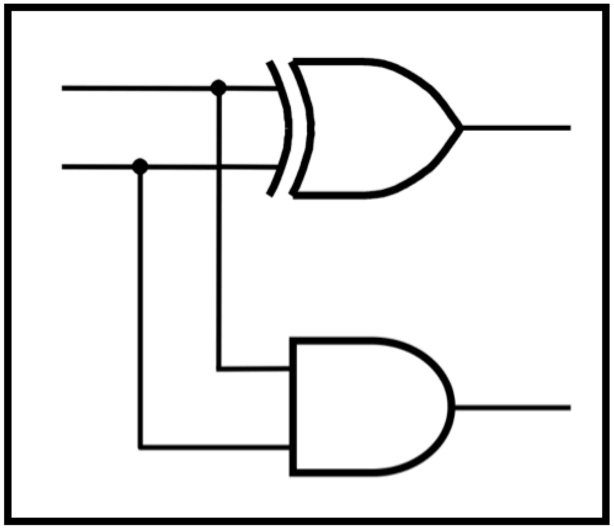


CprE 281: Digital Logic

Instructor: Alexander Stoytchev

http://www.ece.iastate.edu/~alexs/classes/

Floating Point Numbers

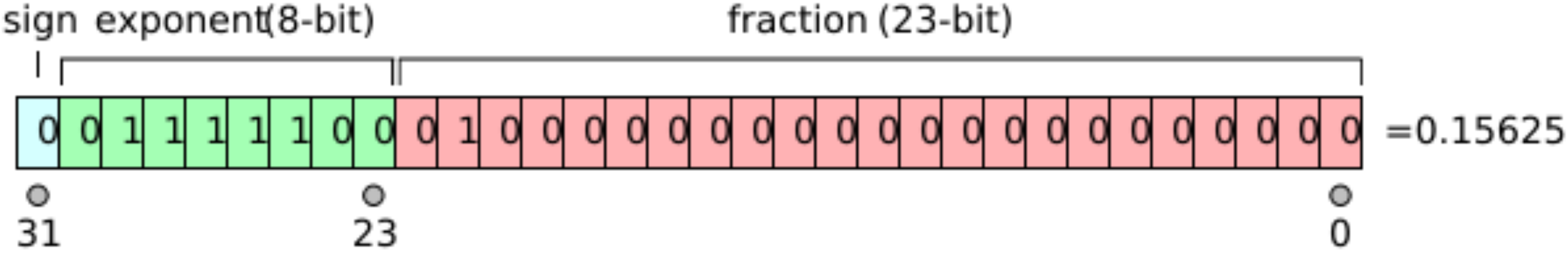
CprE 281: Digital Logic
Iowa State University, Ames, IA

Administrative Stuff

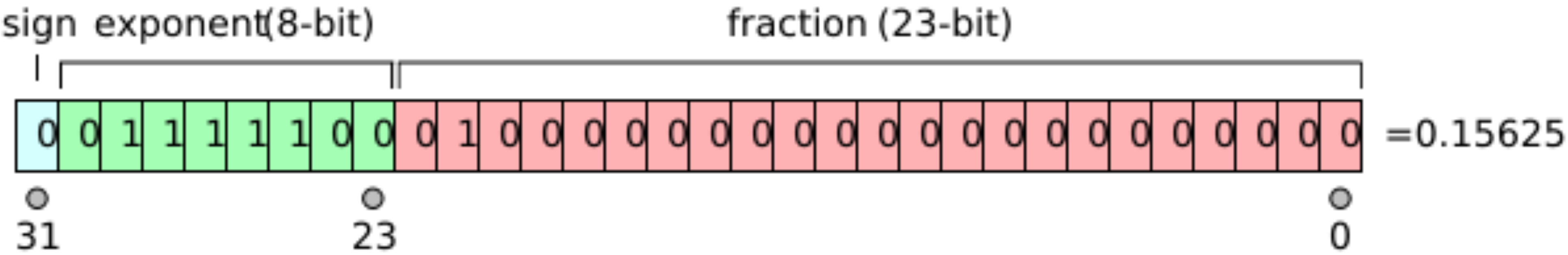
- **HW 6 is out**

- **It is due on Monday Oct 9 @ 4pm**

The story with floats is more complicated
IEEE 754-1985 Standard



[http://en.wikipedia.org/wiki/IEEE_754]



$$v = (-1)^{sign} \times 2^{exponent - exponent\ bias} \times 1.fraction$$

s = +1 (positive numbers and +0) when the sign bit is 0

s = −1 (negative numbers and −0) when the sign bit is 1

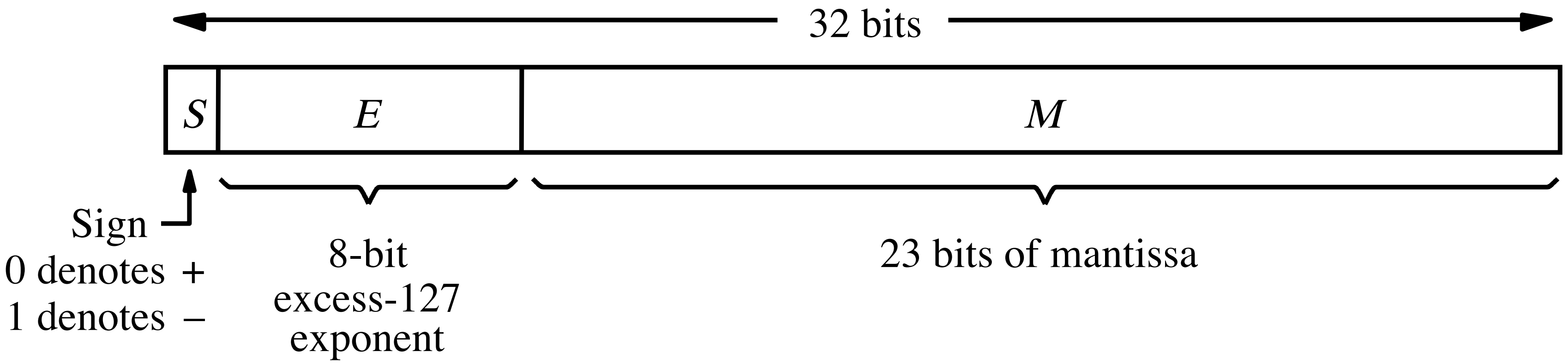
e = *exponent* − 127 (in other words the exponent is stored with 127 added to it, also called "biased with 127")

In the example shown above, the *sign* bit is zero, the *exponent* is 124, and the significand is 1.01 (in binary, which is 1.25 in decimal). The represented number is

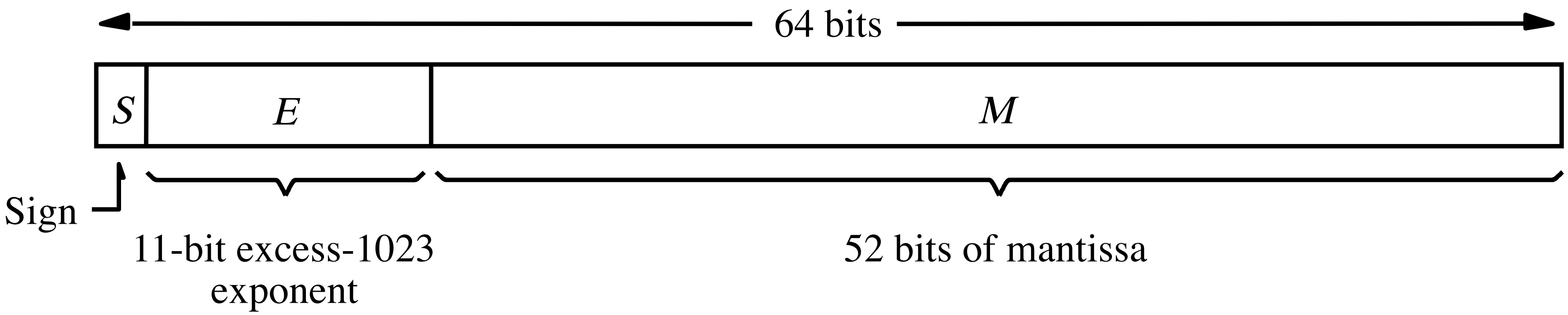
$$(-1)^{0} \times 2^{(124 - 127)} \times 1.25 = +0.15625.$$

[http://en.wikipedia.org/wiki/IEEE_754]

Float (32-bit) vs. Double (64-bit)



(a) Single precision



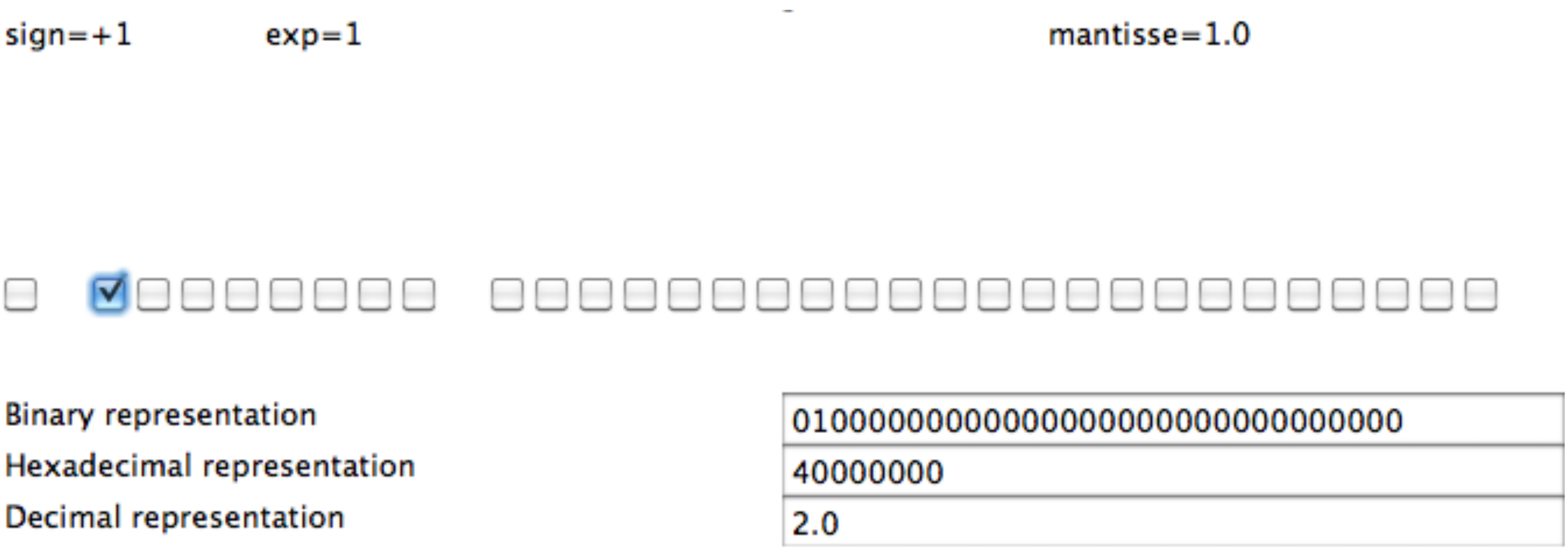
(b) Double precision

[Figure 3.37 from the textbook]

On-line IEEE 754 Converter

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

Representing 2.0



sign=+1 exp=1 mantisse=1.0

☐ ☑☐☐☐☐☐☐☐ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

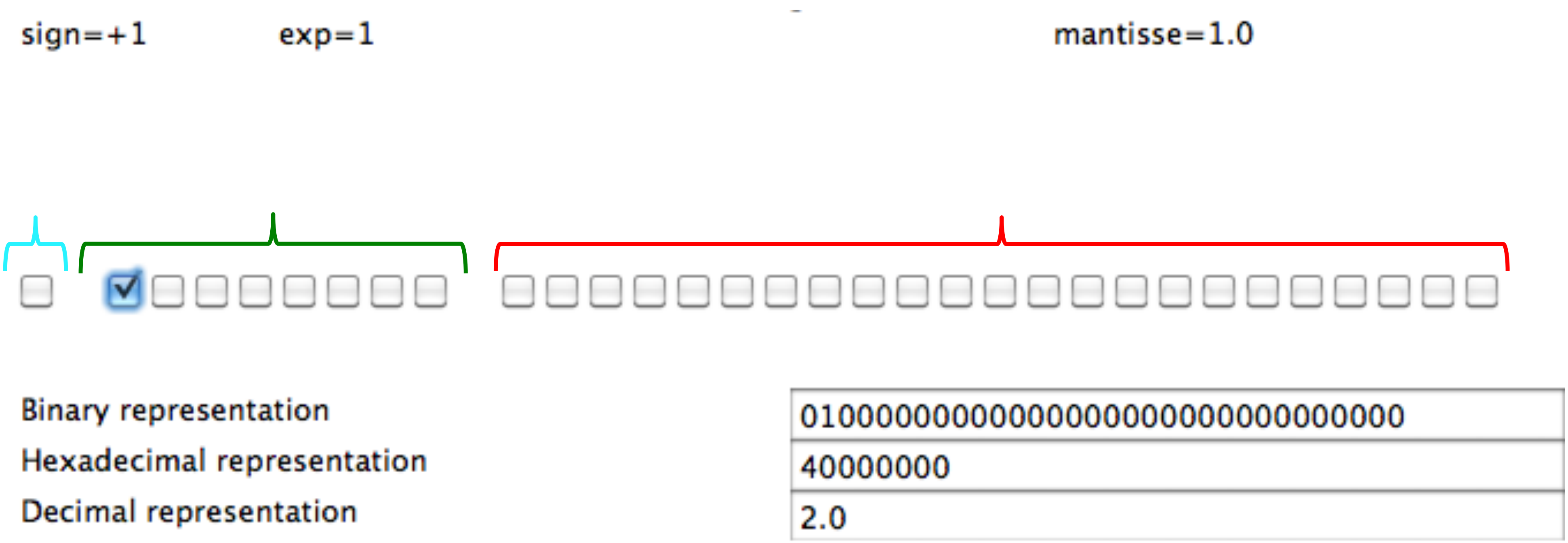
Binary representation: 01000000000000000000000000000000

Hexadecimal representation: 40000000

Decimal representation: 2.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 2.0



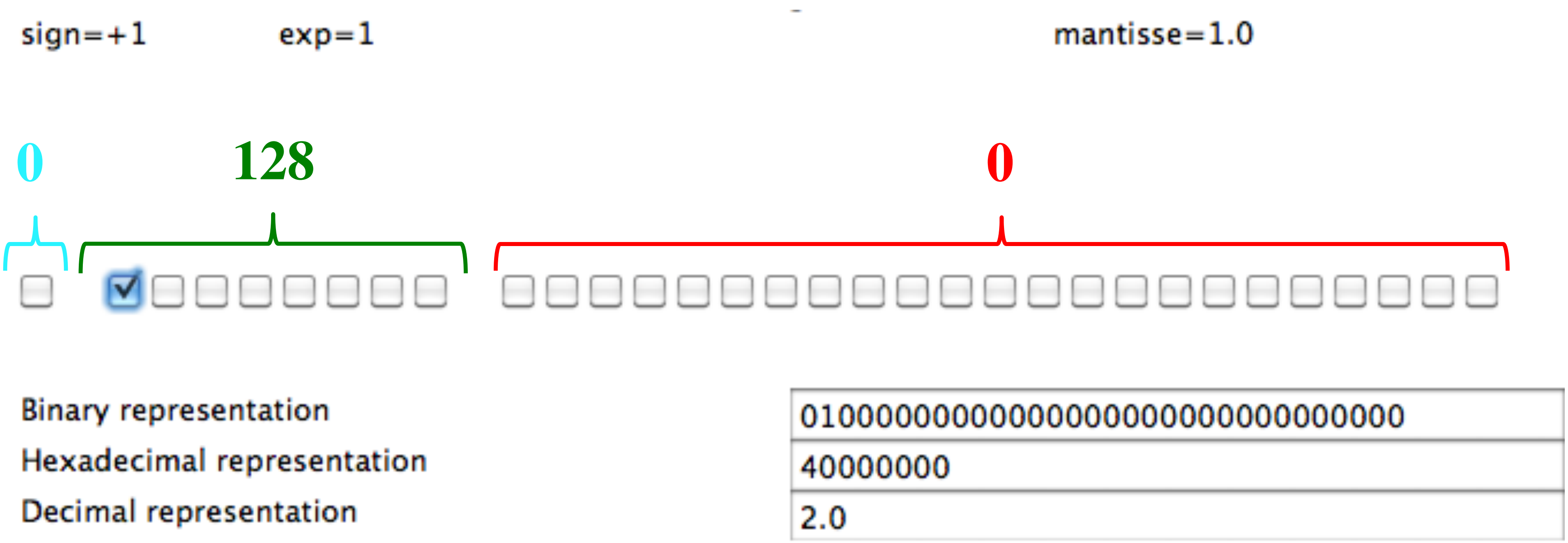
sign=+1 exp=1 mantisse=1.0

☐ ☑☐☐☐☐☐☐☐ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	01000000000000000000000000000000
Hexadecimal representation	40000000
Decimal representation	2.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 2.0



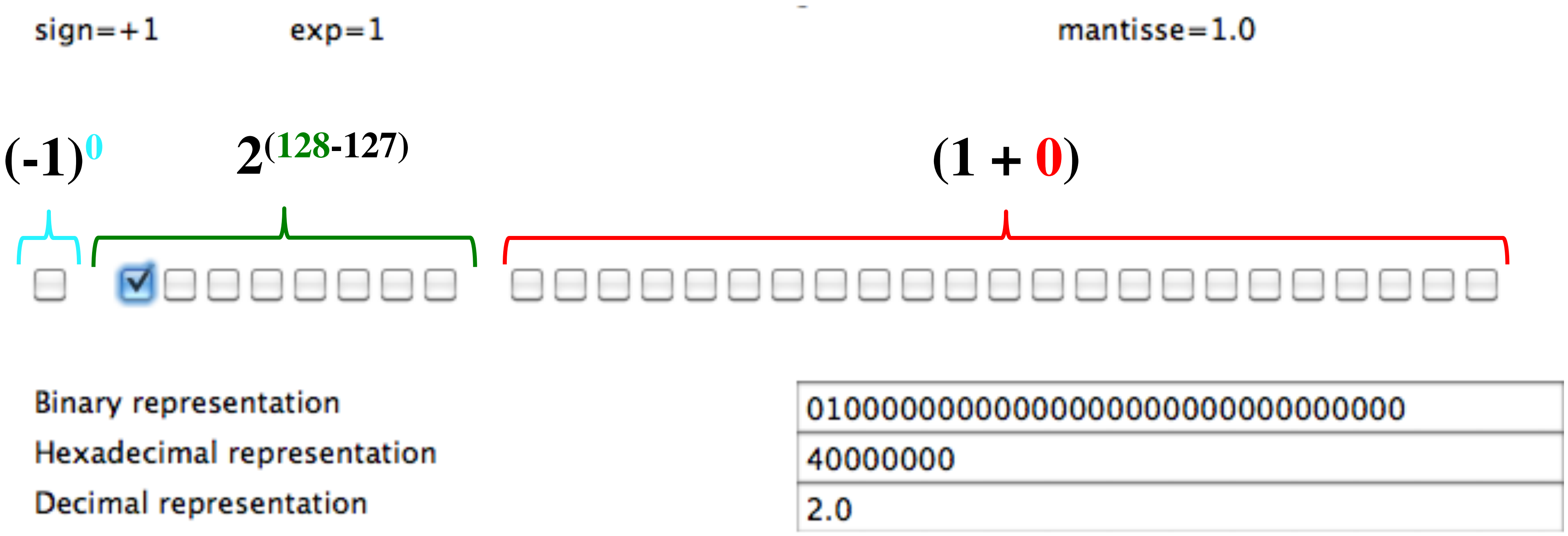
sign=+1 exp=1 mantisse=1.0

☐ ☑☐☐☐☐☐☐☐ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	01000000000000000000000000000000
Hexadecimal representation	40000000
Decimal representation	2.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 2.0



sign=+1 exp=1 mantisse=1.0

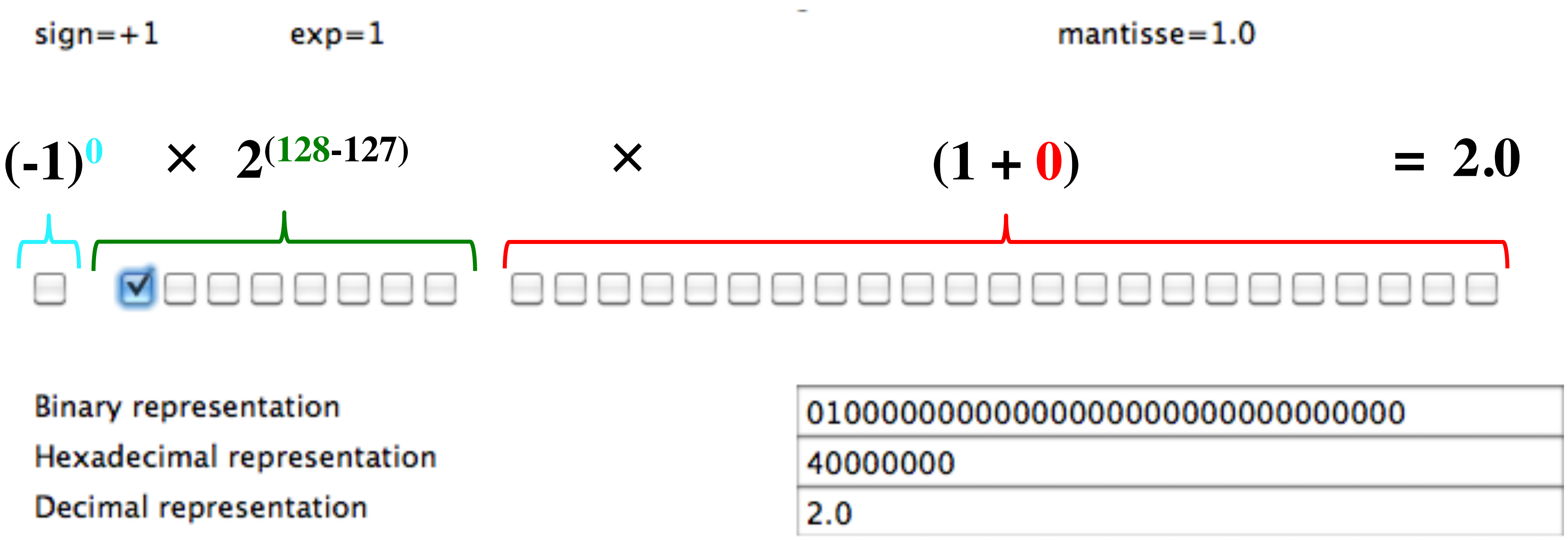
$(-1)^{0}$ $2^{(128-127)}$ $(1 + 0)$

☐ ☑☐☐☐☐☐☐☐ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	01000000000000000000000000000000
Hexadecimal representation	40000000
Decimal representation	2.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 2.0



sign=+1 exp=1 mantisse=1.0

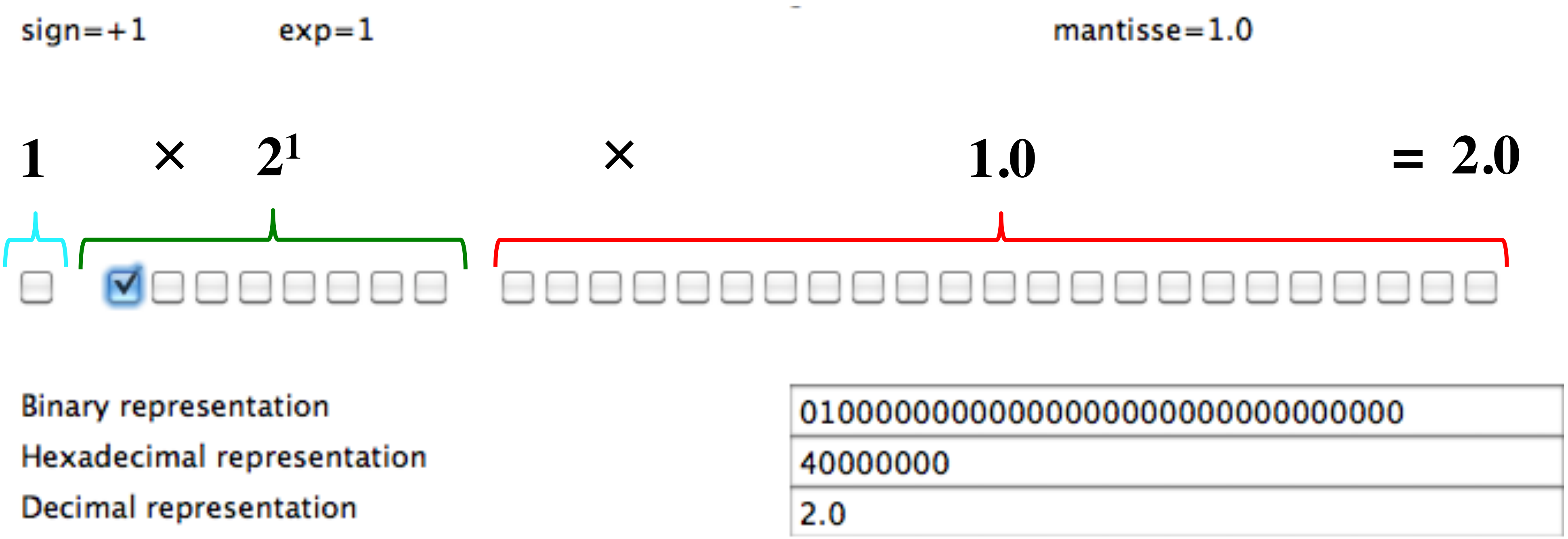
$$(-1)^{0} \times 2^{(128-127)} \times (1 + 0) = 2.0$$

☐ ☑☐☐☐☐☐☐☐ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	01000000000000000000000000000000
Hexadecimal representation	40000000
Decimal representation	2.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 2.0



sign=+1 exp=1 mantisse=1.0

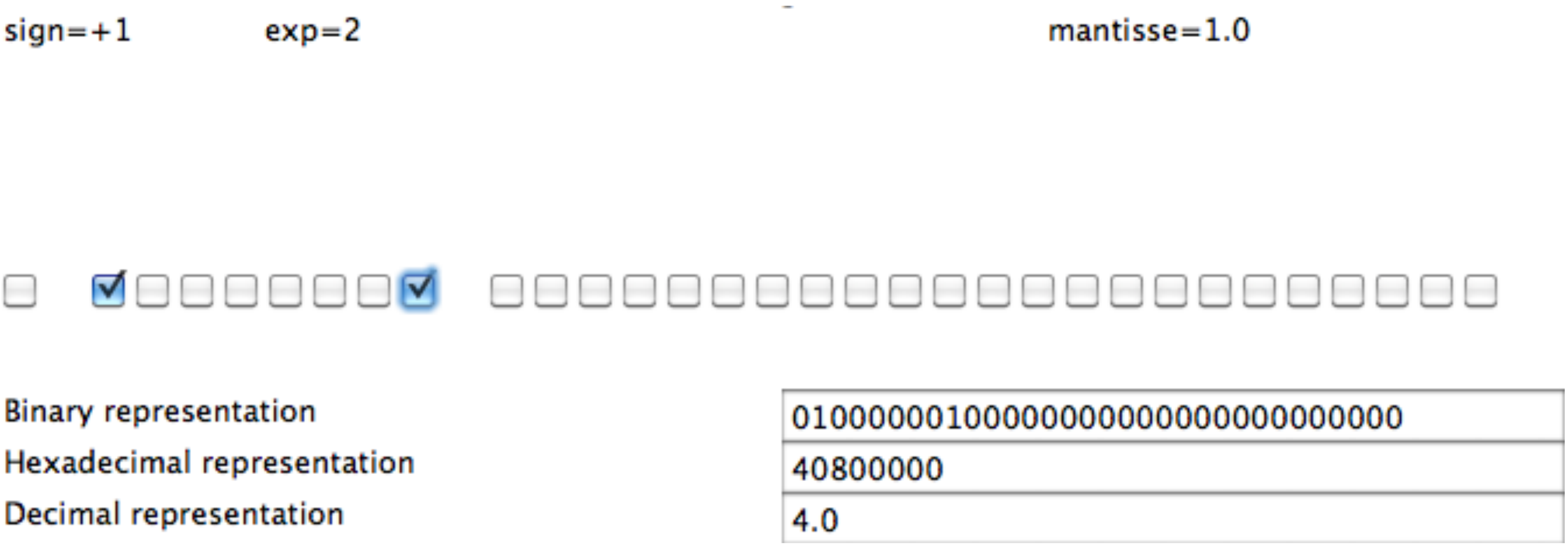
$$\underbrace{1}_{} \times \underbrace{2^1}_{} \times \underbrace{1.0}_{} = 2.0$$

☐ ☑☐☐☐☐☐☐☐ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	01000000000000000000000000000000
Hexadecimal representation	40000000
Decimal representation	2.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 4.0



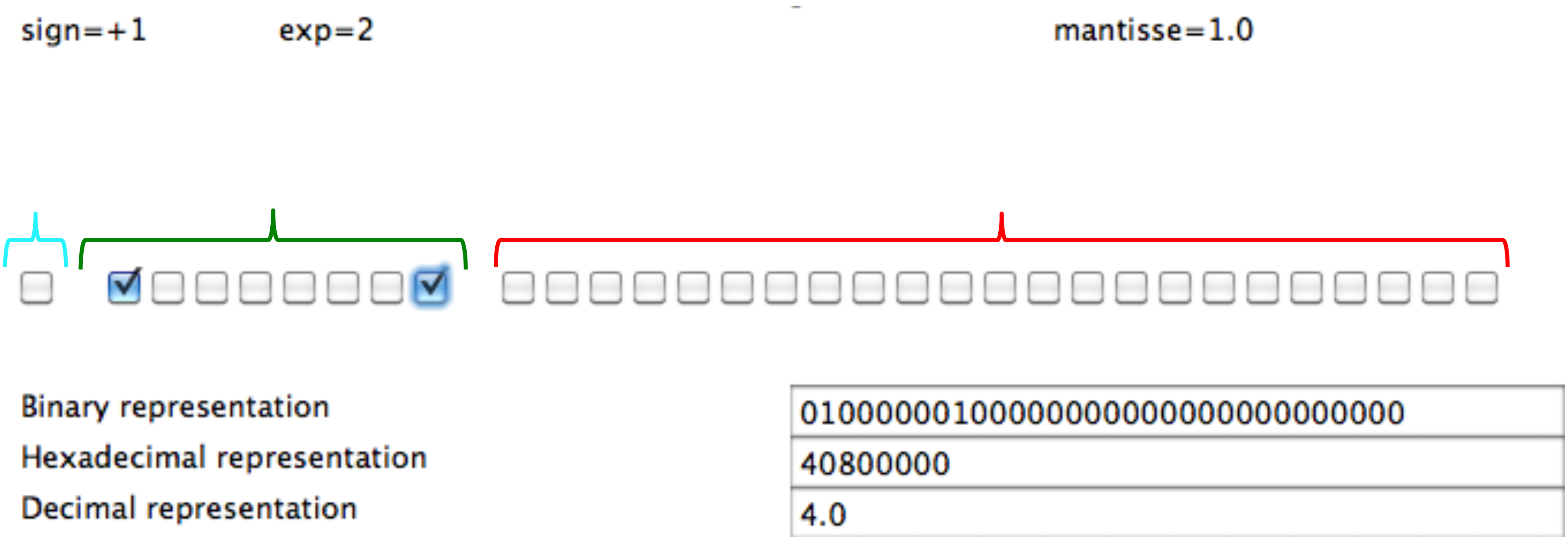
sign=+1 exp=2 mantissa=1.0

☐ ☑☐☐☐☐☐☐☑ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	01000000100000000000000000000000
Hexadecimal representation	40800000
Decimal representation	4.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 4.0



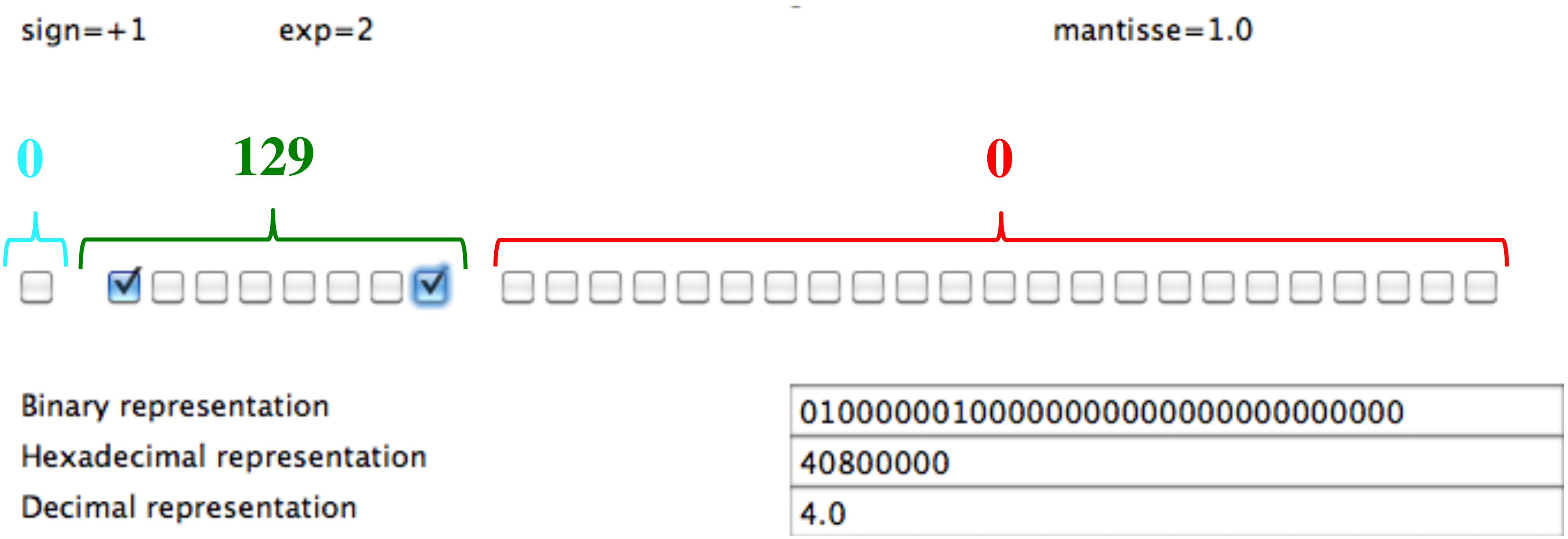
sign=+1 exp=2 mantisse=1.0

☐ ☑☐☐☐☐☐☐☑ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	01000000100000000000000000000000
Hexadecimal representation	40800000
Decimal representation	4.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

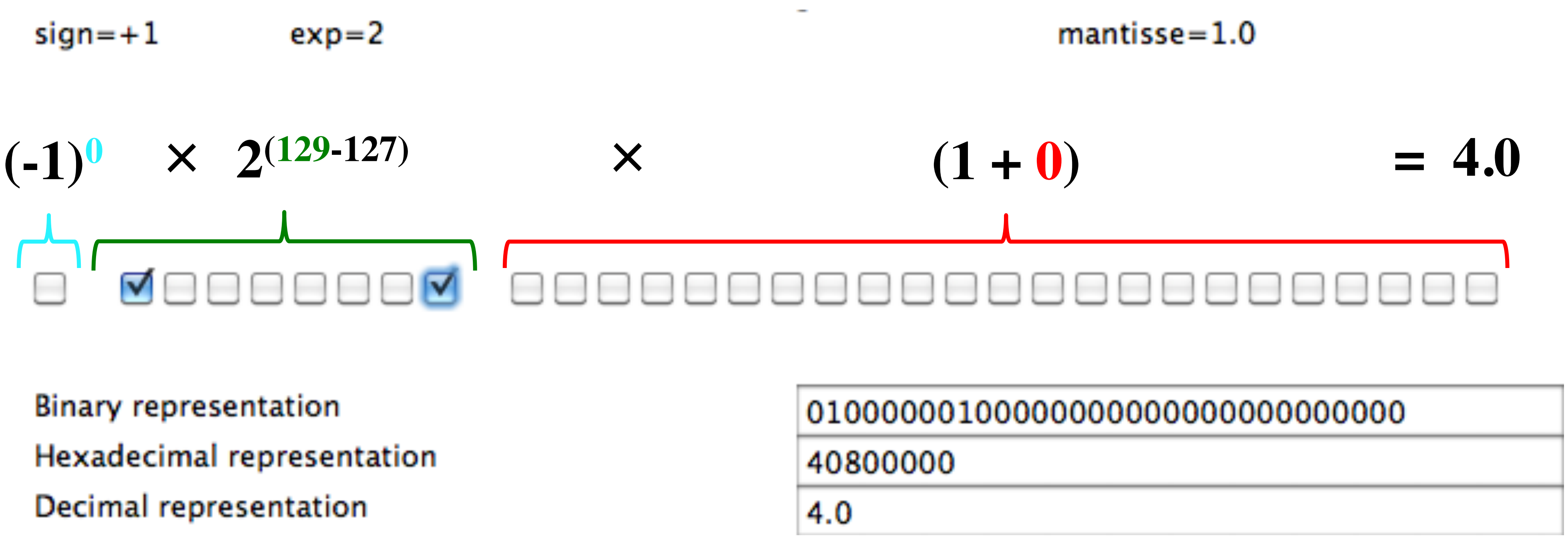
Representing 4.0



Binary representation	01000000100000000000000000000000
Hexadecimal representation	40800000
Decimal representation	4.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 4.0



sign=+1 exp=2 mantisse=1.0

$$\underbrace{(-1)^{0}} \times \underbrace{2^{(129-127)}} \times \underbrace{(1 + 0)} = 4.0$$

☐ ☑☐☐☐☐☐☐☑ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

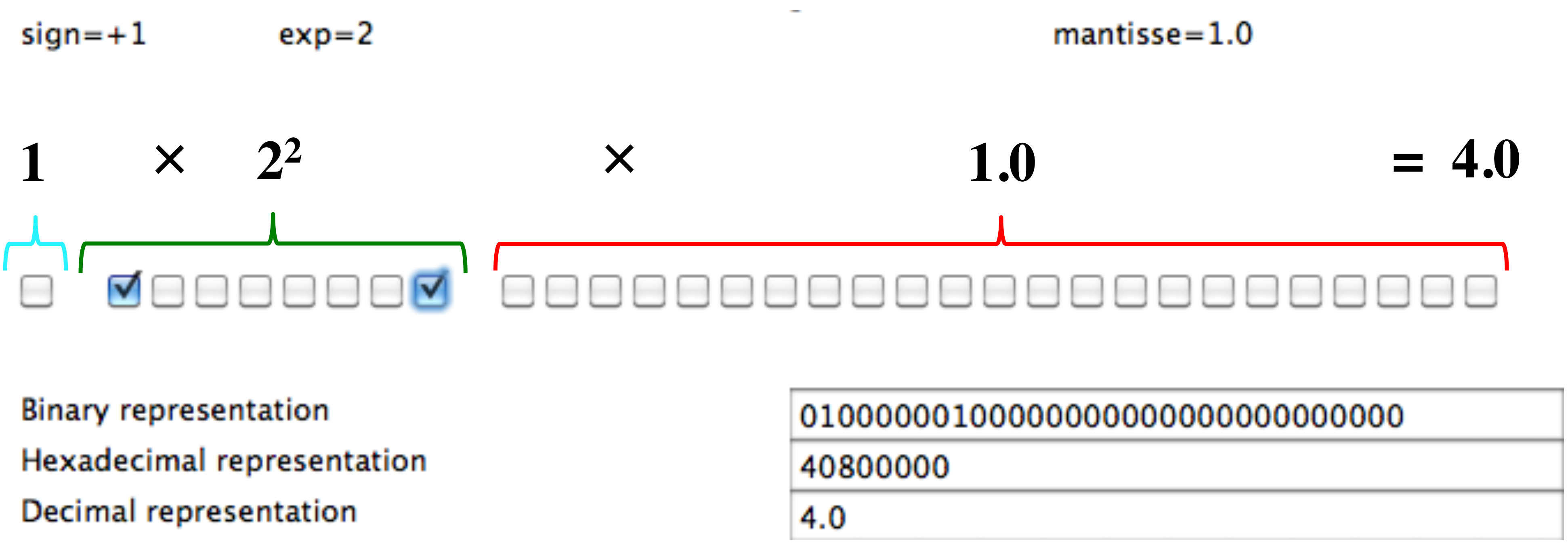
Binary representation: 01000000100000000000000000000000

Hexadecimal representation: 40800000

Decimal representation: 4.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 4.0



$$1 \times 2^2 \times 1.0 = 4.0$$

sign=+1 exp=2 mantisse=1.0

☐ ☑☐☐☐☐☐☐☑ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

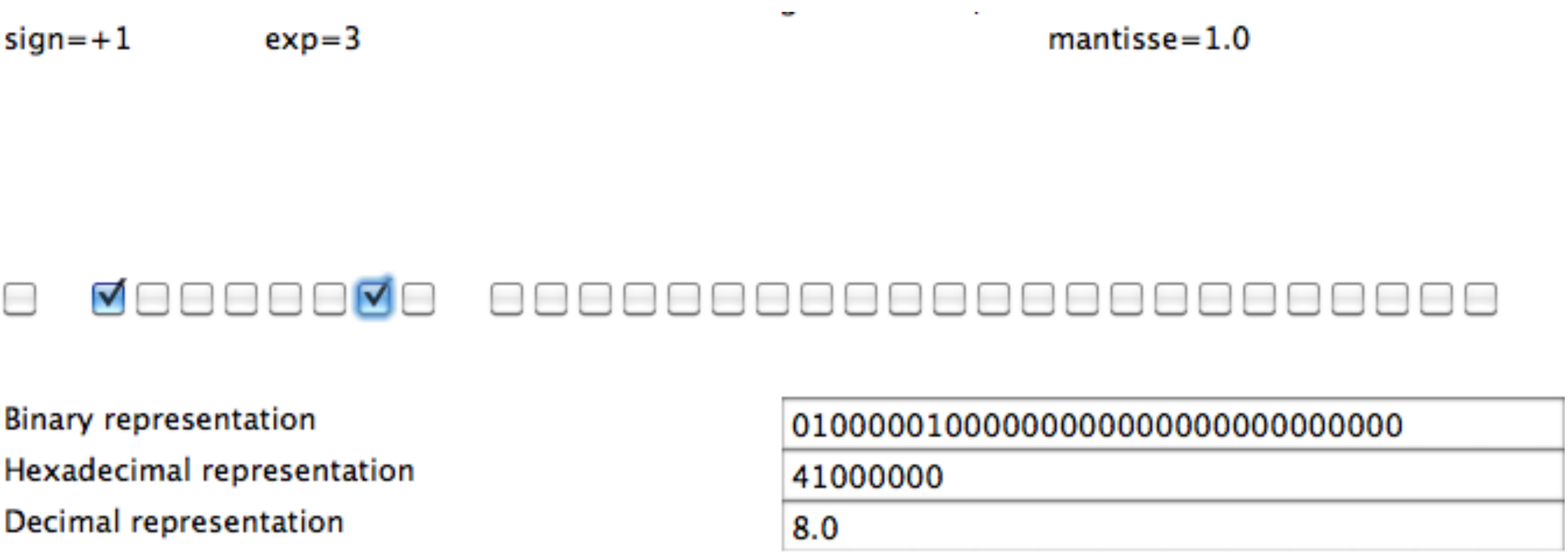
Binary representation: 01000000100000000000000000000000

Hexadecimal representation: 40800000

Decimal representation: 4.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 8.0



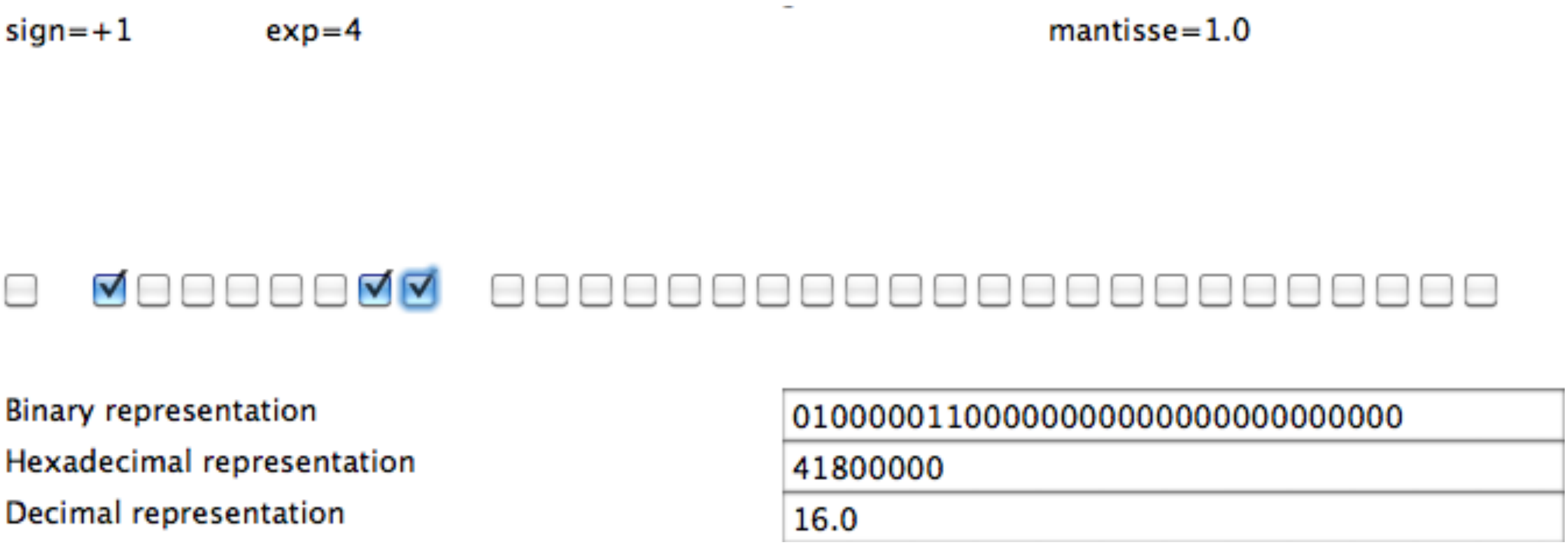
sign=+1 exp=3 mantisse=1.0

☐ ☑☐☐☐☐☐☑☐ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	01000001000000000000000000000000
Hexadecimal representation	41000000
Decimal representation	8.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 16.0



sign=+1 exp=4 mantisse=1.0

☐ ☑☐☐☐☐☐☑☑ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

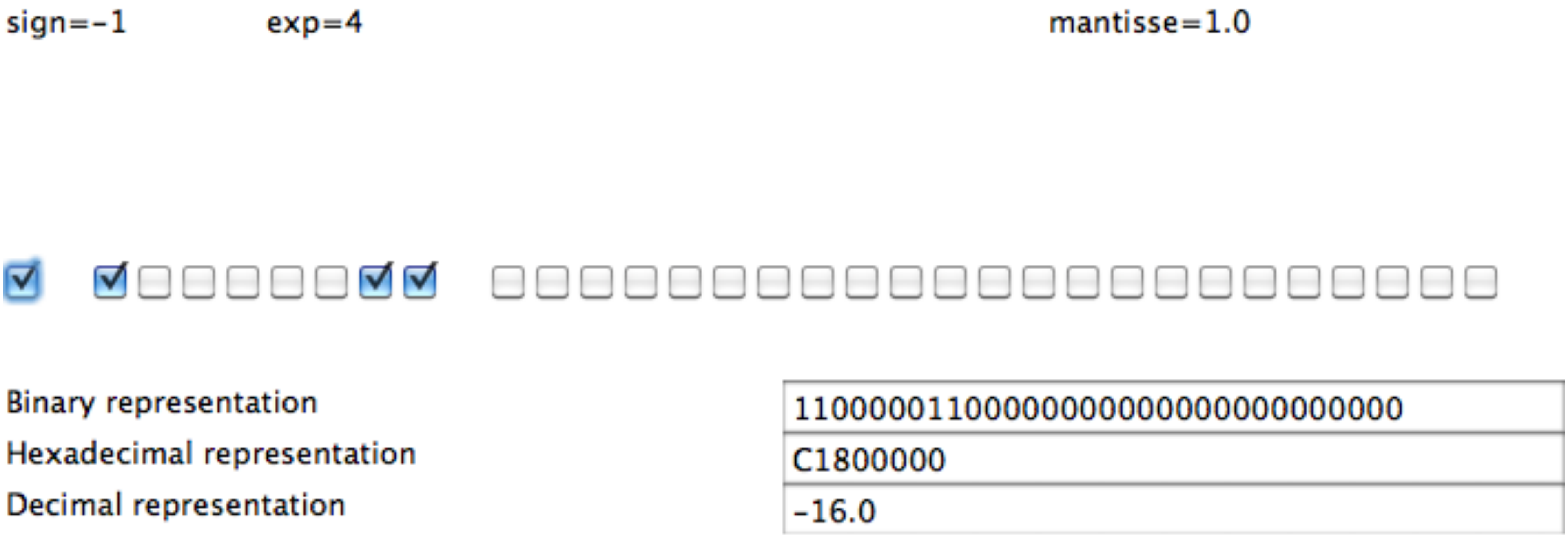
Binary representation: 01000001100000000000000000000000

Hexadecimal representation: 41800000

Decimal representation: 16.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing -16.0



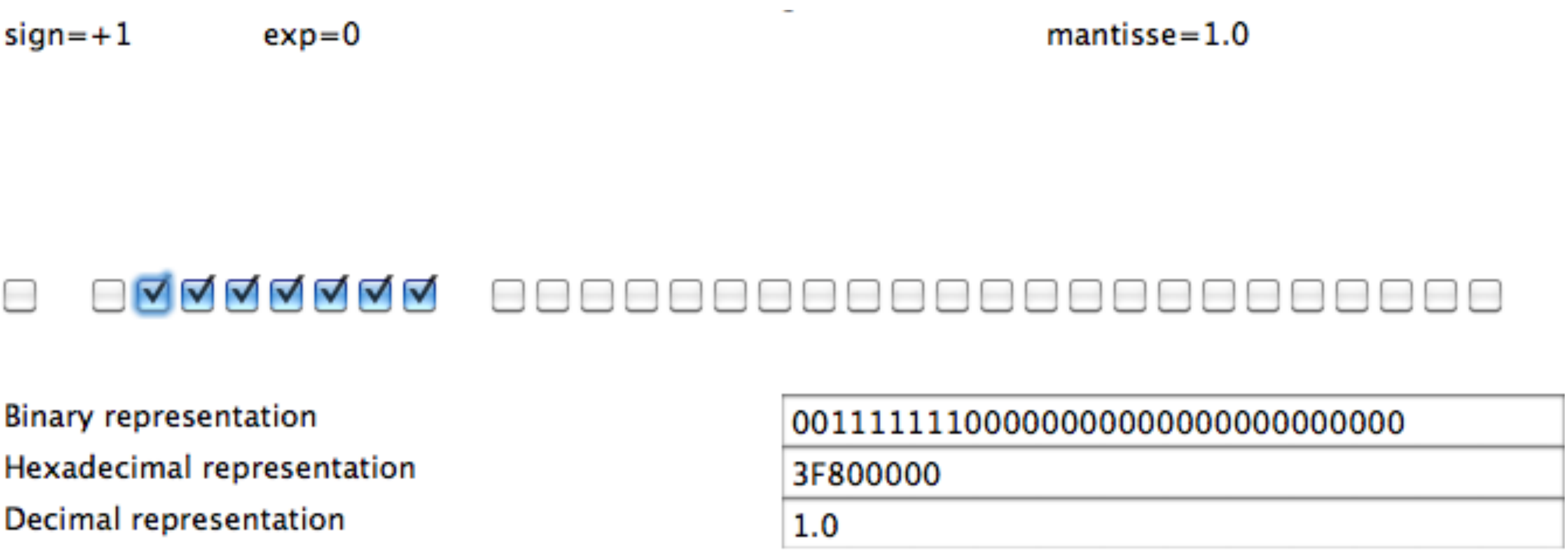
sign=-1 exp=4 mantisse=1.0

☑ ☑☐☐☐☐☐☑☑ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	11000001100000000000000000000000
Hexadecimal representation	C1800000
Decimal representation	-16.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 1.0



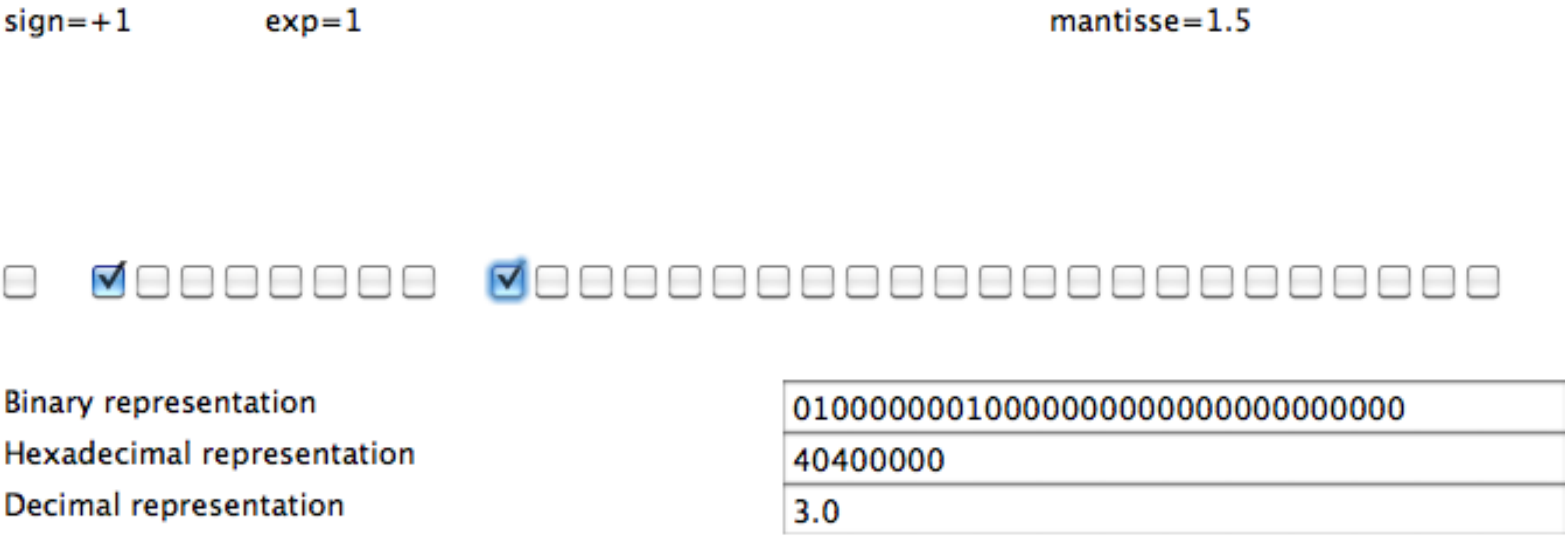
sign=+1 exp=0 mantissa=1.0

☐ ☐☑☑☑☑☑☑☑ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	00111111100000000000000000000000
Hexadecimal representation	3F800000
Decimal representation	1.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 3.0



sign=+1 exp=1 mantissa=1.5

☐ ☑☐☐☐☐☐☐☐ ☑☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

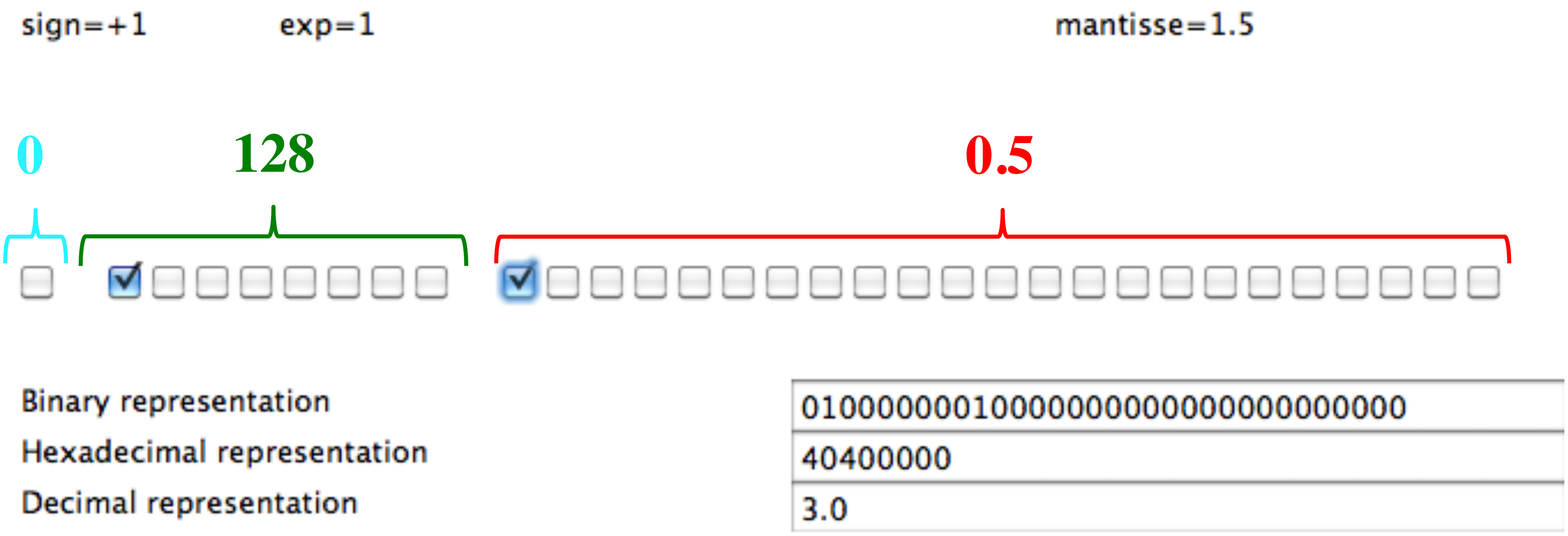
Binary representation: 01000000010000000000000000000000

Hexadecimal representation: 40400000

Decimal representation: 3.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 3.0

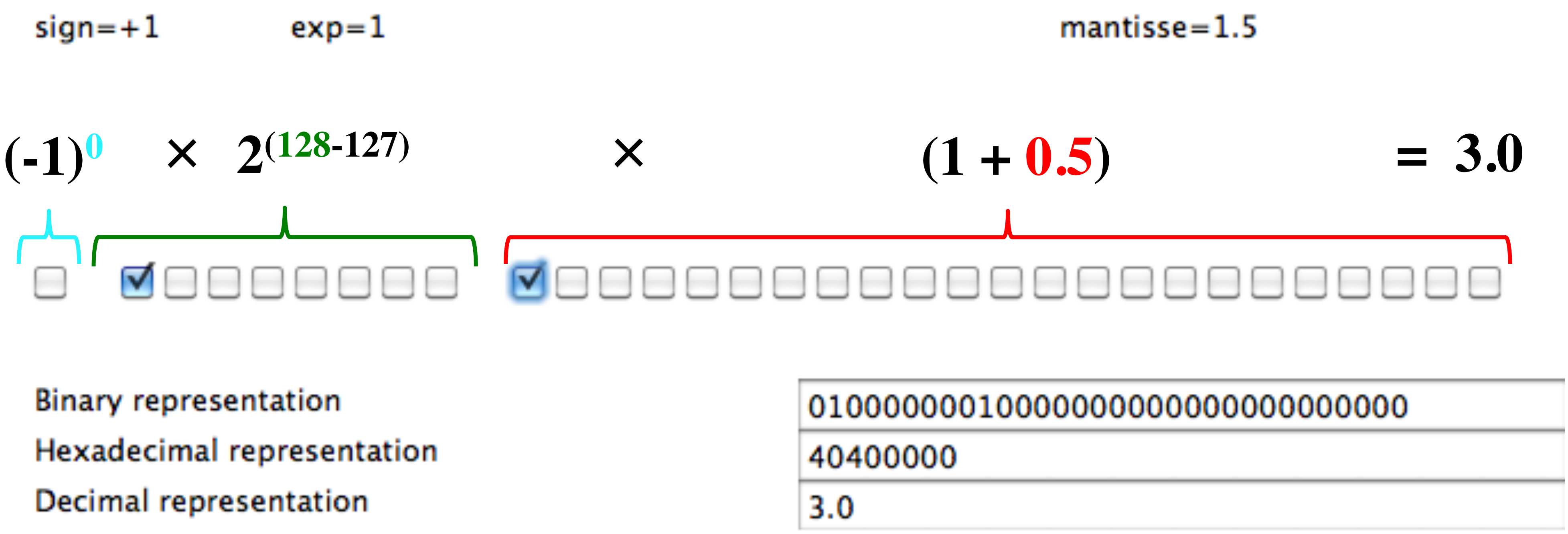


☐ ☑☐☐☐☐☐☐☐ ☑☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	01000000010000000000000000000000
Hexadecimal representation	40400000
Decimal representation	3.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 3.0



sign=+1 exp=1 mantisse=1.5

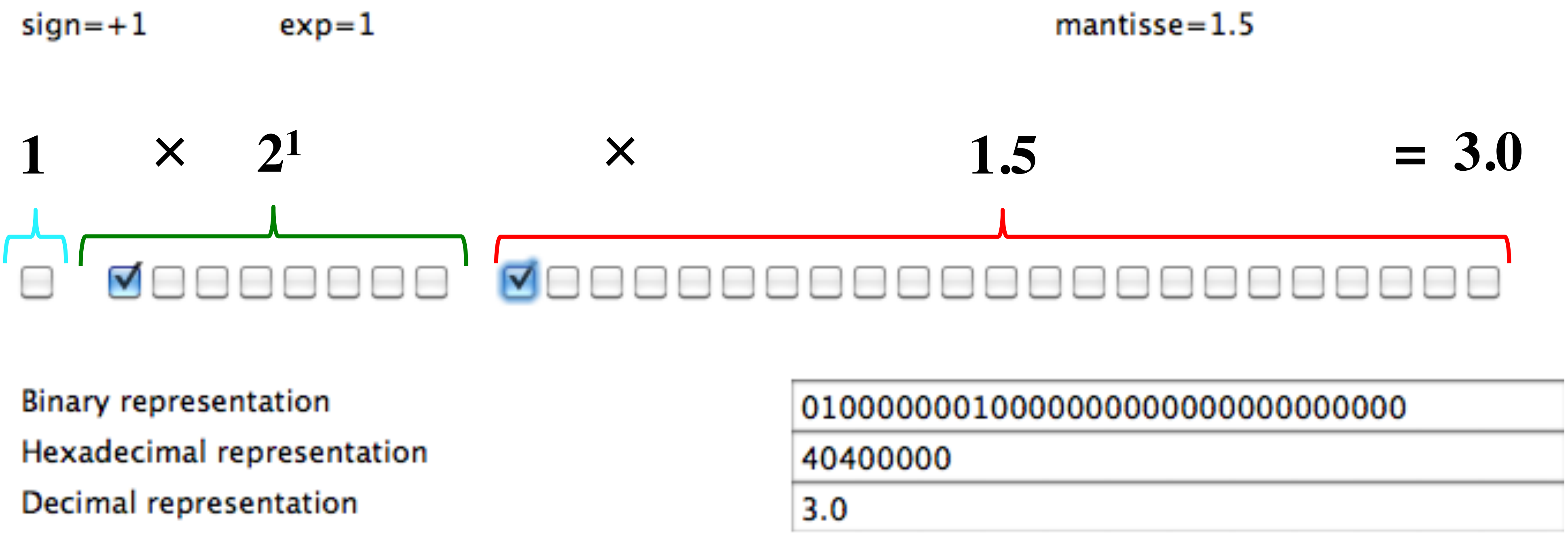
$$(-1)^{0} \times 2^{(128-127)} \times (1 + 0.5) = 3.0$$

☐ ☑☐☐☐☐☐☐☐ ☑☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	01000000010000000000000000000000
Hexadecimal representation	40400000
Decimal representation	3.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 3.0



sign=+1 exp=1 mantisse=1.5

$1 \times 2^1 \times 1.5 = 3.0$

☐ ☑☐☐☐☐☐☐☐ ☑☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

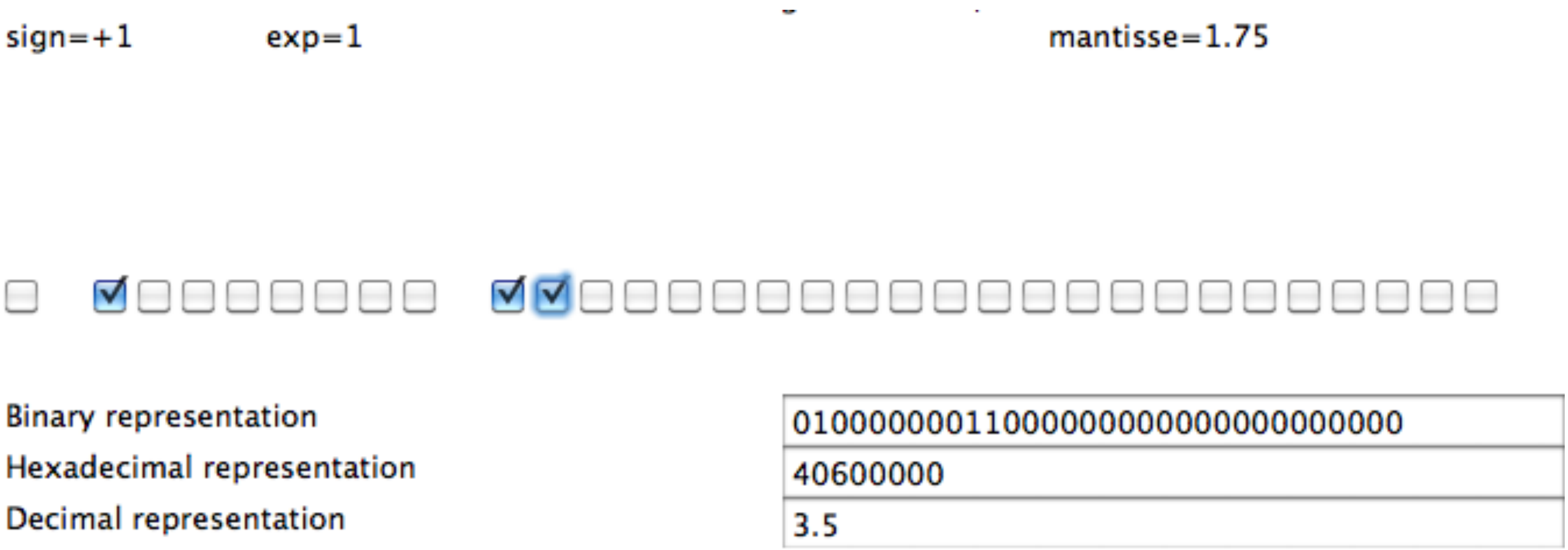
Binary representation: 01000000010000000000000000000000

Hexadecimal representation: 40400000

Decimal representation: 3.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 3.5



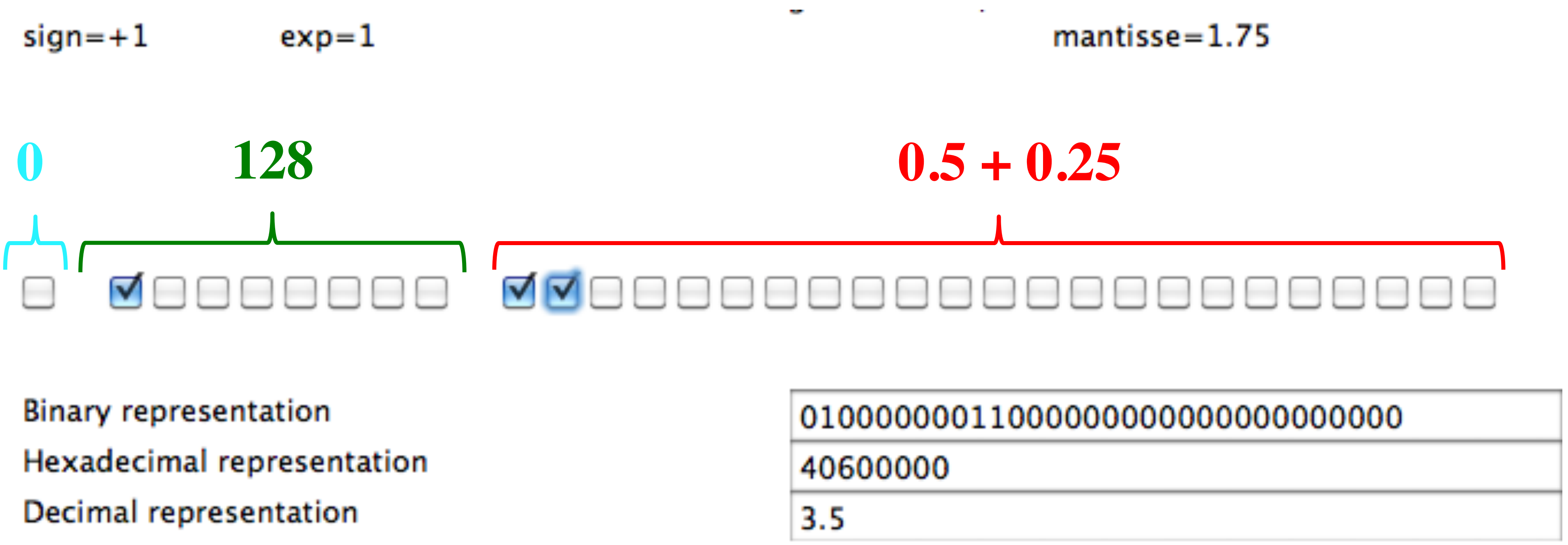
sign=+1 exp=1 mantisse=1.75

☐ ☑☐☐☐☐☐☐☐ ☑☑☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	01000000011000000000000000000000
Hexadecimal representation	40600000
Decimal representation	3.5

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 3.5



sign=+1 exp=1 mantisse=1.75

☐ ☑☐☐☐☐☐☐☐ ☑☑☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

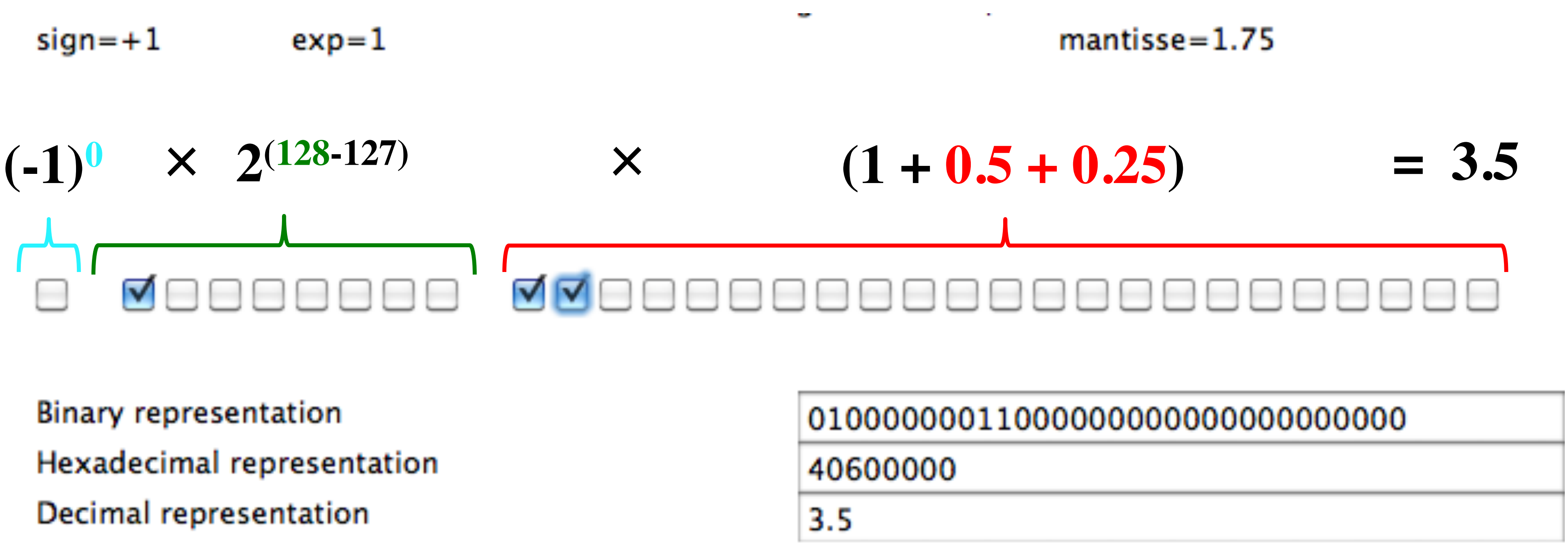
Binary representation: 01000000011000000000000000000000

Hexadecimal representation: 40600000

Decimal representation: 3.5

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 3.5



sign=+1 exp=1 mantisse=1.75

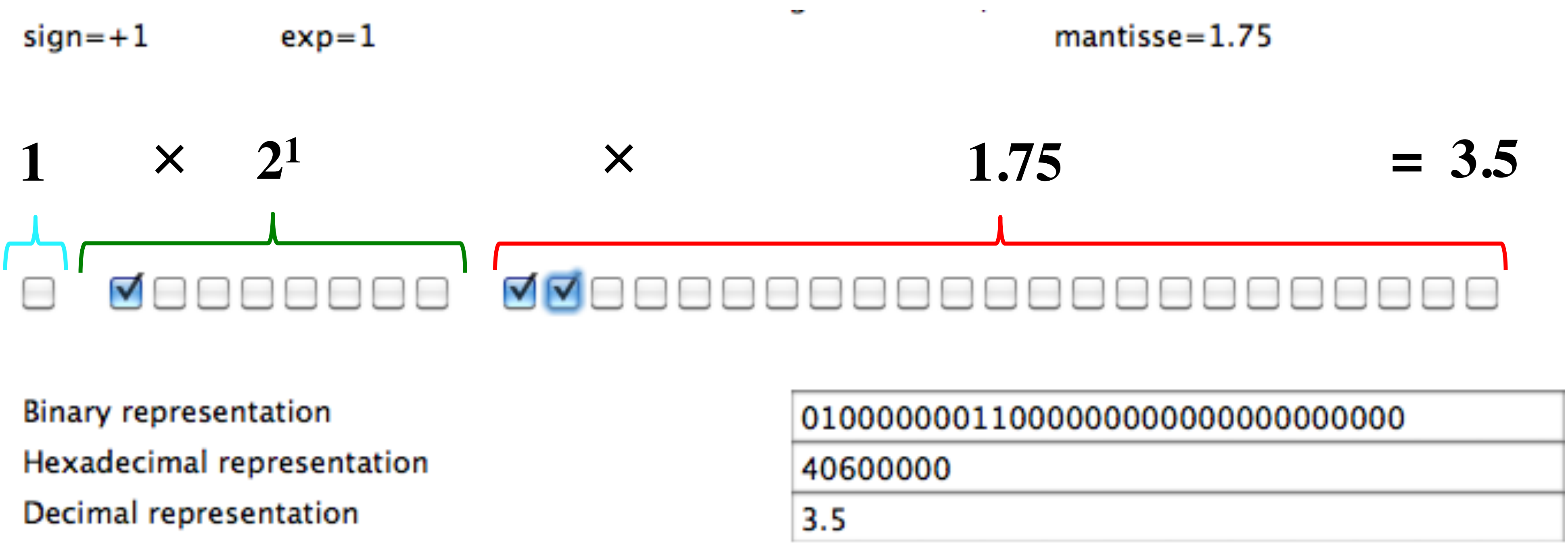
$$(-1)^{0} \times 2^{(128-127)} \times (1 + 0.5 + 0.25) = 3.5$$

☐ ☑☐☐☐☐☐☐☐ ☑☑☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	01000000011000000000000000000000
Hexadecimal representation	40600000
Decimal representation	3.5

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 3.5



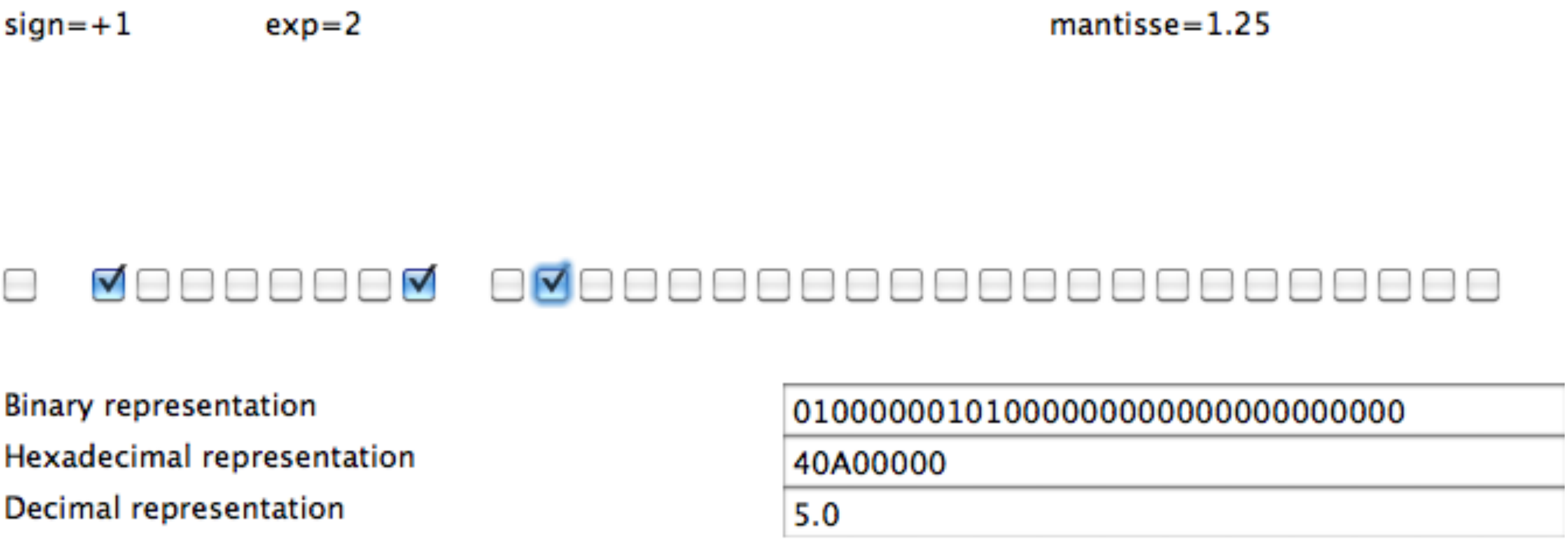
$$1 \times 2^1 \times 1.75 = 3.5$$

☐ ☑☐☐☐☐☐☐☐ ☑☑☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	01000000011000000000000000000000
Hexadecimal representation	40600000
Decimal representation	3.5

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 5.0



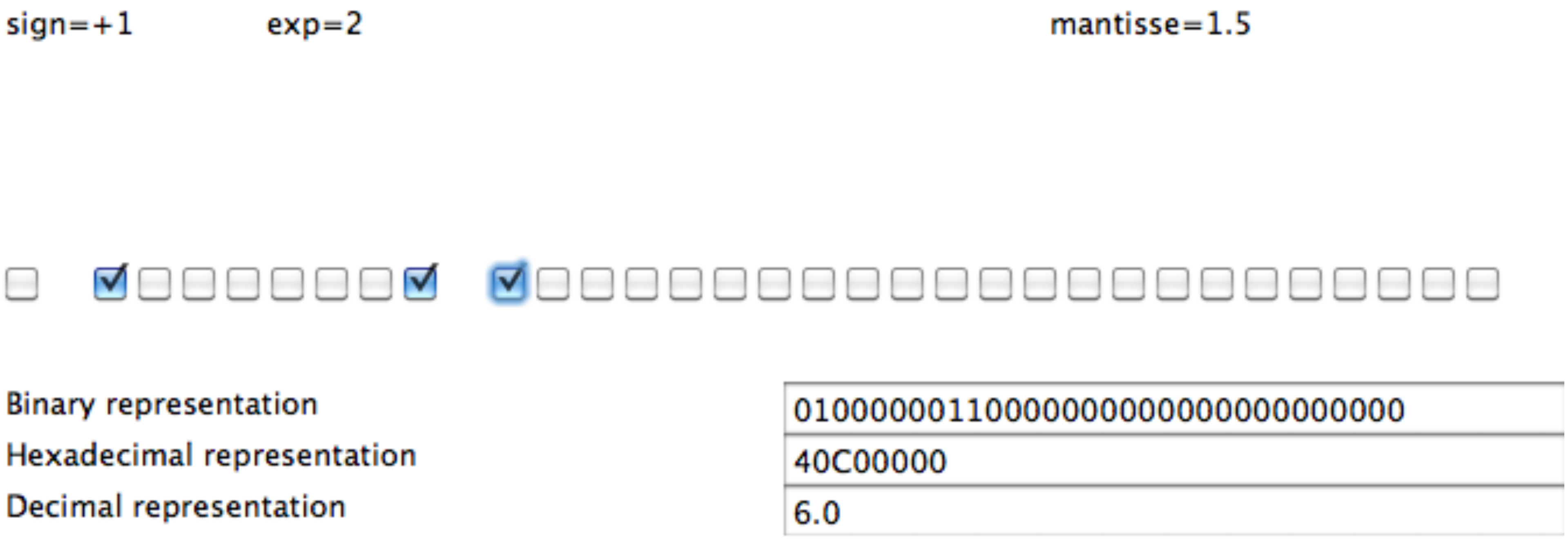
sign=+1 exp=2 mantisse=1.25

☐ ☑☐☐☐☐☐☐☑ ☐☑☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	01000000101000000000000000000000
Hexadecimal representation	40A00000
Decimal representation	5.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 6.0



sign=+1 exp=2 mantisse=1.5

☐ ☑☐☐☐☐☐☐☑ ☑☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

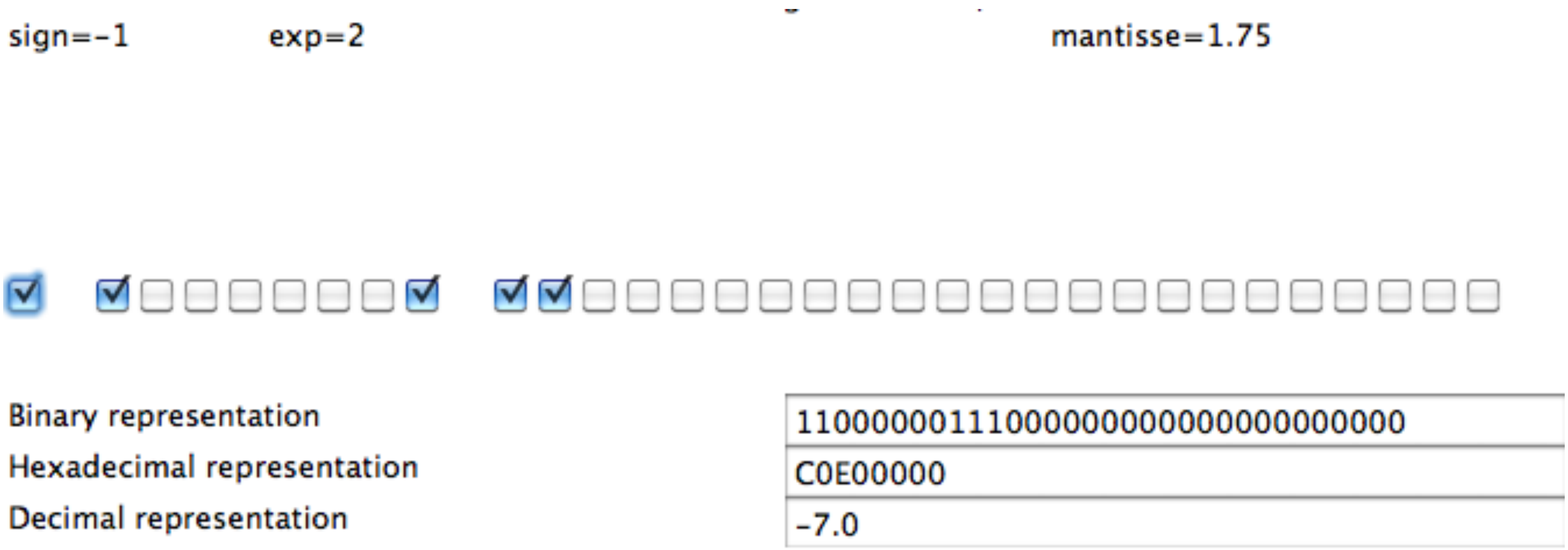
Binary representation: 01000000110000000000000000000000

Hexadecimal representation: 40C00000

Decimal representation: 6.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing -7.0



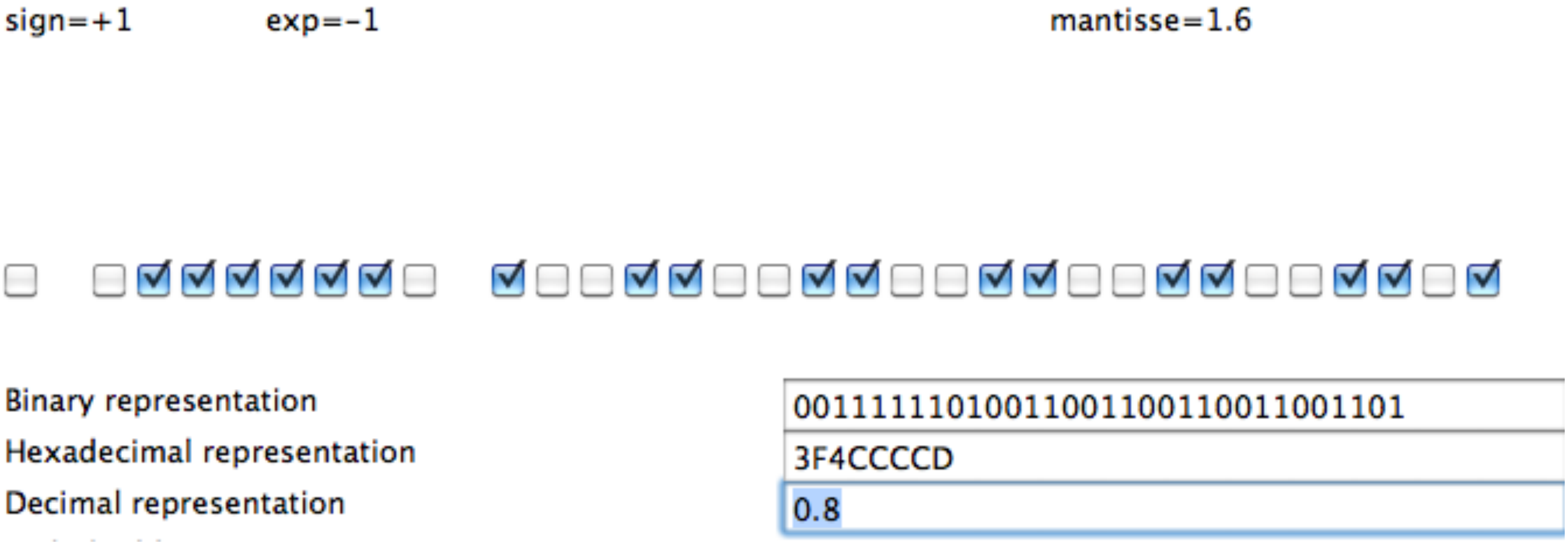
sign=-1 exp=2 mantisse=1.75

☑ ☑☐☐☐☐☐☐☑ ☑☑☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	11000000111000000000000000000000
Hexadecimal representation	C0E00000
Decimal representation	-7.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 0.8



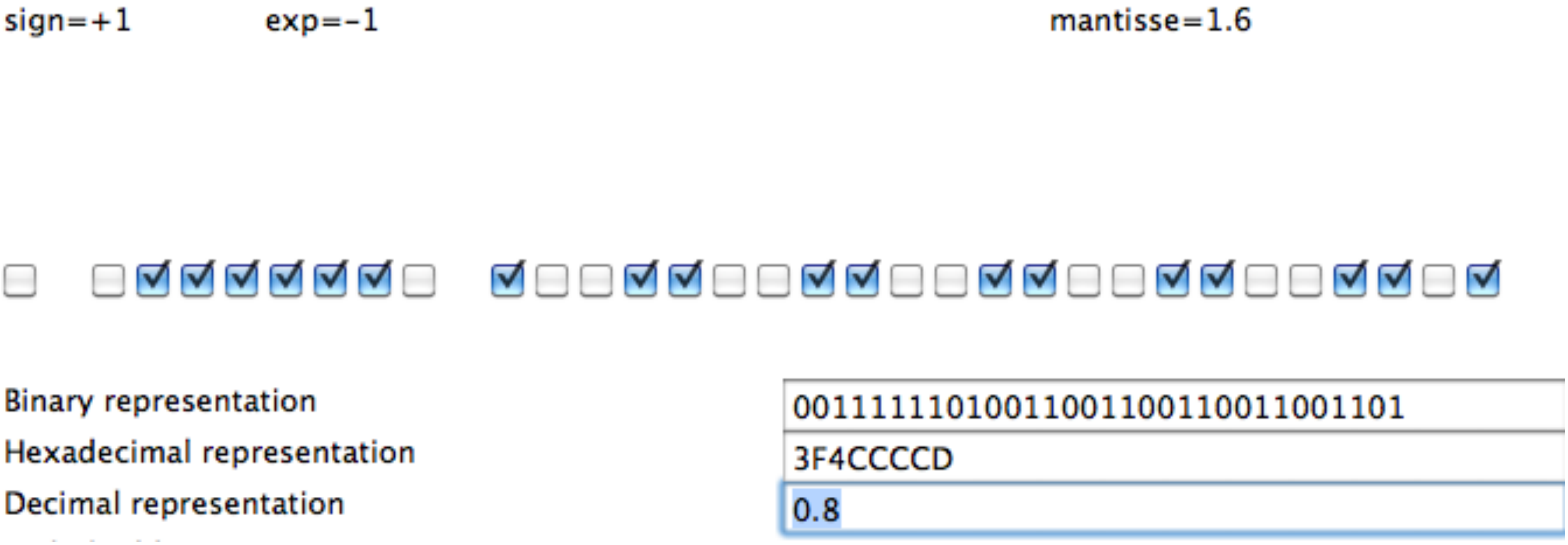
sign=+1 exp=-1 mantisse=1.6

☐ ☐☑☑☑☑☑☑☐ ☑☐☐☑☑☐☐☑☑☐☐☑☑☐☐☑☑☐☐☑☑☐☑

Binary representation	00111111010011001100110011001101
Hexadecimal representation	3F4CCCCD
Decimal representation	0.8

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 0.8



sign=+1 exp=-1 mantissa=1.6

☐ ☐☑☑☑☑☑☑☐ ☑☐☐☑☑☐☐☑☑☐☐☑☑☐☐☑☑☐☐☑☑☐☑

Binary representation 00111111010011001100110011001101

Hexadecimal representation 3F4CCCCD

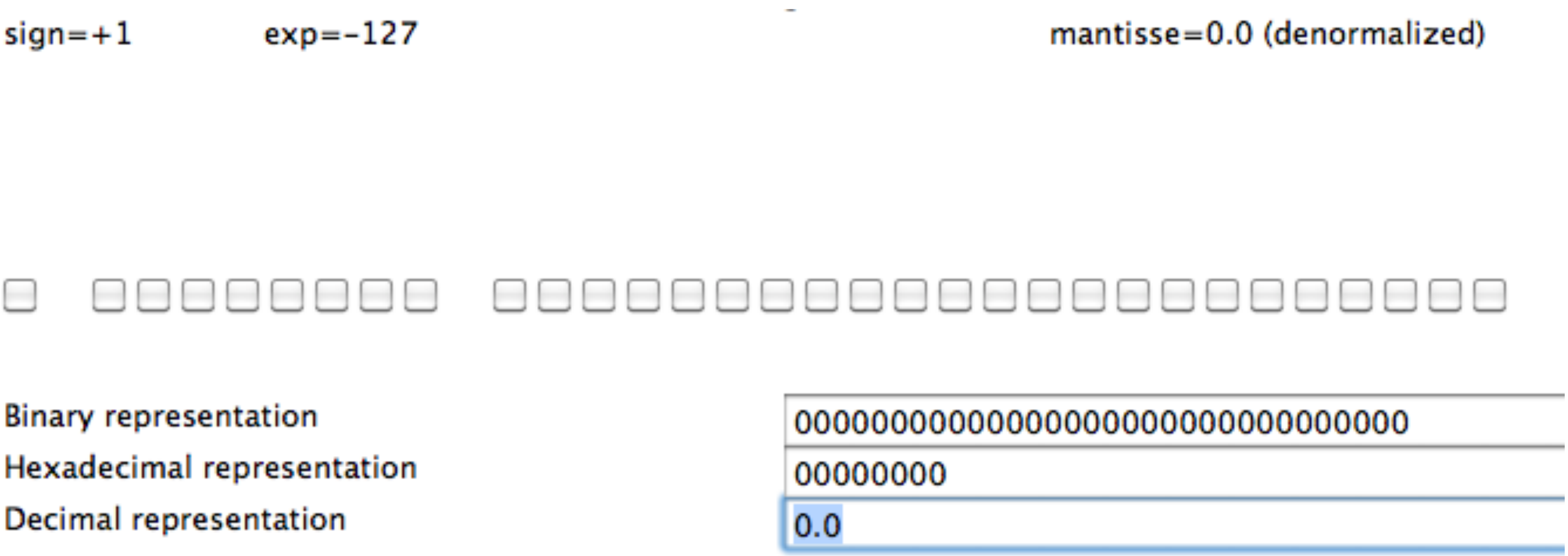
Decimal representation 0.8

This decimal number cannot be stored perfectly in this format!
The bits in the mantissa are periodic and will extend to infinity.

Think of storing 1/3 = 0.33333(3) with fixed number of decimal digits.
This is similar: 0.8_{10} has no finite representation in IEEE 754.

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing 0.0



sign=+1 exp=-127 mantisse=0.0 (denormalized)

☐ ☐☐☐☐☐☐☐☐ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

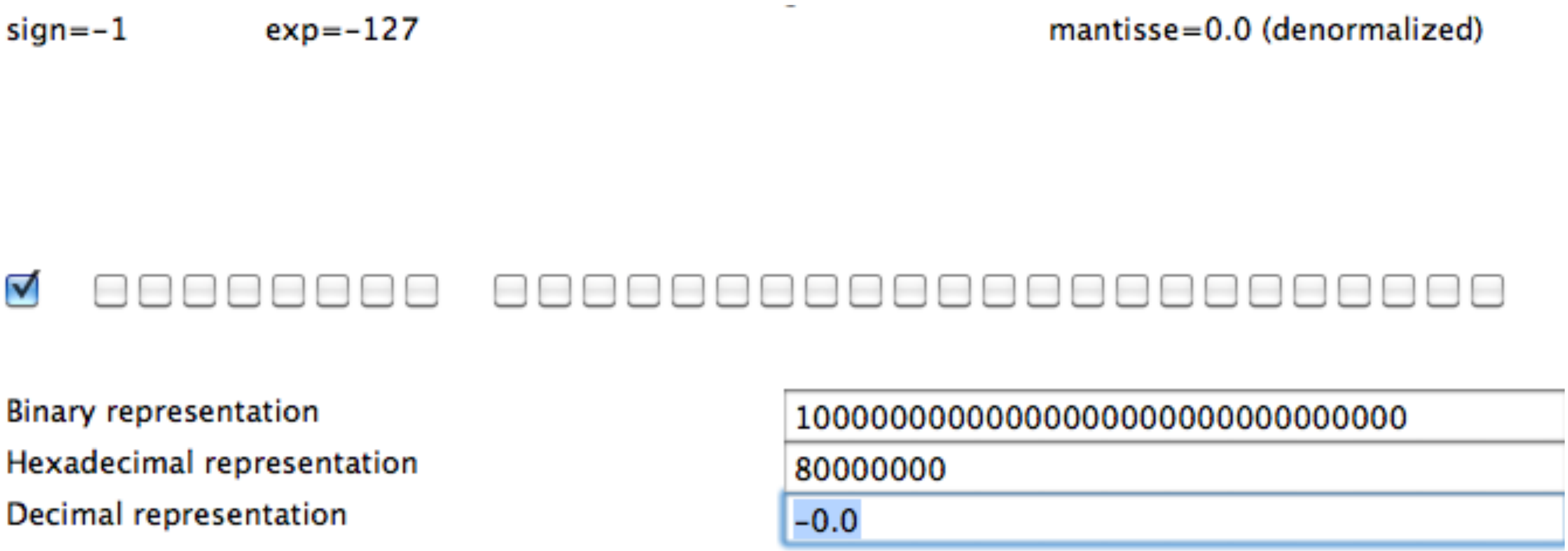
Binary representation 00000000000000000000000000000000

Hexadecimal representation 00000000

Decimal representation 0.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing -0.0



sign=-1 exp=-127 mantisse=0.0 (denormalized)

☑ ☐☐☐☐☐☐☐☐ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

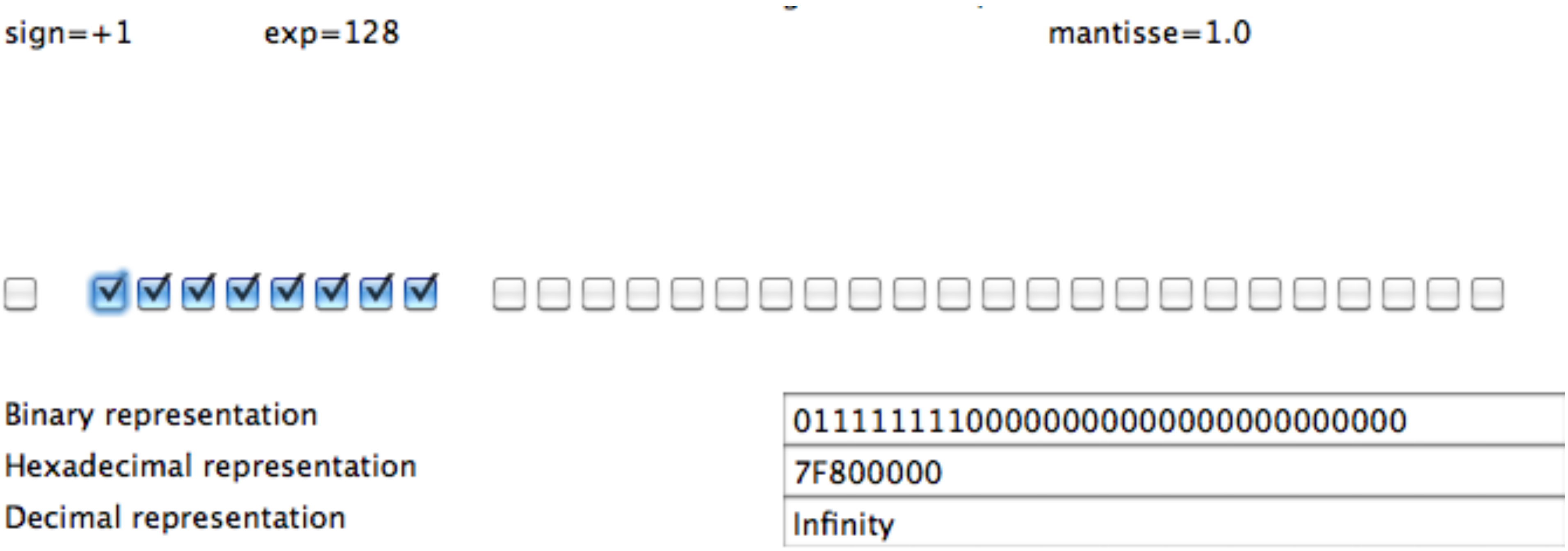
Binary representation 10000000000000000000000000000000

Hexadecimal representation 80000000

Decimal representation -0.0

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing +Infinity



sign=+1 exp=128 mantissa=1.0

☐ ☑☑☑☑☑☑☑☑ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

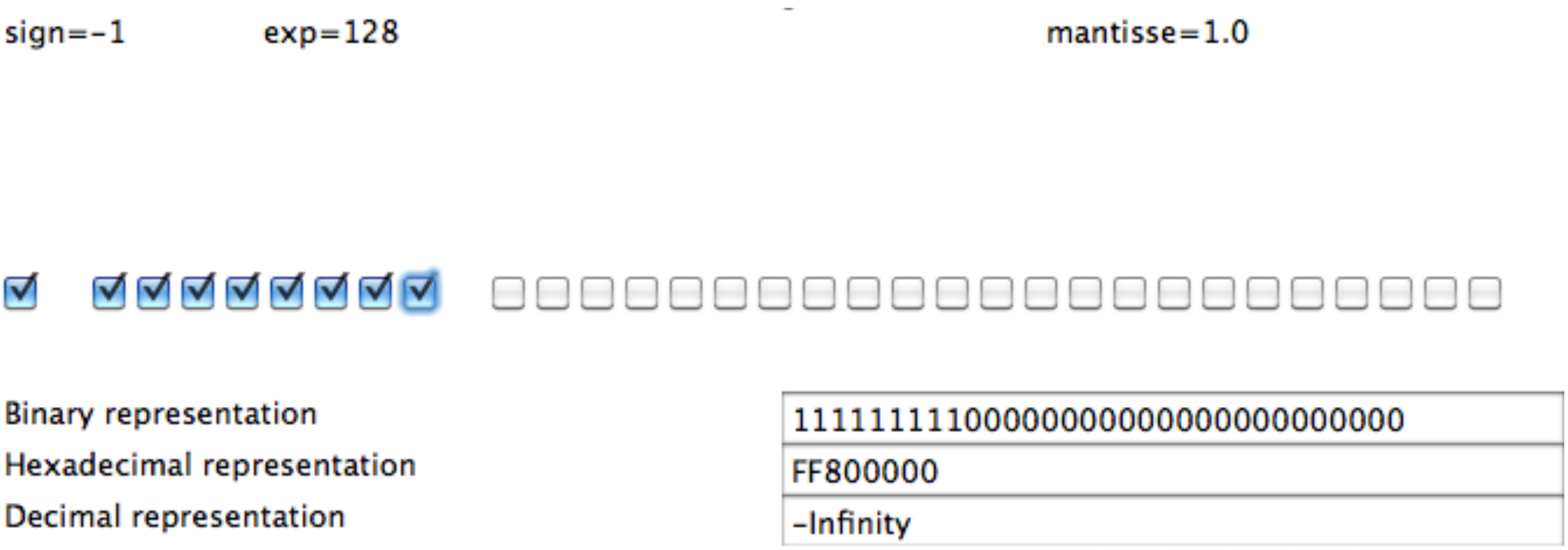
Binary representation: 01111111100000000000000000000000

Hexadecimal representation: 7F800000

Decimal representation: Infinity

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing -Infinity



sign=-1 exp=128 mantisse=1.0

☑ ☑☑☑☑☑☑☑☑ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

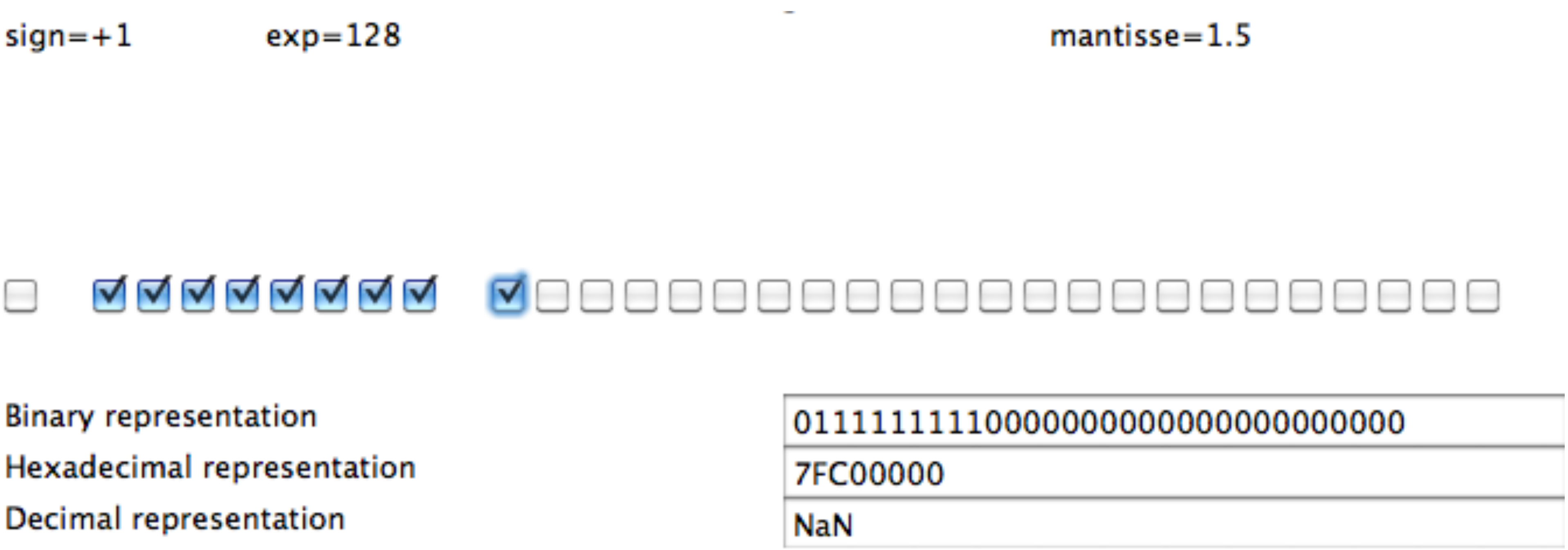
Binary representation: 11111111100000000000000000000000

Hexadecimal representation: FF800000

Decimal representation: -Infinity

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing NaN



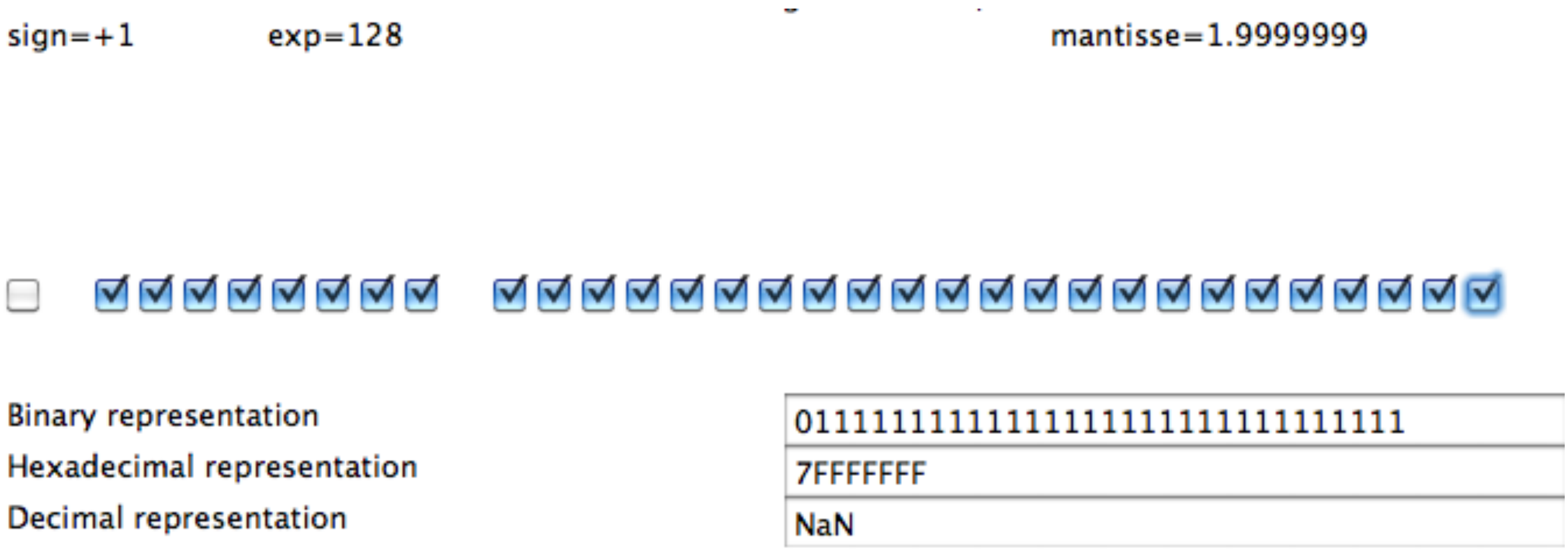
sign=+1 exp=128 mantisse=1.5

☐ ☑☑☑☑☑☑☑☑ ☑☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Binary representation	01111111110000000000000000000000
Hexadecimal representation	7FC00000
Decimal representation	NaN

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing NaN



sign=+1 exp=128 mantisse=1.9999999

☐ ☑☑☑☑☑☑☑☑ ☑☑☑☑☑☑☑☑☑☑☑☑☑☑☑☑☑☑☑☑☑☑☑

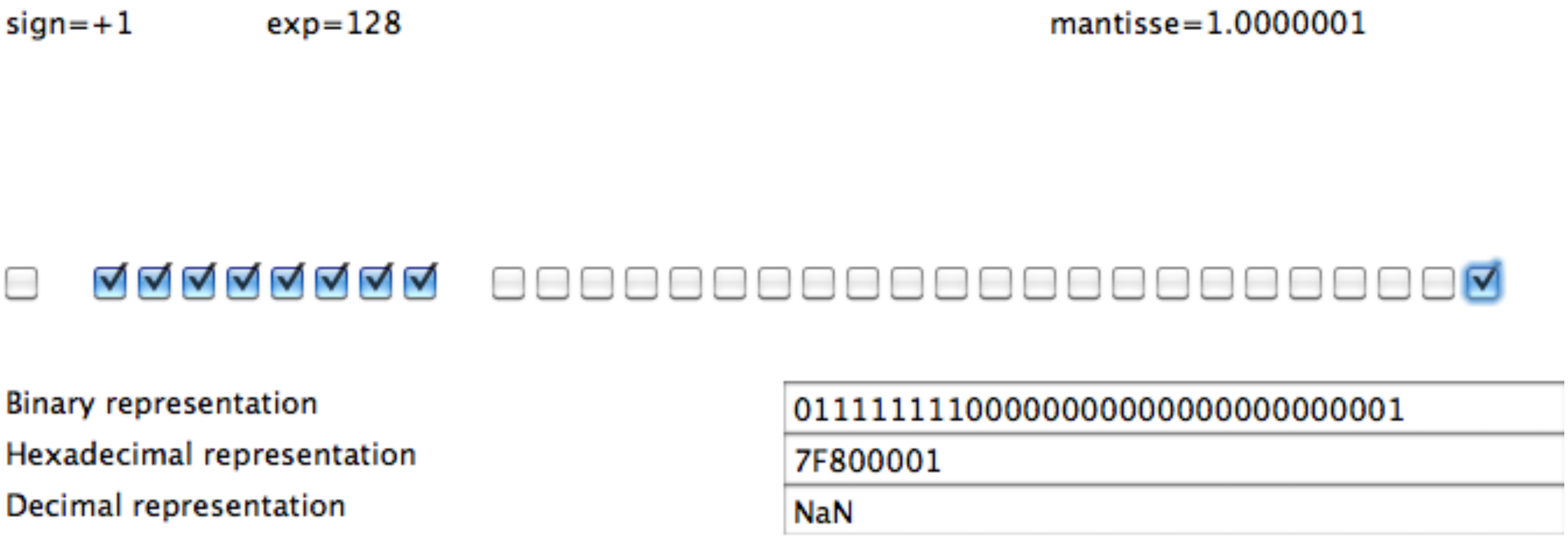
Binary representation 01111111111111111111111111111111

Hexadecimal representation 7FFFFFFF

Decimal representation NaN

[http://www.h-schmidt.net/FloatApplet/IEEE754.html]

Representing NaN

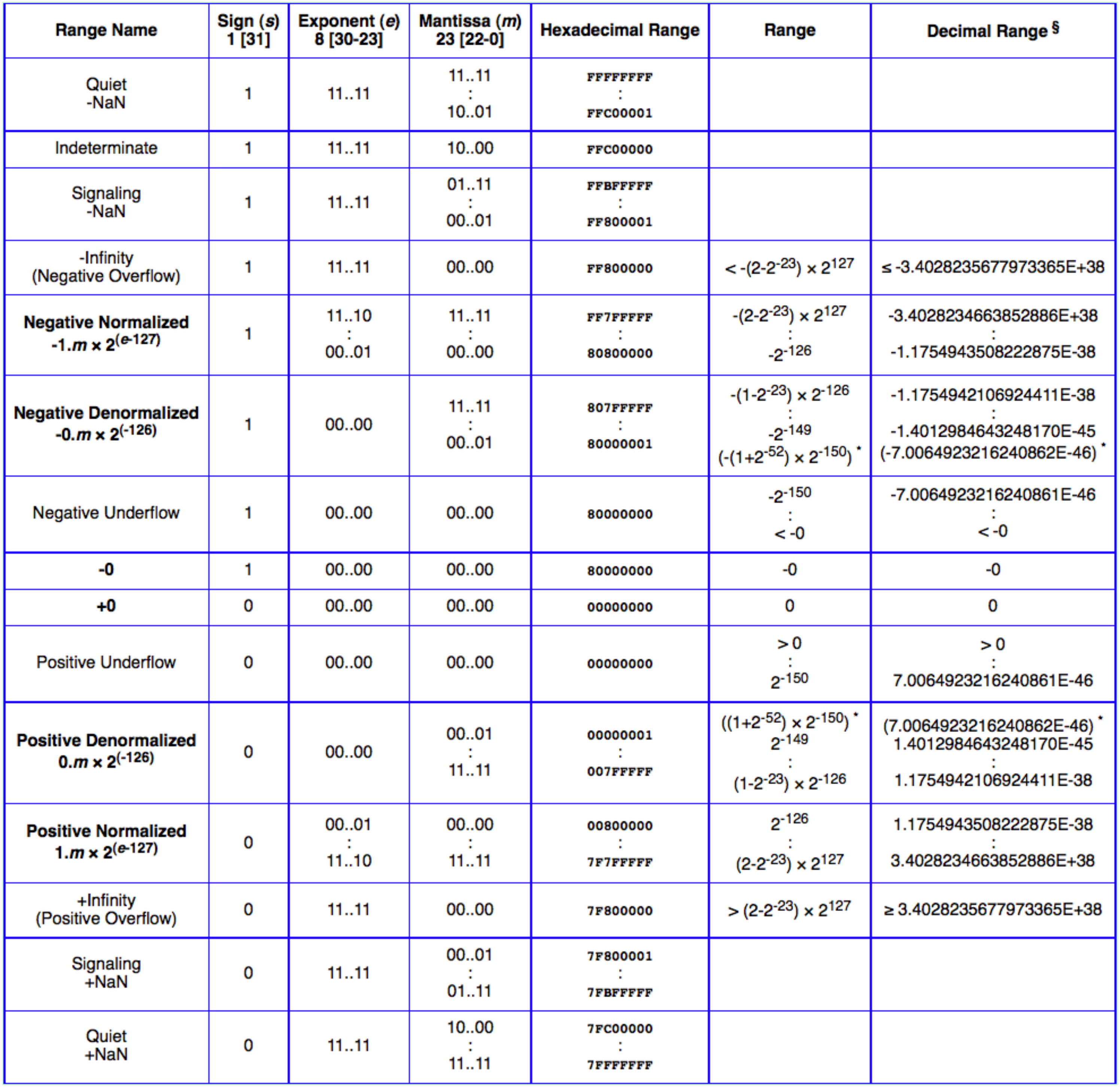


sign=+1 exp=128 mantisse=1.0000001

☐ ☑☑☑☑☑☑☑☑ ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☑

Binary representation	01111111100000000000000000000001
Hexadecimal representation	7F800001
Decimal representation	NaN

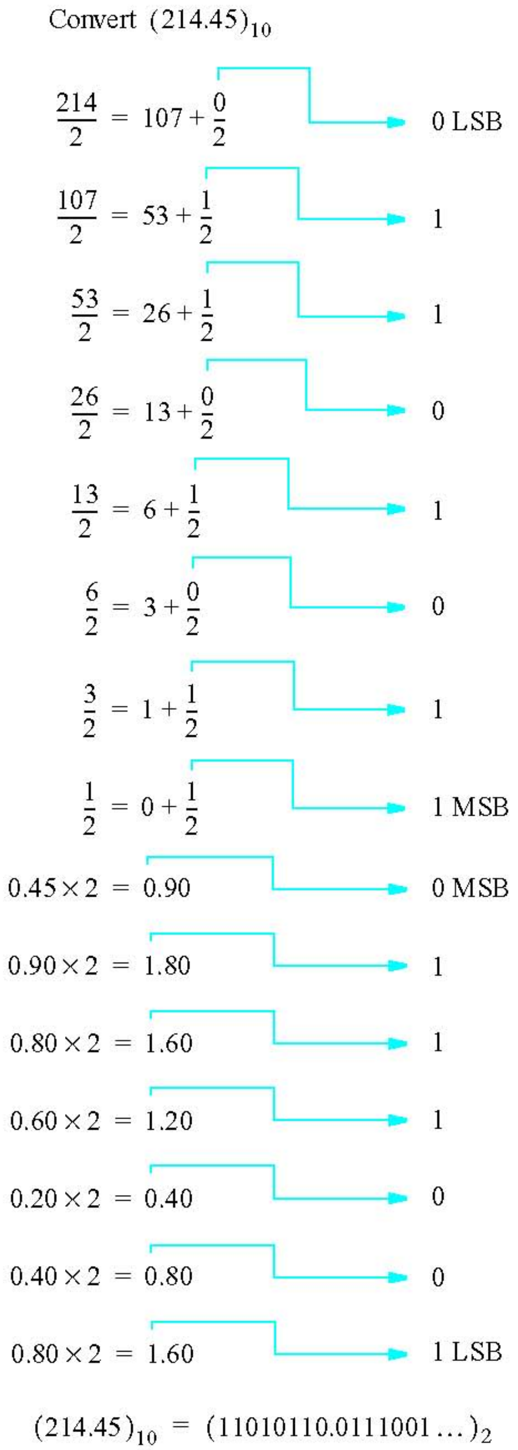
[http://www.h-schmidt.net/FloatApplet/IEEE754.html]



Range Name	Sign (s) 1 [31]	Exponent (e) 8 [30-23]	Mantissa (m) 23 [22-0]	Hexadecimal Range	Range	Decimal Range §
Quiet -NaN	1	11..11	11..11 : 10..01	FFFFFFFF : FFC00001		
Indeterminate	1	11..11	10..00	FFC00000		
Signaling -NaN	1	11..11	01..11 : 00..01	FFBFFFFF : FF800001		
-Infinity (Negative Overflow)	1	11..11	00..00	FF800000	$< -(2-2^{-23}) \times 2^{127}$	≤ -3.4028235677973365E+38
Negative Normalized **$-1.m \times 2^{(e-127)}$**	1	11..10 : 00..01	11..11 : 00..00	FF7FFFFF : 80800000	$-(2-2^{-23}) \times 2^{127}$: -2^{-126}	-3.4028234663852886E+38 : -1.1754943508222875E-38
Negative Denormalized **$-0.m \times 2^{(-126)}$**	1	00..00	11..11 : 00..01	807FFFFF : 80000001	$-(1-2^{-23}) \times 2^{-126}$: -2^{-149} $(-(1+2^{-52}) \times 2^{-150})$ *	-1.1754942106924411E-38 : -1.4012984643248170E-45 (-7.0064923216240862E-46) *
Negative Underflow	1	00..00	00..00	80000000	-2^{-150} : < -0	-7.0064923216240861E-46 : < -0
-0	1	00..00	00..00	80000000	-0	-0
+0	0	00..00	00..00	00000000	0	0
Positive Underflow	0	00..00	00..00	00000000	> 0 : 2^{-150}	> 0 : 7.0064923216240861E-46
Positive Denormalized **$0.m \times 2^{(-126)}$**	0	00..00	00..01 : 11..11	00000001 : 007FFFFF	$((1+2^{-52}) \times 2^{-150})$ * 2^{-149} : $(1-2^{-23}) \times 2^{-126}$	(7.0064923216240862E-46) * 1.4012984643248170E-45 : 1.1754942106924411E-38
Positive Normalized **$1.m \times 2^{(e-127)}$**	0	00..01 : 11..10	00..00 : 11..11	00800000 : 7F7FFFFF	2^{-126} : $(2-2^{-23}) \times 2^{127}$	1.1754943508222875E-38 : 3.4028234663852886E+38
+Infinity (Positive Overflow)	0	11..11	00..00	7F800000	$> (2-2^{-23}) \times 2^{127}$	≥ 3.4028235677973365E+38
Signaling +NaN	0	11..11	00..01 : 01..11	7F800001 : 7FBFFFFF		
Quiet +NaN	0	11..11	10..00 : 11..11	7FC00000 : 7FFFFFFF		

[http://babbage.cs.qc.cuny.edu/IEEE-754.old/References.xhtml]

Conversion of fixed point numbers from decimal to binary



Convert $(214.45)_{10}$

$\frac{214}{2} = 107 + \frac{0}{2}$	0 LSB
$\frac{107}{2} = 53 + \frac{1}{2}$	1
$\frac{53}{2} = 26 + \frac{1}{2}$	1
$\frac{26}{2} = 13 + \frac{0}{2}$	0
$\frac{13}{2} = 6 + \frac{1}{2}$	1
$\frac{6}{2} = 3 + \frac{0}{2}$	0
$\frac{3}{2} = 1 + \frac{1}{2}$	1
$\frac{1}{2} = 0 + \frac{1}{2}$	1 MSB
$0.45 \times 2 = 0.90$	0 MSB
$0.90 \times 2 = 1.80$	1
$0.80 \times 2 = 1.60$	1
$0.60 \times 2 = 1.20$	1
$0.20 \times 2 = 0.40$	0
$0.40 \times 2 = 0.80$	0
$0.80 \times 2 = 1.60$	1 LSB

$(214.45)_{10} = (11010110.0111001\ldots)_2$

[Figure 3.44 from the textbook]

Sample Midterm2 Problem

(a) Convert $3FA00000_{16}$ (a 32-bit float stored in IEEE 754 format) to decimal:

$$0\,\Big|\,\underbrace{011\;1111\;1}_{127}\,\Big|\,0\underset{\nwarrow\,2^{-2}}{1}0\;\;0000\;\;0000\;\;0000\;\;0000\;\;0000$$

$$(-1)^0 \times 2^{127-127} \times \left(1+\frac{1}{4}\right) = 2^0 \times \frac{5}{4} = 1.25$$

Sample Midterm2 Problem

(b) Convert the following 32-bit float number (in IEEE 754 format) to decimal

$$\underset{\text{negative}}{1}\ \Big|\ \underbrace{1\,0\,0\,0\,0\,1\,1\,0}_{128+4+2=134}\ \Big|\ 0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0$$

$$(-1)^1 \times 2^{134-127} \times \left(\overset{8}{1} + \overset{2}{\frac{1}{4}} + \frac{1}{8}\right) = -2^7 \times \frac{11}{8} = -\cancel{2^3} \times 2^4 \times \frac{11}{\cancel{2^3}} = -16 \times 11$$

$$= -176$$

Sample Midterm2 Problem

(c) Write down the 32-bit floating point representation (in IEEE 754 format) for 0110_2

$0110_2 = 6_{10}$

The highest power of 2 less than 6 is $2^2 = 4$.

$6/4 = 1.5$

-4

20

-20

0

$$6 = (-1)^0 \times \underbrace{2^2}_{2^{129-127}} \times \left(1 + \frac{1}{2}\right)$$

$$0 \,\vdots\, 10000001 \,\vdots\, 1\underbrace{000\ldots\ldots0}_{22 \text{ zeros}}$$

positive

Sample Midterm2 Problem

(d) Write down the 32-bit floating point representation (in IEEE 754 format) for -7_{10}

7/4= 1.75

−4

30

−28

20

−20

0

$$(-1)^1 \times \underbrace{2^2}_{2^{129-127}} \times \left(1 + \frac{1}{2} + \frac{1}{4}\right)$$

$$1 \vdots 1000000\underline{1} \vdots 11\underbrace{00\ldots\ldots\ldots0}_{21 \text{ zeros}}$$

negative

Memory Analogy

Address 0

Address 1

Address 2

Address 3

Address 4

Address 5

Address 6

Memory Analogy (32 bit architecture)

Address 0

Address 4

Address 8

Address 12

Address 16

Address 20

Address 24

Memory Analogy (32 bit architecture)

Address 0x00

Address 0x04

Address 0x08

Address 0x0C

Address 0x10

Address 0x14

Address 0x18

Hexadecimal

Address 0x0A

Address 0x0D

Storing a Double

Address 0x08

Address 0x0C

Storing 3.14

- **3.14 in binary IEEE-754 double precision (64 bits)**

sign	exponent	mantissa
0	10000000000	1001000111101011100001010001111010111000010100011111

- **In hexadecimal this is (hint: groups of four):**

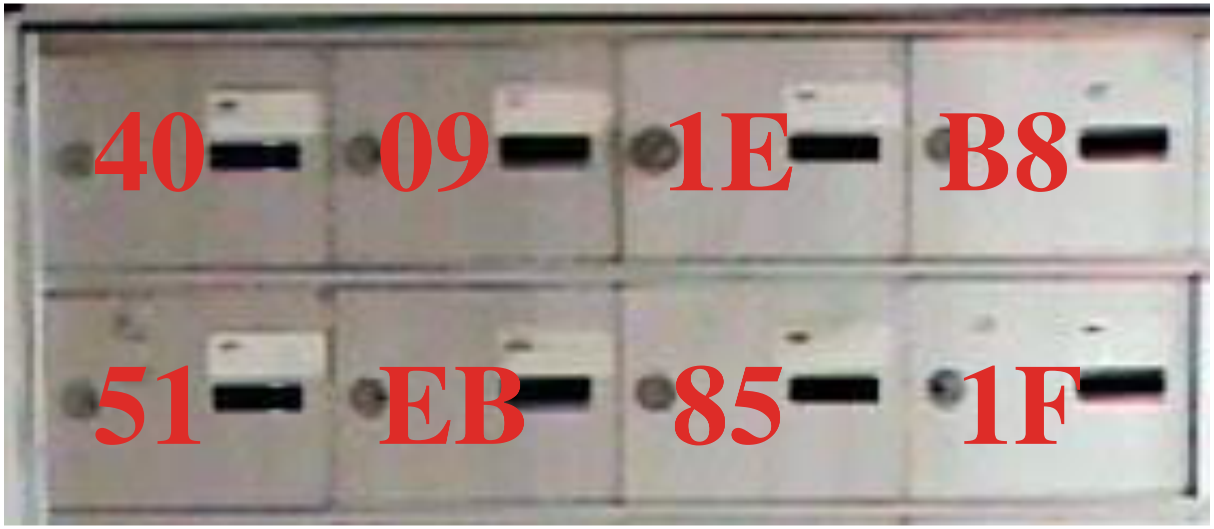
0100	0000	0000	1001	0001	1110	1011	1000	0101	0001	1110	1011	1000	0101	0001	1111
4	0	0	9	1	E	B	8	5	1	E	B	8	5	1	F

Storing 3.14

- **So 3.14 in hexadecimal IEEE-754 is 40091EB851EB851F**
- **This is 64 bits.**
- **On a 32 bit architecture there are 2 ways to store this**

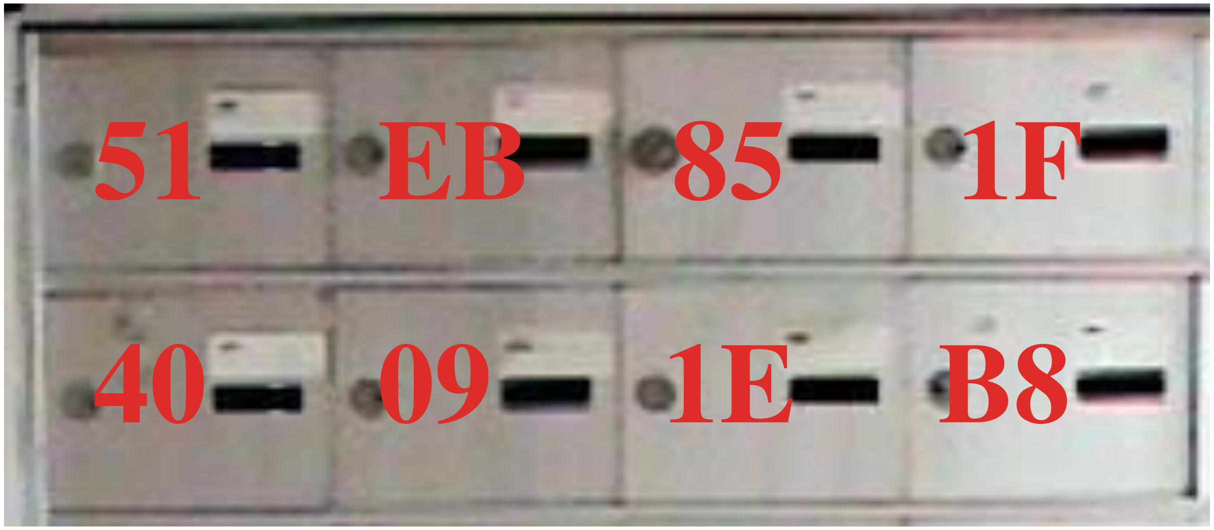
Small address:	**40091EB8**	**51EB851F**
Large address:	**51EB851F**	**40091EB8**
	Big-Endian	Little-Endian
Example CPUs:	Motorola 6800	Intel x86

Storing 3.14



Address 0x08	40	09	1E	B8
Address 0x0C	51	EB	85	1F

Big-Endian



Address 0x08	51	EB	85	1F
Address 0x0C	40	09	1E	B8

Little-Endian

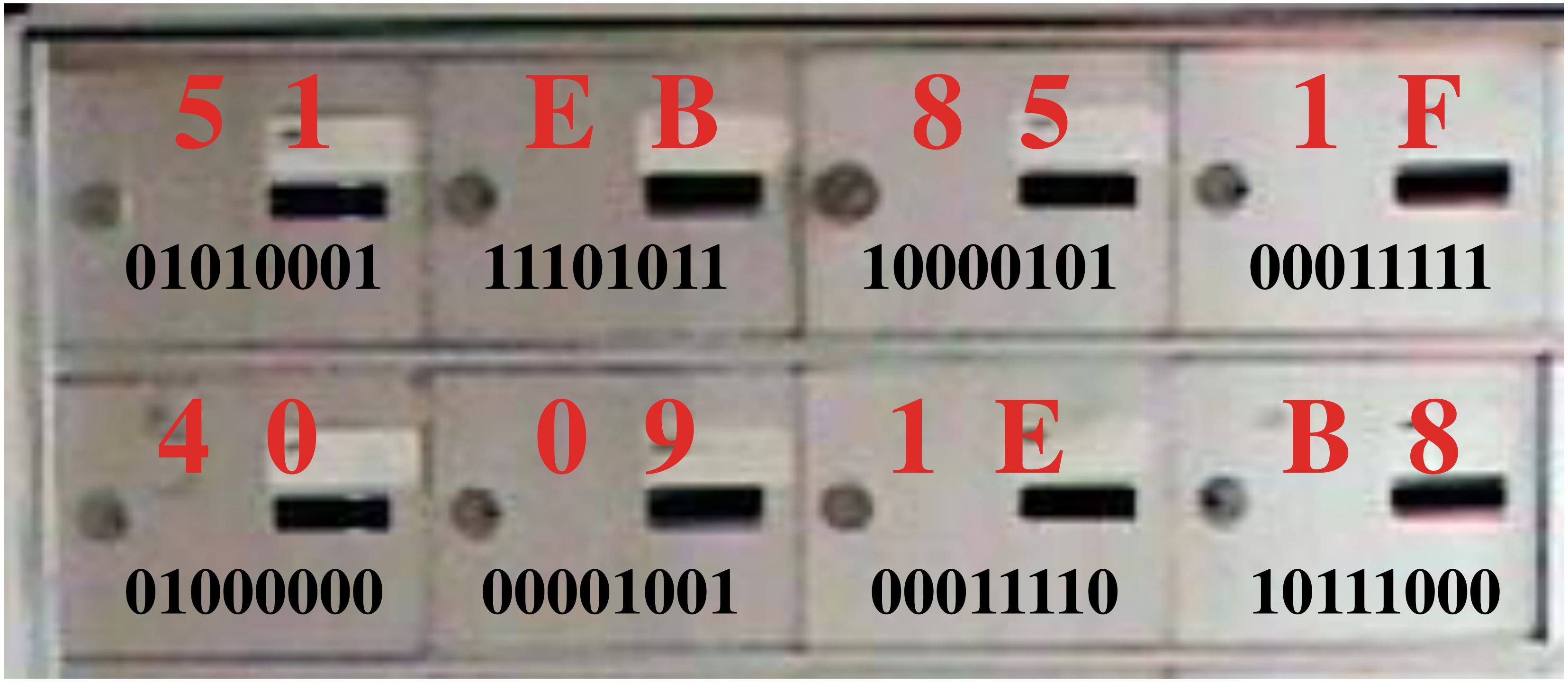
Storing 3.14 on a Little-Endian Machine (these are the actual bits that are stored)

Address 0x08	01010001	11101011	10000101	00011111
Address 0x0C	01000000	00001001	00011110	10111000

Once again, 3.14 in IEEE-754 double precision is:

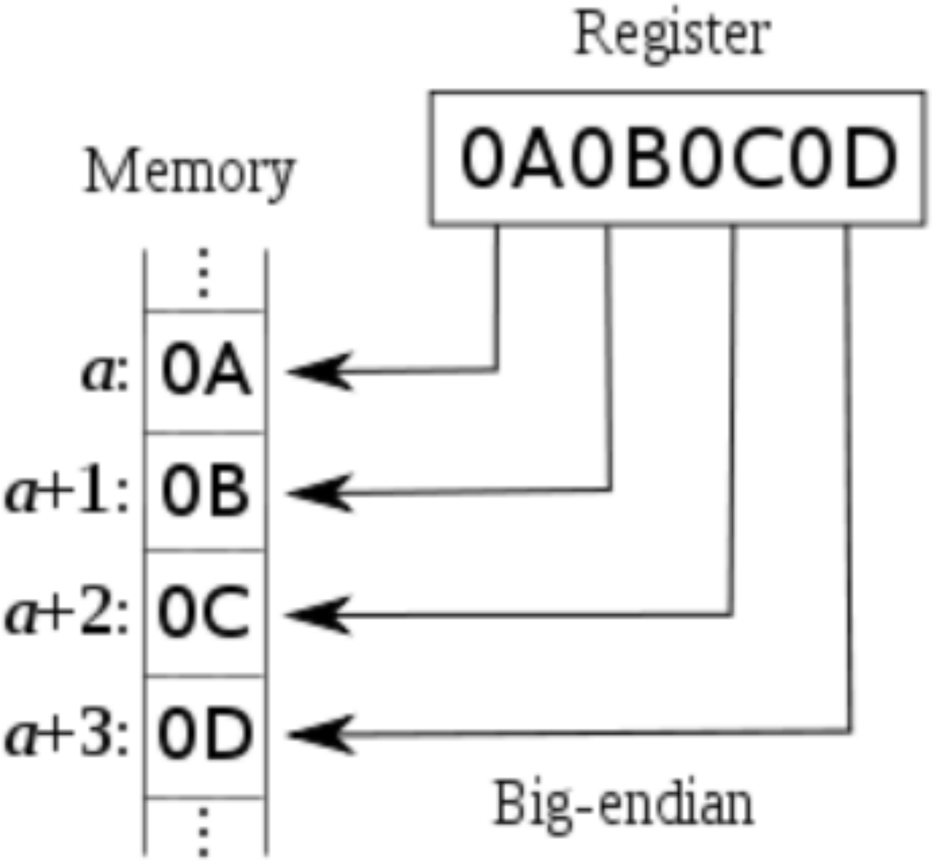
sign	exponent	mantissa
0	10000000000	1001000111101011100001010001111010111000010100011111

They are stored in binary
(the hexadecimals are just for visualization)



Address 0x08	5 1 01010001	E B 11101011	8 5 10000101	1 F 00011111
Address 0x0C	4 0 01000000	0 9 00001001	1 E 00011110	B 8 10111000

Big-Endian



increasing addresses →

...	0x0A	0x0B	0x0C	0x0D	...

(8-bit architecture)

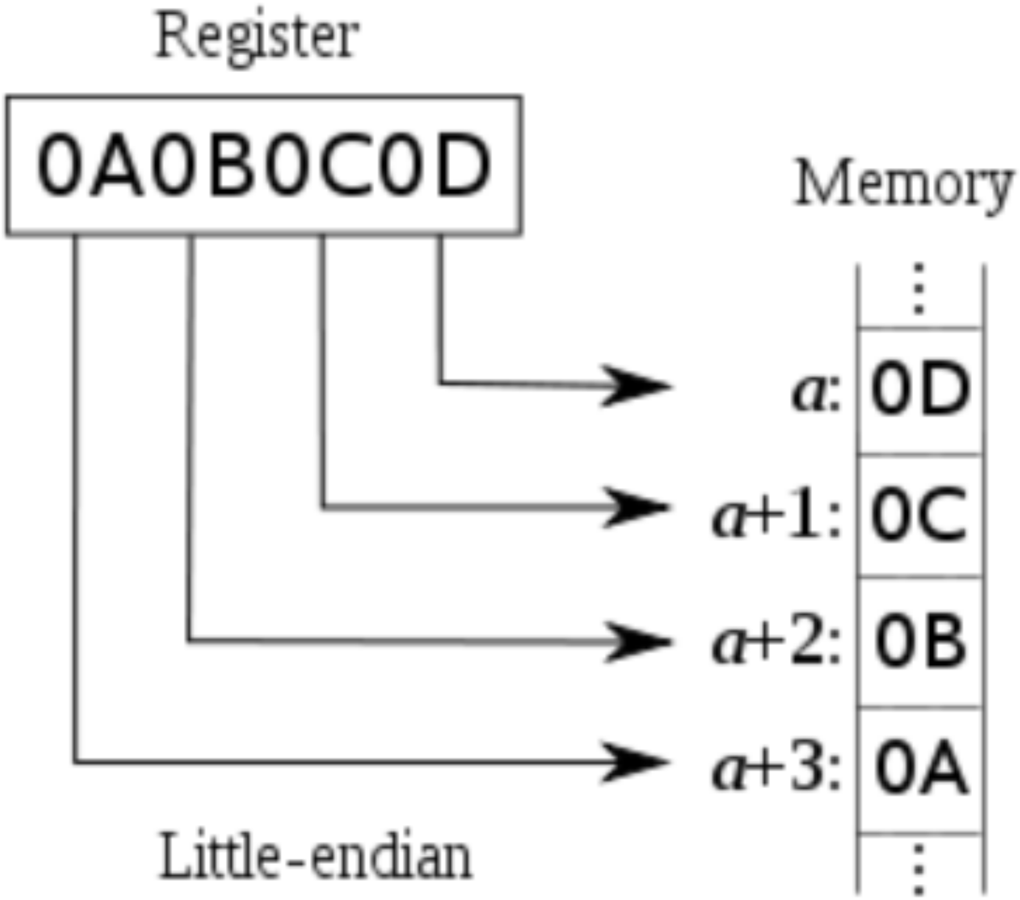
increasing addresses →

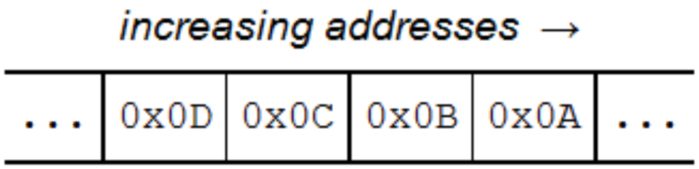
...	0x0A0B	0x0C0D	...

(16-bit architecture)

http://en.wikipedia.org/wiki/Endianness

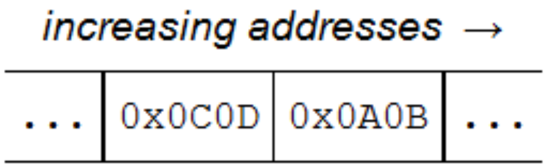
Little Endian




increasing addresses →

...	0x0D	0x0C	0x0B	0x0A	...

(8-bit architecture)


increasing addresses →

...	0x0C0D	0x0A0B	...

(16-bit architecture)

http://en.wikipedia.org/wiki/Endianness

Big-Endian/Little-Endian analogy

[image fom http://www.simplylockers.co.uk/images/PLowLocker.gif]

Big-Endian/Little-Endian analogy

[image fom http://www.simplylockers.co.uk/images/PLowLocker.gif]

Big-Endian/Little-Endian analogy

[image fom http://www.simplylockers.co.uk/images/PLowLocker.gif]

What would be printed?
(don’t try this at home)

```
double pi = 3.14;
printf(“%d”,pi);
```

- **Result: 1374389535**

Why?

- **3.14 = 40091EB851EB851F (in double format)**
- **Stored on a little-endian 32-bit architecture**
  - **51EB851F (1374389535 in decimal)**
  - **40091EB8 (1074339512 in decimal)**

What would be printed?
(don’t try this at home)

```
double pi = 3.14;
printf(“%d %d”, pi);
```

- **Result: 1374389535 1074339512**

Why?

- **3.14 = 40091EB851EB851F (in double format)**
- **Stored on a little-endian 32-bit architecture**
  - **51EB851F (1374389535 in decimal)**
  - **40091EB8 (1074339512 in decimal)**

  - **The second %d uses the extra bytes of pi that were not printed by the first %d**

What would be printed? (don’t try this at home)

```
double a = 2.0;
printf(“%d”,a);
```

- **Result: 0**

Why?

- 2.0 = 40000000 00000000 (in hex IEEE double format)
- Stored on a little-endian 32-bit architecture
  - 00000000 (0 in decimal)
  - 40000000 (1073741824 in decimal)

What would be printed? (an even more advanced example)

```
int a[2];                  // defines an int array
a[0]=0;
a[1]=0;
scanf("%lf", &a[0]);      // read 64 bits into 32 bits
// The user enters 3.14
printf("%d %d", a[0], a[1]);
```

- **Result: 1374389535 1074339512**

Why?

- **3.14 = 40091EB851EB851F (in double format)**
- **Stored on a little-endian 32-bit architecture**
  - **51EB851F (1374389535 in decimal)**
  - **40091EB8 (1074339512 in decimal)**
- **The double 3.14 requires 64 bits which are stored in the two consecutive 32-bit integers named a[0] and a[1]**

Questions?

THE END