

CSE 210: Computer Architecture

Lecture 21: Floating Point

Stephen Checkoway

Oberlin College

Apr 15, 2022

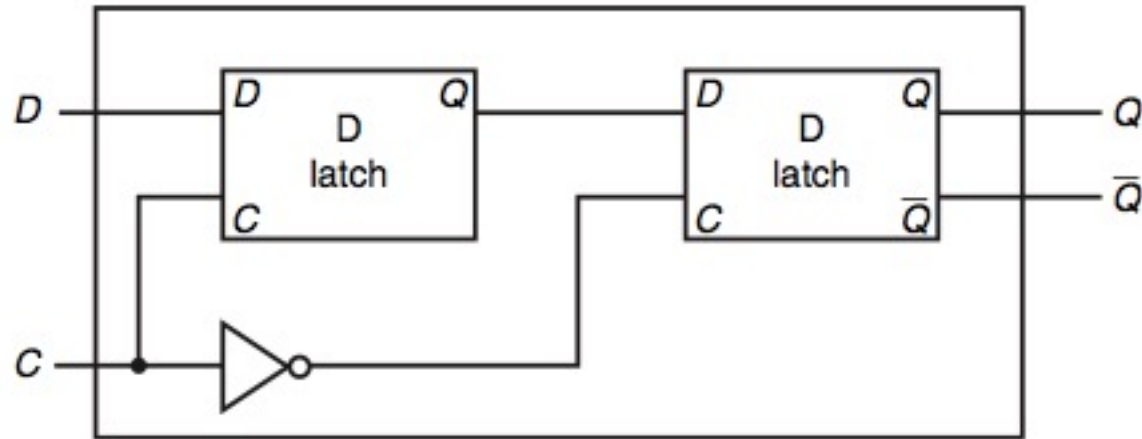
Slides from Cynthia Taylor

Announcements

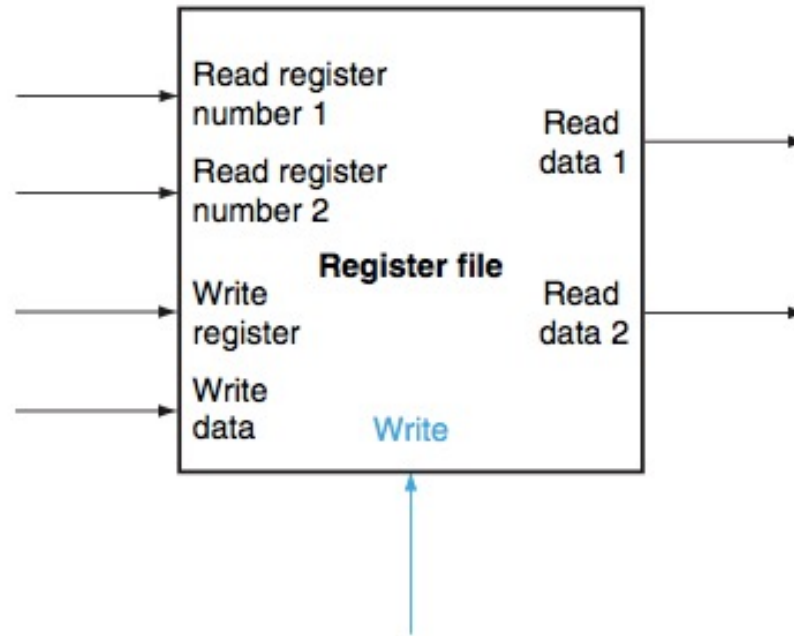
- Problem Set 6 due today
- Lab 5 due Sunday
- Office Hours today 13:30 – 14:30

Registers

- Each 32-bit register will consist of 32 1-bit D flip-flops

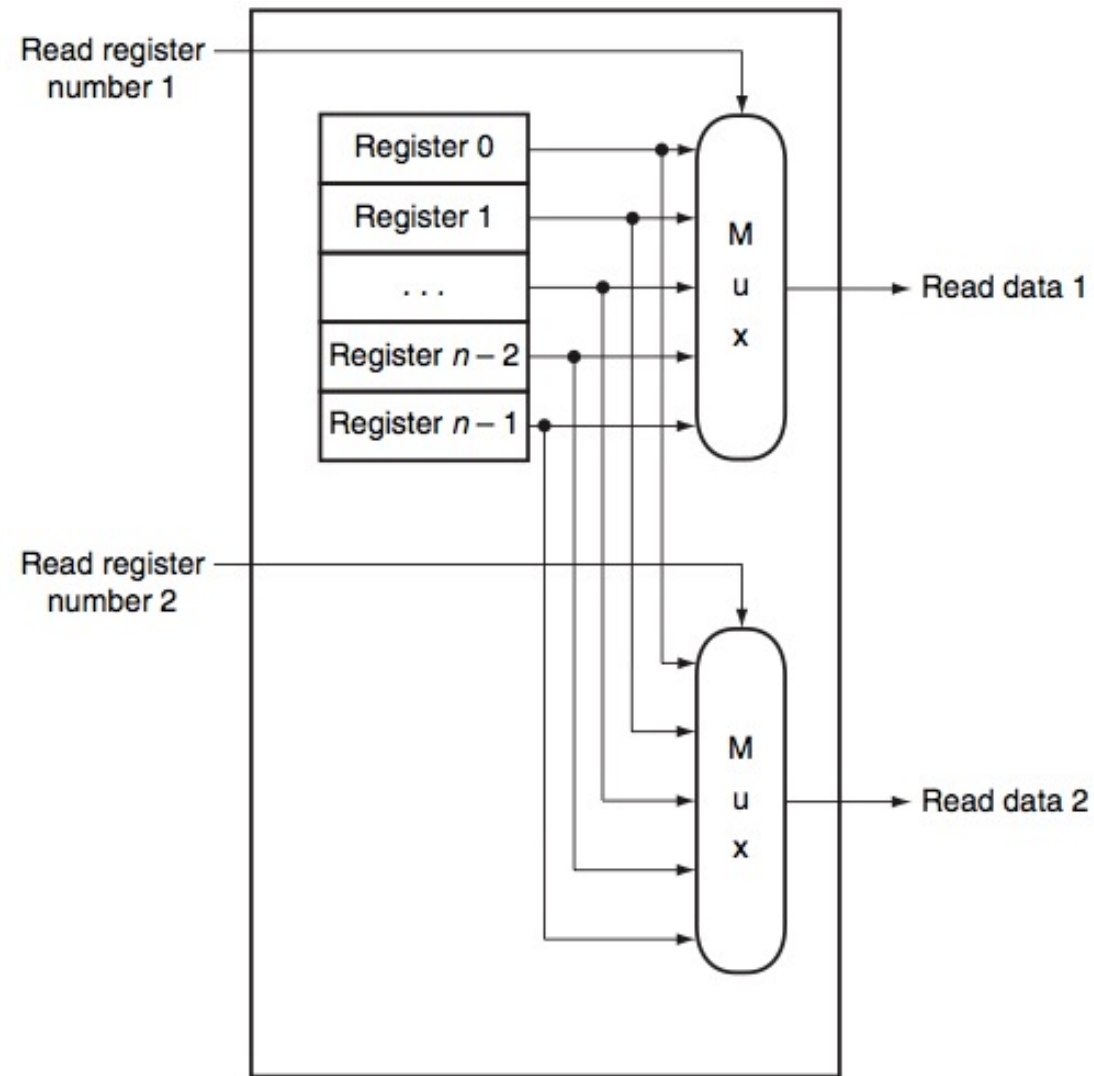


Register File

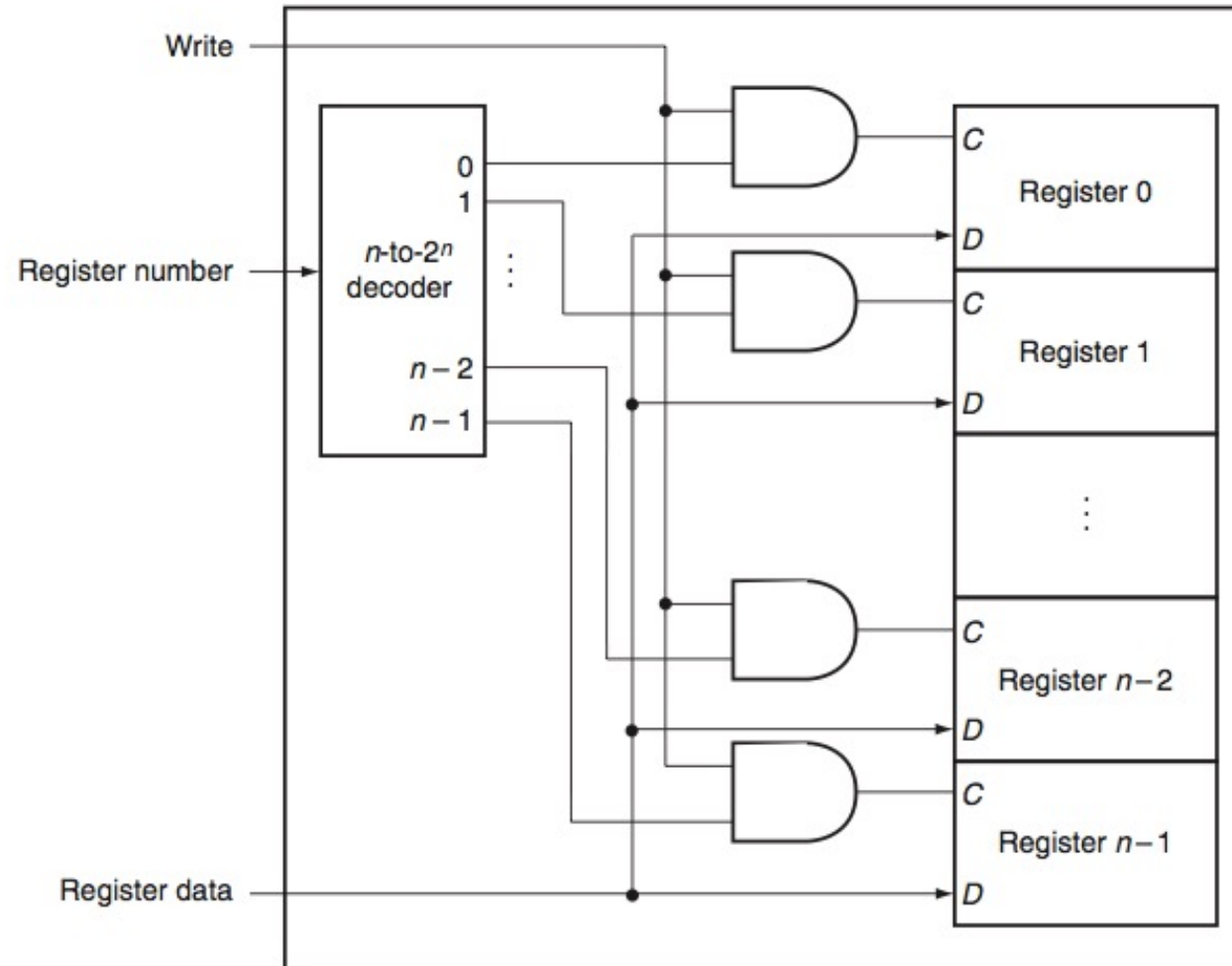


- Set of registers that can be written/read by supplying a register number

Read Function

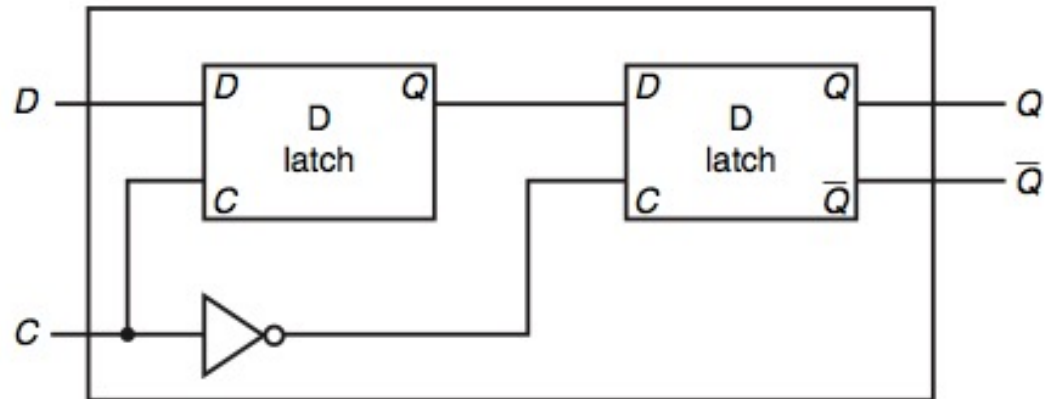


Write Function



What will happen if we read and write to a register in the same clock cycle?

- A. The read will get the previous value
- B. The read will get the just written value
- C. It is ambiguous
- D. None of the above



Register Questions?

Floating Point

- Problem: Need a way to store non-integer values
- Including numbers with very large and very small magnitudes

How Humans Do This

- Scientific Notation
 - $1.2825 * 10^2$
 - $2.004 * 10^{38}$
 - $3.74 * 10^{-27}$
 - $-7.888889 * 10^{40}$
- Normalized Form
 - Always multiply by power of 10
 - Always 1 digit before the decimal point

How Computers Do This

- Floating Point Notation
 - $1.11_2 \times 2^2$
 - $1.0101_2 \times 2^{127}$
 - $1.110001_2 \times 2^{-126}$
 - $-1.0001_2 \times 2^{80}$
- Normalized Form
 - One digit before ~~decimal~~ binary point
 - Multiplied by power of two

$$101.10001_2$$

- $101.10001_2 = 2^2 + 2^0 + 2^{-1} + 2^{-5}$
- Integer part is $101_2 = 4 + 1 = 5$
- Fractional part is $0.10001_2 = 1/2 + 1/2^5 = 0.503125$
- Total is 5.503125

We know $101.10001_2 = 5.503125$. What is
 $1.0110001_2 \times 2^2$

A. 1.37578125

B. 5.503125

C. 22.0125

D. None of the above

-17.125 in binary

- Step 1. Convert integer part: $17 = 10001_2$
- Step 2. Convert fractional part: $.125 = 1/8 = 0.001_2$
- Step 3. Add integer and fractional parts: $17.125 = 10001.001_2$
- Step 4. Normalize: $10001.001_2 = 1.0001001_2 \times 2^4$
- Step 5. Add sign: $-17.125 = -1.0001001_2 \times 2^4$

−0.75 in Binary is

A. $-1.1_2 \times 2^{-1}$

B. $-1.1_2 \times 2^{-2}$

C. $-1.001011_2 \times 2^{-1}$

D. $-1.001011_2 \times 2^{-2}$

E. None of the above

1.2825 * 10² in Binary is

- A. $1.000000001_2 \times 2^{-7}$
- B. $1.000000001_2 \times 2^6$
- C. $1.1001000011001_2 \times 2^6$
- D. $1.000000001_2 \times 2^7$
- E. None of the above

Want to Represent $(-1)^s * 1.x * 2^e$ in 32 bits

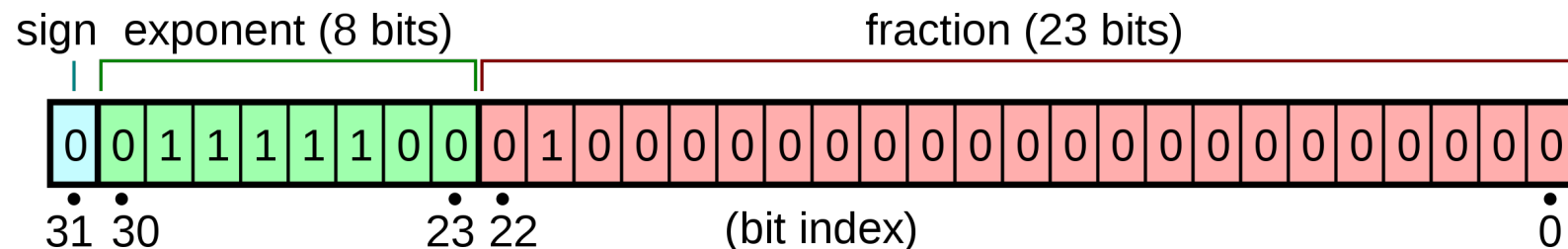
- Divide up 32 bits into different sections
- 1 bit for sign s (1 = negative, 0 = nonnegative)
- 8 bits for exponent e
- 23 bits for significand $1.x$

Goal: Get the most out of 32 bits

- The first number before our ~~decimal~~ binary point is always 1
 - $1.0001 * 2^4$
 - $-1.1011 * 2^{-16}$
- We don't need to represent it in our remaining 23 bits—it is implicit!

$$(-1)^s * 1.x * 2^e$$

- 1 bit for sign s (1 = negative, 0 = positive)
- 8 bits for exponent e
- 0 bits for implicit leading 1 (called the “hidden bit”)
- 23 bits for significand (without hidden bit)/fraction/mantissa x



$1.001100101 * 2^7$ as a single word

- $1.001100101 * 2^7$ as a single word becomes
 - Sign = 0 (positive)
 - Exponent = 00000111
 - Significand = 001100101000000000000000

If we gave more bits to the exponent, and fewer to the fraction, we could represent

- A. Fewer individual numbers
- B. More individual numbers
- C. Numbers with greater magnitude, but less precision
- D. Numbers with smaller magnitude, but greater precision

Want To Make Comparisons Easy

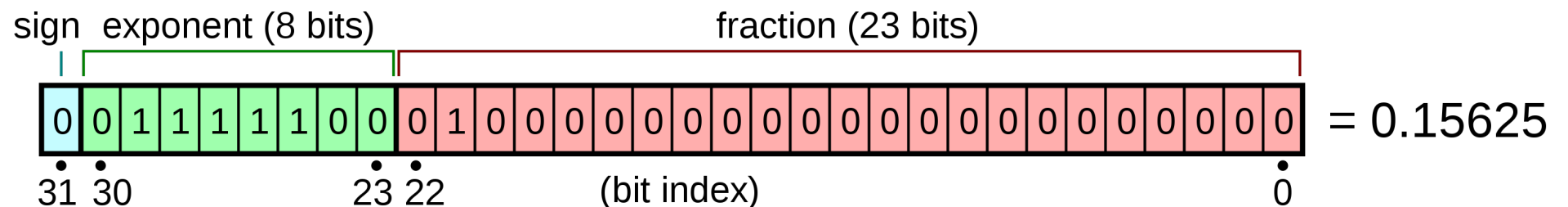
- Can easily tell if number is positive or negative
 - Just check MSB bit
- Exponent is in higher magnitude bits than the fraction
 - Numbers with higher values will look bigger
 - 0 00000111 10000000000000000000000000000000 = $1.1 * 2^7$
 - 0 00001000 10000000000000000000000000000000 = $1.1 * 2^8$

Problem with Two's Complement

- 0 00000111 100000000000000000000000000000 = $1.1 * 2^7$
- 0 00001000 100000000000000000000000000000 = $1.1 * 2^8$
- 0 11111000 100000000000000000000000000000 = $1.1 * 2^{-8}$
- Solution: Get rid of negative exponents!
 - We can represent $2^8 = 256$ numbers: normal exponents -126 to 127 and two special values for zero, infinity, (and NaN and subnormals)
 - Add 127 to value of exponent to encode it, subtract 127 to decode

$$(-1)^s * 1.x * 2^e$$

- 1 bit for sign s (1 = negative, 0 = positive)
- 8 bits for exponent $e + 127$
- 0 bits for implicit leading 1 (called the “hidden bit”)
- 23 bits for significand (without hidden bit)/fraction/~~mantissa~~ x



Encode 1.000000001×2^7 in 32-bit Floating Point

- A. 0 00000111 000000001000000000000000
- B. 0 00000111 100000000100000000000000
- C. 0 10000110 000000001000000000000000
- D. 0 10000110 100000000100000000000000
- E. None of the above

Reading

- Next lecture: Floating Point
- Problem Set 6 due today
- Lab 5 due Sunday