COMPUTER SYSTEMS ARCHITECTURE - LECTURE 0x05

INTEGER MULTIPLICATION/DIVISION, FLOATING-POINT REPRESENTATION

Cristian Rusu, Bogdan Macovei

LAST TIME

- **details of a binary addition circuit**
- **multiplexing**
- **sequential circuits**
  - SR Latch
  - D Latch
  - D Flip Flop
  - sequential addition circuit
- **state representation of a circuit**

OUTLINE

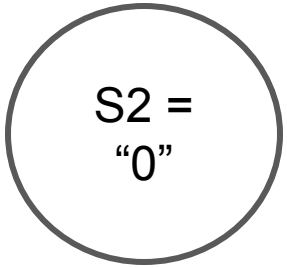
- sequential + combinational logic, an example
- multiplication of binary integers
- division of binary integers
- floating-point number representation
- working with floating-point numbers

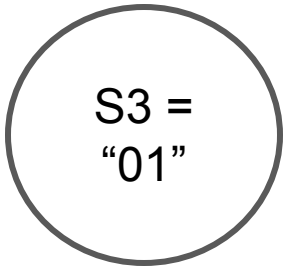
SEQUENTIAL, SEMINAR 0X02, EX. 10

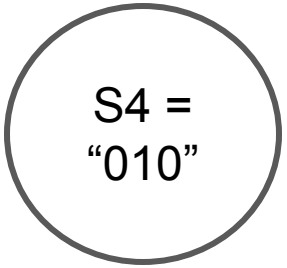
- a digital signal A can take values {0, 1} over time, and we want to detect whether the signal has the value 010 at some point. If this bit sequence is detected in A, then a variable Y is set to 1; otherwise this variable is 0.
- we define 4 states

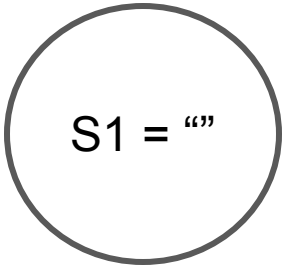
SEQUENTIAL, SEMINAR 0X02, EX. 10

- **a digital signal A can take values {0, 1} over time, and we want to detect whether the signal has the value 010 at some point. If this bit sequence is detected in A, then a variable Y is set to 1; otherwise this variable is 0**
- **we define 4 states**



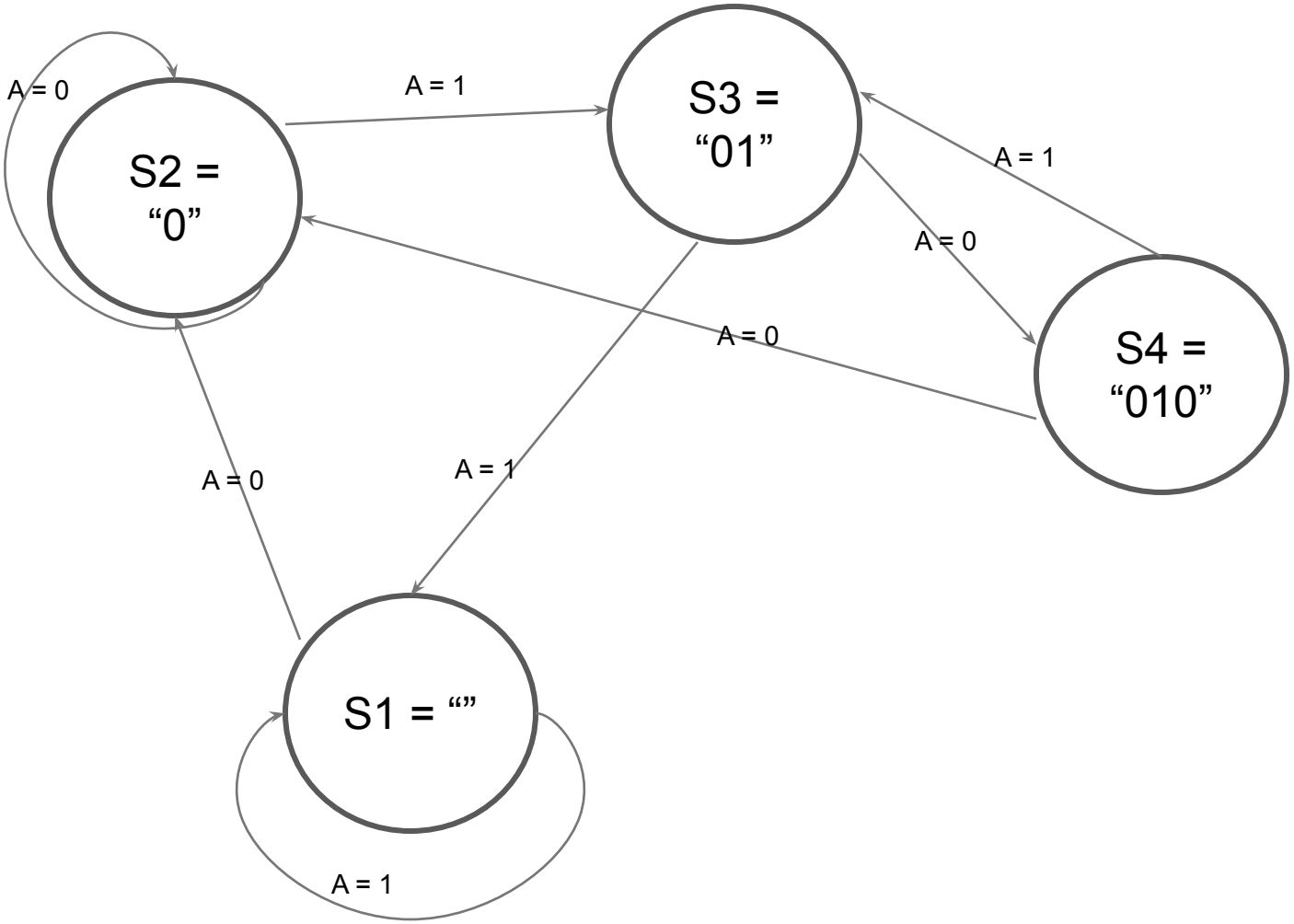




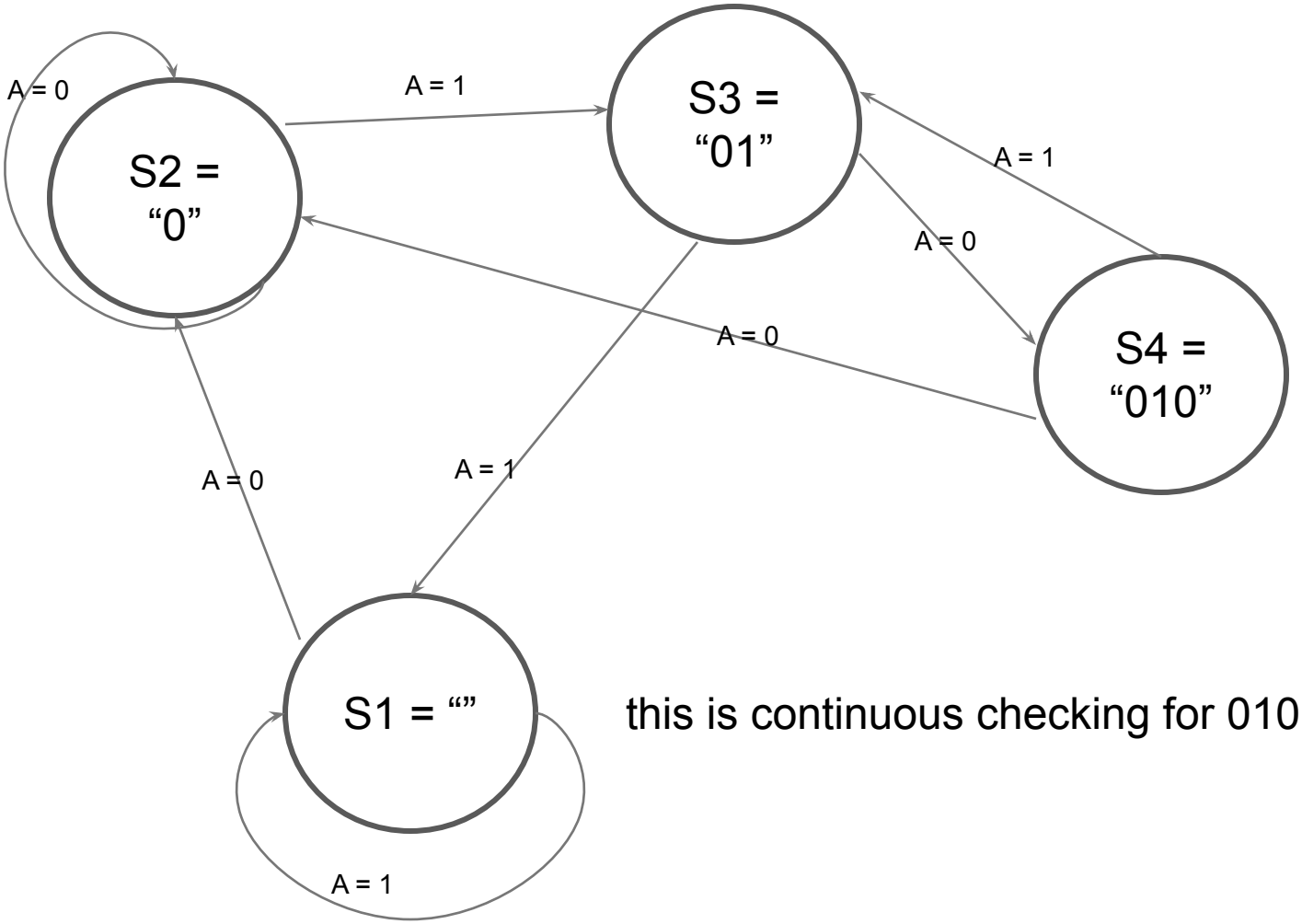


what are the transitions between these states?

SEQUENTIAL, SEMINAR 0X02, EX. 10

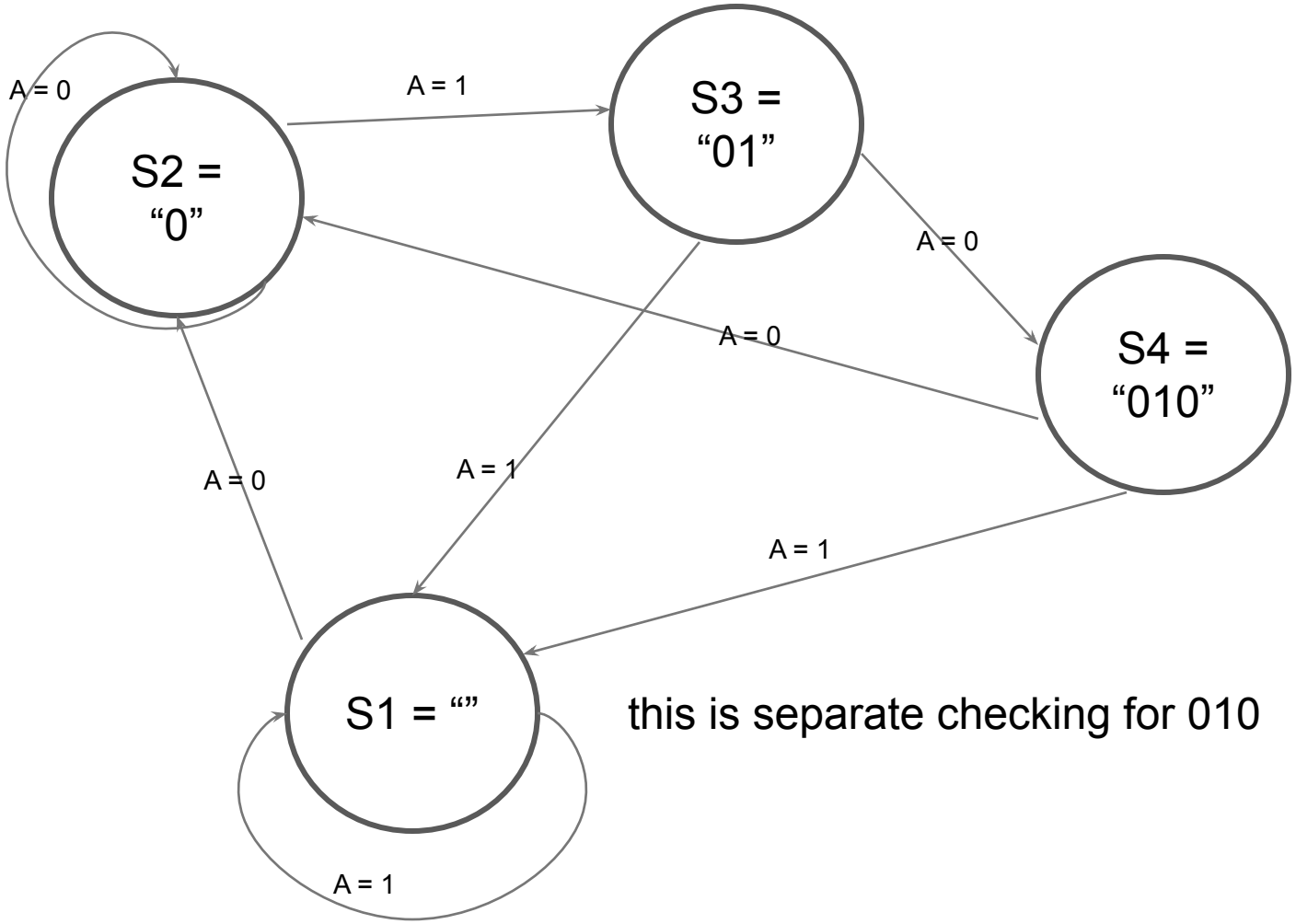


SEQUENTIAL, SEMINAR 0X02, EX. 10



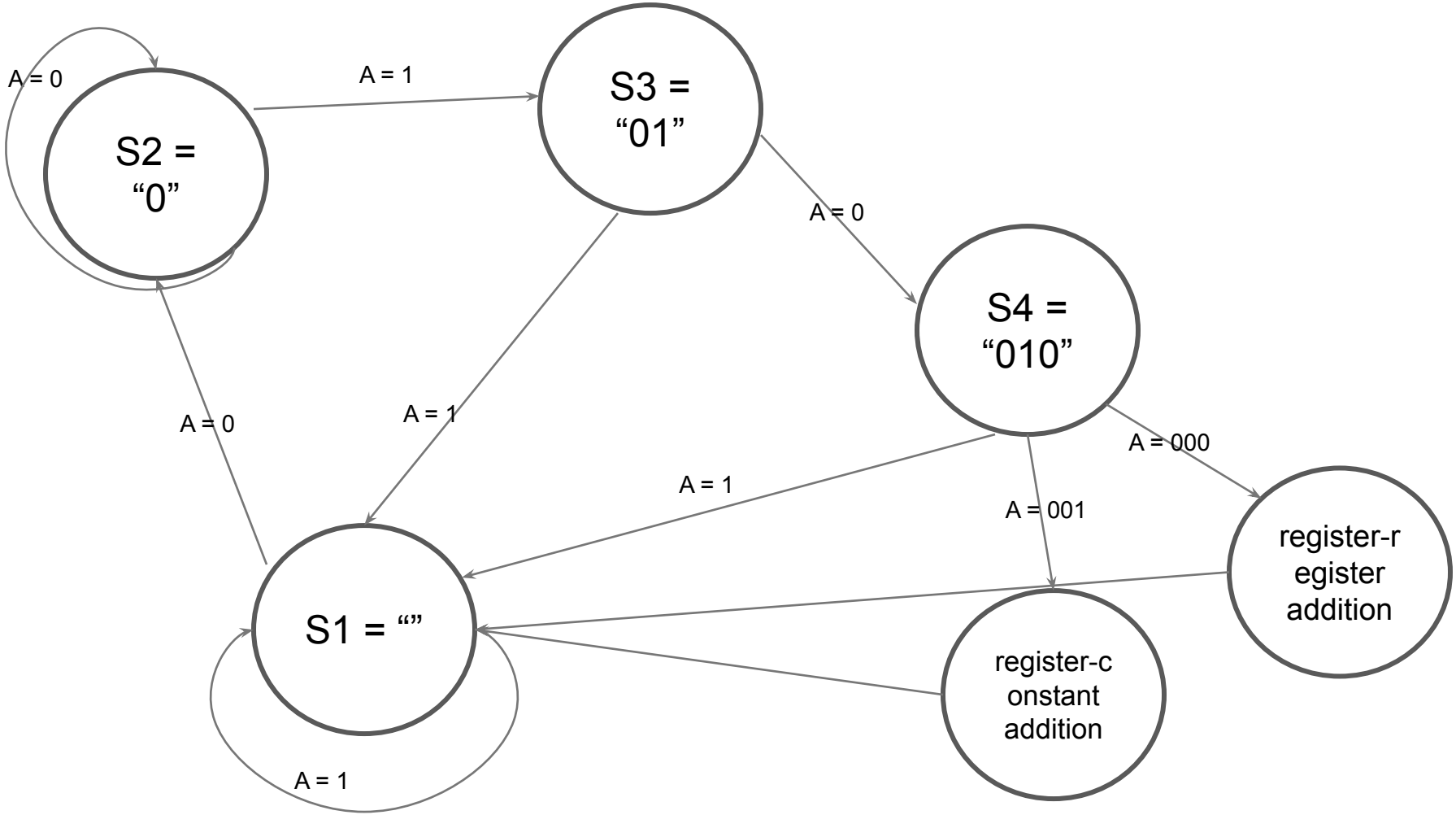
this is continuous checking for 010

SEQUENTIAL, SEMINAR 0X02, EX. 10



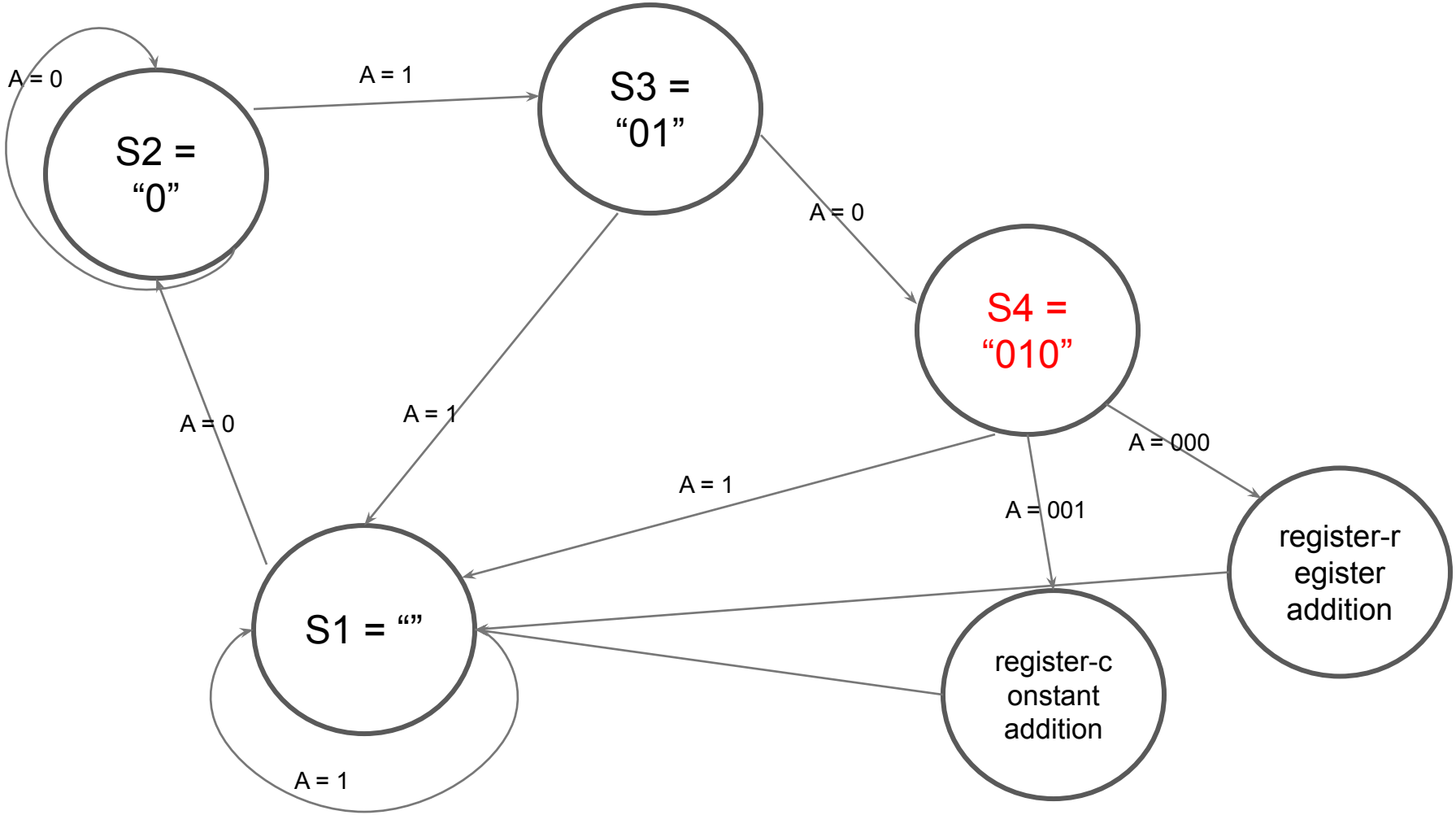
this is separate checking for 010

WHAT A PROCESSOR DOES



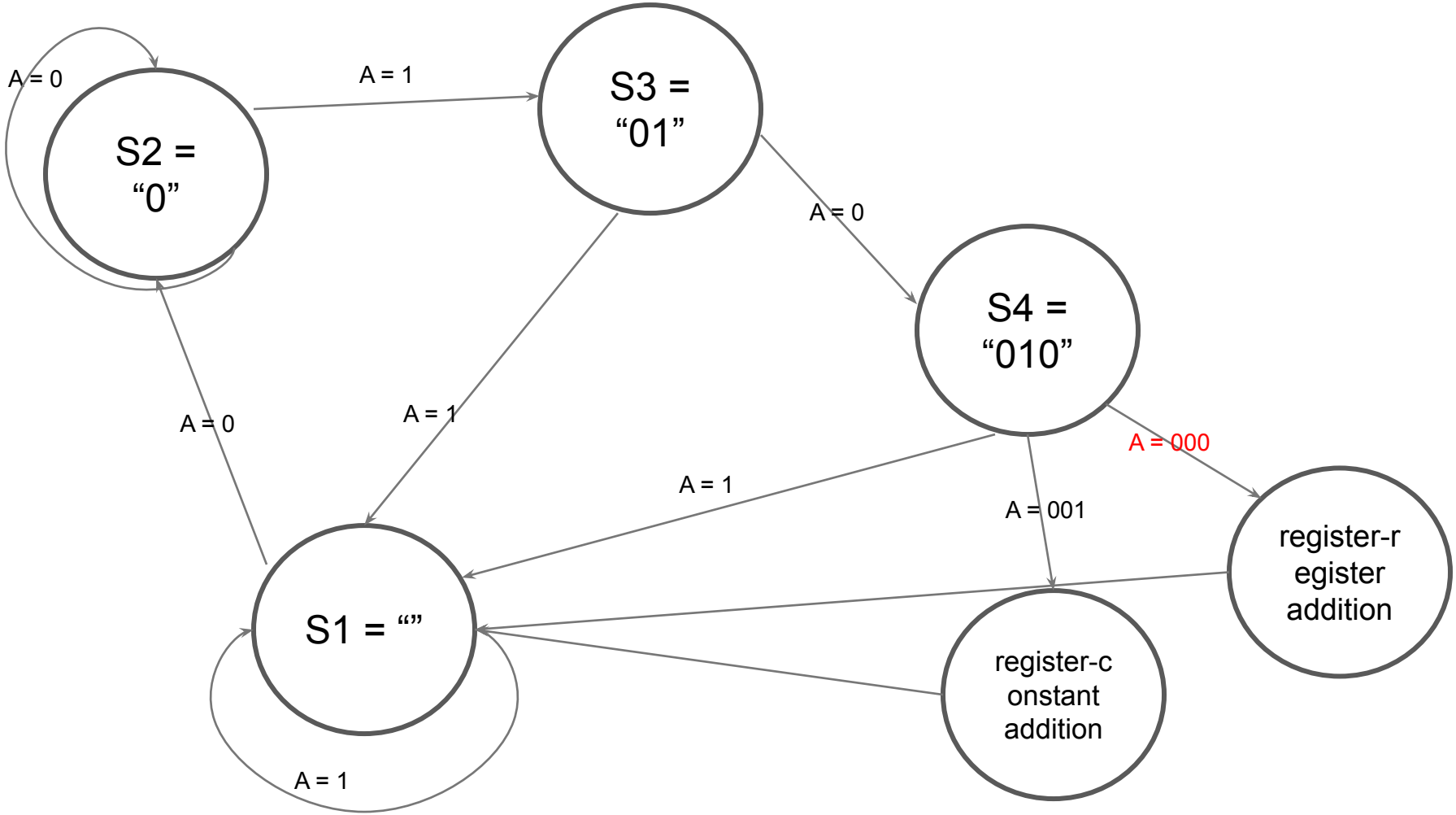
...0100001011010001010000100011001011 ...

WHAT A PROCESSOR DOES



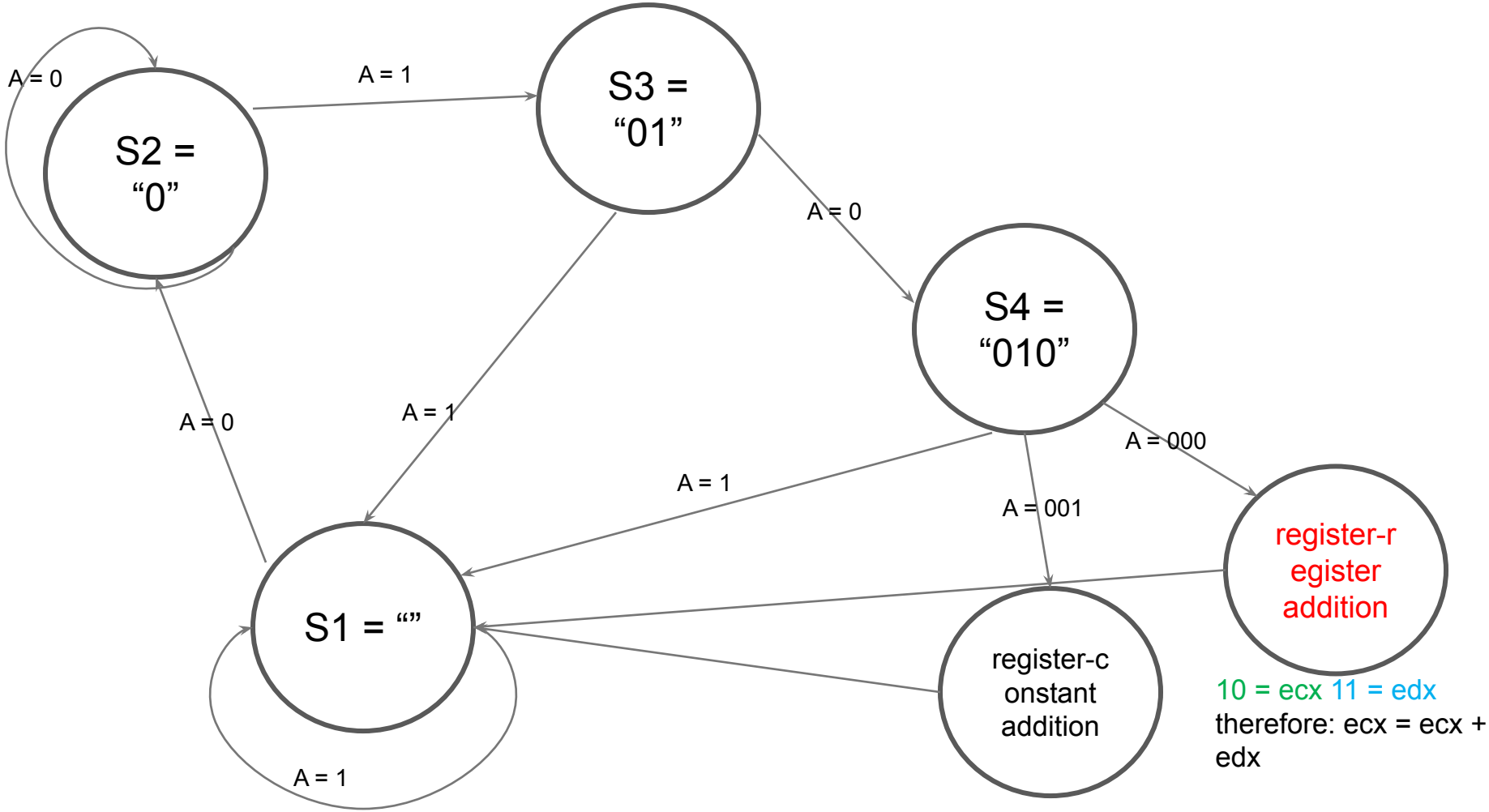
...010000101101000101000010001100101 1 ...

WHAT A PROCESSOR DOES



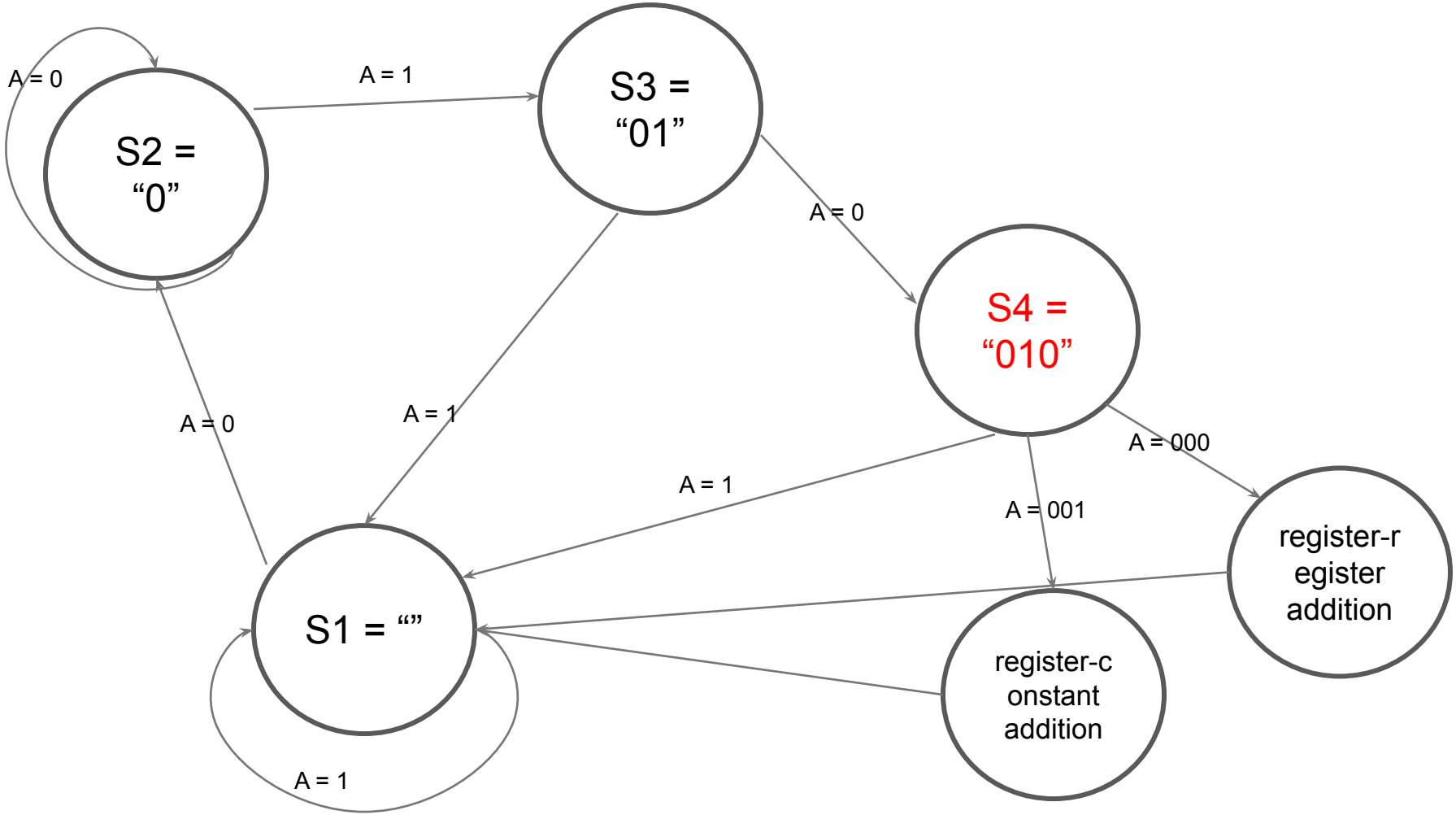
...010000101101000101000010001100101 1 ...

WHAT A PROCESSOR DOES



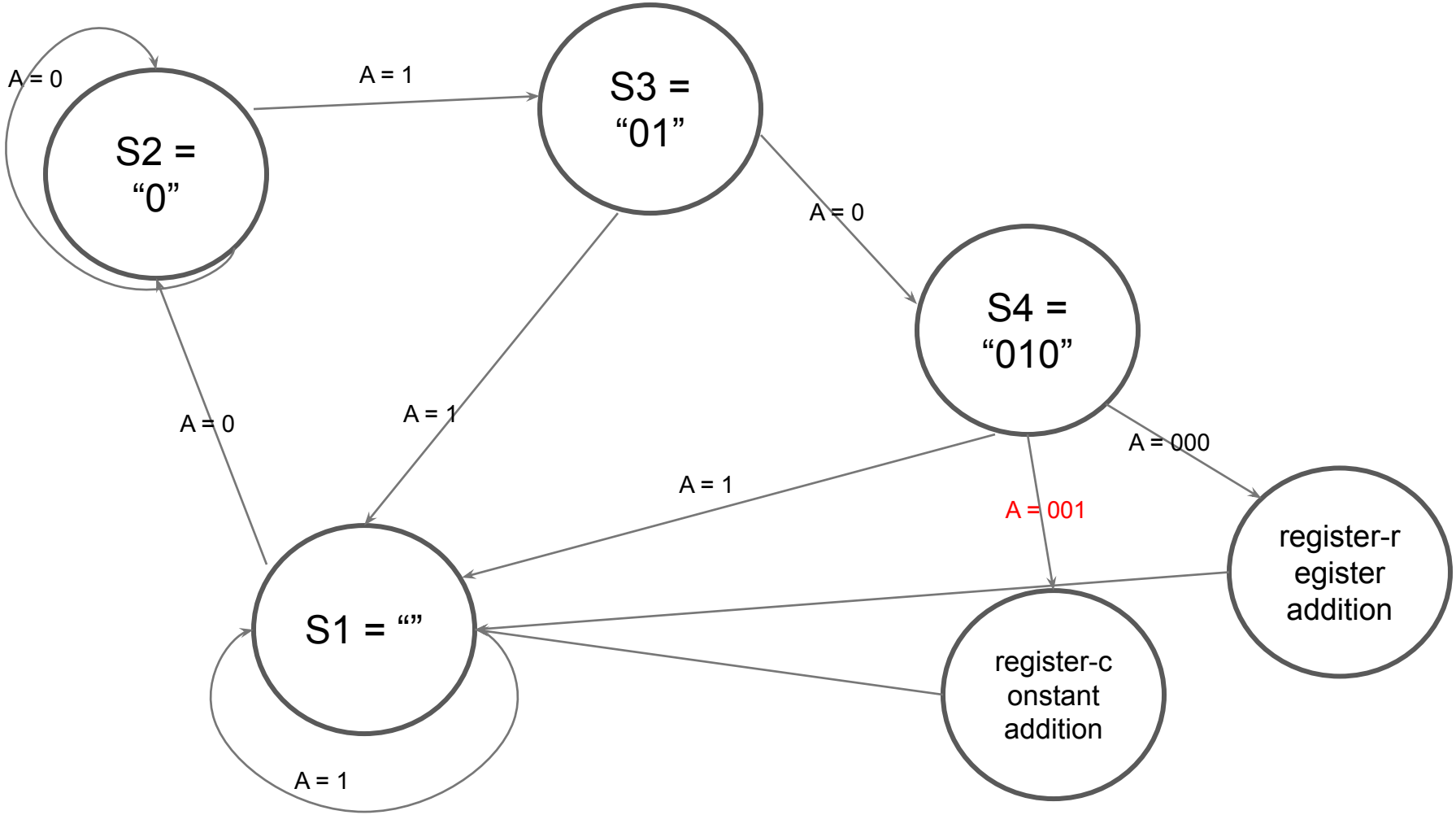
...01000010110100010100001001100 1011 ...

WHAT A PROCESSOR DOES



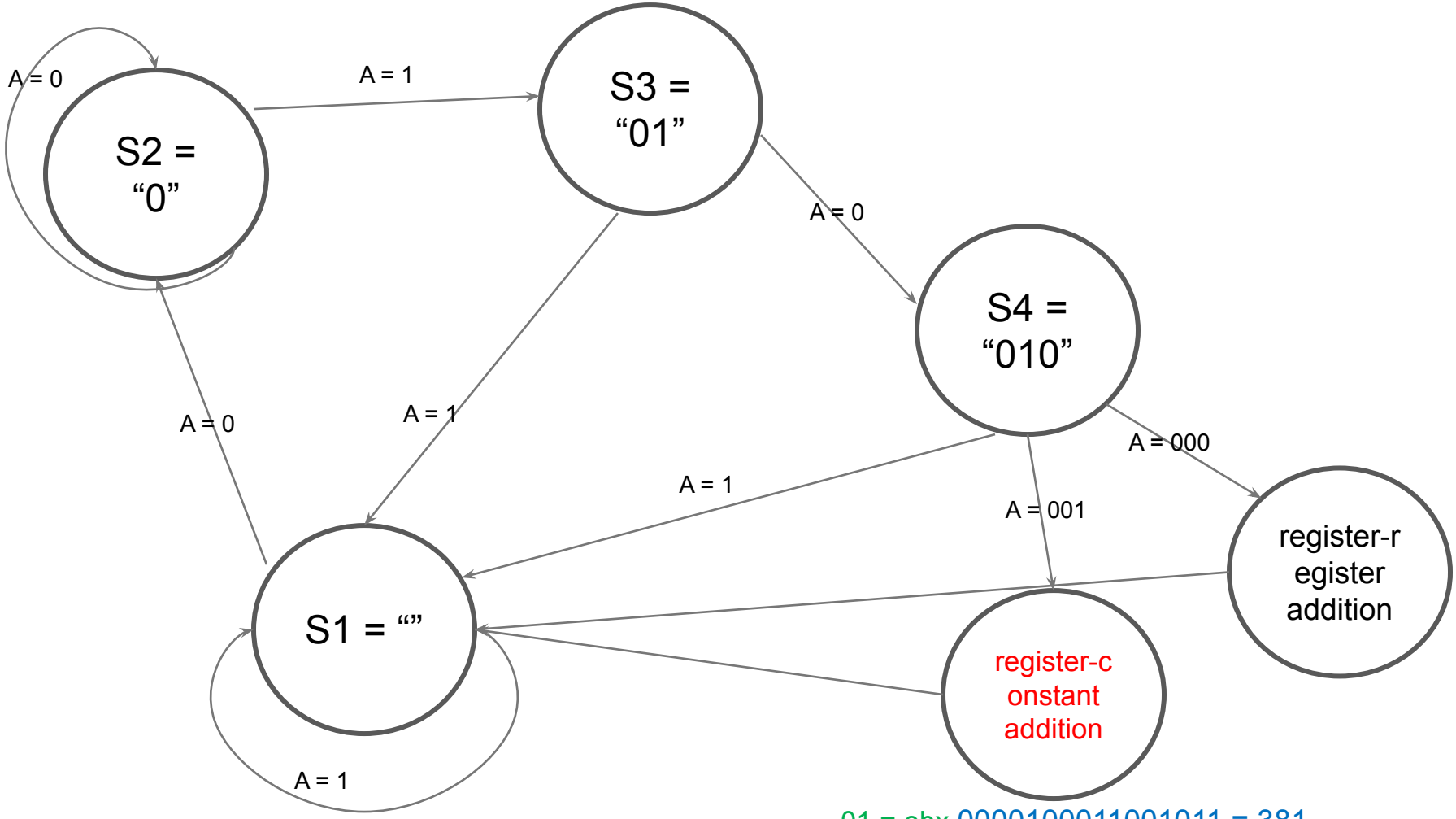
...0100001011010001010000100011001011 ...

WHAT A PROCESSOR DOES



...0100001011010001010000100011001011 ...

WHAT A PROCESSOR DOES

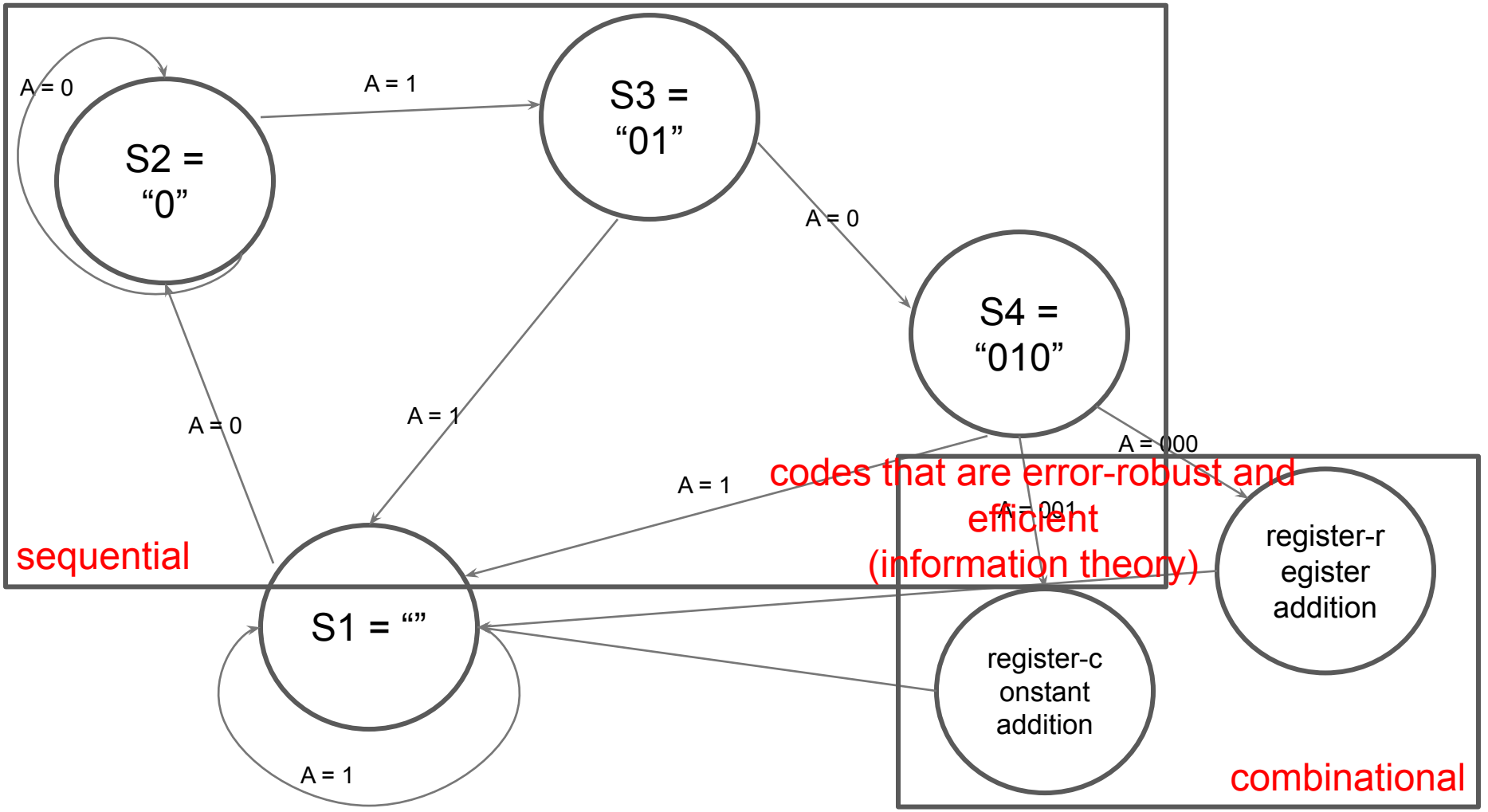


01 = ebx 0000100011001011 = 381

therefore: ebx = ebx + 381

...01000010110100010100001000110010 11 ...

WHAT A PROCESSOR DOES



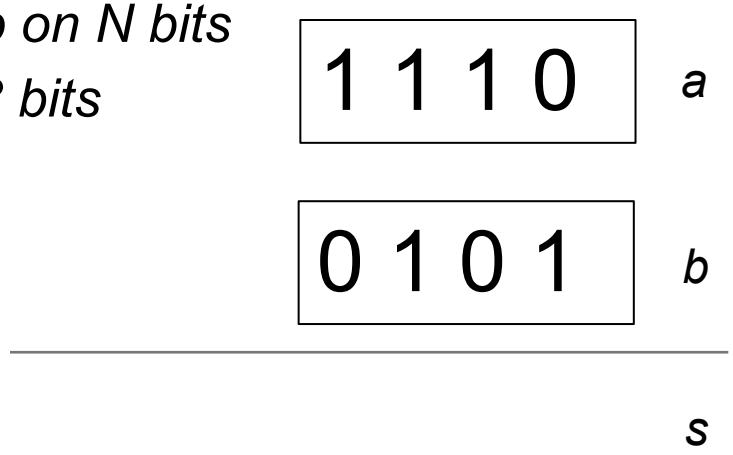
machine code ...0100001011010001010000100011001011 ...

NEW LECTURE CONTENT

- multiplication of binary integers
- division of binary integers
- floating-point number representation
- working with floating-point numbers

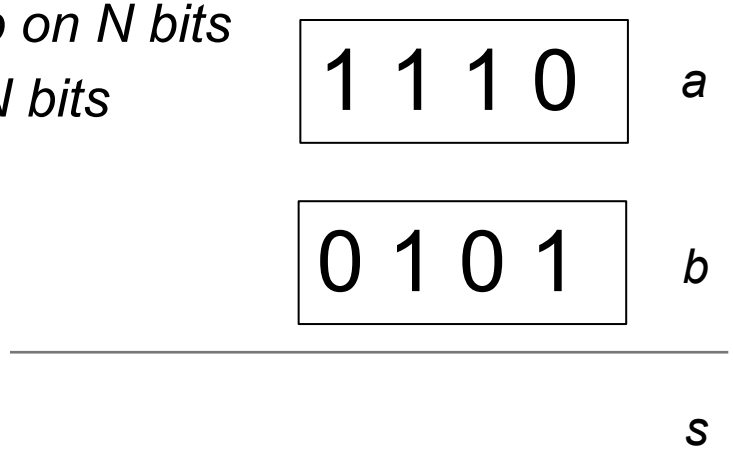
INTEGER MULTIPLICATION

- **example, s = a x b**
  - *a and b on N bits*
  - *s on ?? bits*



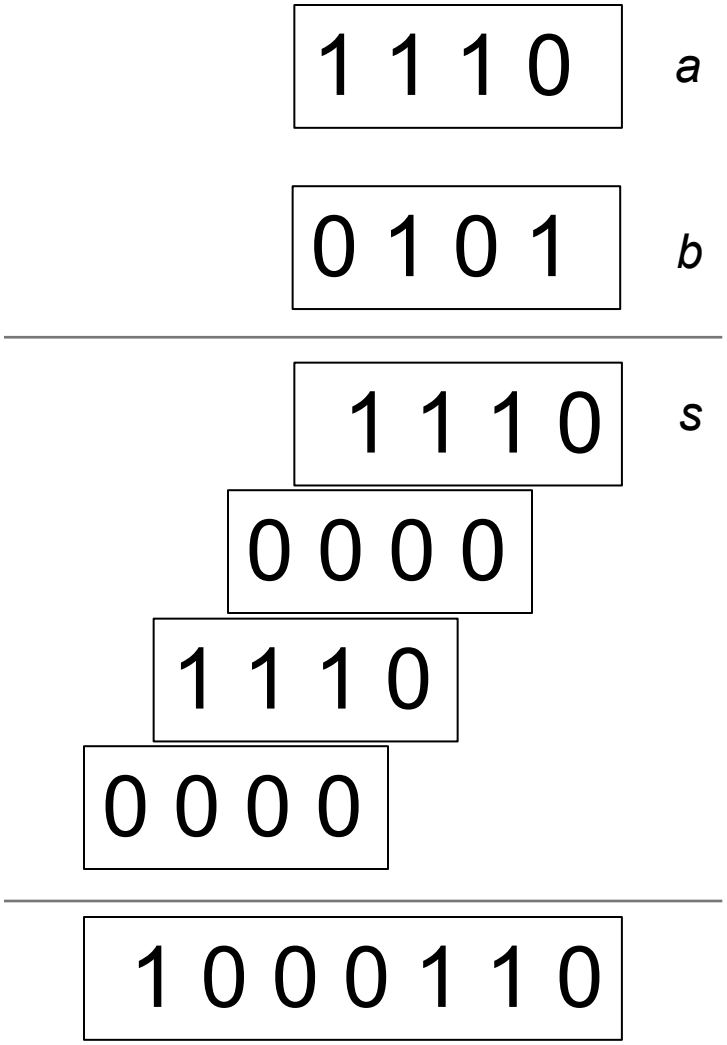
INTEGER MULTIPLICATION

- **example, s = a x b**
  - *a and b on N bits*
  - *s on 2N bits*



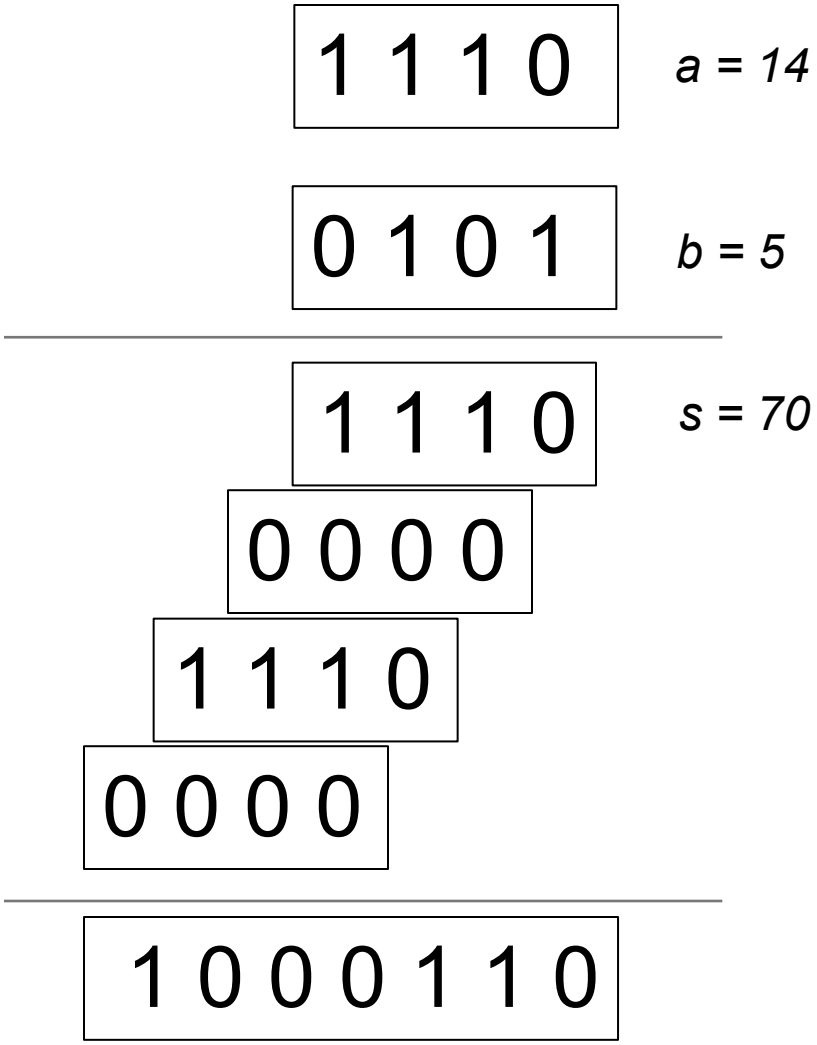
INTEGER MULTIPLICATION

- **example, s = a x b**



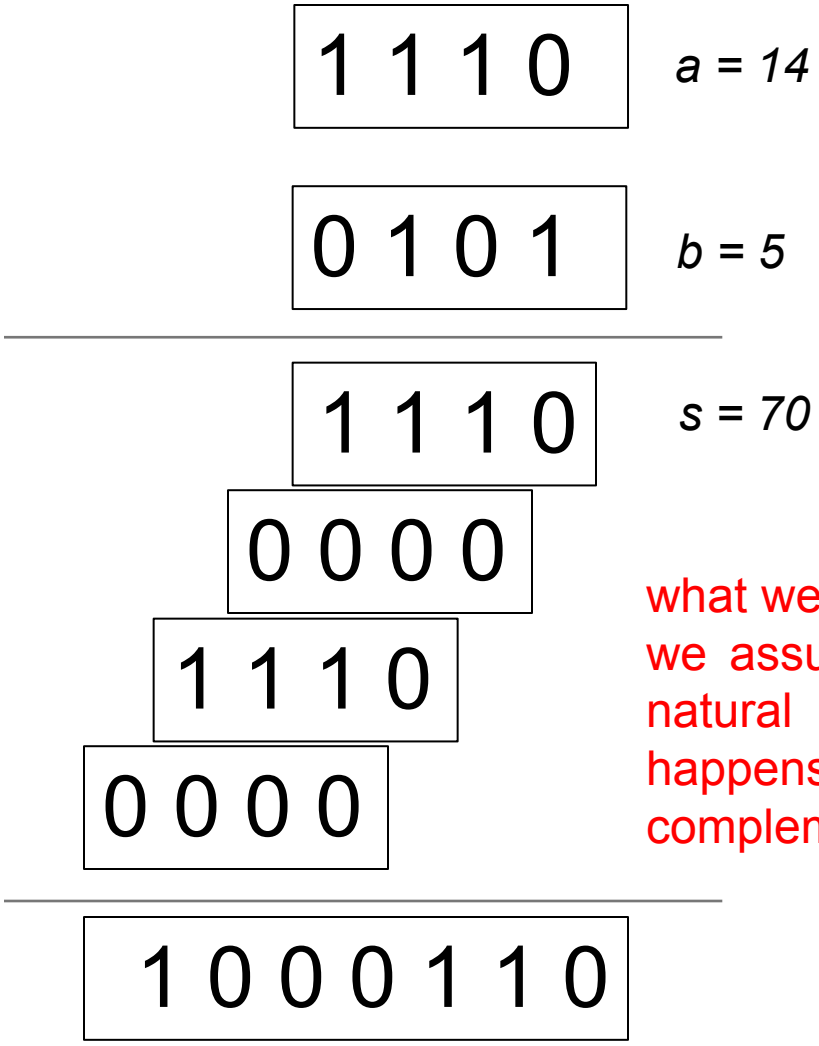
INTEGER MULTIPLICATION

- **example, s = a x b**



INTEGER MULTIPLICATION

- **example, s = a x b**



what we did here is correct, but we assumed that we received natural numbers. what happens if a and b are in two's complement?

INTEGER MULTIPLICATION

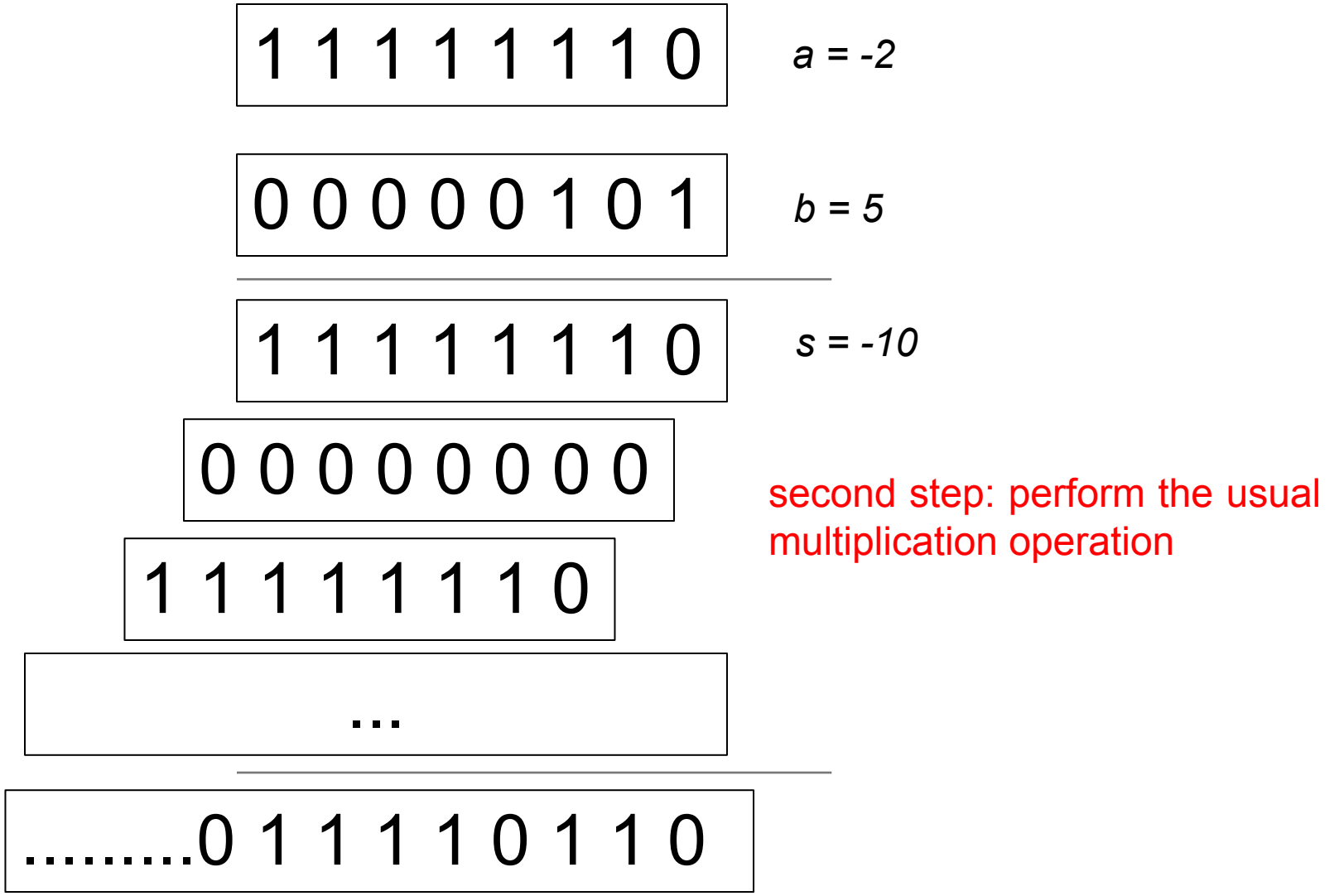
- **example, s = a x b**

1 1 1 0	*a = -2*
0 1 0 1	*b = 5*
	s = -10

first step: extend the operands to 8 bits

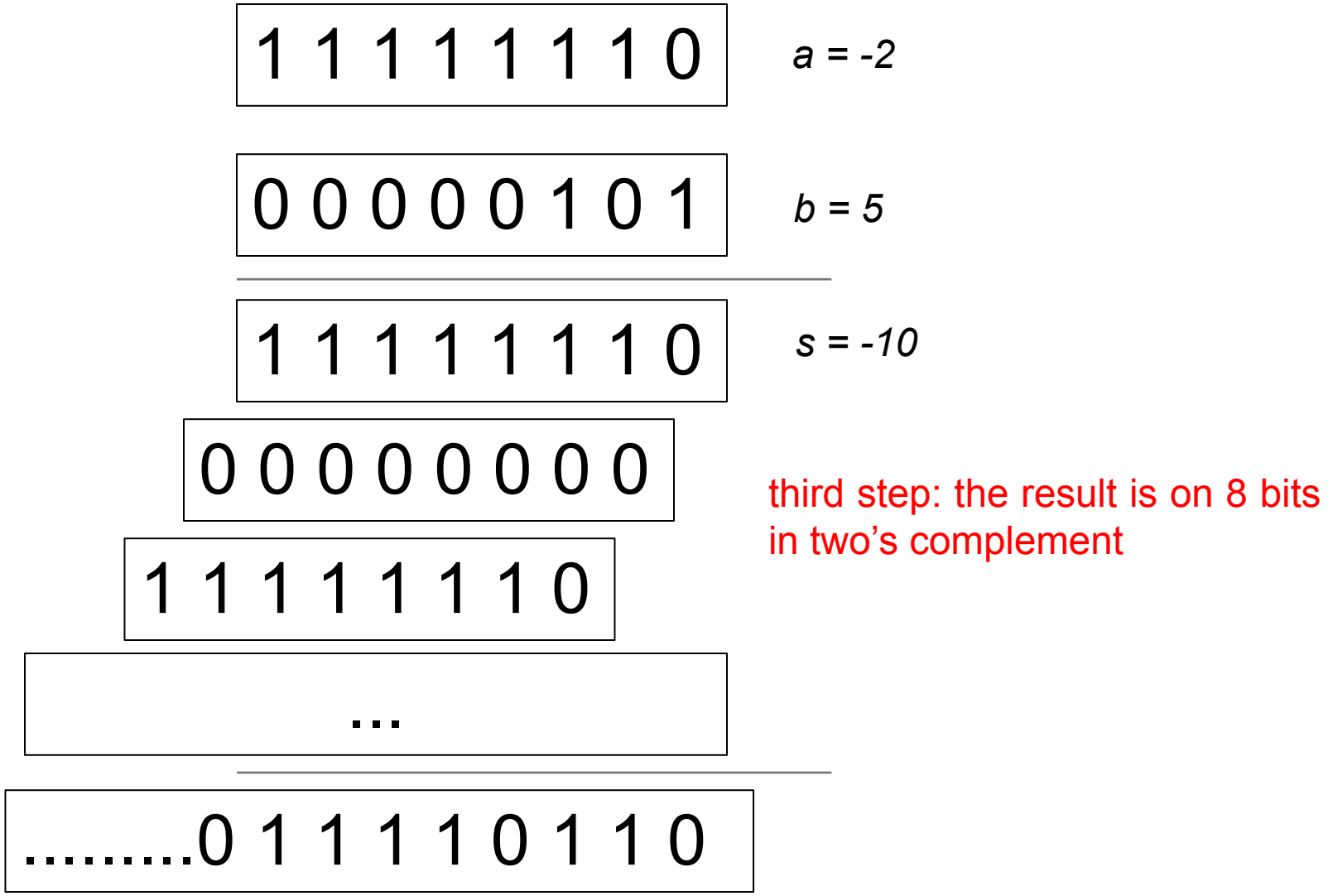
INTEGER MULTIPLICATION

- **example, s = a x b**



INTEGER MULTIPLICATION

- **example, s = a x b**

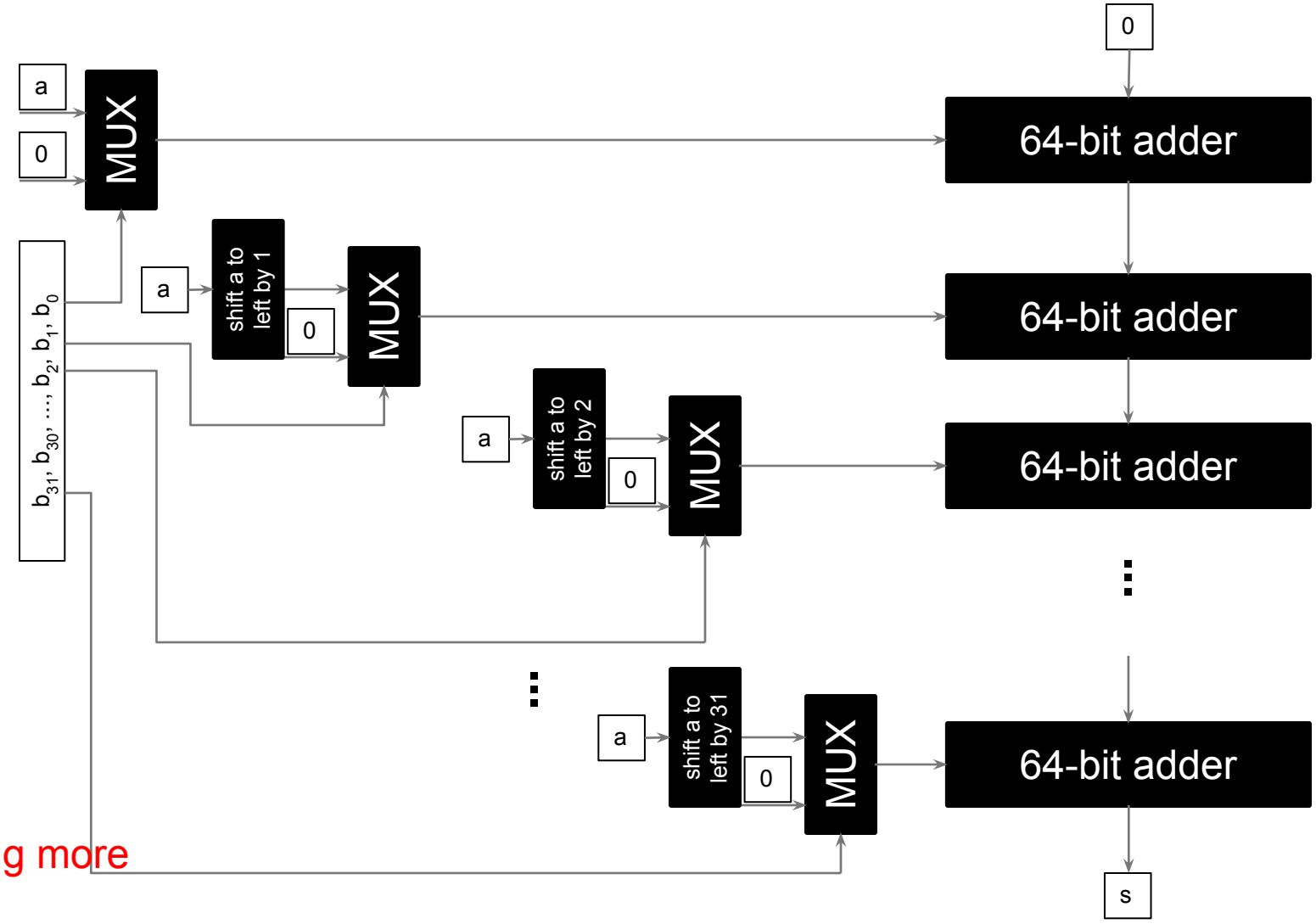


INTEGER MULTIPLICATION

- **the combinational circuit**
  - s = a x b, a and b are 32-bit numbers
  - how do we do it?

INTEGER MULTIPLICATION

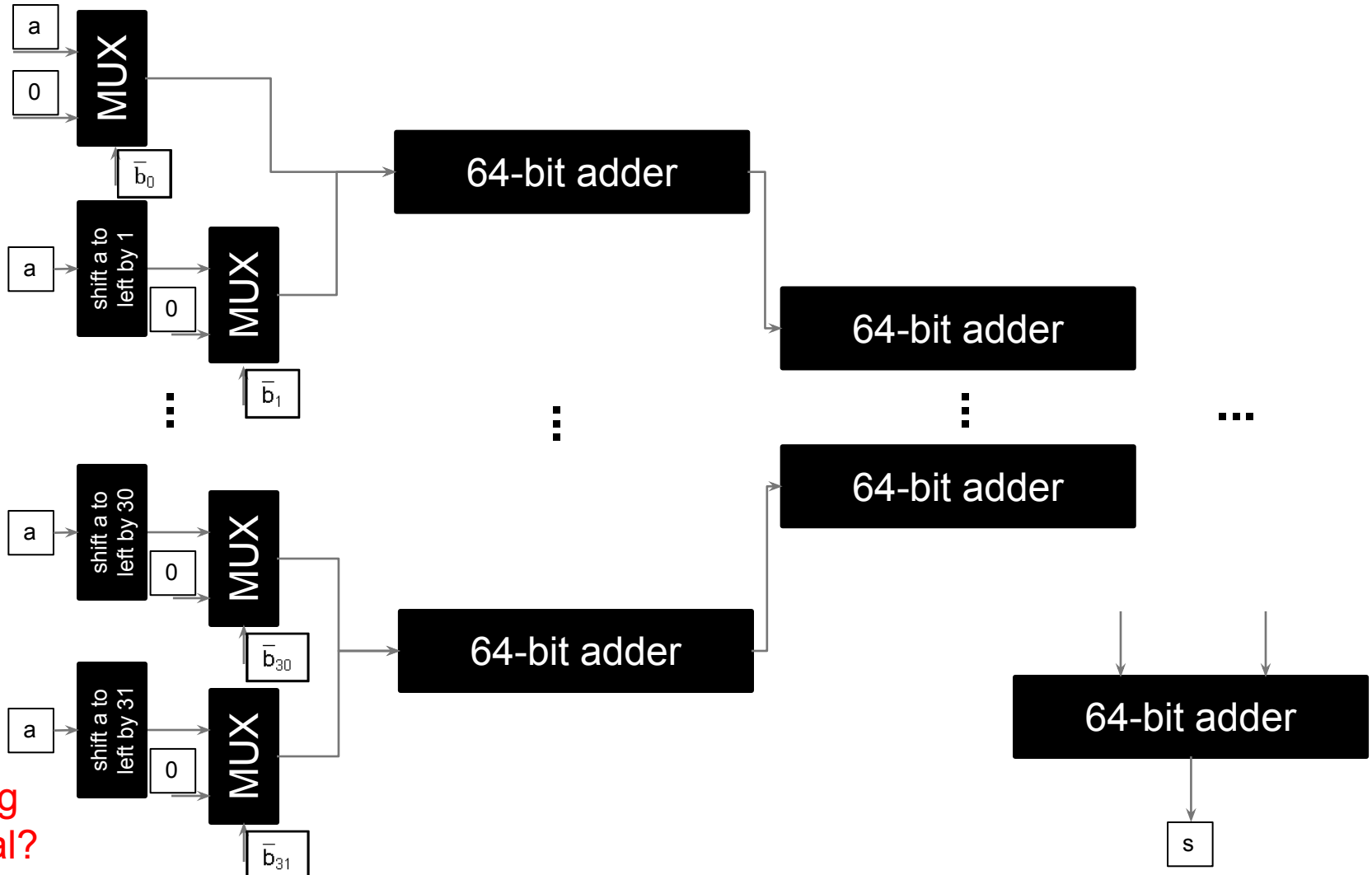
- **the combinational circuit**
  - s = a x b, a and b are 32-bit numbers



something more efficient?

INTEGER MULTIPLICATION

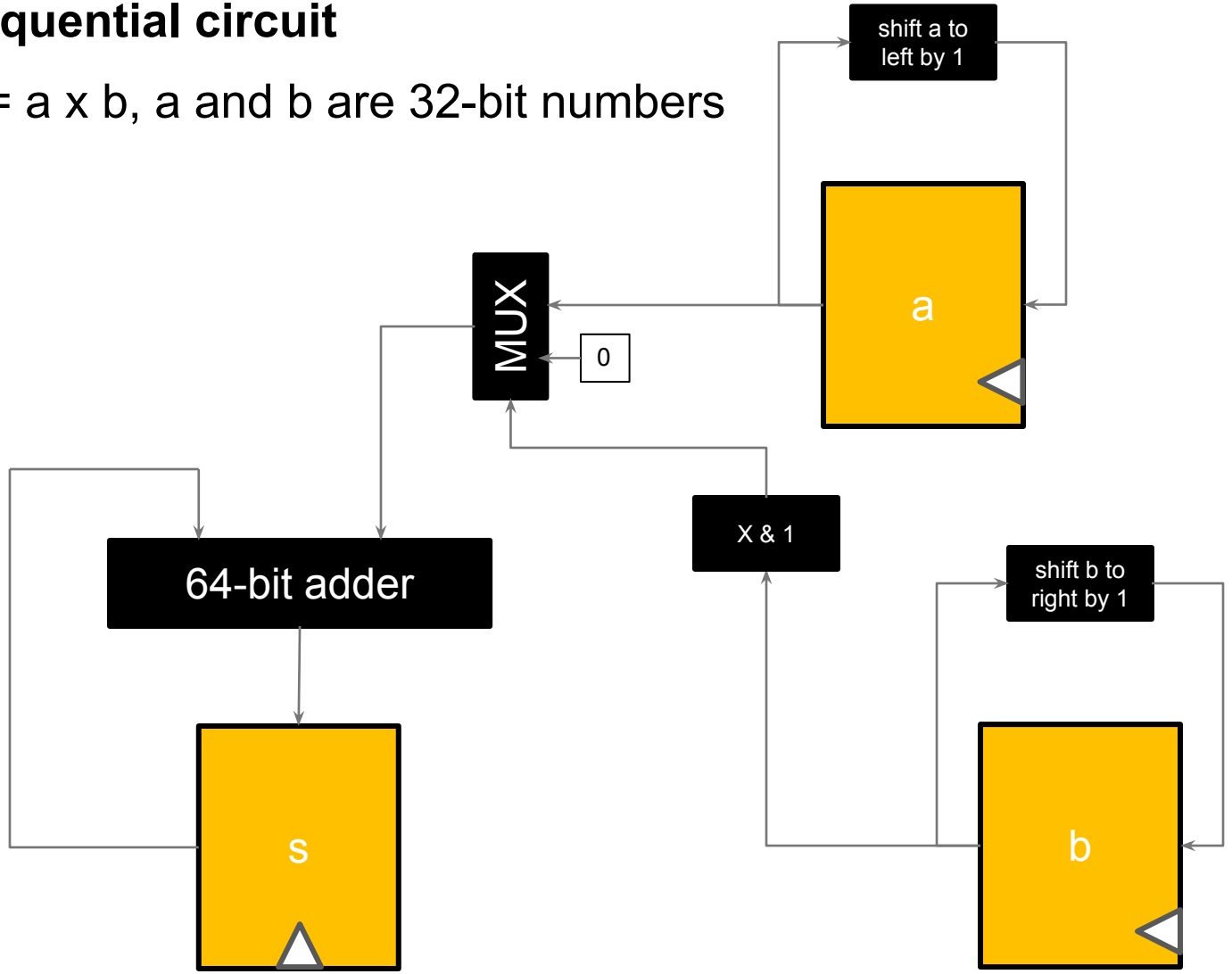
- **the combinational circuit**
  - s = a x b, a and b are 32-bit numbers



something sequential?

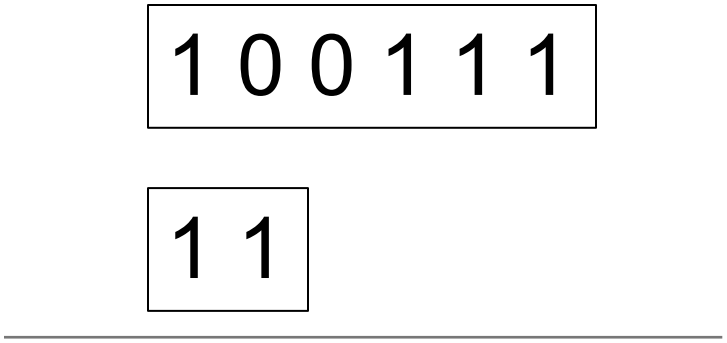
INTEGER MULTIPLICATION

- **the sequential circuit**
  - s = a x b, a and b are 32-bit numbers



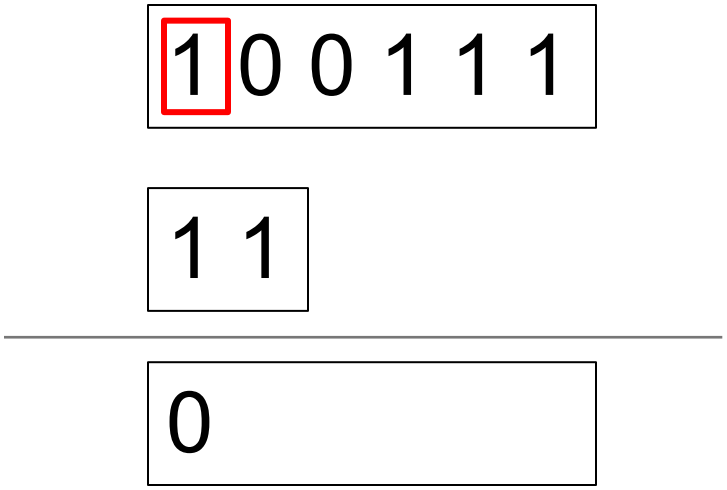
INTEGER DIVISION

- **example, s = a ÷ b**



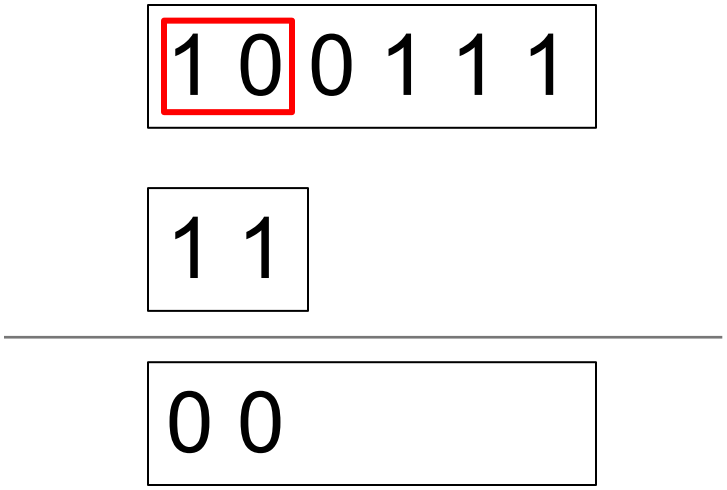
INTEGER DIVISION

- **example, s = a ÷ b**



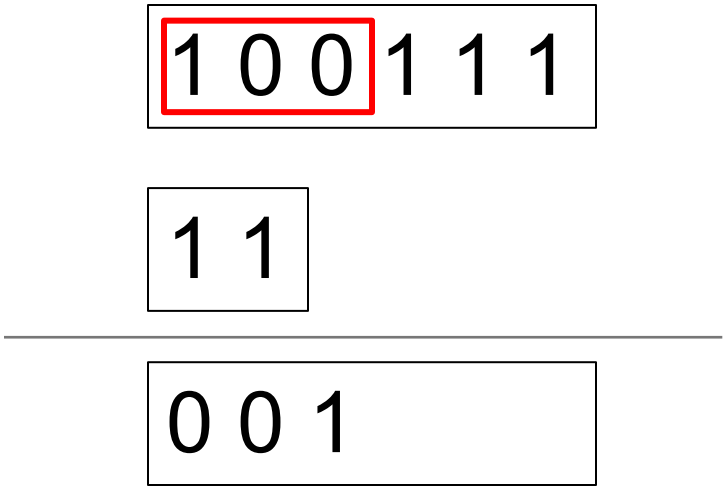
INTEGER DIVISION

- **example, s = a ÷ b**



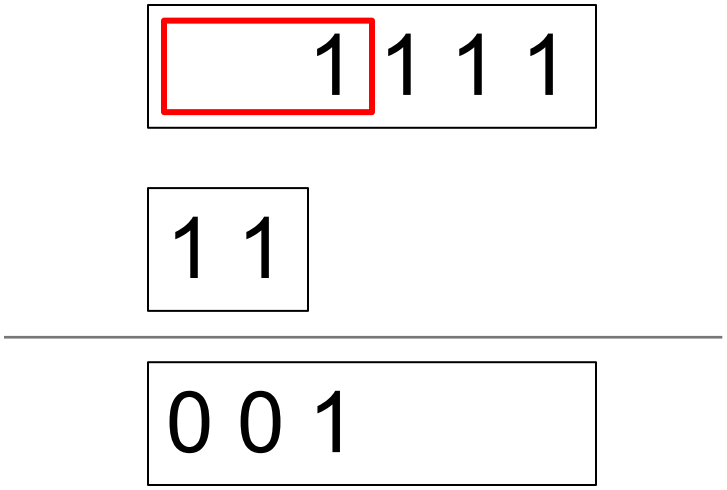
INTEGER DIVISION

- **example, s = a ÷ b**



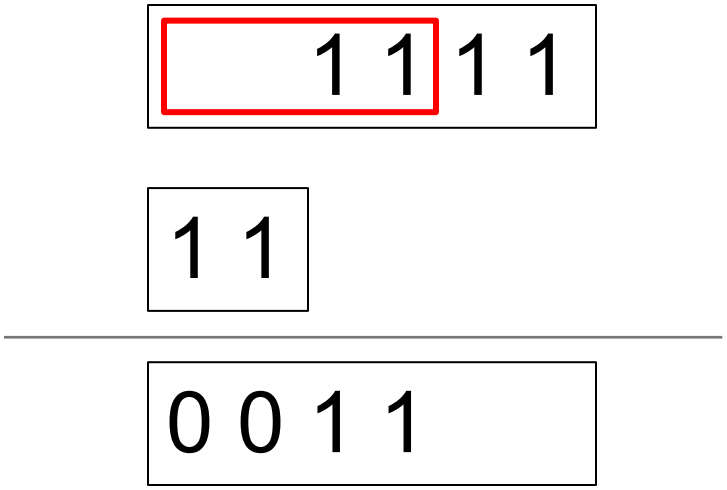
INTEGER DIVISION

- **example, s = a ÷ b**



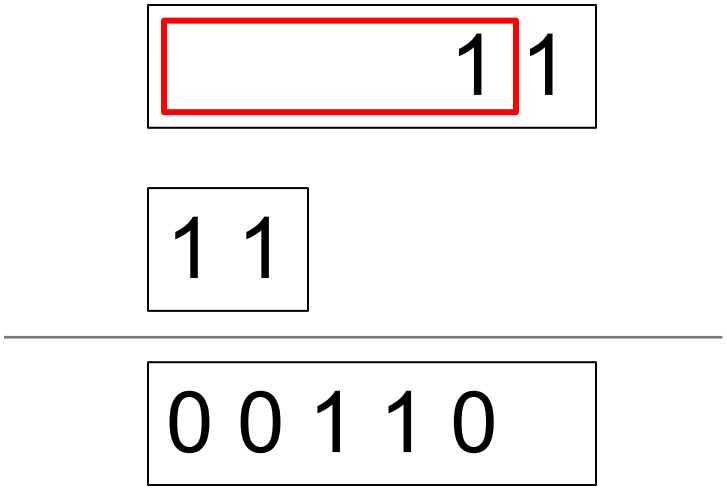
INTEGER DIVISION

- **example, s = a ÷ b**



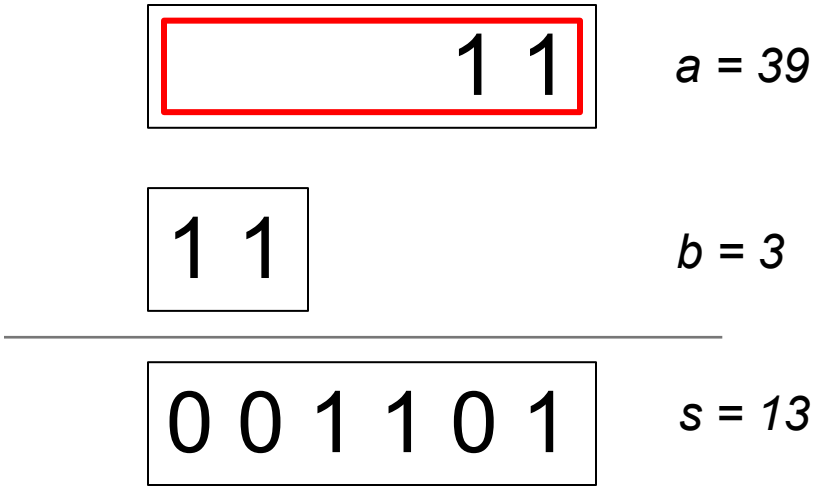
INTEGER DIVISION

- **example, s = a ÷ b**



INTEGER DIVISION

- **example, s = a ÷ b**

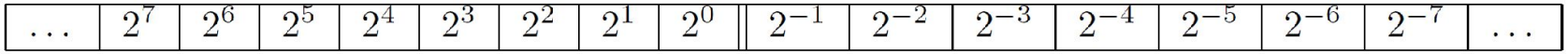


INTEGER DIVISION

- **s = a ÷ b**
  - what happens if a or b are negative variables?
  - the result is negative if a and b have different signs (logical XOR)
  - in general
    - a = s x b + r
    - the sign of r is the sign of a
  - we will not even try to build the division circuit
  - because of this high complexity, compilers and computing systems will do everything possible to avoid division
  - we will see more examples in the seminar ...

FLOATING-POINT REPRESENTATION

- **we discussed fixed-point representation in Seminar 0x00**



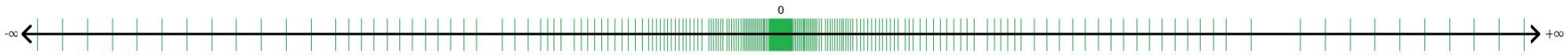
- **example: 7.5 is written as 111.1**

- **what is the problem with this representation?**
  - the integer part is separated from the fractional part
  - each needs a predefined number of bits
  - this can be inefficient
  - we want the total number of bits to be allocated “dynamically”, depending on the number we need to represent

FLOATING-POINT REPRESENTATION

- **when we need to represent a real number**
  - we cannot have infinite precision
  - we have a finite number of bits, so we can write the bits in circuits
  - we need variable precision
  - we can have “infinite” precision if we use rational numbers (and store the numerator and denominator separately as integers)

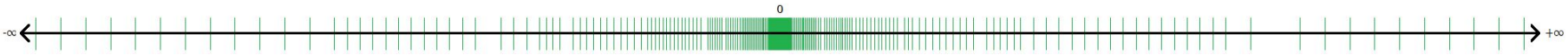
- **the standard: IEEE 754 Floating Point**
  - the density is not uniform on the real line



https://en.wikipedia.org/wiki/Floating-point_arithmetic

FLOATING-POINT REPRESENTATION

- **the standard: IEEE 754 Floating Point**
  - the density is not uniform on the real line



- **there are several consequences**
  - (0.1 + 0.2) == 0.3 versus (0.2 + 0.3) == 0.5 (rounding)
  - math.sqrt(3)*math.sqrt(3) == 3 versus math.sqrt(3*3) == 3
  - (0.7 + 0.2) + 0.1 versus (0.7 + 0.1) + 0.2 (we do not have associativity)
  - the difference with integers
    - if we use an integer data type: 16777216 + 1 = 16777217
    - if we use an FP data type: 16777216.0 + 1 = 16777216.0
    - float(123456789101112) + 1.0 = 123456789101113.0
    - float(1234567891011121) + 1.0 = 1234567891011122.0
    - float(12345678910111213) + 1.0 = 1.2345678910111212e+16

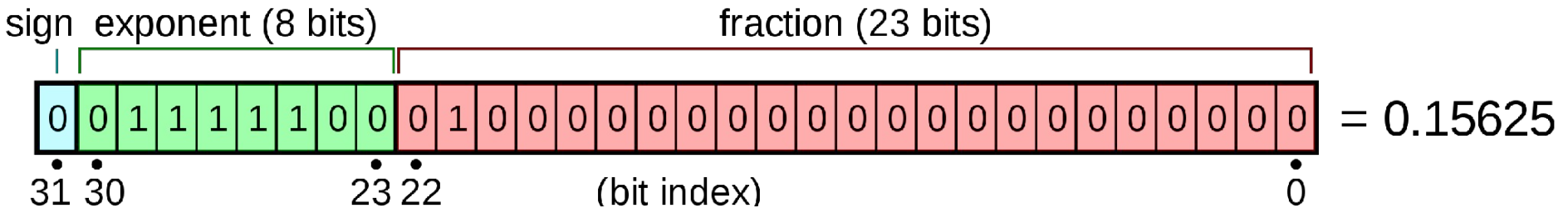
the tests are done in C and Python with variables that are single precision (32-bit FP); the next number after 16777216 is 16777218
another example: after 167772160 comes 167772176 (as numbers grow, the distances between two consecutive values become larger — as in the figure on the previous page

FLOATING-POINT REPRESENTATION

- **scientific representation**
  - $12345 = 1.2345 \times 10^{4}$
  - $5024 = 5.024 \times 10^{3}$
  - $0.00925 = 9.25 \times 10^{-3}$
  - float(12345678910111213) + 1.0 = 1.2345678910111212e+16
  - $101010 = 1.01010 \times 2^{5}$
    - in the binary system, the first bit in the representation is always 1

FLOATING-POINT REPRESENTATION

- **the standard: IEEE 754 Floating Point**



- **$x = (-1)^s \ 1.mmmmmmmmmmmmmmmmmmmmmmm \ 2^{(eeeeeeee)_2 - 127}$**

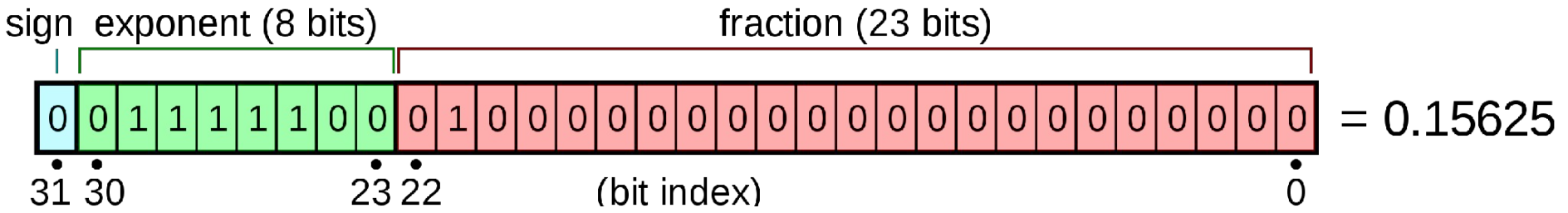
- **examples:**
  - $0.15625 = (-1)^0 \ 1.0100...0 \ 2^{b01111100 - 127} = 1.25 \ 2^{-3} = 1.25/8$
  - other examples:
  - $(-1)^0 \ 1.1000...0 \ 2^{b01111100 - 127} = 1.5 \ 2^{-3} = 1.5/8 = 0.1875$
  - $1 = (-1)^0 \ 1.00000000000000000000000 \ 2^{b01111111-127}$
  - $-1 = (-1)^1 \ 1.00000000000000000000000 \ 2^{b01111111-127}$
  - $2 = (-1)^0 \ 1.00000000000000000000000 \ 2^{b10000000-127}$
  - $\infty = (-1)^0 \ 1.00000000000000000000000 \ 2^{b11111111-127}$
  - $-\infty = (-1)^1 \ 1.00000000000000000000000 \ 2^{b11111111-127}$

https://en.wikipedia.org/wiki/Floating-point_arithmetic

https://www.h-schmidt.net/FloatConverter/IEEE754.html

FLOATING-POINT REPRESENTATION

- **the standard: IEEE 754 Floating Point**



- **$x=(-1)^s 1.mmmmmmmmmmmmmmmmmmmmmmm 2^{(eeeeeeee)_2 - 127}$**

- **examples:**
  - $\pi \approx 3.14159265 = (-1)^0\ 1.10010010000111111011010\ 2^{b10000000\ -127}$
  - $+0 = (-1)^0\ \ 1.00000000000000000000000\ 2^{00000000-127}$
  - $-0 = (-1)^1\ \ 1.00000000000000000000000\ 2^{00000000-127}$
  - signaling NaN: 0x7F800001 or 0x7FBFFFFF or between 0xFF800001 and 0xFFBFFFFF
  - quiet NaN: 0x7FC00000 or 0x7FFFFFFF or between 0xFFC00000 and 0xFFFFFFFF

more examples in Seminar 0x03

https://en.wikipedia.org/wiki/Floating-point_arithmetic

https://www.h-schmidt.net/FloatConverter/IEEE754.html

WHAT WE DID TODAY

- sequential + combinational logic, an example
- multiplication of binary integers
- division of binary integers
- floating-point number representation
- working with floating-point numbers

NEXT TIME ...

- we begin discussing computer architectures
- we begin discussing instruction sets

FURTHER READING

- **PH book**
  - 3.3 Multiplication
  - 3.4 Division
  - 3.5 Floating Point

- **Computerphile, Floating Point Numbers, https://www.youtube.com/watch?v=PZRI1IfStY0**
- **Computephile, Floating Point Numbers (Part1: FP vs Fixed), https://www.youtube.com/watch?v=f4ekifyijlg**
- **Computephile, Floating Point Numbers (Part2: FP Addition), https://www.youtube.com/watch?v=782QWNOD_Z0**

- **Julia Evans, Examples of floating point problems, https://jvns.ca/blog/2023/01/13/examples-of-floating-point-problems/**