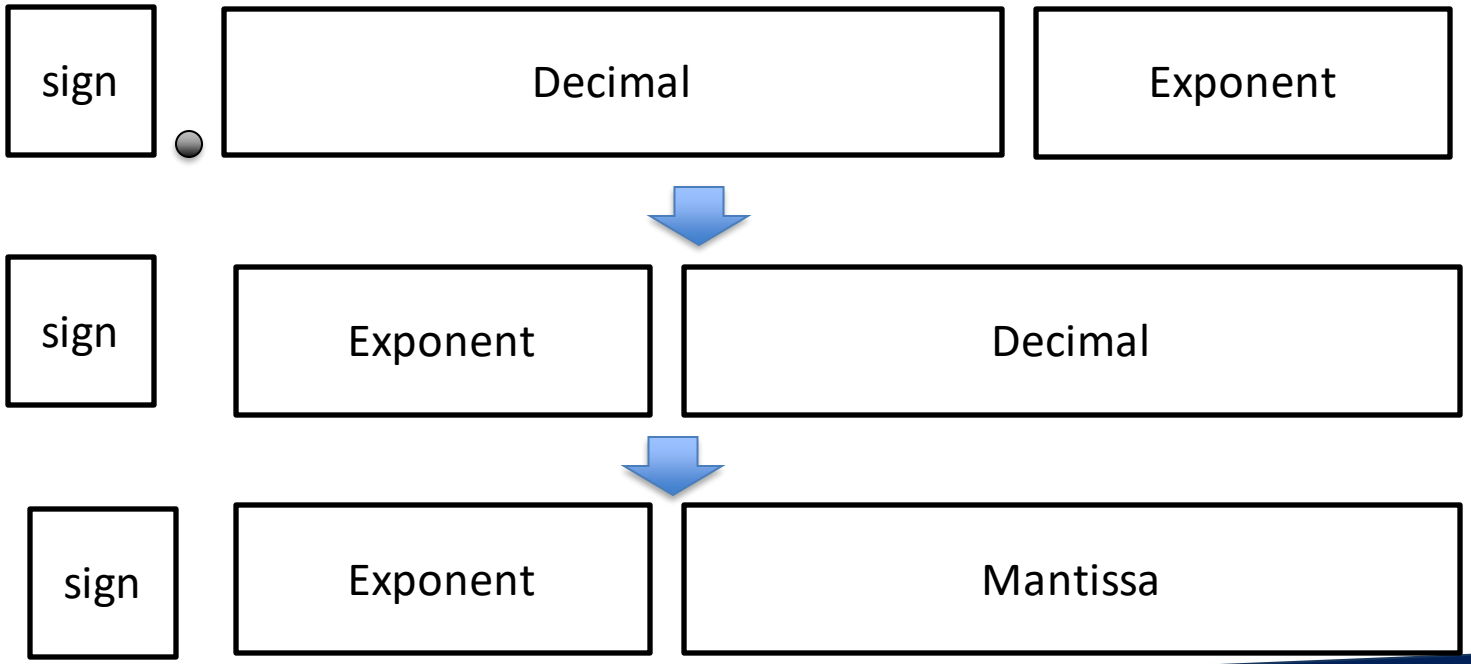
ADDING THE EXPONENT DELETING THE WHOLE NUMBER SECTION



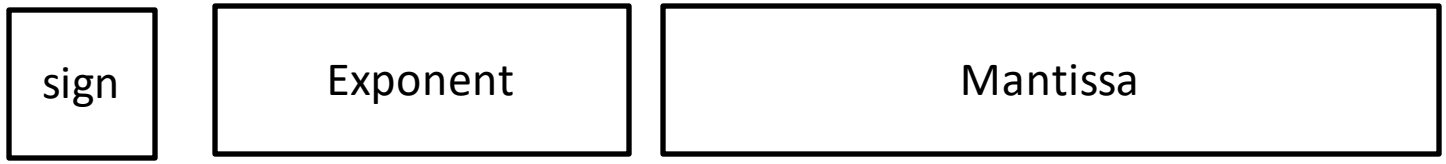
$-74.01 * 10^{2}$

$-0.7401 * 10^{4}$

IEEE 754 FLOATING POINT STANDARD



IEEE 754

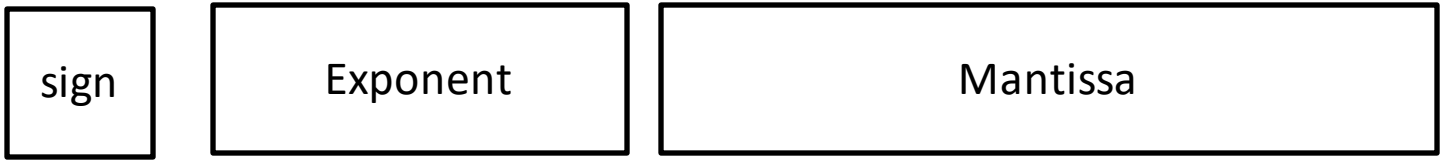


$$number = sign(1 + Mantissa) \times 2^{exponent - bias}$$

On 32-bit machines bias is normally 127 (Yes this is bias representation we talked about earlier)
0b01111111 (8 digits) is what we use when adding to the exponent

For 64-bit machines, the bias would be 1023, and this format allows for handling of negative exponents as the biased exponent is always stored as a positive number

IEEE 754



$$number = sign(1 + Mantissa) \times 2^{exponent - bias}$$

Remember this is a base 2 binary string

$1.ffff \times 2^{exponent - bias}$

Remember 2^0 is one

BINARY STRING

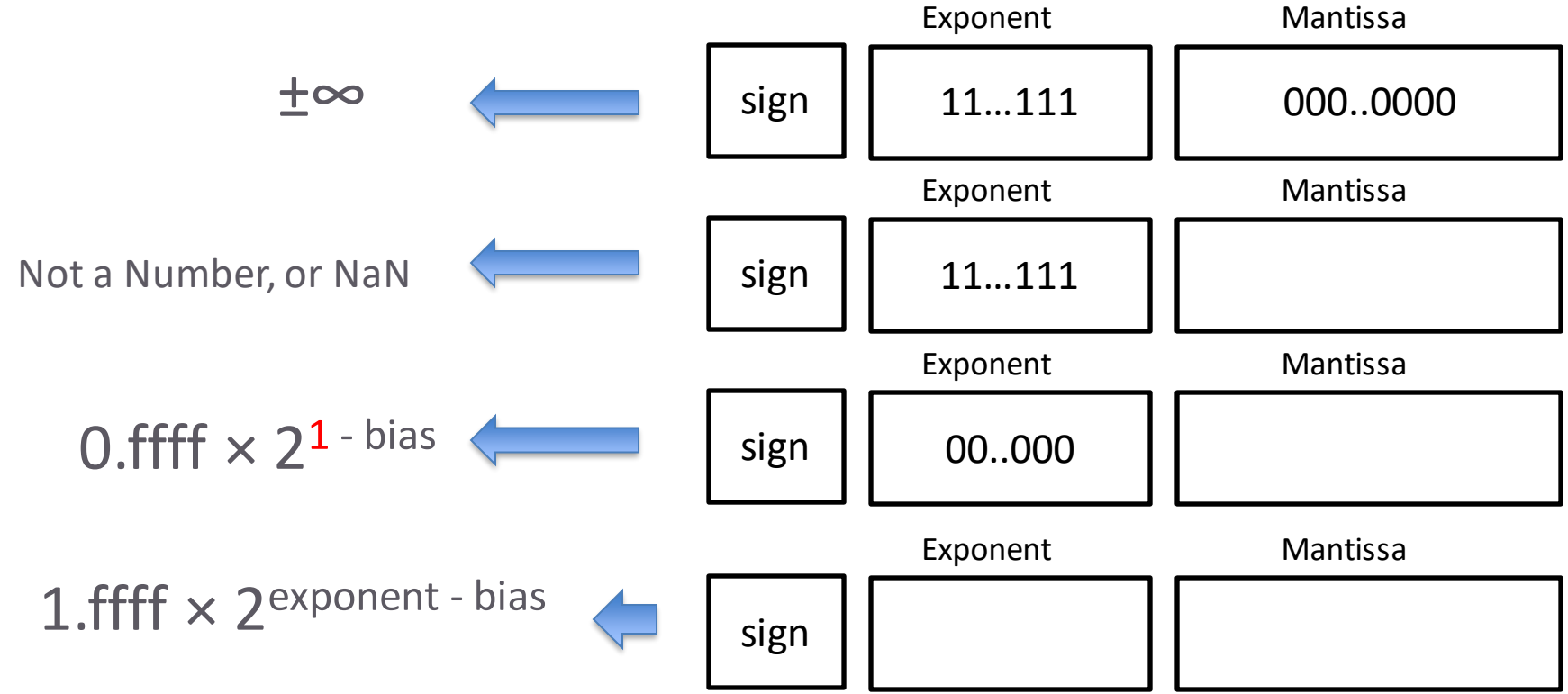
$0.1101 = 1.101 \times 2^{-1}$

$0.01101 = 1.101 \times 2^{-2}$

$0.001101 = 1.101 \times 2^{-3}$

Keep going until you get to your first 1. (one 1 left to the decimal point)

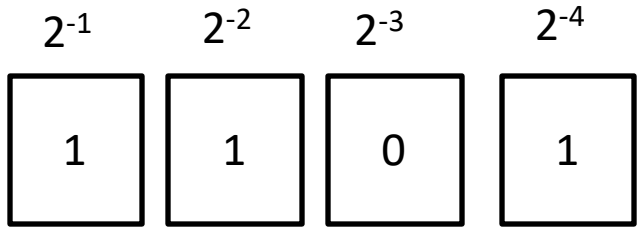
IEEE 754 FOUR CASES



CONVERSION EXAMPLE

Let's convert 0.8125 to floating-point representation

0.8125 x 2=1.6250 **1**
0.6250 x 2=1.2500 **1**
0.2500 x 2=0.5000 **0**
0.5000 x 2=1.0000 **1**



1 x 1/2 + 1x 1/4 + 0 x 1/8 + 1 x 1/16 = 13/16

0.1101
= 1.101 x 2^{-1}

CONVERSION EXAMPLE PART 2

$0.1101 = 1.101 \times 2^{-1}$

Sign: 0

Mantissa: 101

Exponent: -1 + 127 = 126(d)
= 01111110(b)

0 01111110 1010000 00000000 00000000

CONVERSION

0.1 x 2 = 0.2 **0**
0.2 x 2 = 0.4 **0**
0.4 x 2 = 0.8 **0**
0.8 x 2 = 1.6 **1**
0.6 x 2 = 1. 2 **1**
0.2 x 2 = 0.4 **0**

…… repeats …

Just like the 1/3, 0.1 keeps repeating in binary

0 01111011 1001100 11001100 1100110

123 ½ + $1/2^4$ + $1/2^5$ + $1/2^8$ + $1/2^9$ + $1/2^{12}$ + $1/2^{13}$ + $1/2^{16}$ + $1/2^{17}$ + $1/2^{20}$ + $1/2^{21}$

$(-1)^s \times (1 + m) \times 2^{\text{exponent} - \text{bias}}$

0.099999994039535522460937 5 = $(-1)^0 \times \left(1 + \frac{1258291}{2097152}\right) \times 2^{123 - 127}$

Not quite 0.1

EXAM QUESTION

Page 5: Floating Point and Bitwise Operations

8. [9 points] Write the following binary number as an 8-bit floating point number assuming a 4-bit exponent value.

```
+0.00001110011
```

Answer

EXAM QUESTION

CS 2130 Exam 1, Spring 2023 Page 5 of 6 UVA computing id:

Page 5: Floating Point and Bitwise Operations

8. [9 points] Write the following binary number as an 8-bit floating point number assuming a 4-bit exponent value.

```
+0.00001110011
```

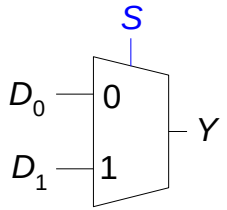
Answer

0 0010 110

TODAY'S LECTURE

MUXES

Example: **2:1 Mux**



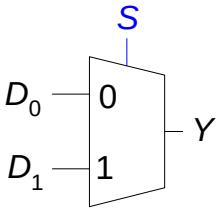
S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

S	Y
0	D_0
1	D_1

- Selects between one of *N* inputs to connect to output
- **Select** input is **$\log_2 N$ bits** – control input

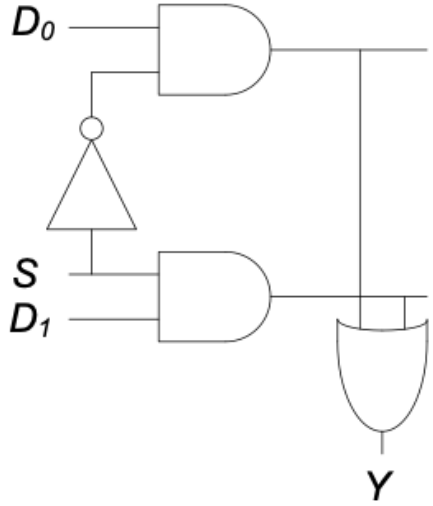
1 BIT MUX

$$Y = D_0\overline{S} + D_1S$$

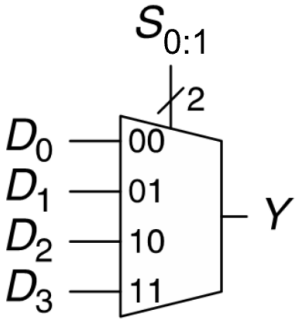


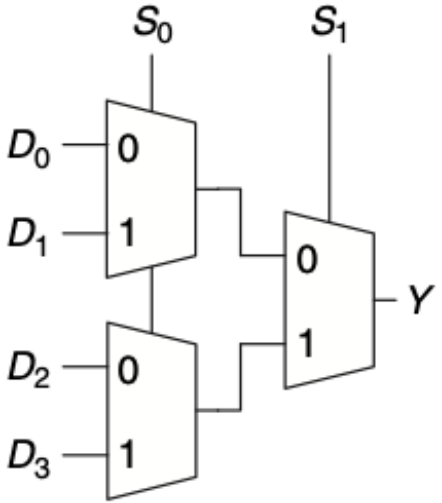
S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

S	Y
0	D_0
1	D_1

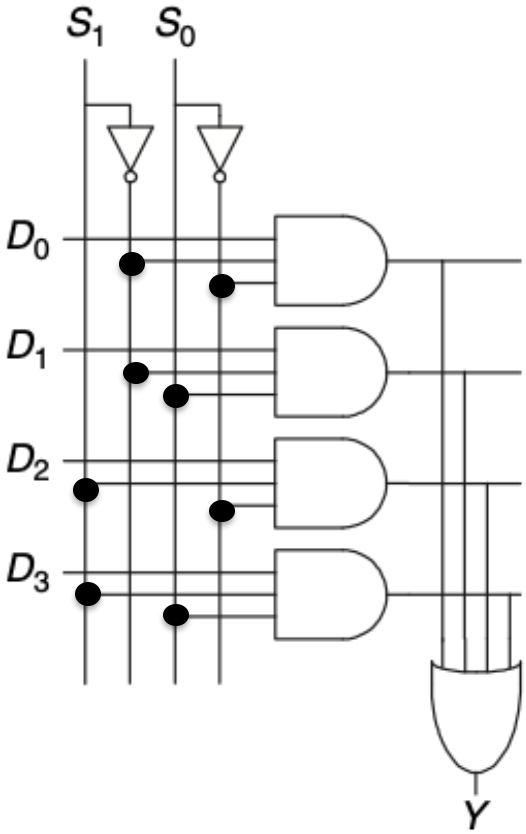


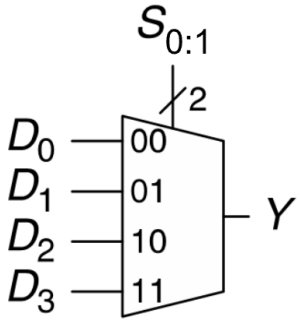
2 BIT MUX





2 BIT MUX



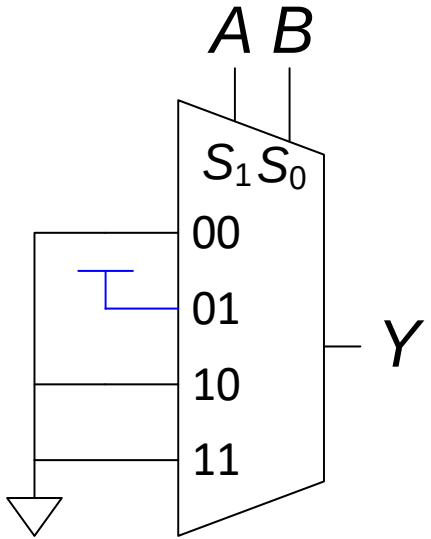


MUX AS A LOOK UP TABLE

Using mux as a **lookup table**

A	B	Y
0	0	0
0	1	1
1	0	0
1	1	0

$Y = AB$

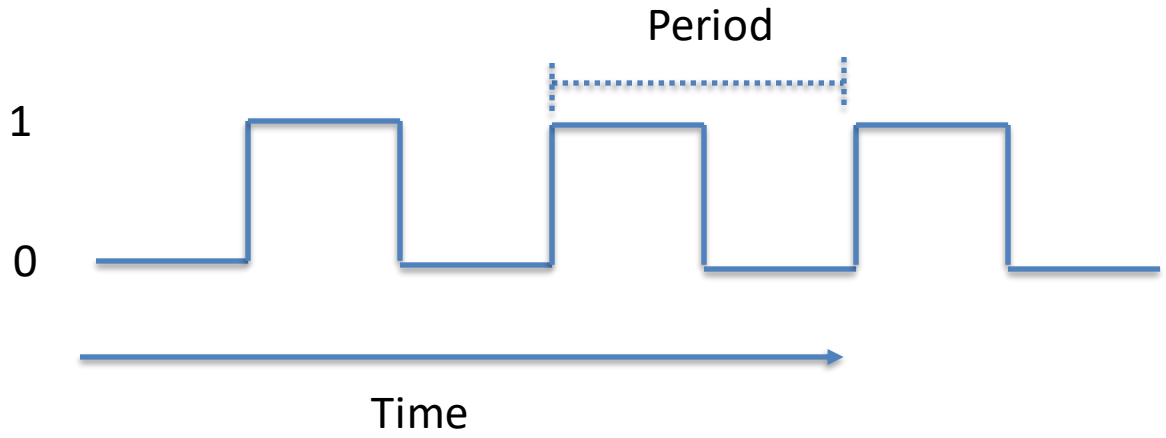


CLOCKS

A clock is something that produces a periodic signal

Period is length of time for one clock cycle

Example

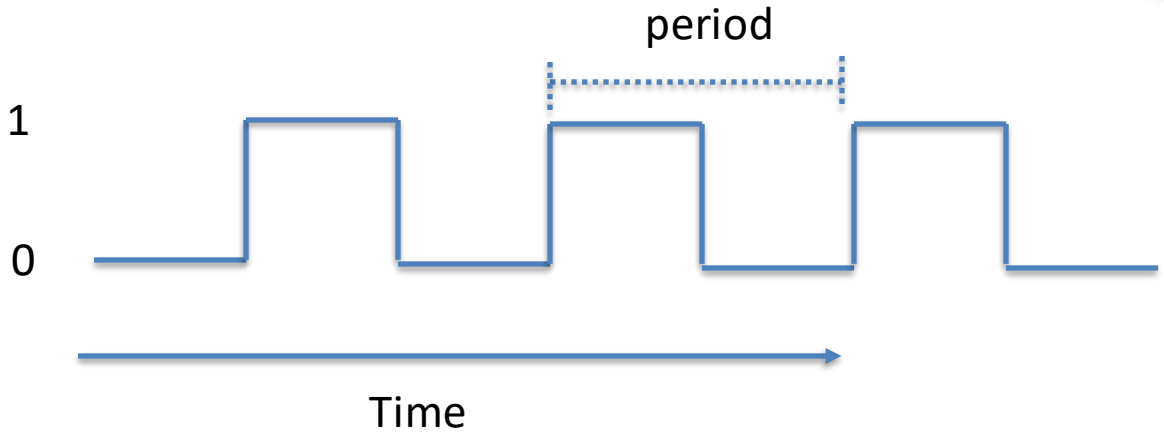


CLOCKS

Clock frequency Intel Core i-9 3.0GHz

Frequency = 1/ period

Period = 1/frequency

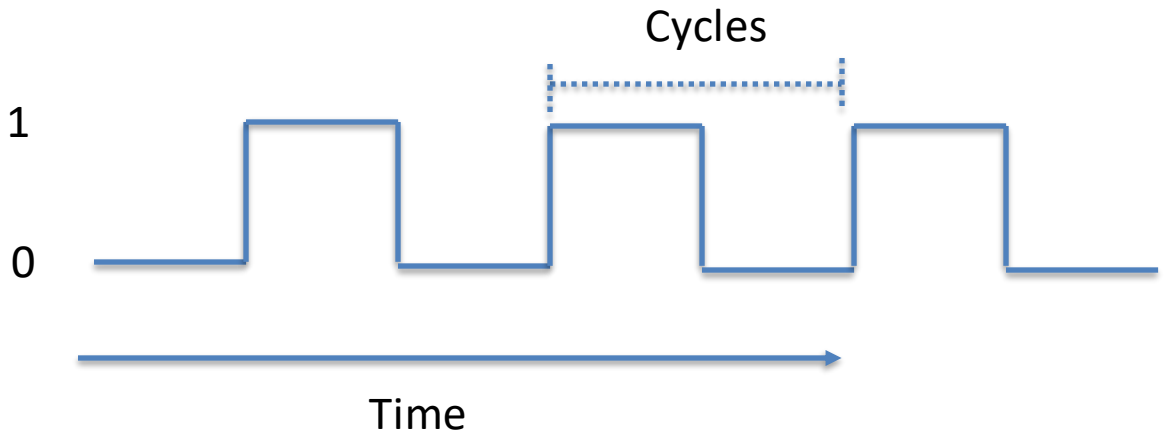


CLOCKS

Clock frequency Intel Core i-9

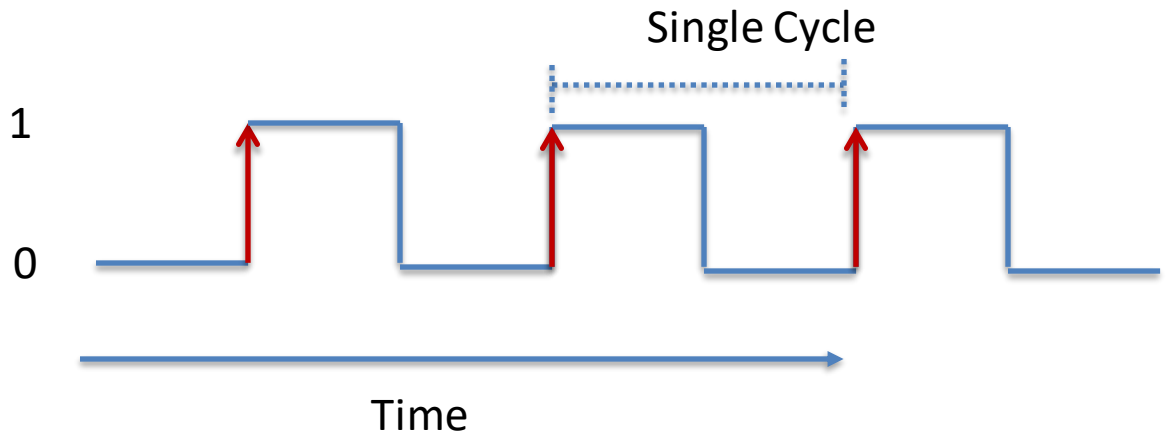
3.0GHz. = $3*10^9$ = 3,000,000,000 cycles per second

That is fast.



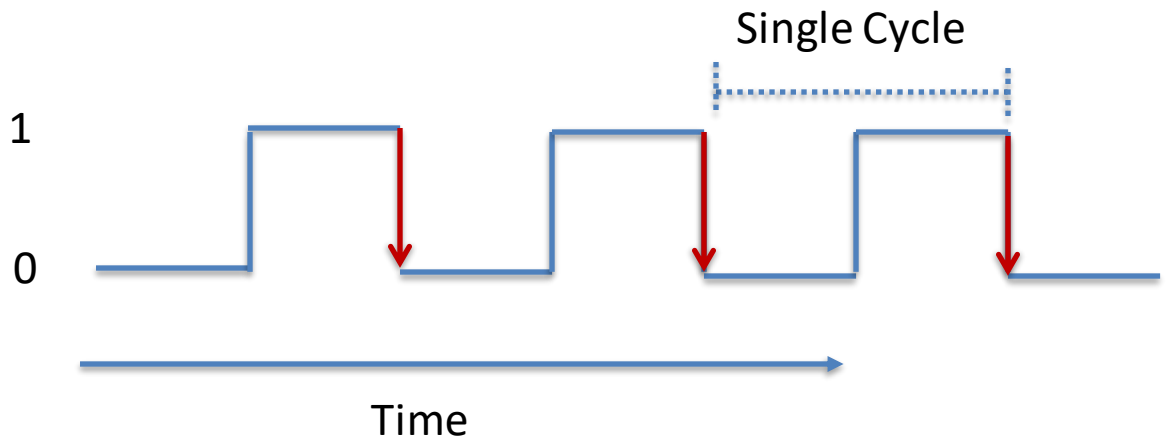
CLOCK EDGES

Rising Edge



CLOCK EDGES

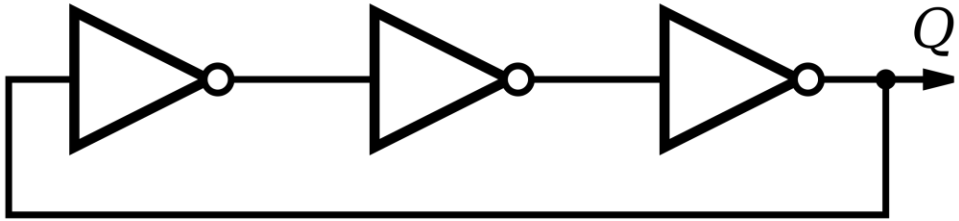
Falling Edge.



We will build a single cycle machine. It will complete all the computation in a single cycle.

USING RING OSCILLATORS TO GENERATE CLOCKS

A clock is something that produces a periodic signal



Let's walk through an example assume that Q starts off as 0

Frequency = 1/(2*t*n)

Where t is the time delay of an inverter and n is number of inverters