

Computer Organization and Networks

(INB.06000UF, INB.07001UF)

Chapter 2 – Number Representation

Winter 2022/2023



Stefan Mangard, www.iaik.tugraz.at

How to Represent Information?

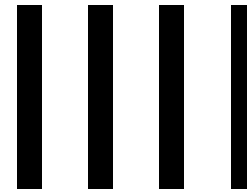
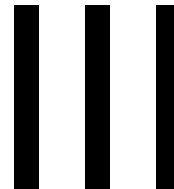
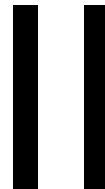
Combinational circuits allow to compute a logic function that takes N input wires and that calculates M output wires.

In order to create circuits for all kinds of data, we need to agree how to represent information. In particular, we need an agreement for:

- Numbers
- Text
- Pictures, photos
- Audio
- Video
- Executables



The Romans



The Romans (ver 2.0)



I



II



III



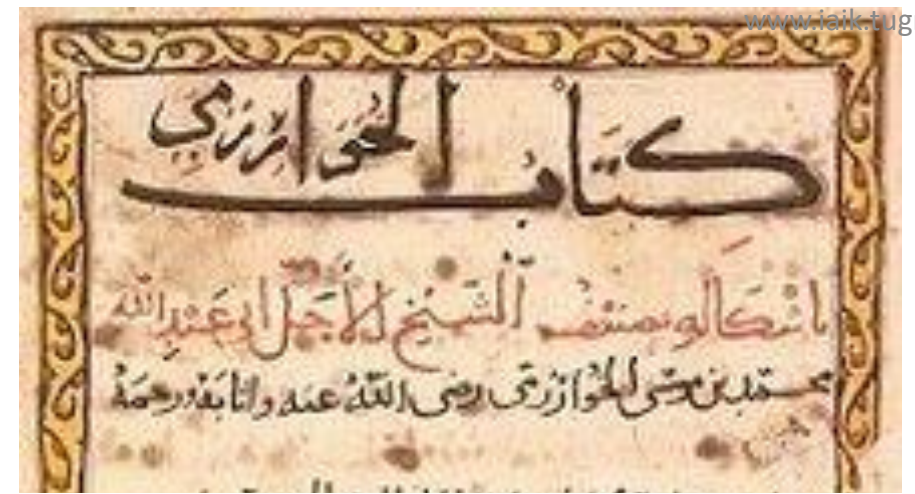
IV

How many symbols do we need / How to continue counting when running out of symbols?





Al-Chwarizmi introduced the positional number system

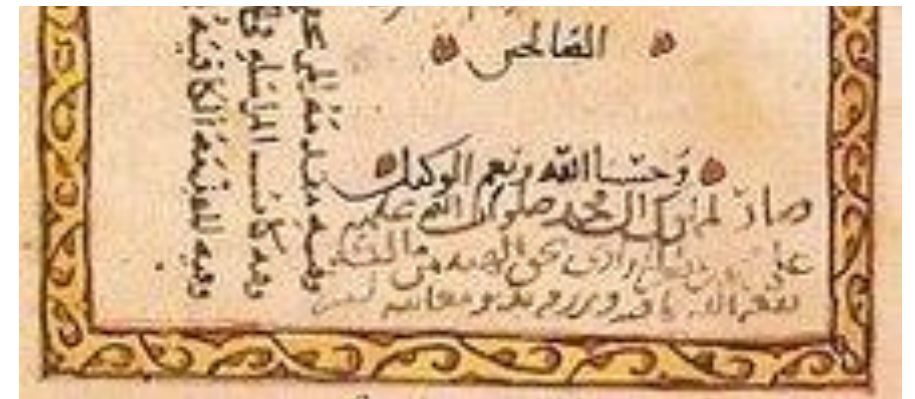


Muhammad ibn Musa al-Khwarizmi

From Wikipedia, the free encyclopedia

"al-Khwārizmī" redirects here. For other uses, see al-Khwārizmī (disambiguation).

Muḥammad ibn Mūsā al-Khwārizmī^[note 1] (Persian: محمد بن موسی خوارزمی Muḥammad Khwārizmī; c. 780 – c. 850), Arabized as al-Khwarizmi and formerly Latinized as *Algorithmi*, was a Persian^{[4][5][6]} polymath who produced vastly influential works in mathematics, astronomy, and geography. Around 820 CE he was appointed as the astronomer and head of the library of the House of Wisdom in Baghdad.^{[7]:14}



Base b

- Number system with base b:
 - ... b^3 b^2 b^1 b^0 . b^{-1} b^{-2} b^{-3} ...
 - b Symbols

Our number system

- Decimal system (base is „ten“):
 -
 - Hundredth 10^{-2}
 - Thenth 10^{-1}
 - **Ones** 10^0
 - Tens 10^1
 - Hundreds 10^2
 - Thousands 10^3
 - ...
- Ten symbols: 0, 1, 2, ..., 9
- With n positions: 10^n possible different values
- $2345 = 2 * 10^3 + 3 * 10^2 + 4 * 10^1 + 5 * 10^0$
- $345.67 = 3 * 100 + 4 * 10 + 5 * 1 + 6 * \frac{1}{10} + 7 * \frac{1}{100}$

Base $b = 2$

- ...
- Quarters 2^{-2}
- Halves 2^{-1}
- **Ones** 2^0
- Twos 2^1
- Fours 2^2
- Eights 2^3
- ...
- Two symbols: 0, 1
- With n positions: 2^n possible different values
- $(1011)_2 = 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$
- $(011.01101)_2 = 2 + 1 + \frac{1}{4} + \frac{1}{8} + \frac{1}{32}$

Counting with binary numbers

$$(0000)_2 = 0 + 0 + 0 + 0 = 0$$

$$(0001)_2 = 0 + 0 + 0 + 1 = 1$$

$$(0010)_2 = 0 + 0 + 2 + 0 = 2$$

$$(0011)_2 = 0 + 0 + 2 + 1 = 3$$

$$(0100)_2 = 0 + 4 + 0 + 0 = 4$$

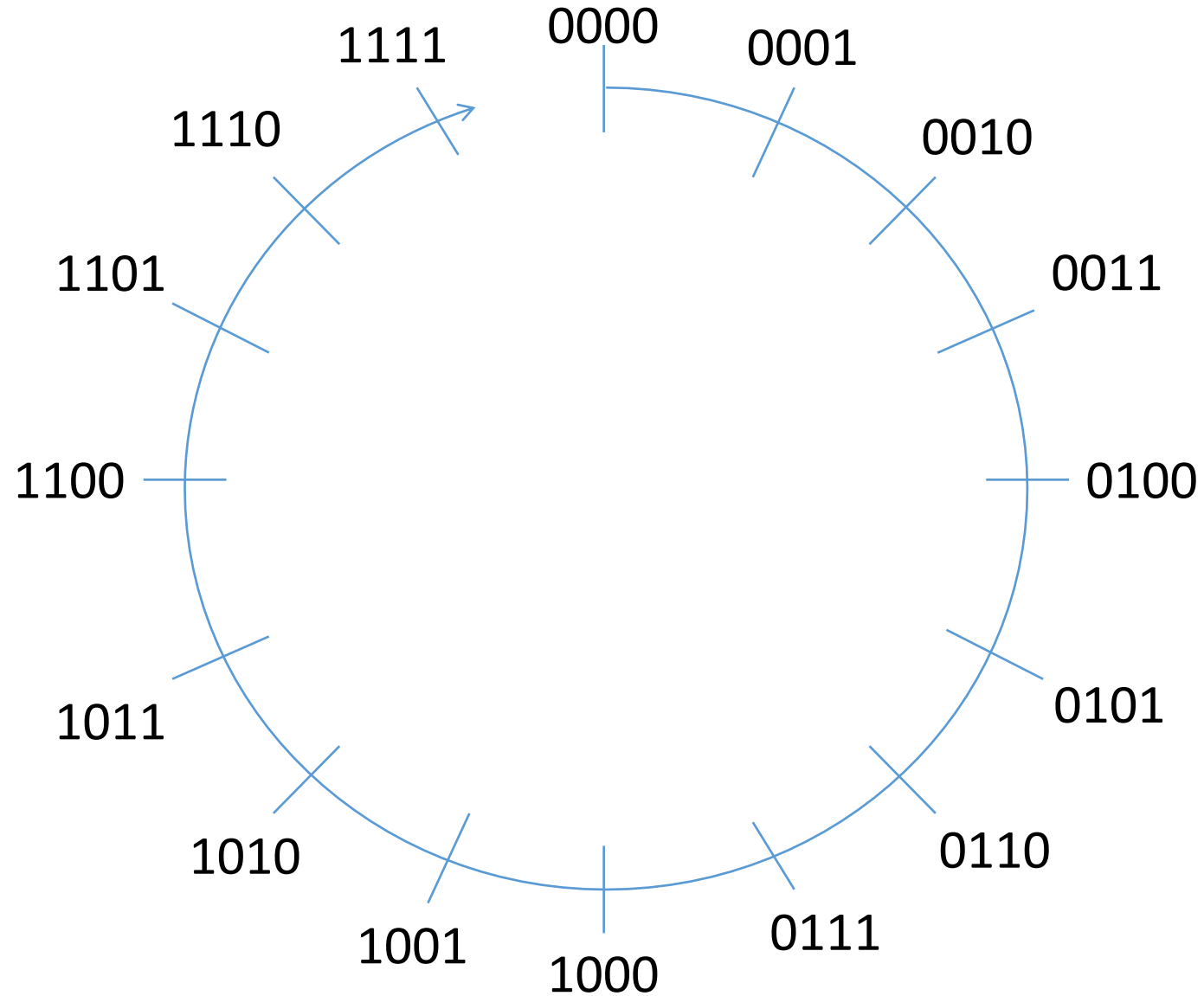
...

$$(1101)_2 = 8 + 4 + 0 + 1 = 13$$

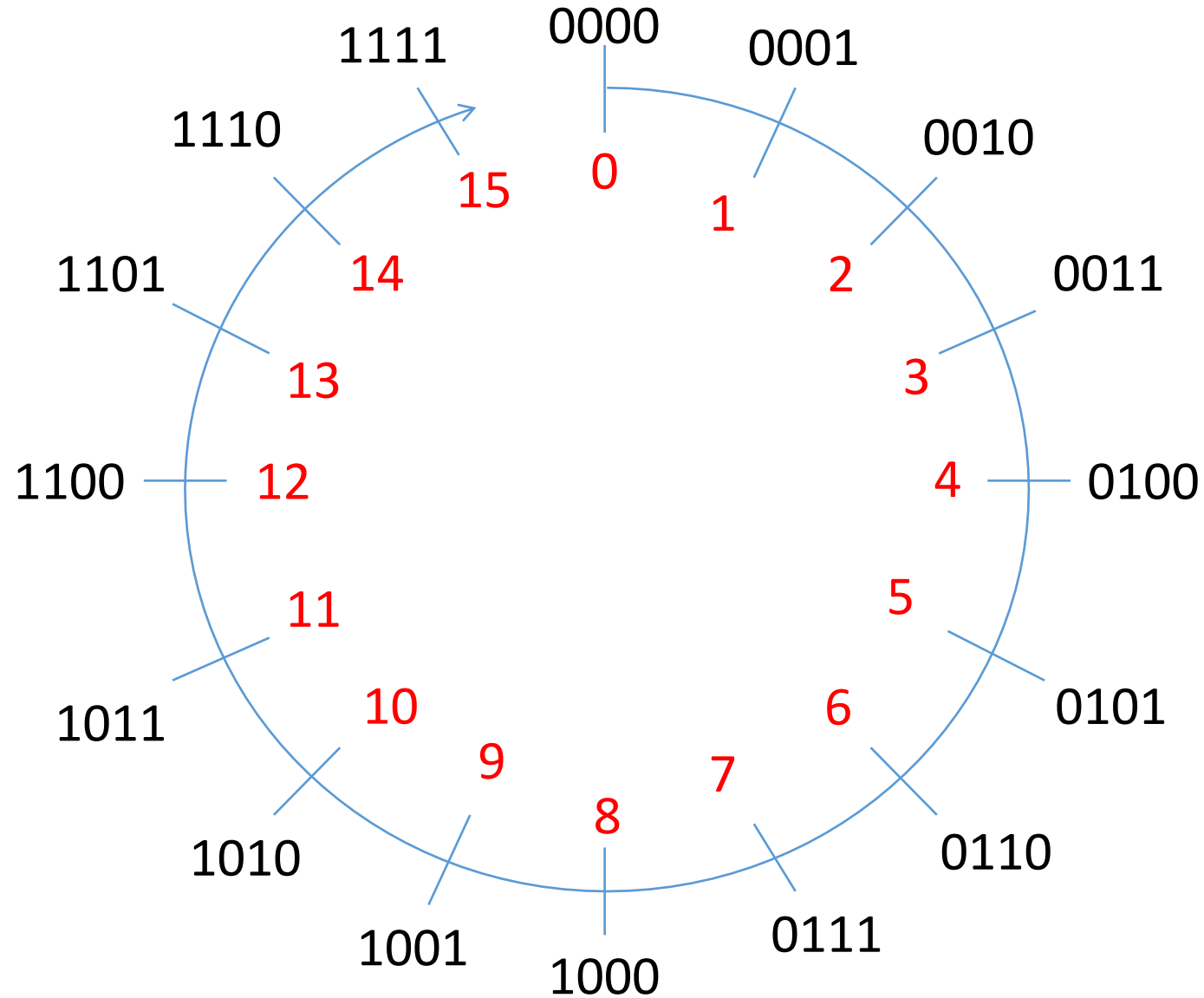
$$(1110)_2 = 8 + 4 + 2 + 0 = 14$$

$$(1111)_2 = 8 + 4 + 2 + 1 = 15$$

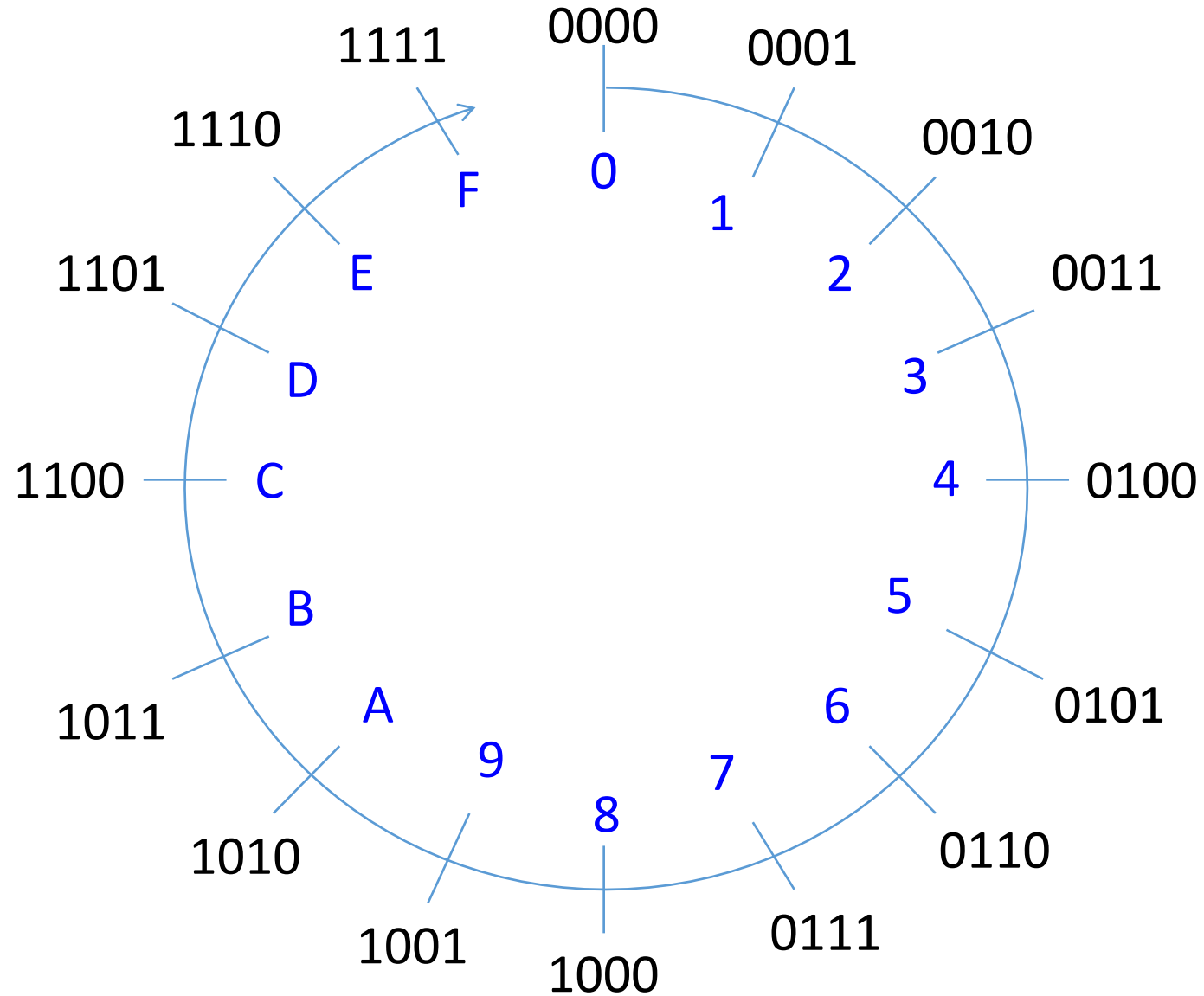
Binary „numbers“ with 4 bits (sorted)



Interpretation as **unsigned** numbers



Abbreviation with hexadecimal numbers



Translation from binary to hexadecimal

hex	binary	dec
0	0 0 0 0	00
1	0 0 0 1	01
2	0 0 1 0	02
3	0 0 1 1	03
4	0 1 0 0	04
5	0 1 0 1	05
6	0 1 1 0	06
7	0 1 1 1	07
8	1 0 0 0	08
9	1 0 0 1	09
A	1 0 1 0	10
B	1 0 1 1	11
C	1 1 0 0	12
D	1 1 0 1	13
E	1 1 1 0	14
F	1 1 1 1	15

1011110000110101

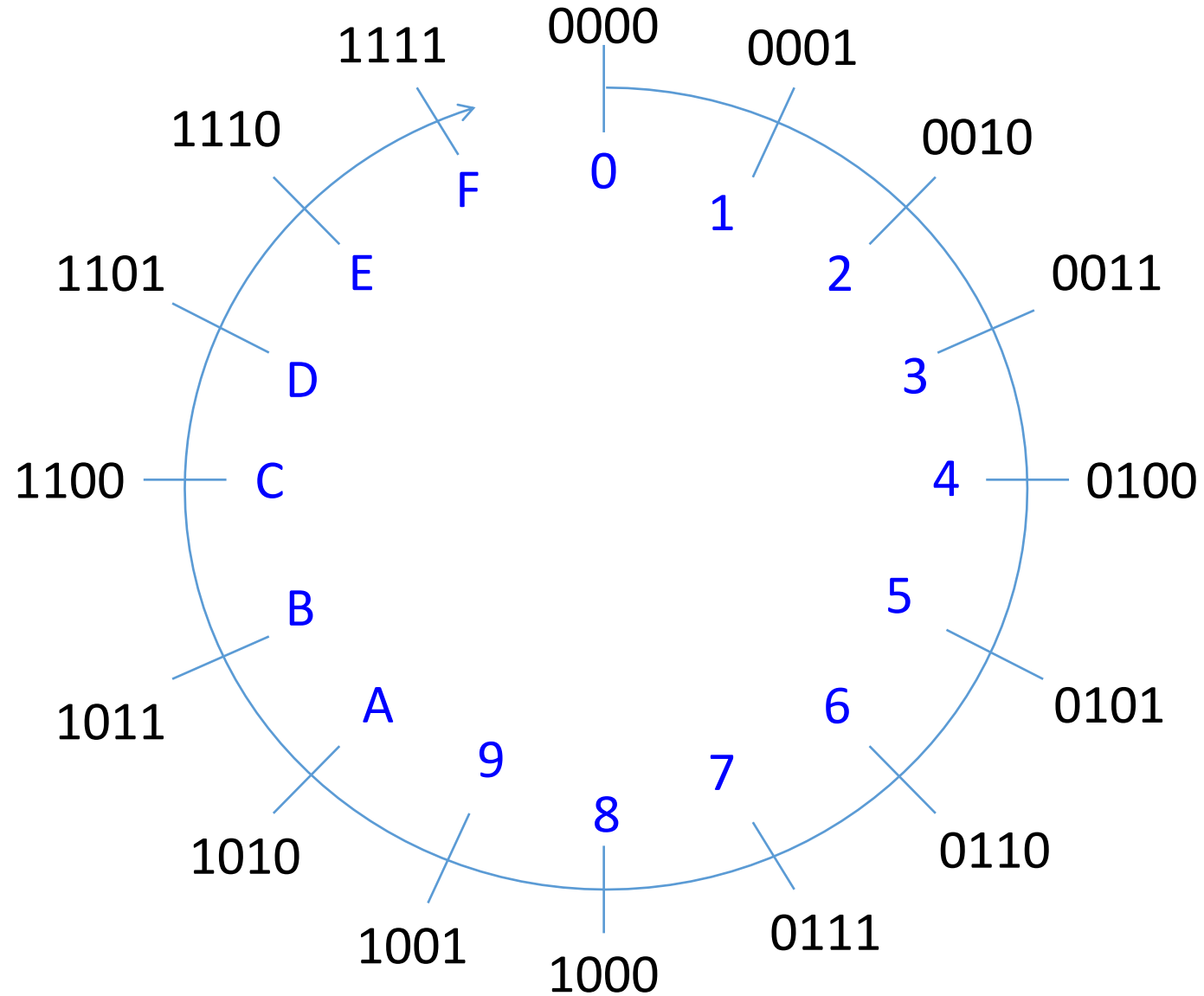
B C 3 5

0xBC35

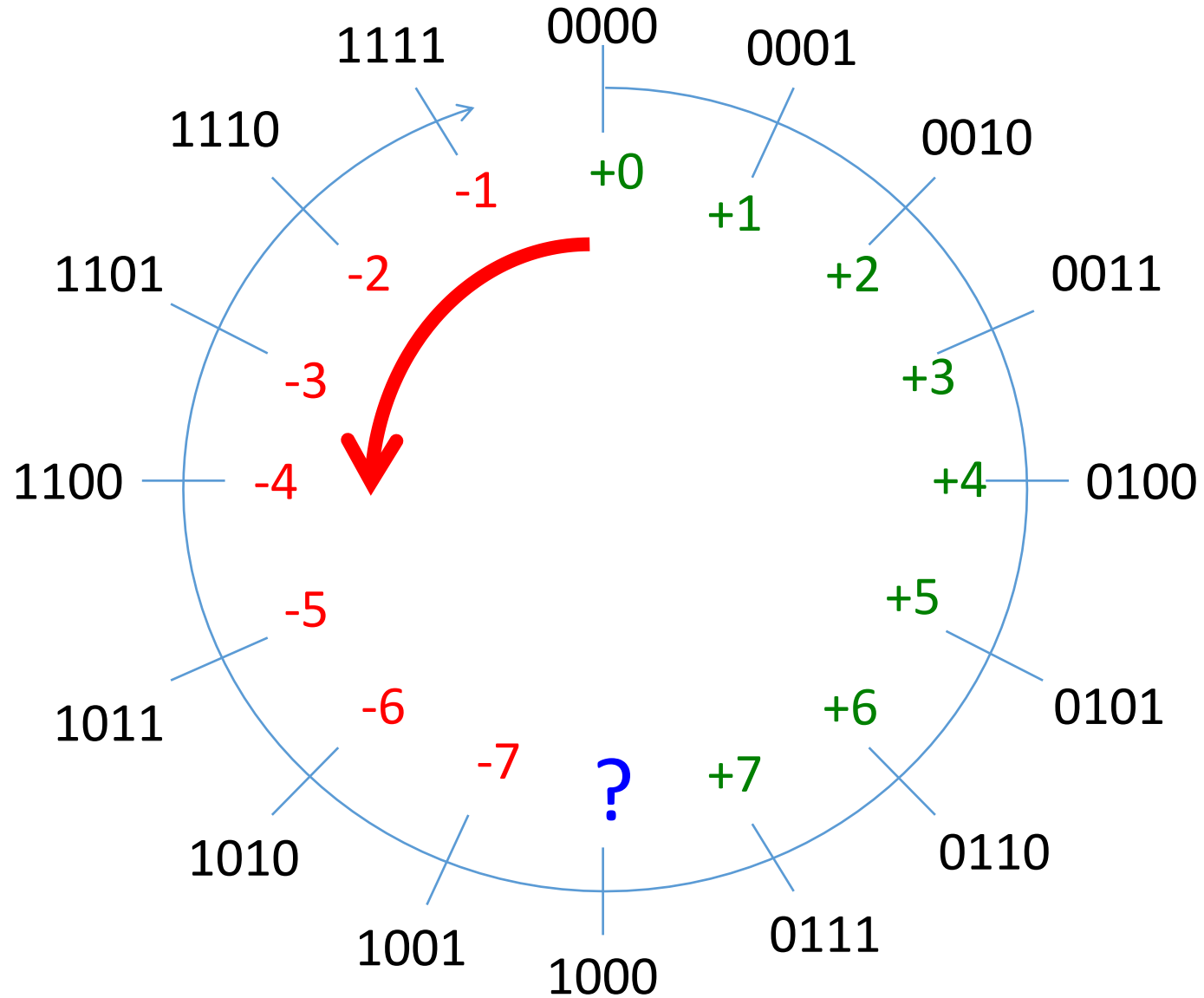
(BC35)₁₆

`hBC35

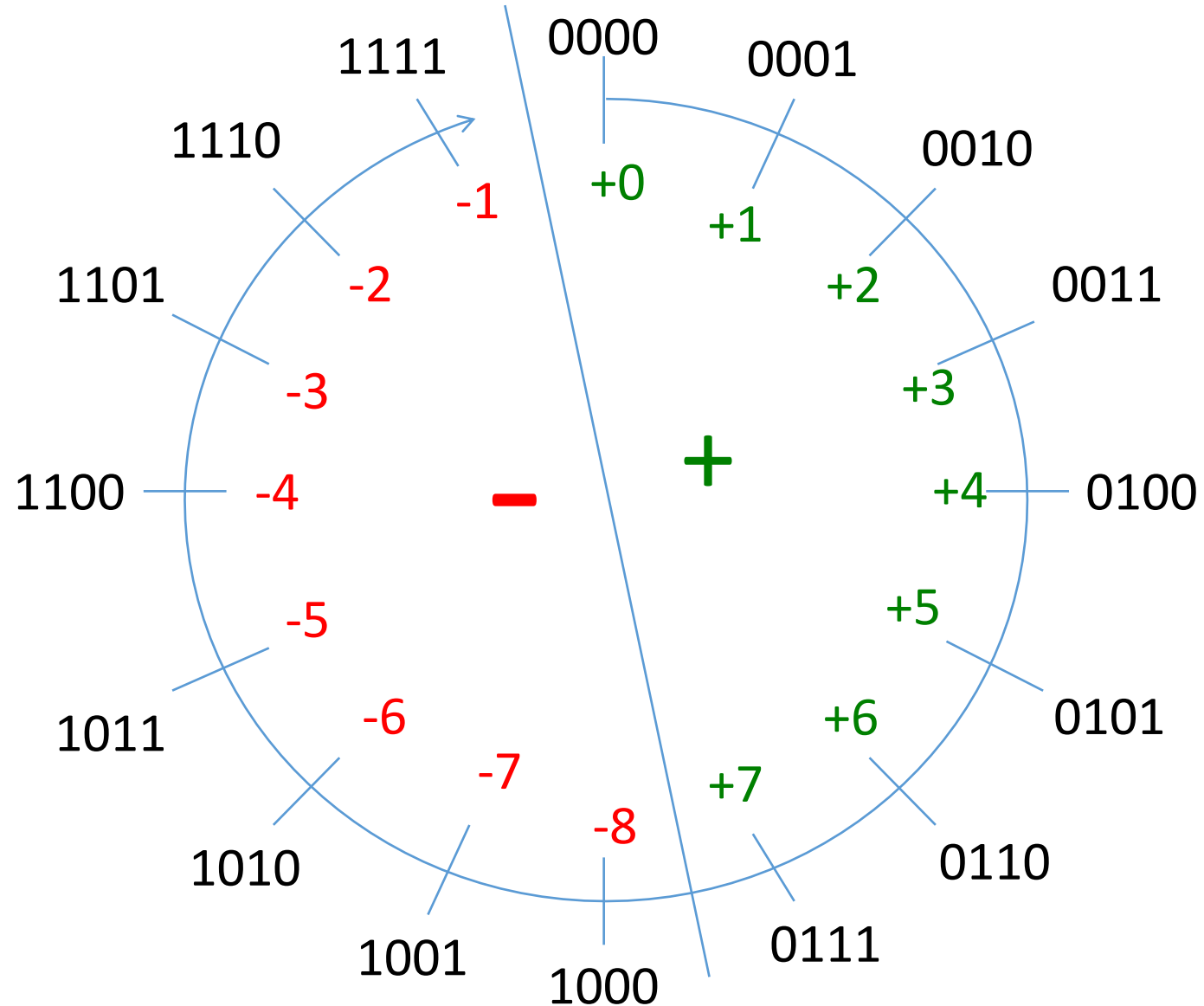
Abbreviation with hexadecimal numbers



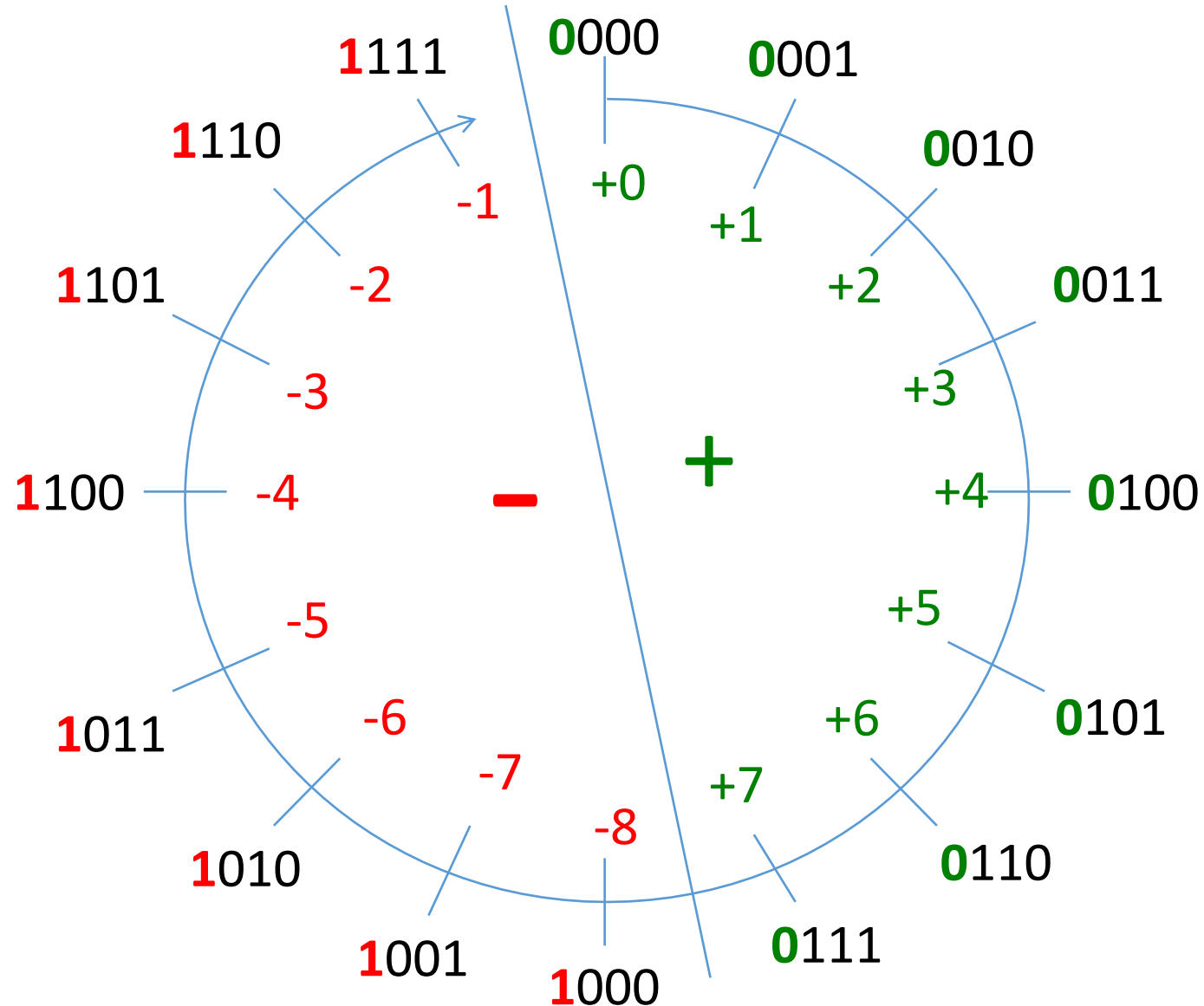
We could also count the other direction



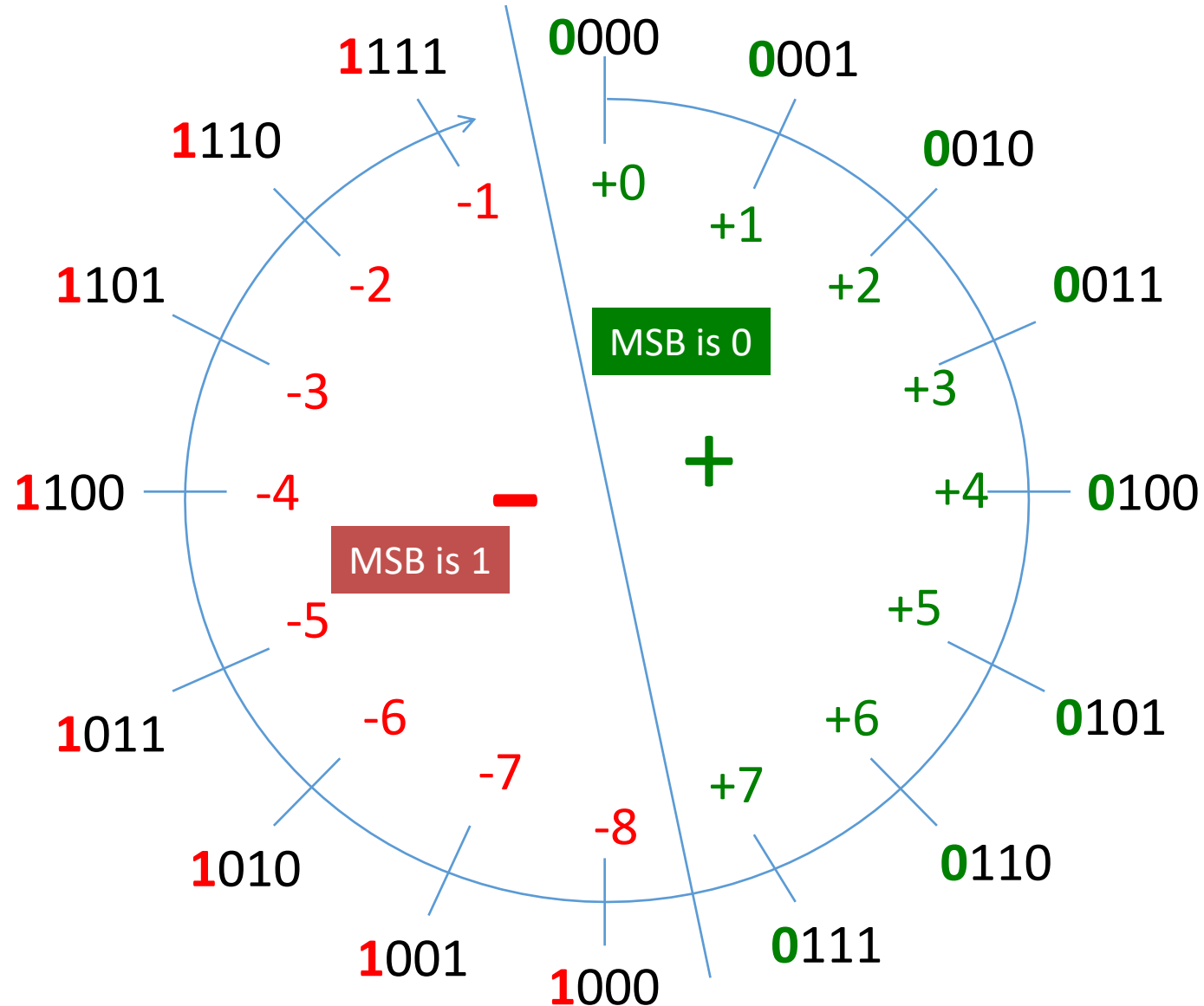
Two halves: **positive** and **negative**



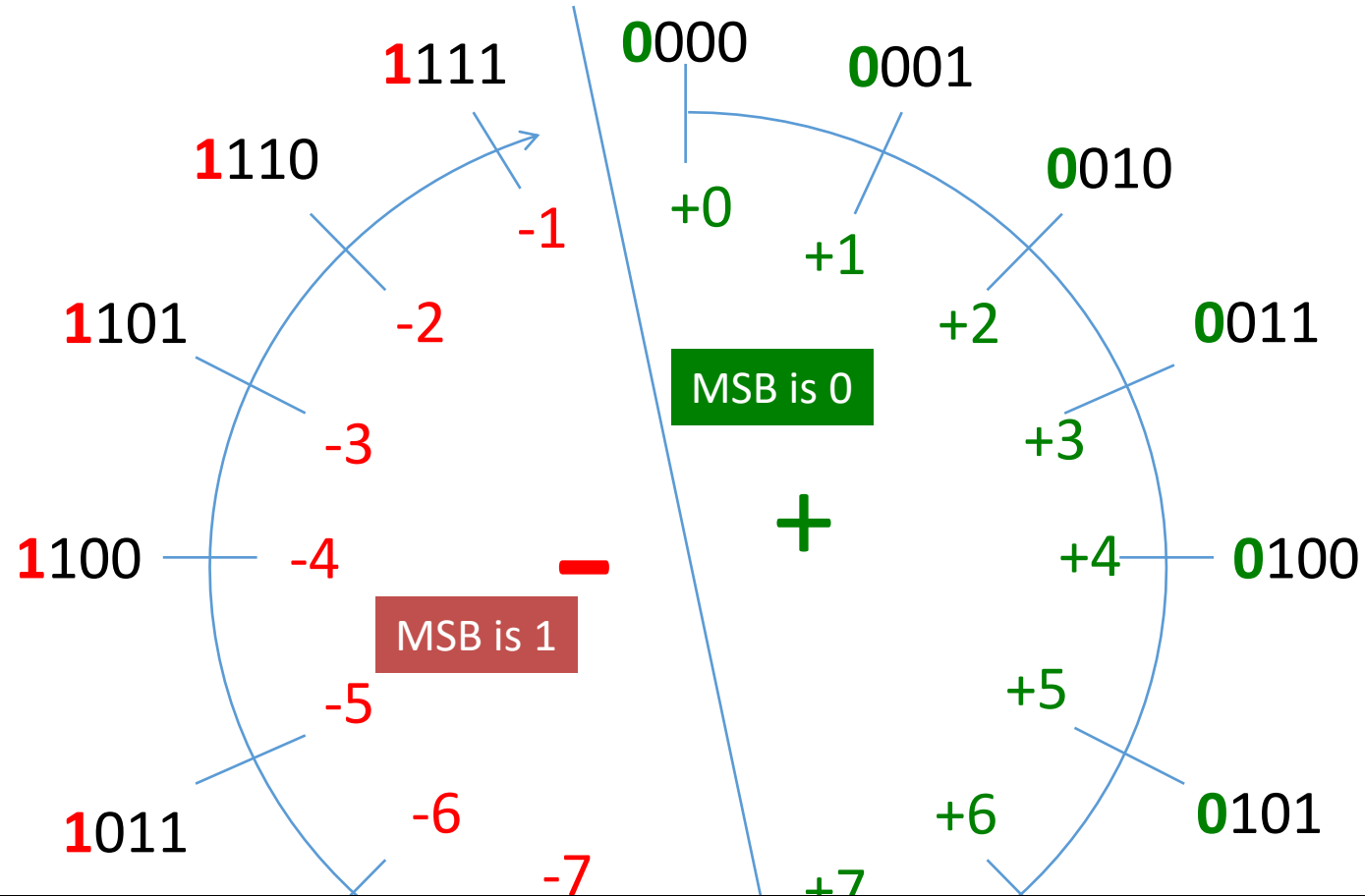
Interesting „side effect“: The most significant bit (MSB) denotes the sign



Interesting „side effect“: The most significant bit (MSB) denotes the sign

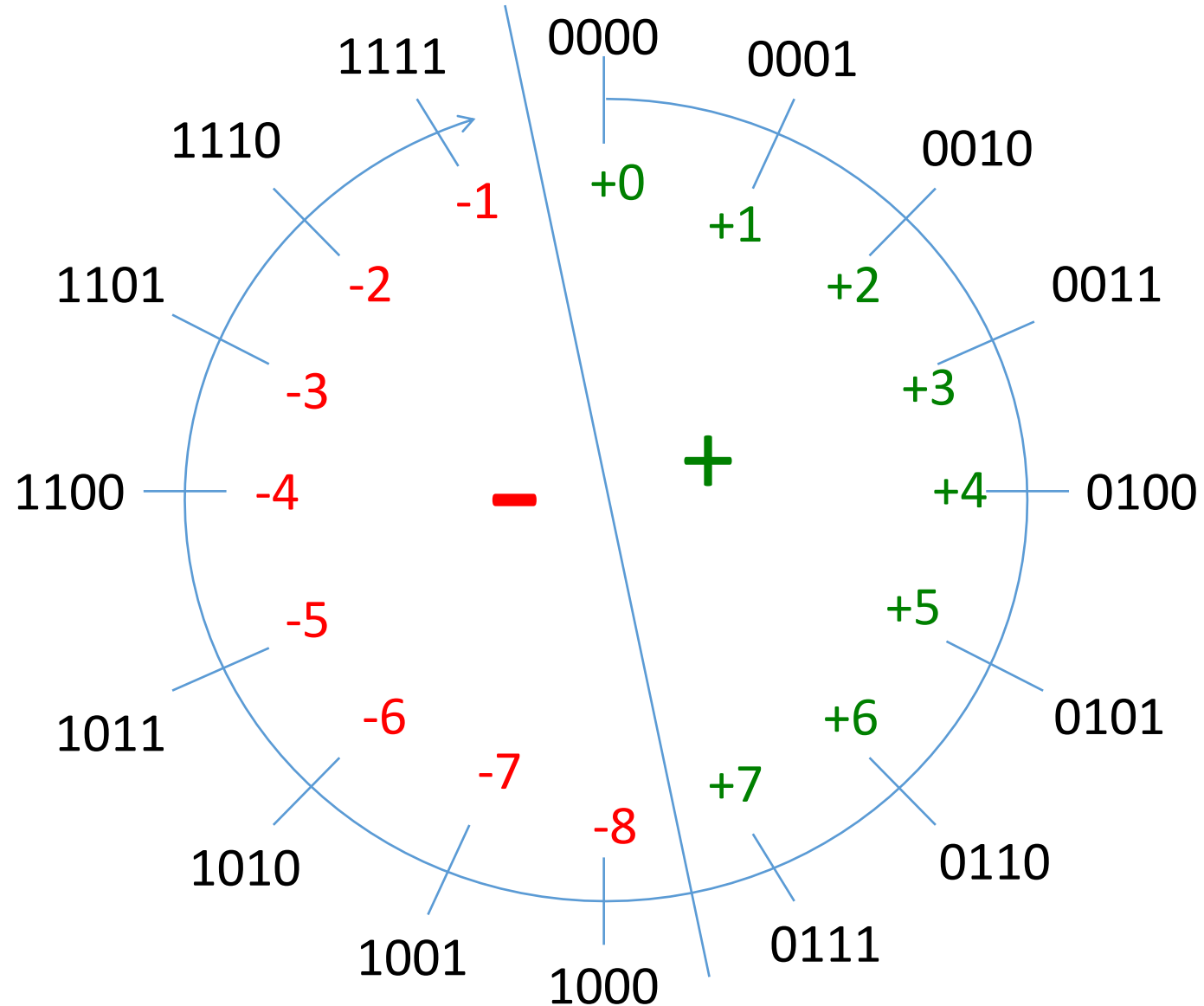


Interesting „side effect“: The most significant bit (MSB) denotes the sign



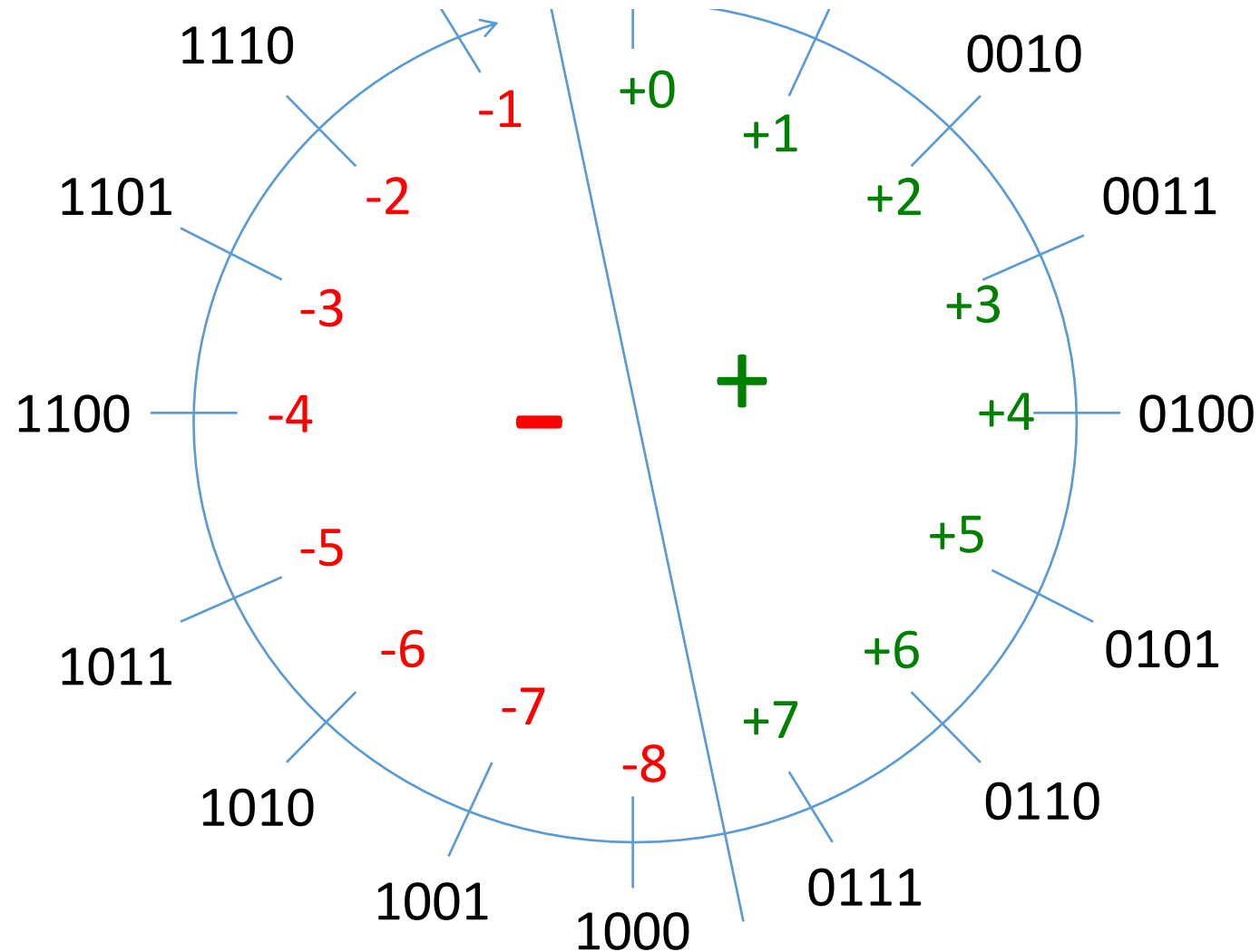
Important: The most significant bit is not the sign in the traditional sense. Here, we do not have the usual representation as “sign” followed by “magnitude of number”. However, the most significant bit “indicates” whether the number is positive or negative.

„Signed“ representation



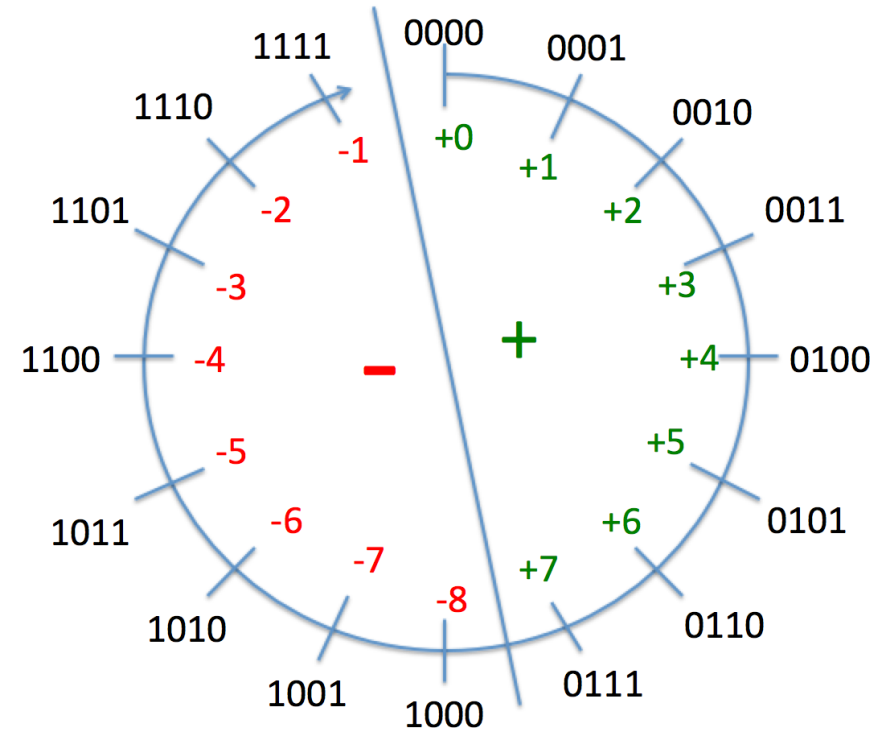
„Signed“ representation

“Two’s complement representation”



How to subtract?

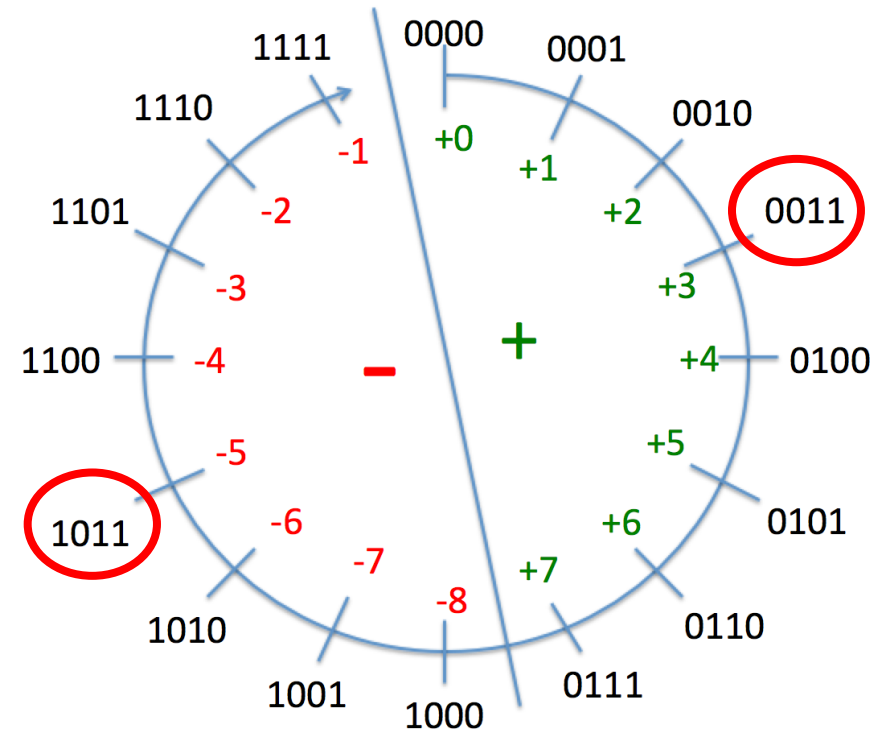
3 - 5



How to subtract?

$$3 - 5$$

$$(+3) + (-5)$$

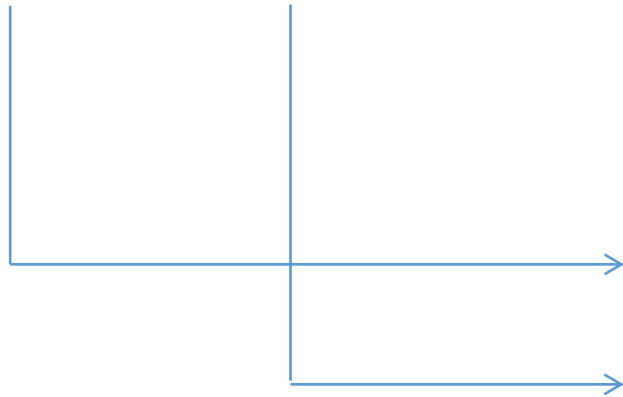


Subtraction is also an addition:
We add the negation of 5.

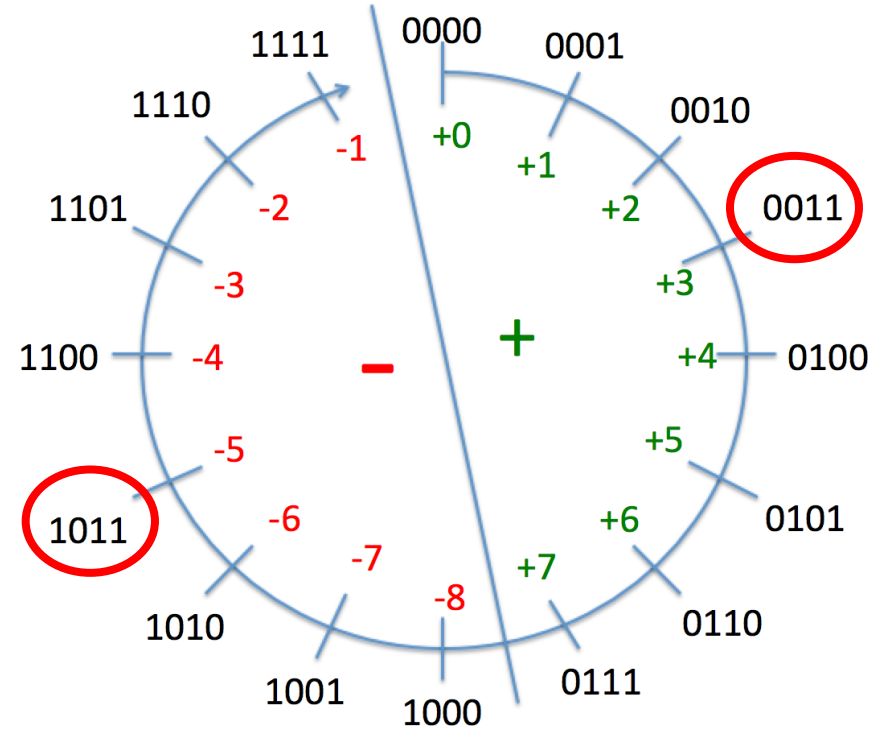
How to subtract

$$3 - 5$$

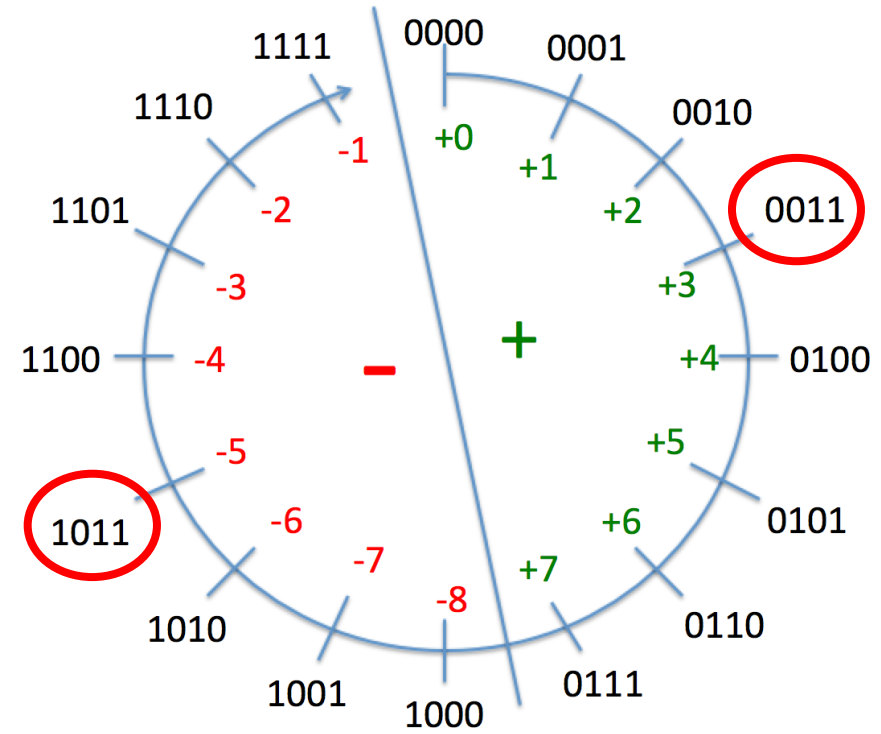
$$(+3) + (-5)$$



$$\begin{array}{r} 0011 \\ + 1011 \\ \hline \end{array}$$



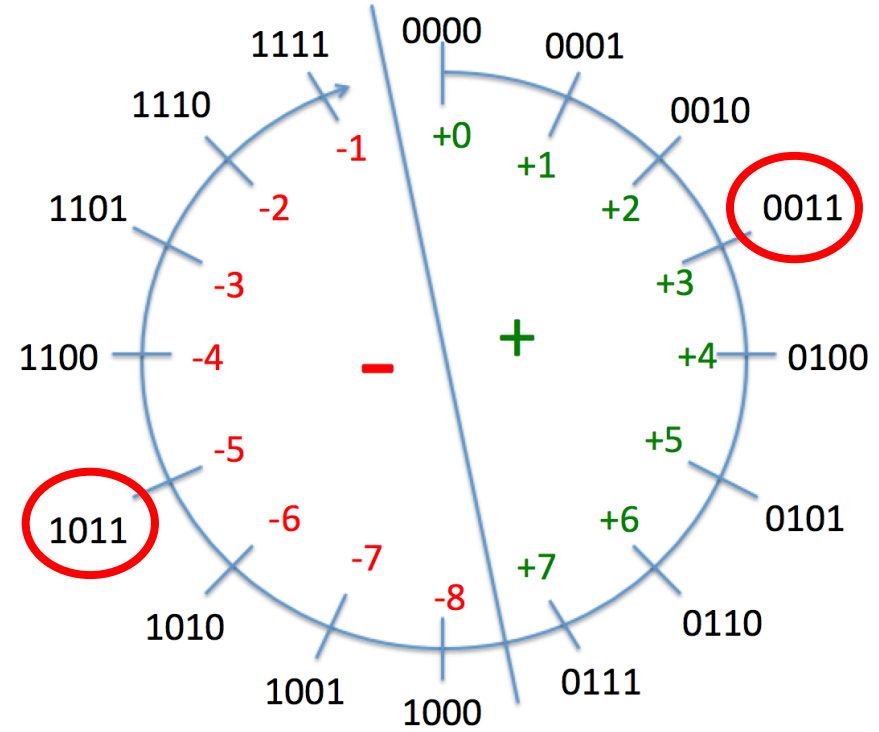
How to subtract?



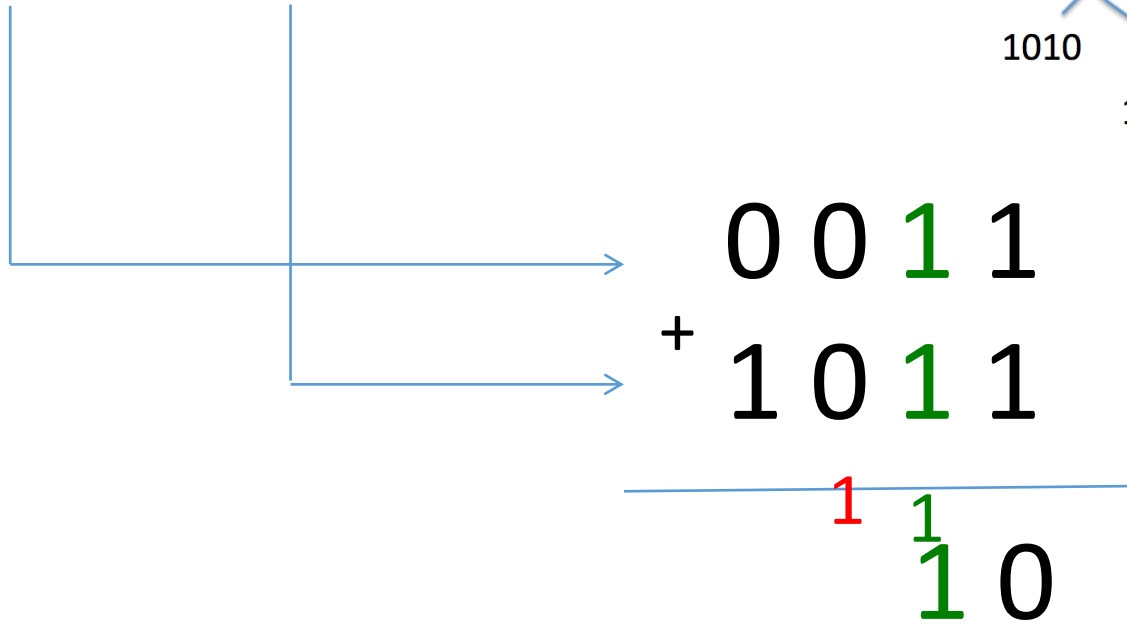
$$(+3) + (-5)$$

$$\begin{array}{r}
 0011 \\
 + 1011 \\
 \hline
 10
 \end{array}$$

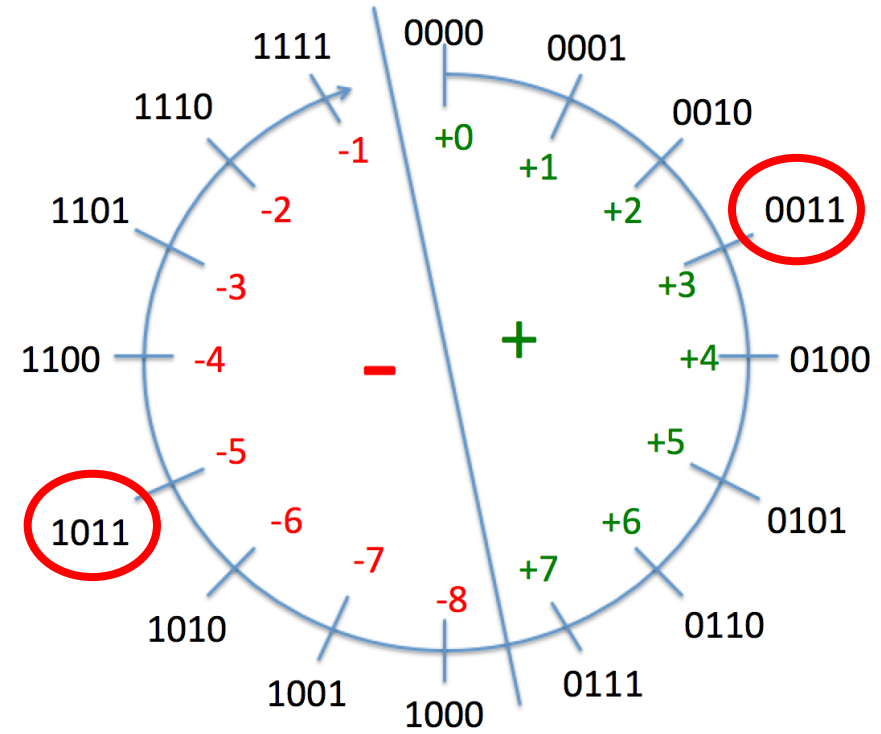
How to subtract?



$$(+3) + (-5)$$



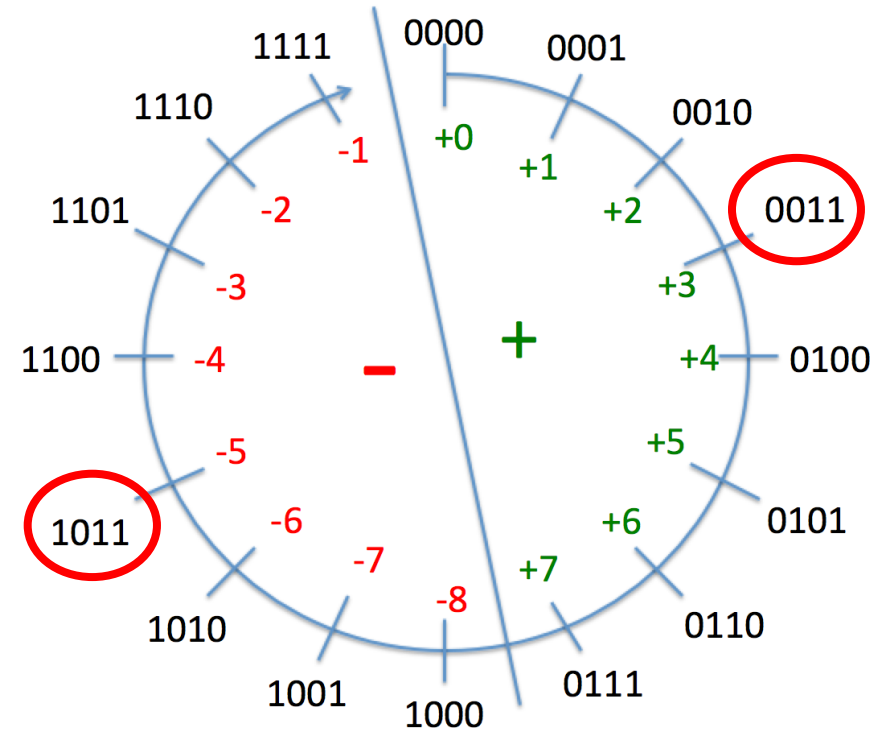
How to subtract?



$$(+3) + (-5)$$

$$\begin{array}{r}
 0011 \\
 + 1011 \\
 \hline
 1100
 \end{array}$$

How to subtract?



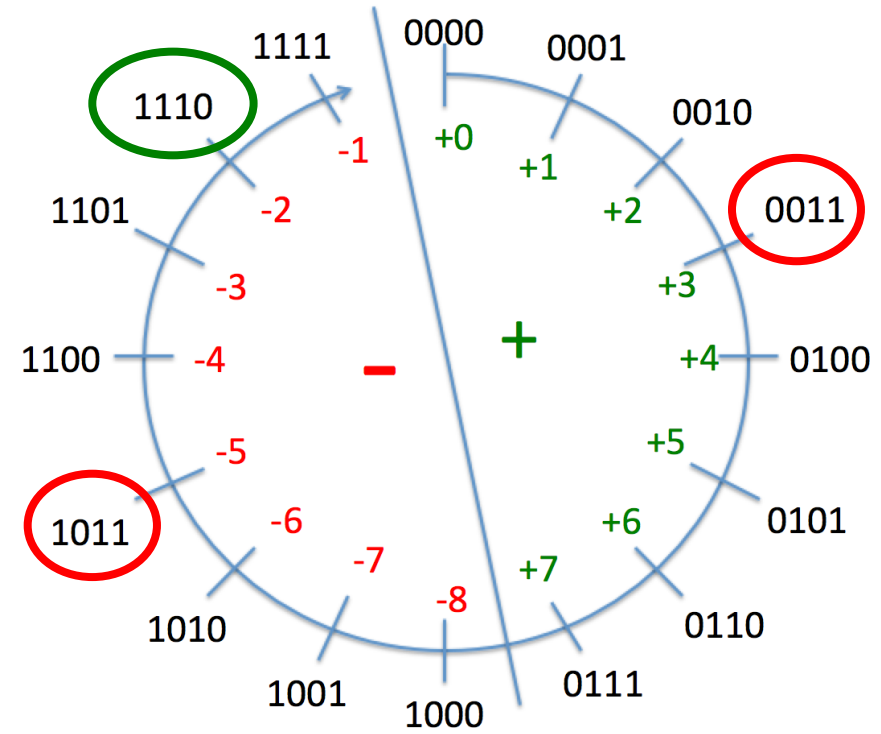
$$(+3) + (-5)$$

$$\begin{array}{r}
 0011 \\
 + 1011 \\
 \hline
 1110
 \end{array}$$

How to subtract?

$$(+3) + (-5) = (-2)$$

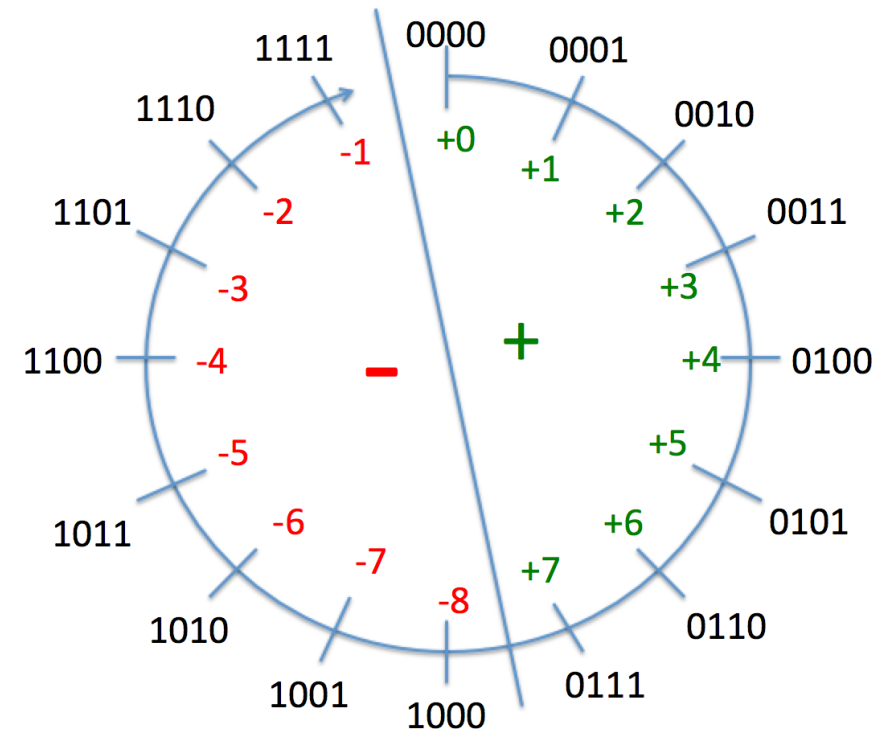
$$\begin{array}{r}
 0011 \\
 + 1011 \\
 \hline
 1110
 \end{array}$$



What happens if we leave the interval: Overflow

$$(+3) + (+6) = (-7)?$$

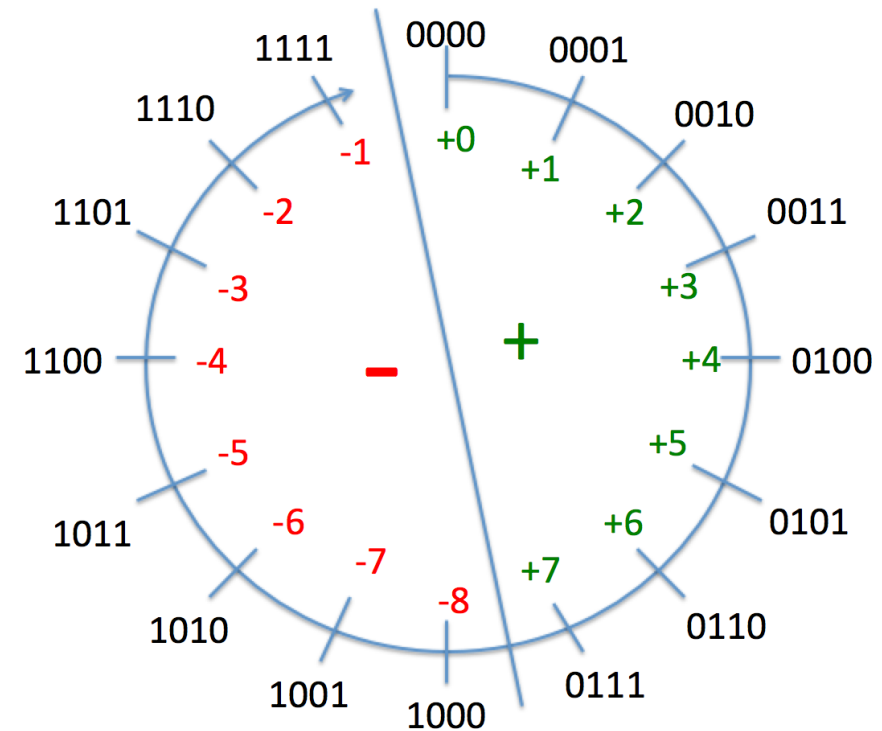
$$\begin{array}{r}
 0011 \\
 + 0110 \\
 \hline
 1001
 \end{array}$$



What happens if we leave the interval: Underflow

$$(-3) + (-6) = (+7)?$$

$$\begin{array}{r}
 1101 \\
 + 1010 \\
 \hline
 \cancel{1}0111
 \end{array}$$

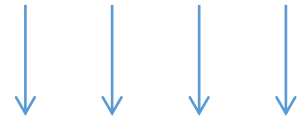


We do not have a 5th bit. We are in a 4-bit world.

Negation of a value: We do the “two’s complement”

+5: **0 1 0 1**

Invert all bits:
„One’s complement“

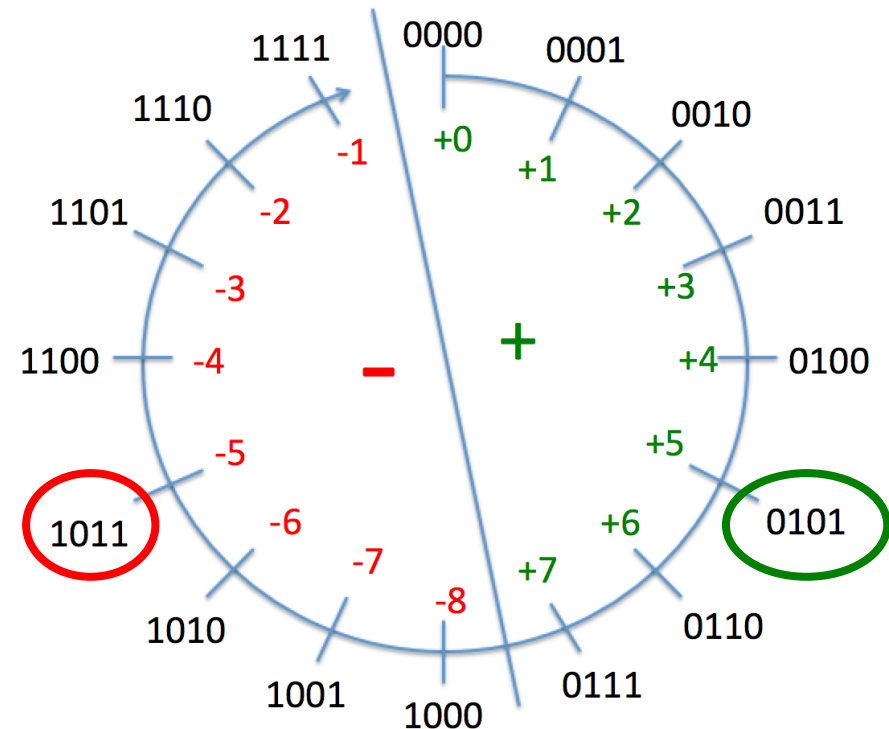


1 0 1 0

adding 1 makes the
„two’s complement“

+ 0 0 0 1

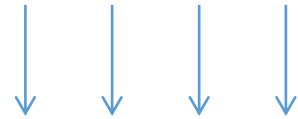
-5: **1 0 1 1**



Two's complement works both ways

-3: **1 1 0 1**

Invert all bits to get the
„one's complement“

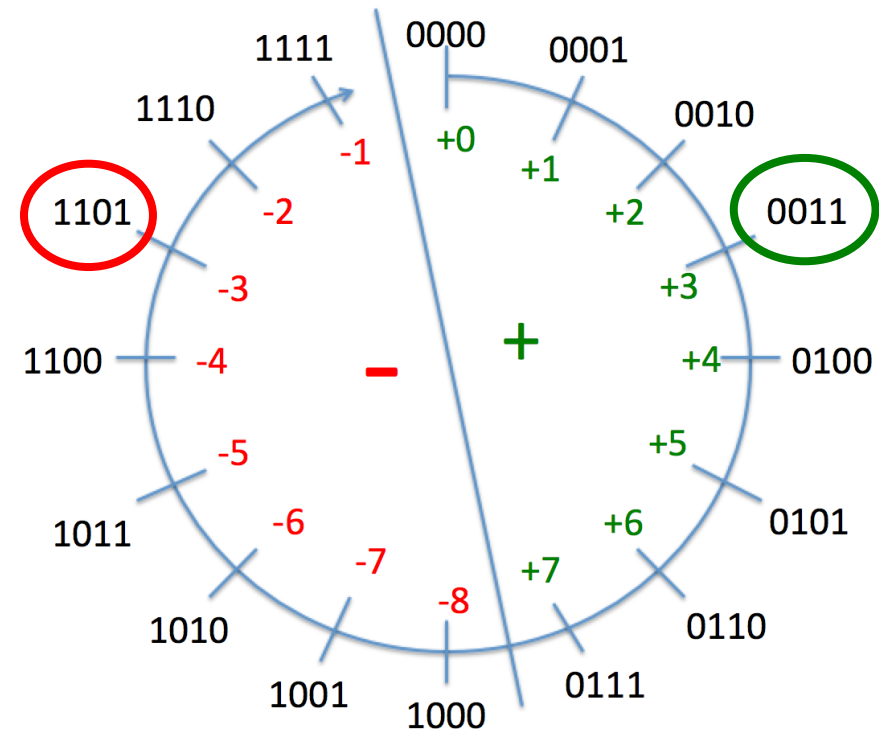


0 0 1 0

Adding 1 makes the
„two's complement“

+ 0 0 0 1

+3: **0 0 1 1**



Check, if Your Computer Behaves as Expected in a 64-bit or 32-bit World

- See examples con02 available at

<https://extgit.iaik.tugraz.at/con/examples-2022.git>

Sign Extension

- Sign extension is necessary, if binary values with different bit width are combined
- Sign extension means that the MSB of the shorter value is replicated until the bit size of the larger value is reached. This ensures correct arithmetic handling.

Sign Extension – Example

- Example: compute $A + B$
 - Value A (16 bit): 7 (Binary: 00000000 00000111)
 - Value B (8 bit): -1 (Binary: 11111111)
- If we would simply add the values without sign extension, this would lead to an incorrect result: 262 decimal (Binary: 00000001 00000110)
- Correct computation with sign extension:
 - Value A (16 bit): 7 (Binary: 00000000 00000111)
 - Value B after sign extension (16 bit): -1 (Binary: 11111111 11111111)
 - Result $A + B$: 6 (Binary: 00000000 00000110)



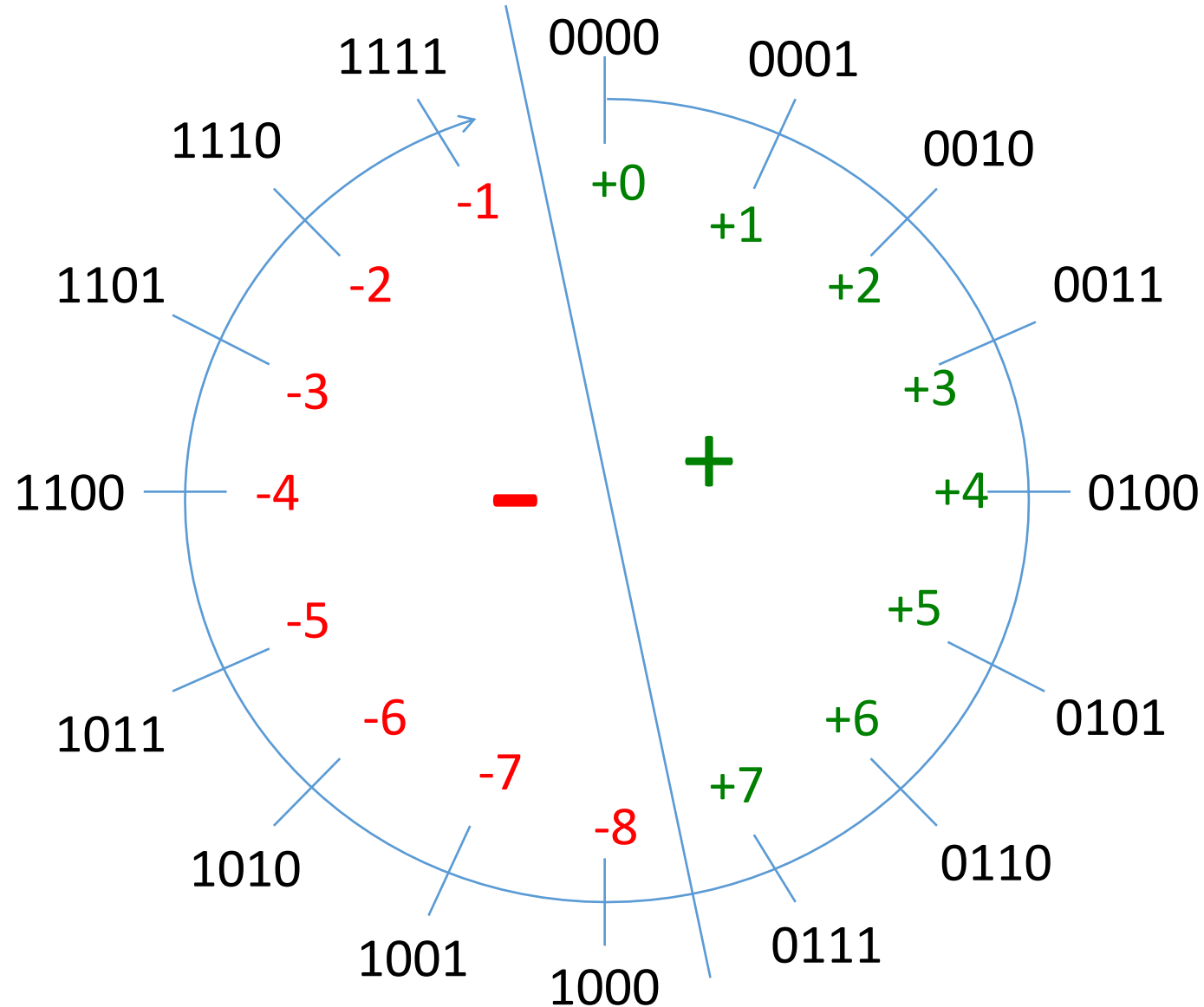
Rational numbers

- To the right of the decimal point:
 - Tenth 10^{-1}
 - Hundredth 10^{-2}
 - Thousandth 10^{-3}
 - ...
- Example: $(0.1234)_{10}$
- Multiply with 10^4 leads to $(1234)_{10}$

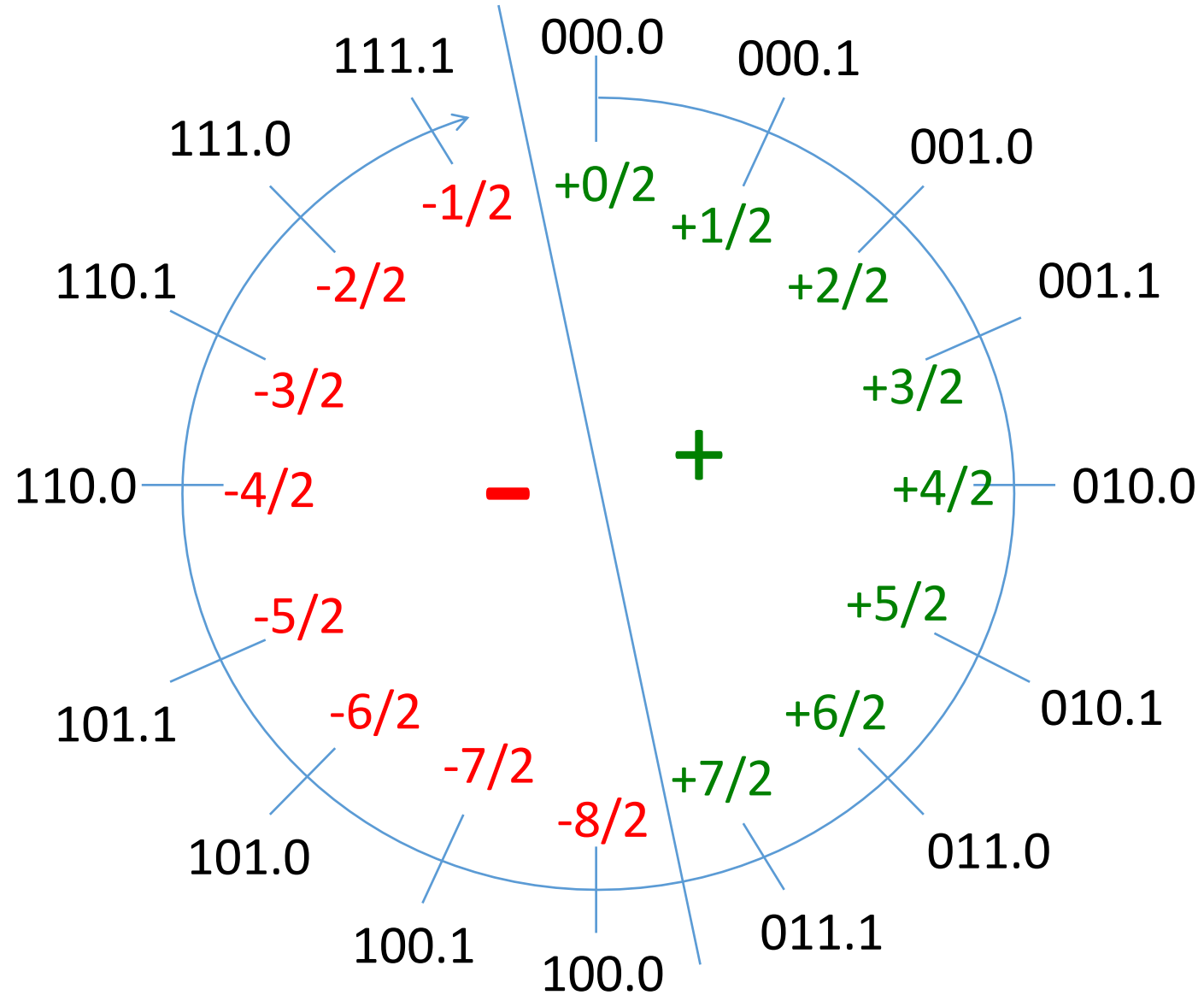
Rational numbers with base 2

- To the right of the „binary“ point:
 - Halves 2^{-1}
 - Quarters 2^{-2}
 - Eighths 2^{-3}
 - ...
- Example: $(0.1101)_2 = \frac{1}{2} + \frac{1}{4} + \frac{1}{16} = \frac{13}{16}$
- Alternative view: Multiply $(0,1101)_2$ with 2^4 and you get $(1101)_2$, i.e. $(13)_{10}$. The multiplication with 2^4 shifts the „binary point“ by 4 positions to the left. Thus, $(0,1101)_2$ is the same as $\frac{13}{16}$.

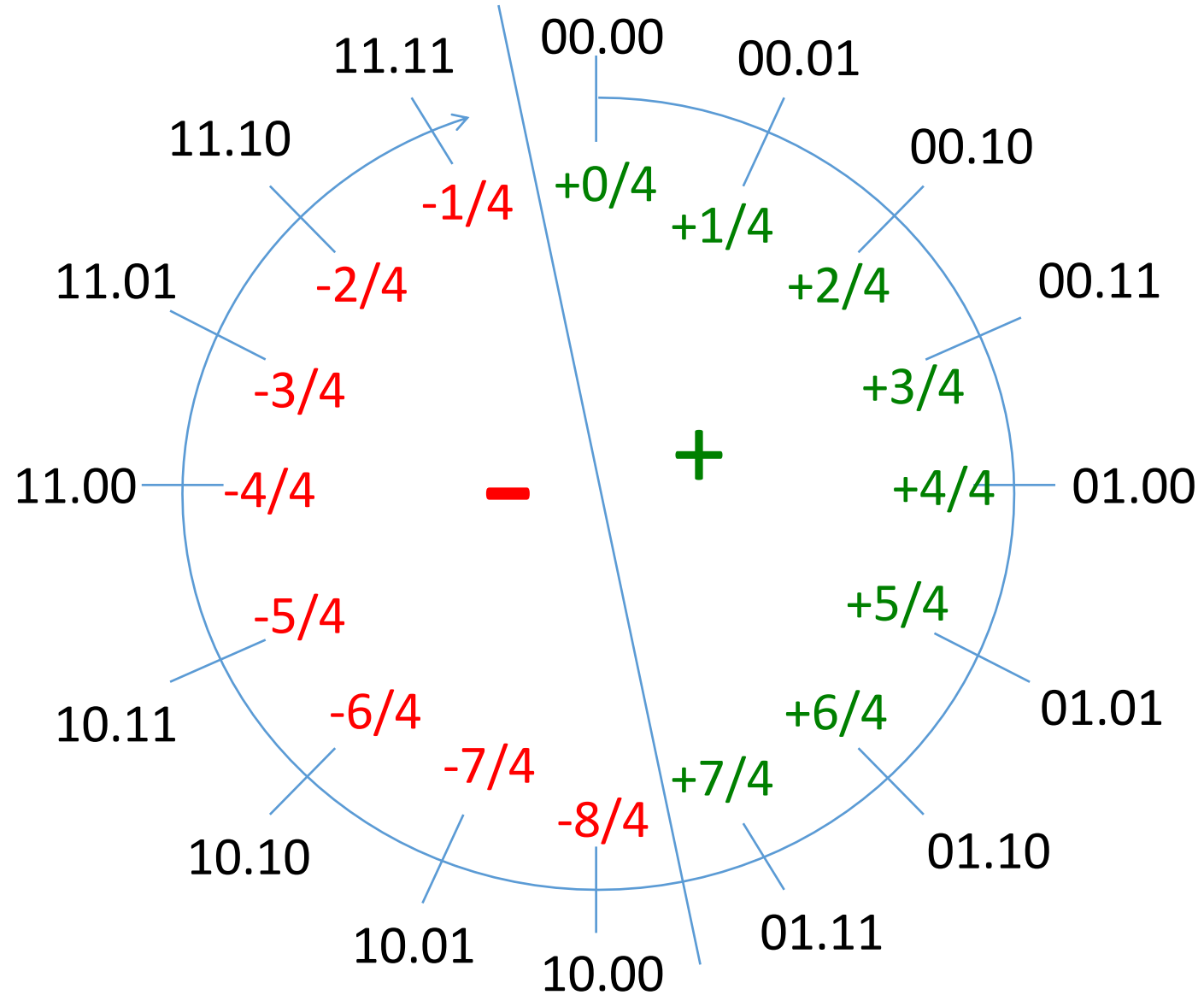
Rational numbers in „signed“ format



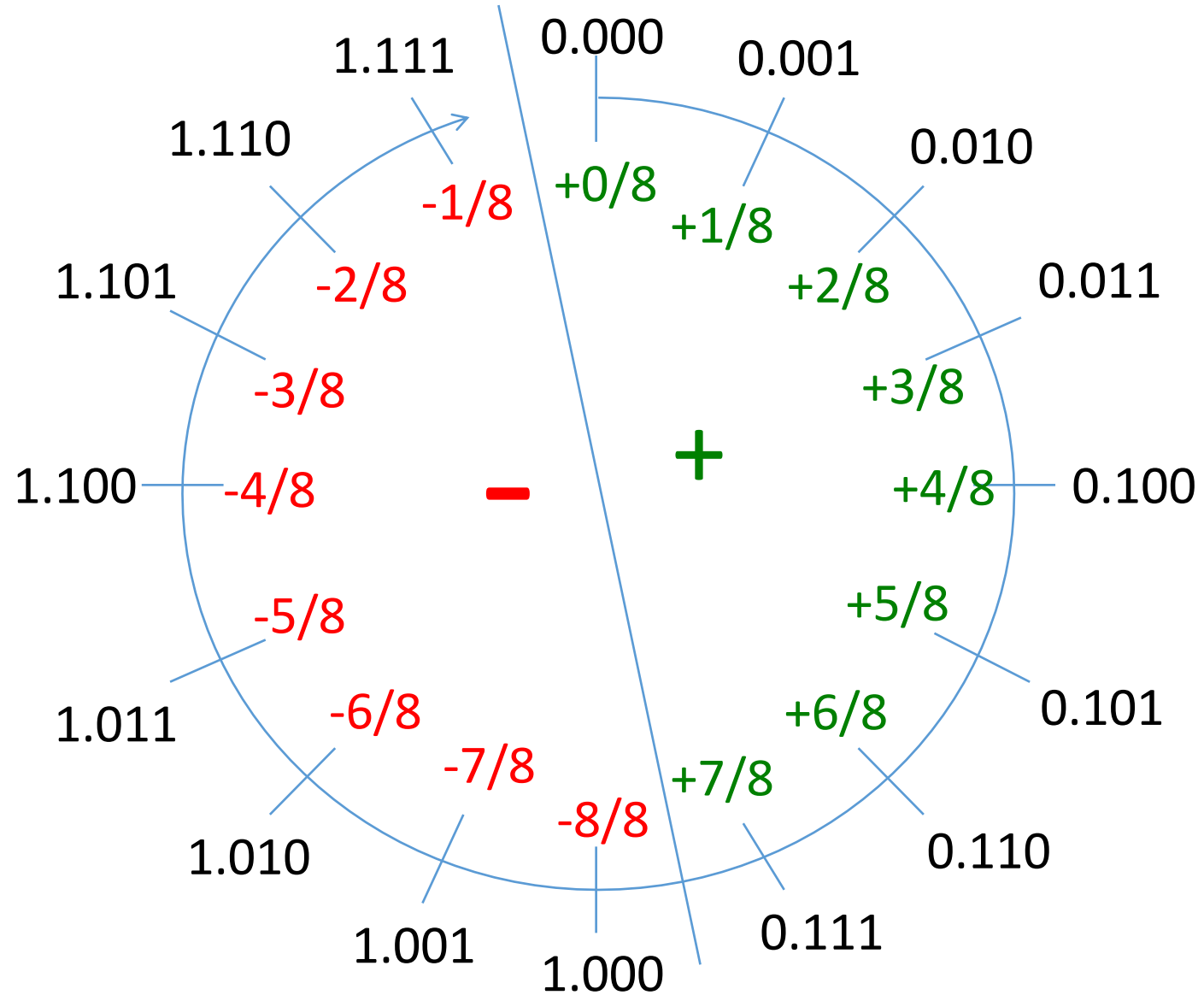
Division by 2



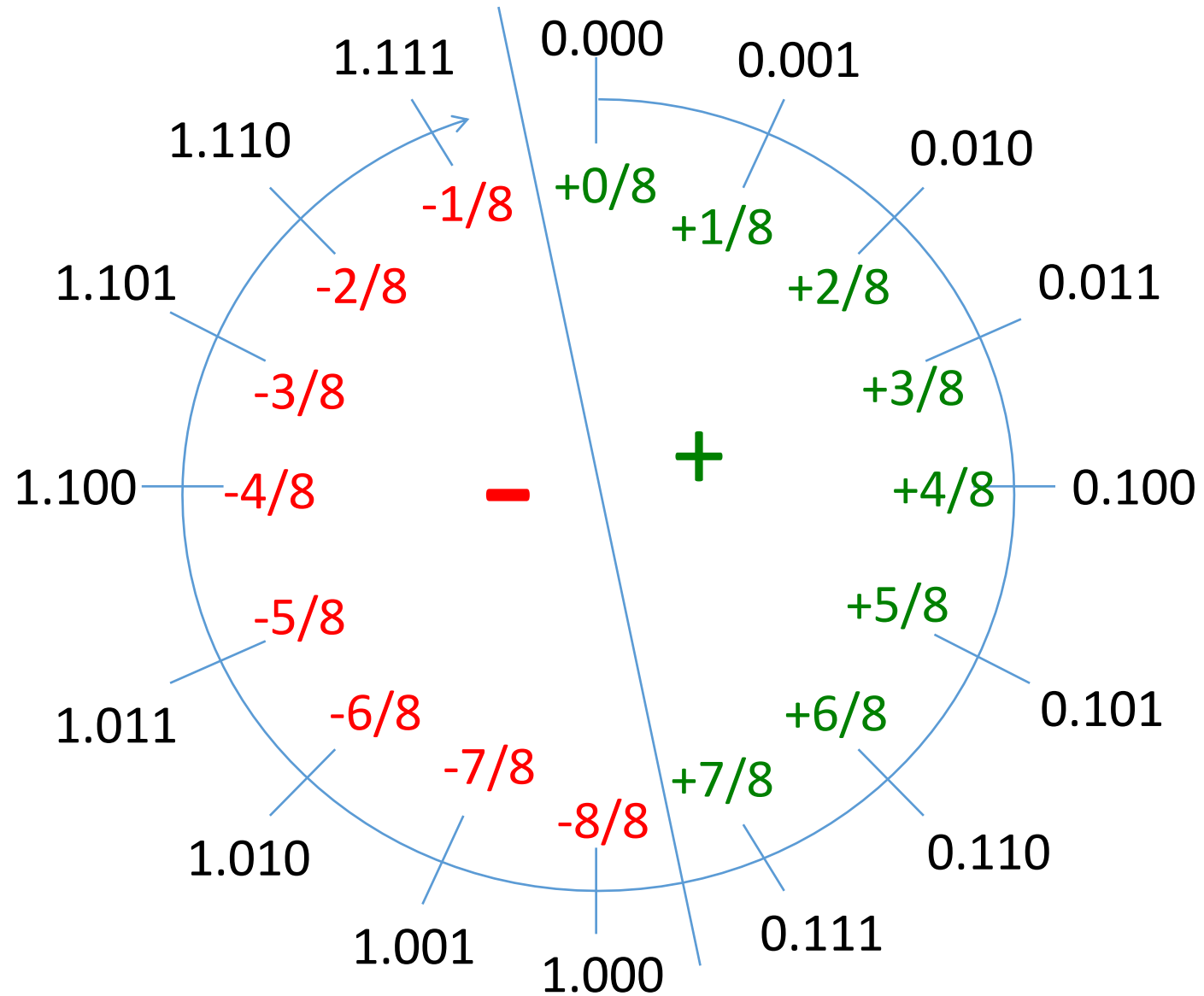
Division by 4



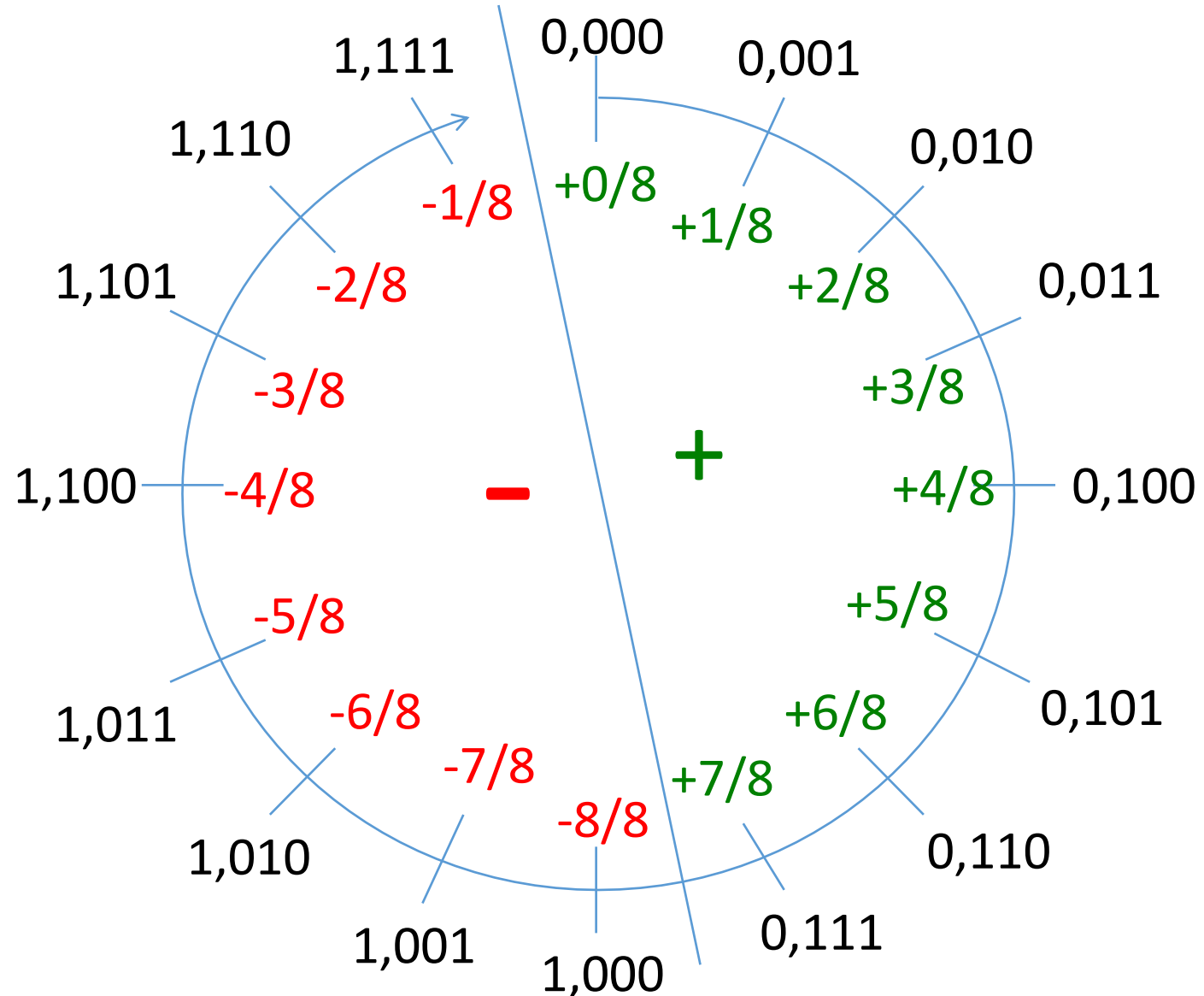
Division by 8



The interval is $[-1, +7/8]$



We cannot divide by 16 since we need at least 1 bit left of the „point“.



Division by the base

- Right shift by one position: $123,4 / 10 = 12,34$
- Arithmetic Shift Right (ASR): All bits get shifted to the right by one position. The value of the most significant bit gets duplicated.
- Logic Shift Right (LSR): All bits get shifted to the right by one position. The most significant bit gets 0.
- Division by 2:
 - Unsigned: ASR oder LSR: Half of $(0110)_2$ is $(0011)_2$
 - Signed: ASR: Half of $(-6)_{10} = (1010)_2$ is $(1101)_2$
 - Half of $(-4/8) = (1,100)_2 = (1,110)_2 = (-2/8)$
- Watch out: negative numbers are rounded towards “lower values”

$$(-1)_2 = (1111)_2$$

$$(-\frac{1}{2})_2 = (1111)_2$$

It does not matter by how many positions you shift to the right:
 $(1111)_2$ stays always $(1111)_2$.

Check out http://en.wikipedia.org/wiki/Arithmetic_shift

Building Circuits that Calculate

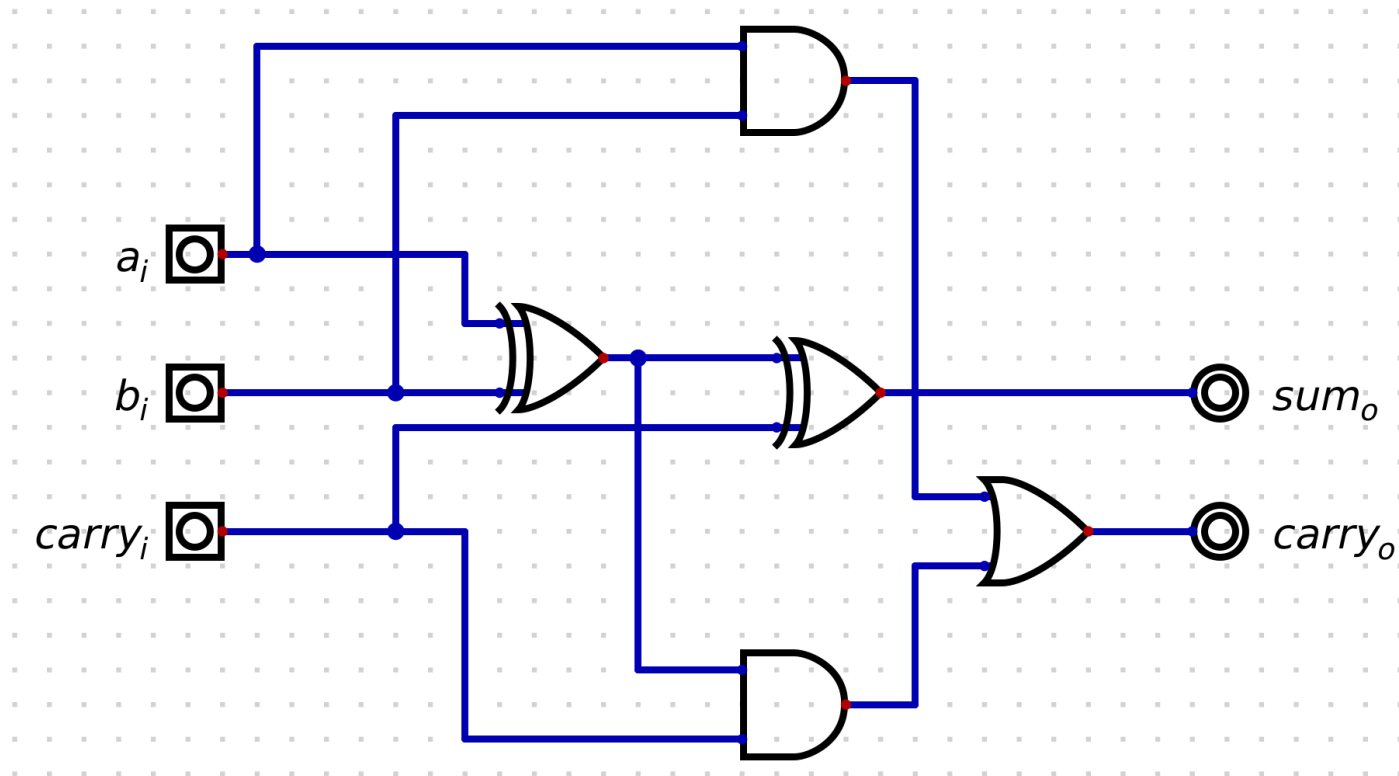
Circuit for a Single Bit Position

- c **0 0 1 0** **Carry signal**
 - $a = 5$ (dec) = 0 1 0 1 (bin)
 - $b = 9$ (dec) = 1 0 0 1 (bin)
 - $s = 14$ (dec) = 1 1 1 0 (bin)
-
- Inputs:
 - a_i
 - b_i
 - c_{i-1} (“carry in”)
 - Outputs:
 - s_i
 - c_i (“carry out”)

Truth Table of a Full Adder

a	b	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Schematic of a Full Adder



Addition and Subtraction in SystemVerilog

- See examples con02 available at

<https://extgit.iaik.tugraz.at/con/examples-2022.git>

- A simple “+” operator in SystemVerilog creates an adder of the necessary size for you.
 - Try it out and build a 16-bit adder
 - Change the sign from “-”
 - ...