CS 211: Intro to Computer Architecture

3.1: Character and Integer Representations Cntd.

Minesh Patel

Spring 2025 – Thursday 4 February

Announcements

- **Due dates**
  - PA1 on Thursday (online via Gradescope)
  - WA1 due Friday (online via Gradescope)
- **WA2 and PA2 to be assigned on Friday (maybe Saturday…)**
  - WA2: 1 week
  - PA2: 1.5 weeks
- **TA's recitation activities**
  - A variety of activities if you need more practice material or individual attention

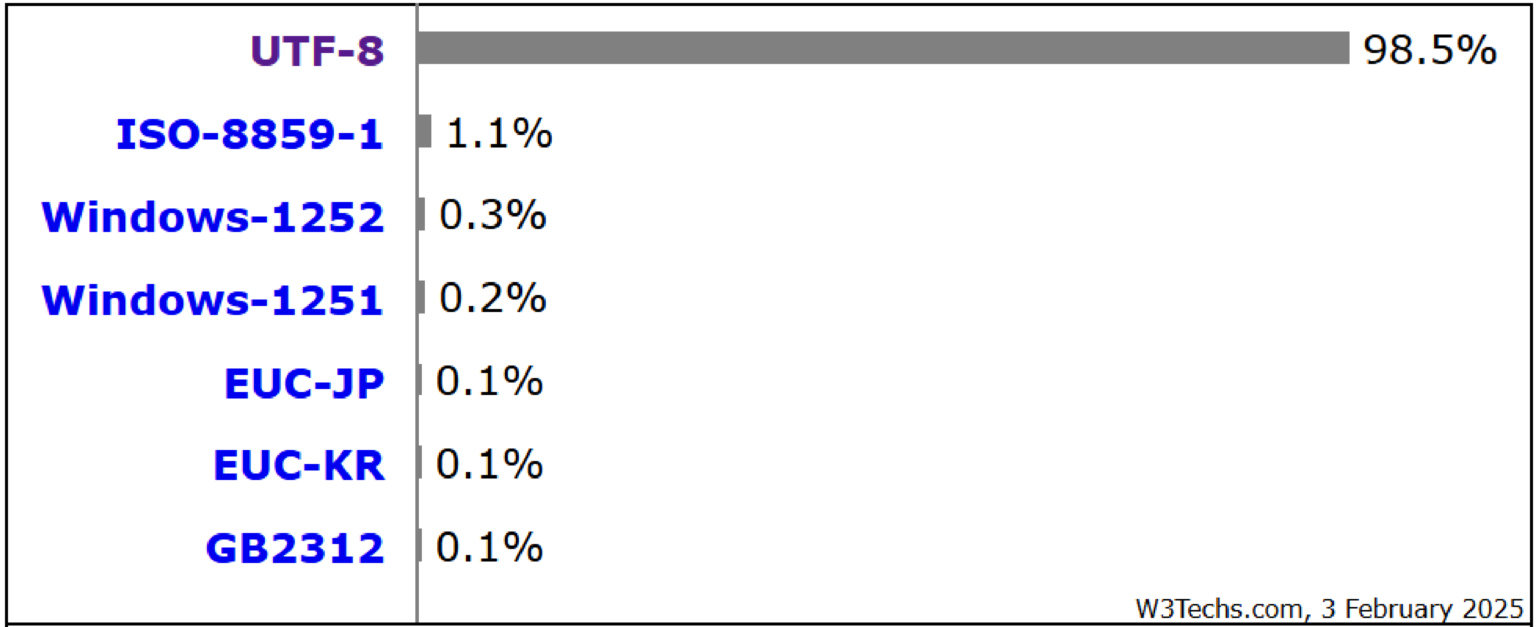
Agenda

- **Endianness**

- Two’s Complement
  - Useful properties

- Other signed representations
  - Bias-K Representation
  - Sign Magnitude
  - One’s Complement

Webpage Encodings by Popularity



Percentages of websites using various character encodings
Note: a website may use more than one character encoding

The following character encodings are used by less than 0.1% of the websites

- Windows-1250
- Shift JIS
- ISO-8859-2
- Big5
- ISO-8859-15
- ISO-8859-9
- GBK
- US-ASCII
- Windows-1254
- Windows-874
- Windows-1256
- Windows-1255
- UTF-16
- Windows-1253
- GB18030
- Windows-1257
- KOI8-R
- KS C 5601
- ISO-2022-JP
- ISO-8859-4
- UTF-7
- ISO-8859-8
- ISO-8859-5
- ISO-8859-6
- Windows-31J
- ANSI_X3.110-1983
- KOI8-U
- Windows-1258
- ISO-8859-16
- ISO-8859-13
- Big5 HKSCS
- ISO-8859-3
- ISO-8859-10
- ISO-8859-14
- ISO-8859-11
- IBM850
- Windows-949

UTF-8 and UTF-16

Text

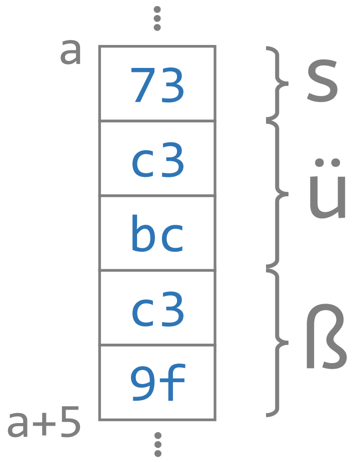
süß

Unicode Codepoints

U+0073	s	LATIN SMALL LETTER S
U+00FC	ü	LATIN SMALL LETTER U WITH DIAERESIS
U+00DF	ß	LATIN SMALL LETTER SHARP S

UTF-8 (variable-width)

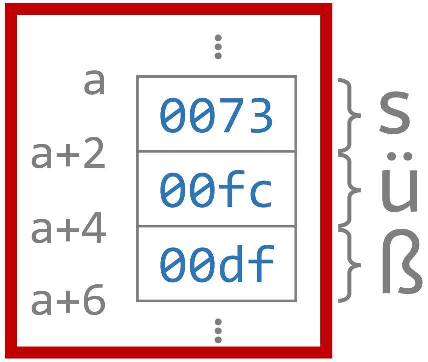
`0x73_c3bc_c39f`



UTF-16 (fixed-width)

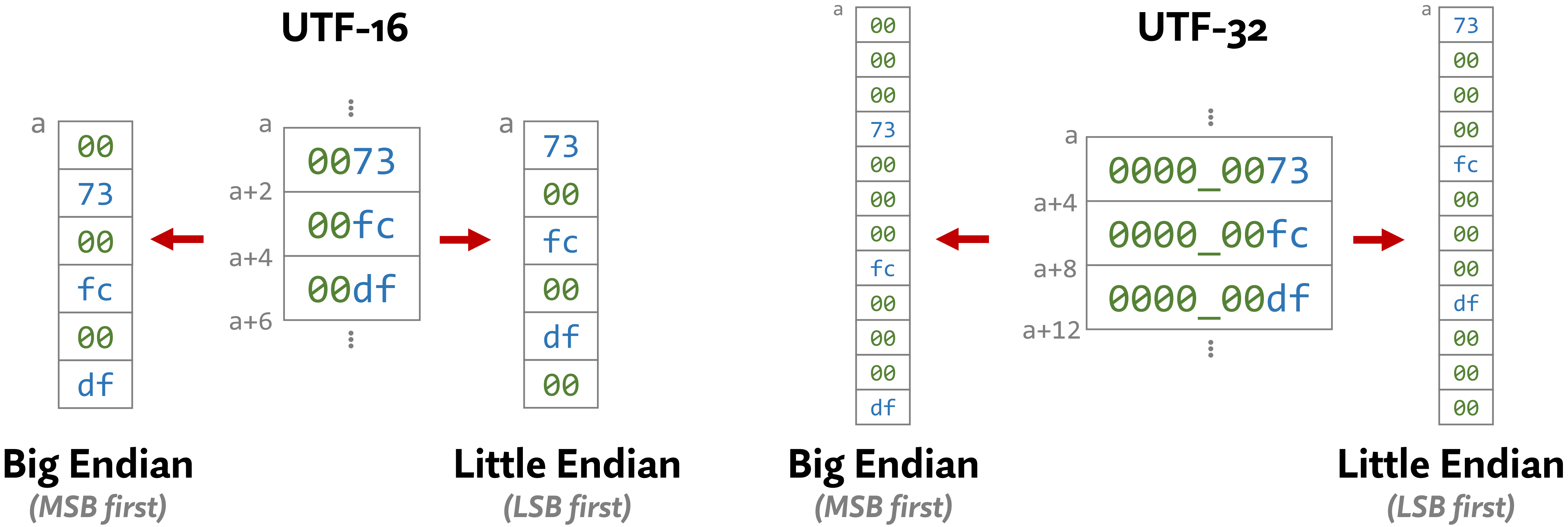
`0x0073_00fc_00df`

Let's take a closer look



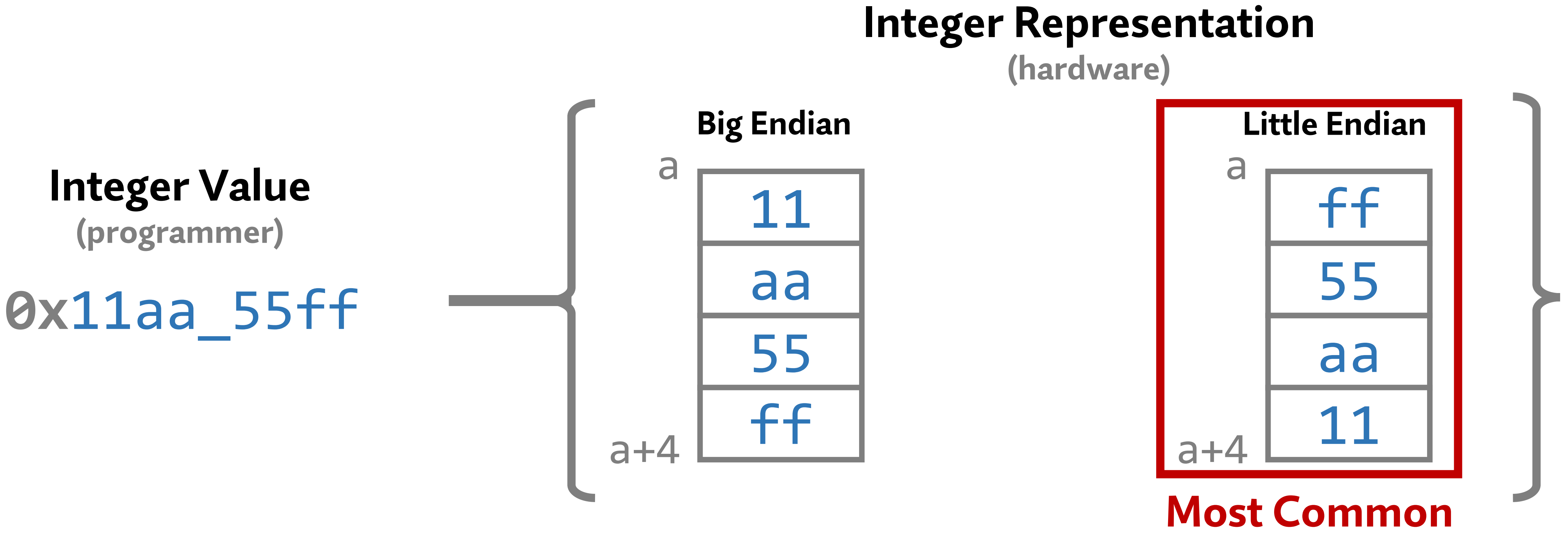
Byte Order Ambiguity

- **Recall:** memory is a contiguous array of **bytes**
- Need a convention for how **N-byte "words"** are represented as **bytes**



Endianness More Generally

- Big and little endianness is just a matter of **convention**
  - Hardware and software must agree on that convention



- Endianness must be clearly specified for any **multi-byte** representation

Endianness in Action

• File containing 32-bit (N=4-byte) integers

```
mp2099@ilab3:~/cs211/experiment$ file integers.dat
integers.dat: data
mp2099@ilab3:~/cs211/experiment$ cat integers.dat; echo
"3DUfw���������
```

```
mp2099@ilab3:~/cs211/experiment$ xxd -g 4 integers.dat
00000000: 00112233 44556677 8899aabb ccddeeff  .."3DUfw........
mp2099@ilab3:~/cs211/experiment$ xxd -e integers.dat
00000000: 33221100 77665544 bbaa9988 ffeeddcc  .."3DUfw........
```

Agenda

- Endianness
- **Two's Complement**
  - Useful properties
- Other signed representations
  - Bias-K Representation
  - Sign Magnitude
  - One's Complement

Recap: Signed vs. Unsigned Numbers

- **Sign:** the plus/minus sign you put in front of a number

sign → `-10` / `+10`

TIL Java doesn't have these ☹

	Signed Positive or negative integers	Unsigned Nonnegative integers
Range	`(-inf, inf)`	`[0, inf)`
Examples	`-10` `10`	`0` `10` `31`

Use Cases	**Most use cases, really ☺**
	• Degrees Centigrade [-273, infinity]
	• Integer arithmetic
		When you're:
		• **Not doing arithmetic**
		  • ID numbers (no add/subtract)
		  • Using bitwise operations
		• **100% sure you don't need a negative number**
		  • Degrees Kelvin [0, inf)

Recap: Unsigned Number Representations

- It's just the binary representation of our value

Data value

$(97)_{10}$

Convert to base 2 →

8-Bit Representation

0
1

0110_0001

- With N bits…

Smallest Number

0
1

0000…0000

Value = 0

Largest Number

0
1

1111…1111

Value = 2^N-1

Today's Question

How can we represent **negative numbers** using bits?

Data value		**8-Bit Representation**
$(-27)_{10}$	??? →	0 / 1: ????_????

Ideal Properties of a Signed Representation

- Arithmetic should "make sense"

1 Let C : value → representation

We want that $C(0) = 0$

Arithmetic on Values
Programmer's Job

Arithmetic on Representations
Hardware's Job

2

$v_0 + v_1 = v_2$
$v_0 - v_1 = v_2$
$v_0 * v_1 = v_2$

Programmer and hardware ↔ should do the same thing

$C(v_0) + C(v_1) = C(v_2)$
$C(v_0) - C(v_1) = C(v_2)$
$C(v_0) * C(v_1) = C(v_2)$

"Sensible" Arithmetic

- Operations on **representations** should mirror operations on **values**

Data values

```
int8_t a = -27;
```

```
int8_t b = 27;
```

```
c = a + b;
```

Programmer thinks a base-10 add

Hardware performs a base 2 add

8-Bit Representations

&a: $a_7 a_6 a_5 a_4 \ a_3 a_2 a_1 a_0$

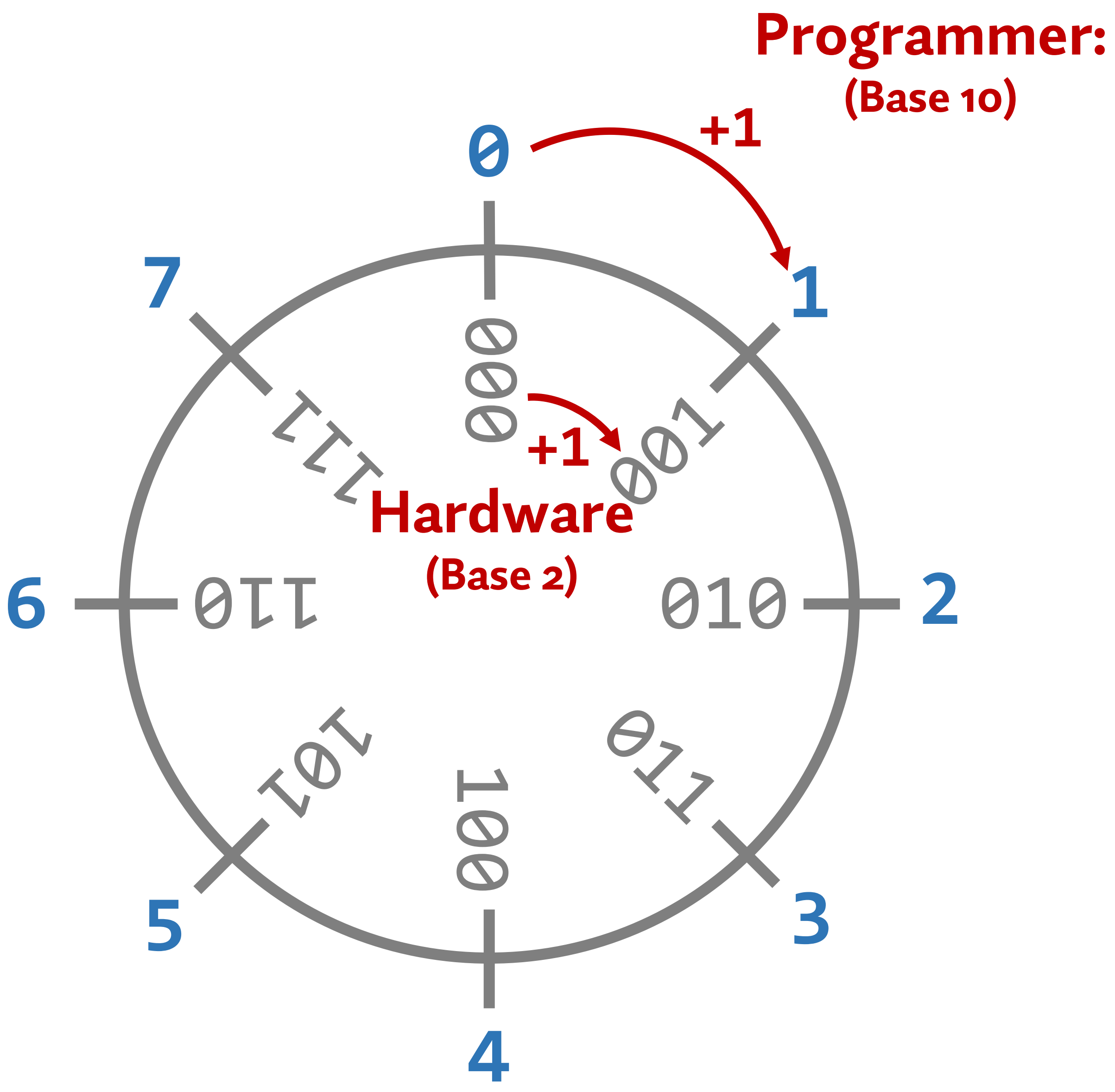
+

&b: $b_7 b_6 b_5 b_4 \ b_3 b_2 b_1 b_0$

=

&c: 0000 0000

Unsigned Representation Properties



✓ Easy to convert from value to representation

- R = $(V)_2$

✓ C(0) = 0

✓ Sensible arithmetic (mod 2^3)

- Overflow is OK
- Programmer and hardware do the same thing

✗ Negative numbers

Extending Unsigned to Signed

- **Key idea**: generate the negative numbers through **subtraction**
  - Find the "additive inverse" of each positive representation

$(001)_2 + X = (000)_2$

$(001)_2 + X = (1000)_2$ (equivalent mod 2^3)

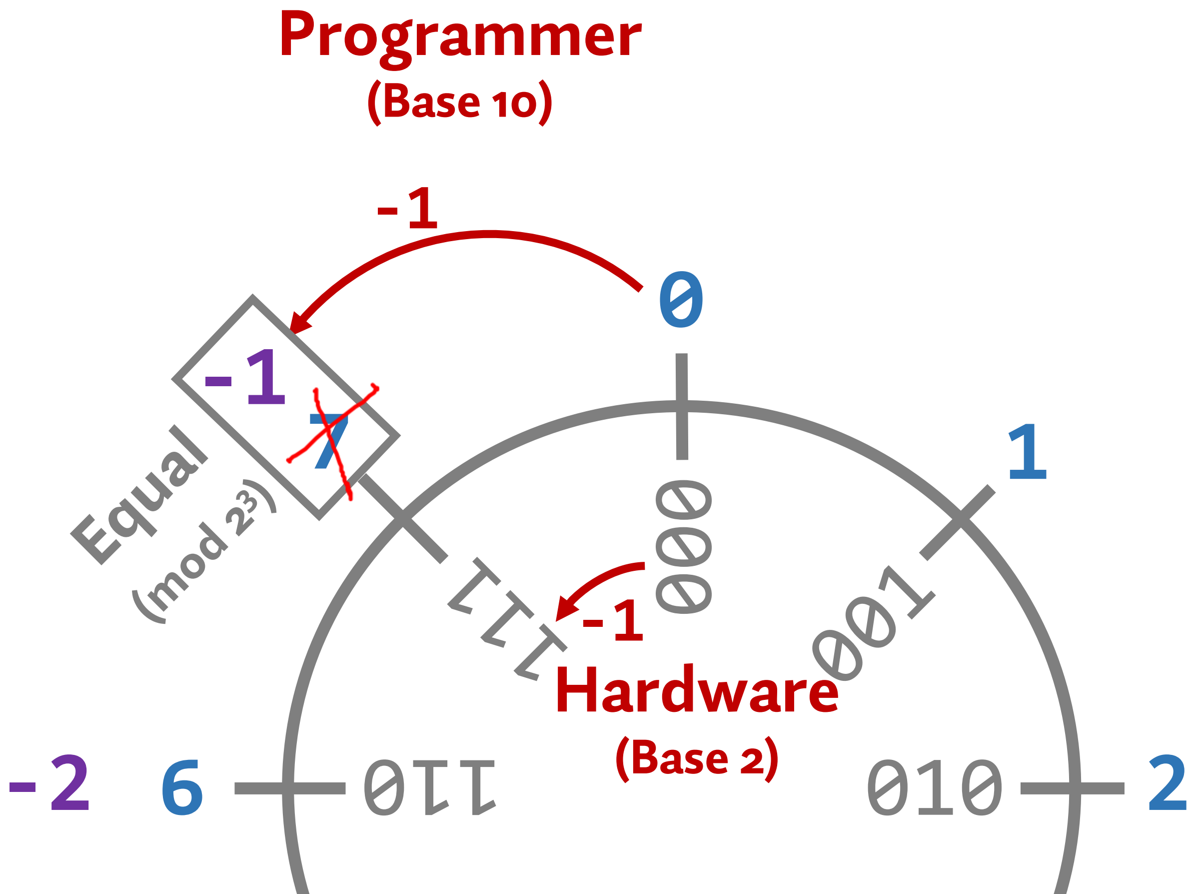
$X = (1000)_2 - (001)_2$

$$\begin{array}{r} 1000 \\ -\quad 1 \\ \hline 111 \end{array}$$

$X = (111)_2 \rightarrow (-1)_{10}$

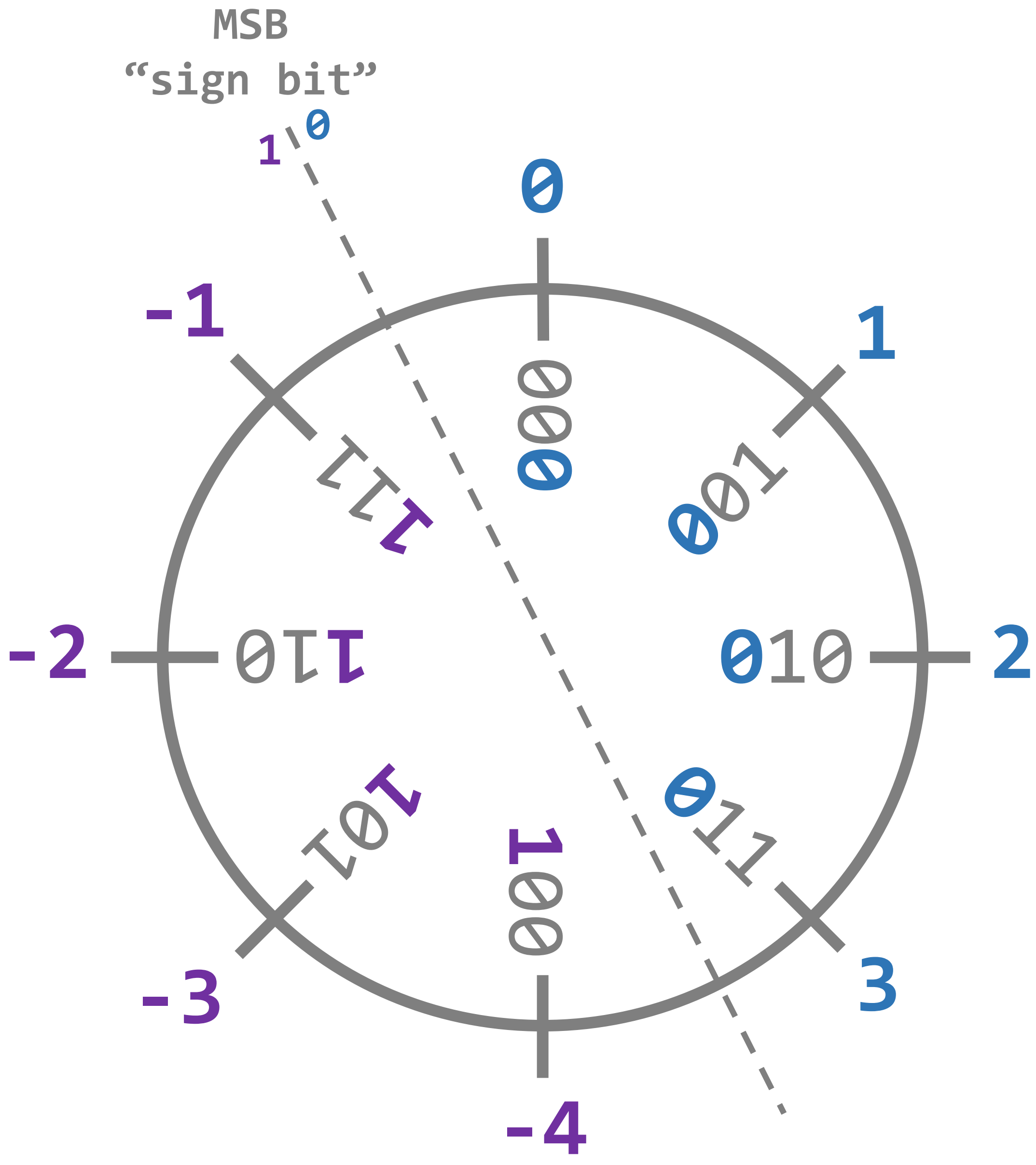
Extending Unsigned to Signed

- **Key idea**: generate the negative numbers through **subtraction**
  - Find the "additive inverse" of each positive representation

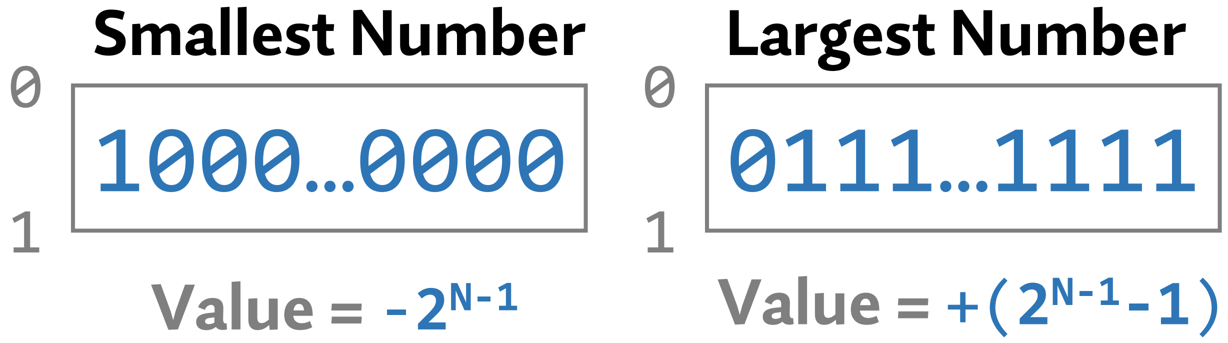


Programmer (Base 10, mod 8)	Hardware (Base 2, mod 2^3)
1 + (-1) = 0	001 + 111 = 1000
2 + (-2) = 0	010 + 110 = 1000
X + (-X) = 0	X + -X = 1000

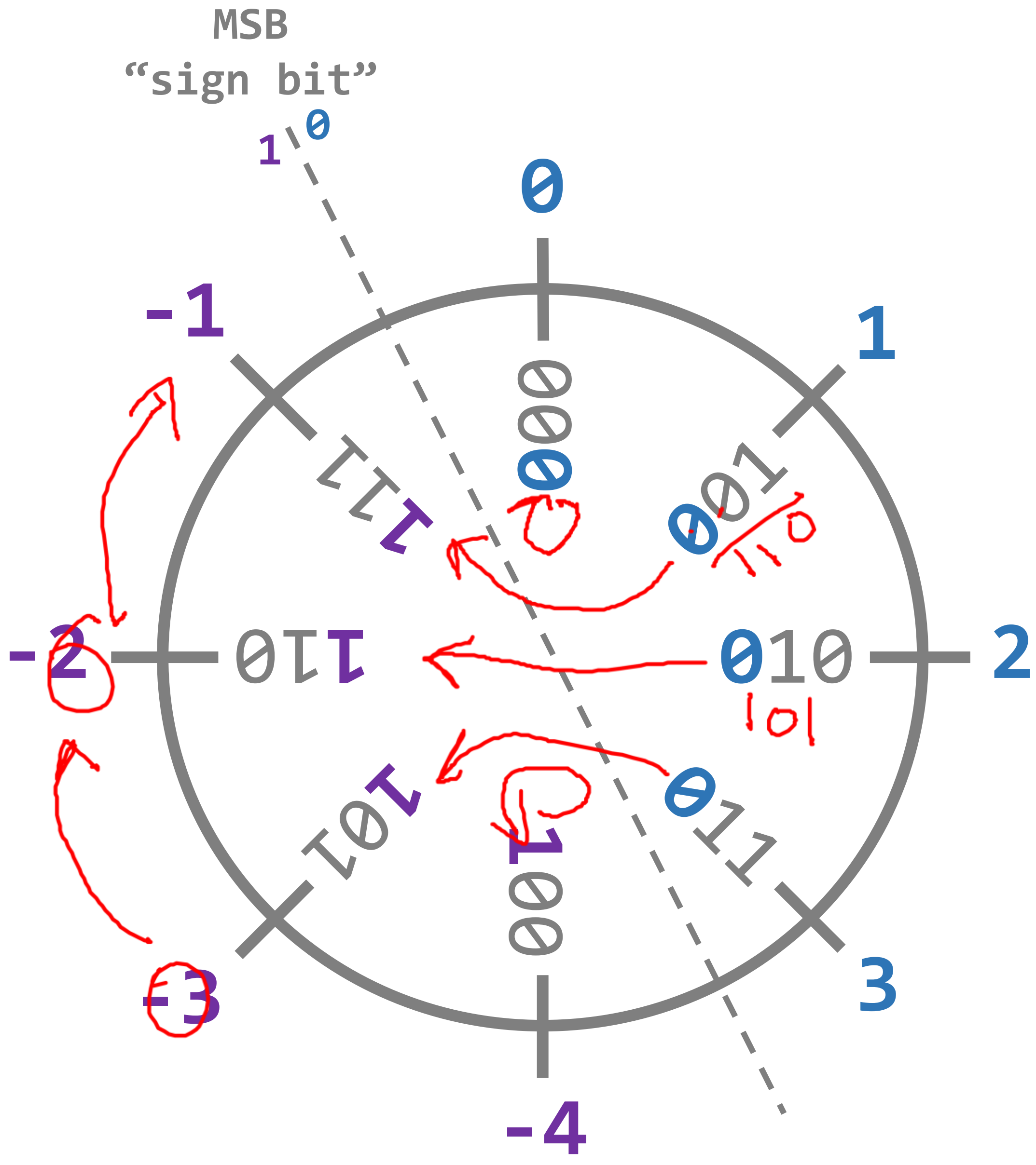
Two's Complement Representation



- "Sign bit" helps make hardware efficient



Two's Complement Representation



- Converting to 2's complement

$$R = \begin{cases} (V)_2 & V \geq 0 \\ (\sim|V| + 1)_2 & V < 0 \end{cases}$$

1. "Flip" the bits of $(|V|)_2$
2. Add one

- Converting from 2's complement

$$V = \begin{cases} (R)_{10} & MSB = 0 \\ -(\sim|R| + 1)_{10} & MSB = 1 \end{cases}$$

1. "Flip" the bits of R
2. Add one
3. Add the minus sign

N-Bit Two's Complement to Decimal

$$V = \begin{cases} (R)_{10} & MSB = 0 \\ -(\sim|R| + 1)_{10} & MSB = 1 \end{cases}$$

1. "Flip" the bits of R
2. Add one
3. Add the minus sign

$(+9)_{10} = (01001)_2$

$(-9)_{10} = (10111)_2$

Decimal to N-Bit Two's Complement

$$R = \begin{cases} (V)_2 & V \geq 0 \\ (\sim|V| + 1)_2 & V < 0 \end{cases}$$

1. "Flip" the bits of $(|V|)_2$
2. Add one

$(+9)_{10} = (01001)_2$

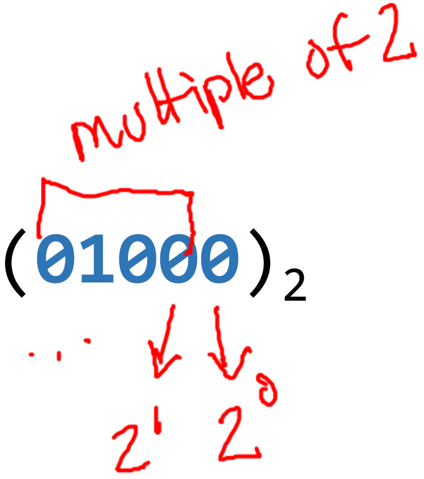
$(-9)_{10} = (10111)_2$

- Always double-check the sign bit
  - $|V|$ must fit in $N-1$ bits

Sanity Checks with Two's Complement

- **Check the MSB:** sign is positive (0) or negative (1)
- **Check the LSB:** the number is even (0) or odd (1)

$(+8)_{10} = (01000)_2$

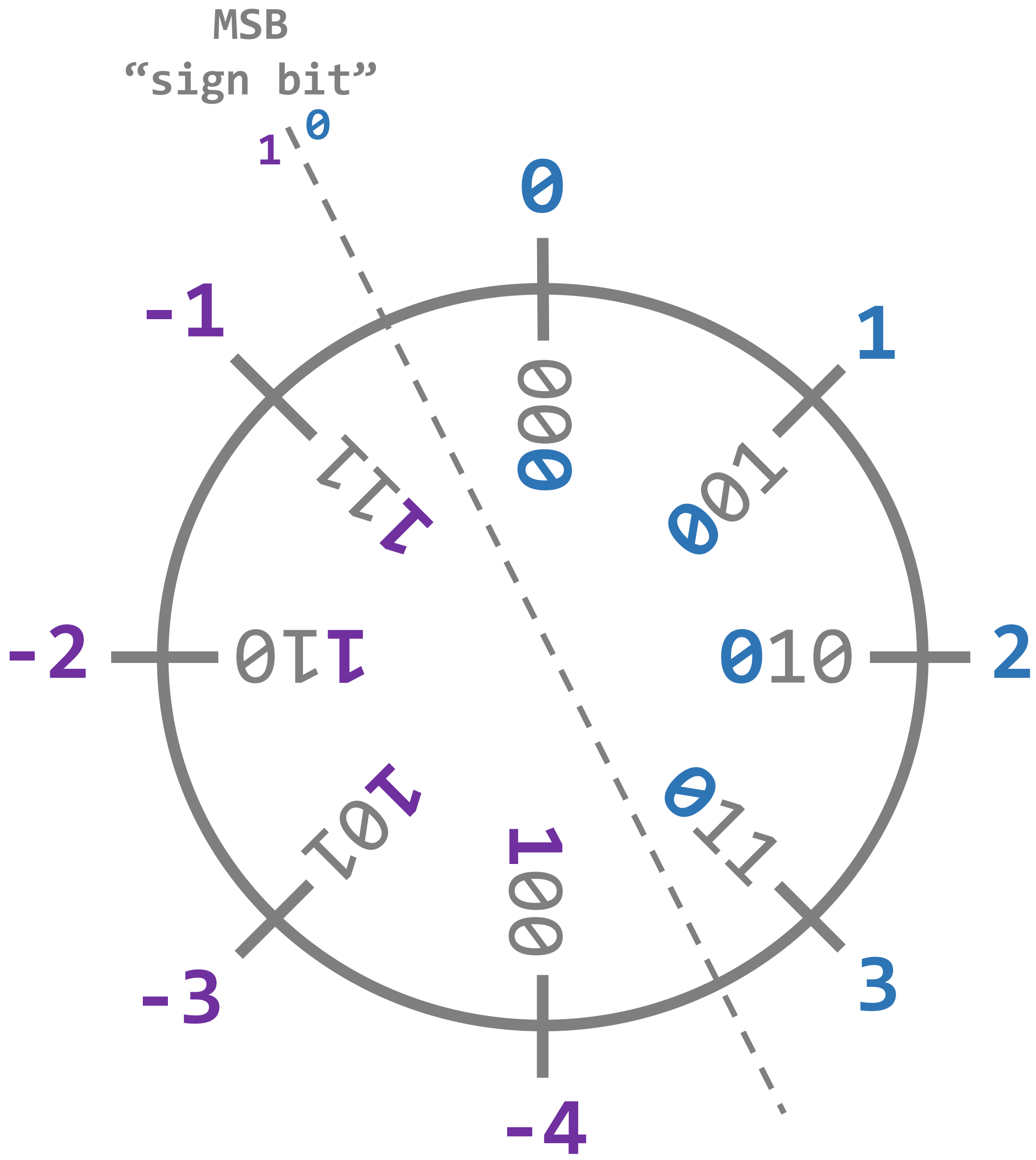


$(+9)_{10} = (01001)_2$

$(-8)_{10} = (11000)_2$

$(-9)_{10} = (10111)_2$

Two's Complement Properties



- ✗ Strange conversion from value to representation
  - $R = \begin{cases} (V)_2 & V \geq 0 \\ (\sim|V| + 1)_2 & V < 0 \end{cases}$
- ✓ $C(0) = 0$
- ✓ Sensible arithmetic (mod 2^3)
  - Overflow is OK
  - Programmer and hardware do the same thing
- ✓ Negative numbers

Integer Representations

Unsigned

✓ Simple code
✓ `C(0) = 0`
✓ Sensible arithmetic
✗ Negative numbers

Two's Complement

✗ Simple code
✓ `C(0) = 0`
✓ Sensible arithmetic
✓ Negative numbers

Agenda

- Endianness
- Two's Complement
  - **Useful properties**
- Other signed representations
  - Bias-K Representation
  - Sign Magnitude
  - One's Complement

Two's Complement and Positional Notation

- MSB's weight is now **a negative power of two**

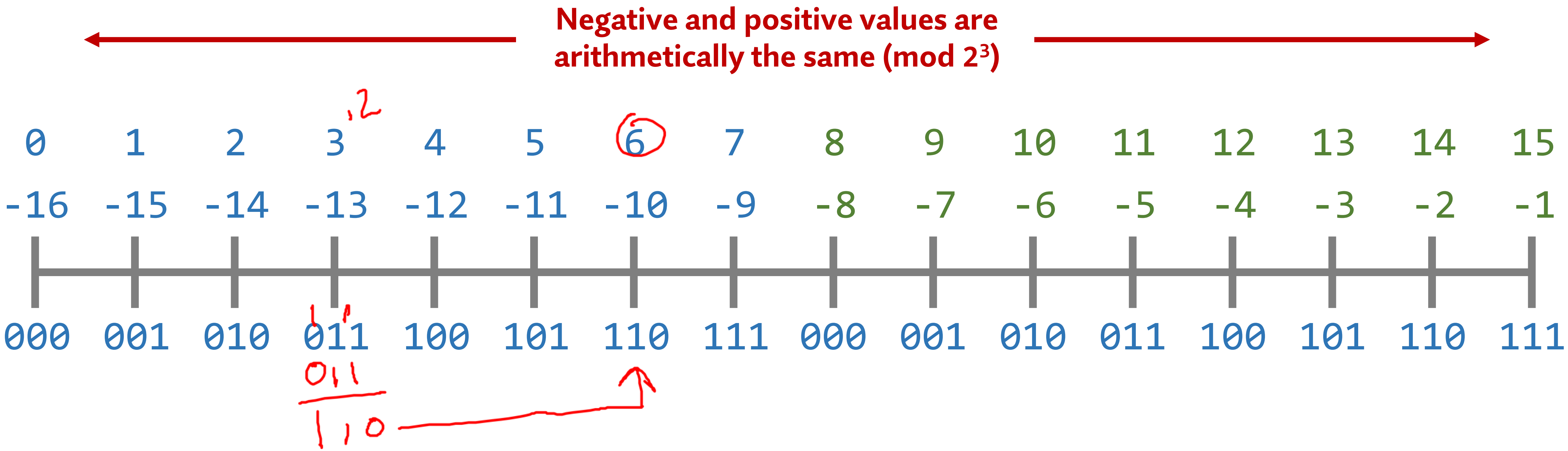
5-Bit Two's Complement

$$(+9)_{10} = -0\cdot 2^4 + 1\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0$$

$$(-9)_{10} = -1\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0$$

$-16 \quad 8+4+2+1=15 \quad = -1$

Unsigned and Two’s Complement Arithmetic



- Unsigned and Two’s Complement arithmetic is identical **by construction**
  - We defined negative numbers to be equivalent to positive numbers **mod 2^3**
  - You can add/sub/mul **two’s complement representations** as if they are unsigned
    - You’ll always get the right answer (mod 2^3)

Unsigned and Two's Complement in C

Unsigned

- `unsigned char`
- `unsigned short`
- `unsigned int`
- `unsigned long`
- `unsigned long long`
- `uint{8,16,32,64}_t`

Signed
(Two's Complement)

- `signed char`
- `signed short`
- `signed int`
- `signed long`
- `signed long long`

Careful! Unstandardized!
Unsigned on some compilers

- ~~`char`~~
- `short`
- `int`
- `long`
- `long long`
- `int{8,16,32,64}_t`

Data value

$(-7)_{10}$ → **int8_t Object**

a
1111_1001 = 0xf9
a+1

Data value

$(249)_{10}$ → **uint8_t Object**

a
1111_1001 = 0xf9
a+1

Agenda

- Endianness
- Two's Complement
  - Useful properties
- **Other signed representations**
  - Bias-K Representation
  - Sign Magnitude
  - One's Complement

Integer Representations

Most computers use these

Unsigned	Two's Complement
✓ Simple code	✗ Simple code
✓ `C(0) = 0`	✓ `C(0) = 0`
✓ Sensible arithmetic	✓ Sensible arithmetic
✗ Negative numbers	✓ Negative numbers

Bias-K **Sign Magnitude**

Part of float representation
(also used in most computers)

One's Complement

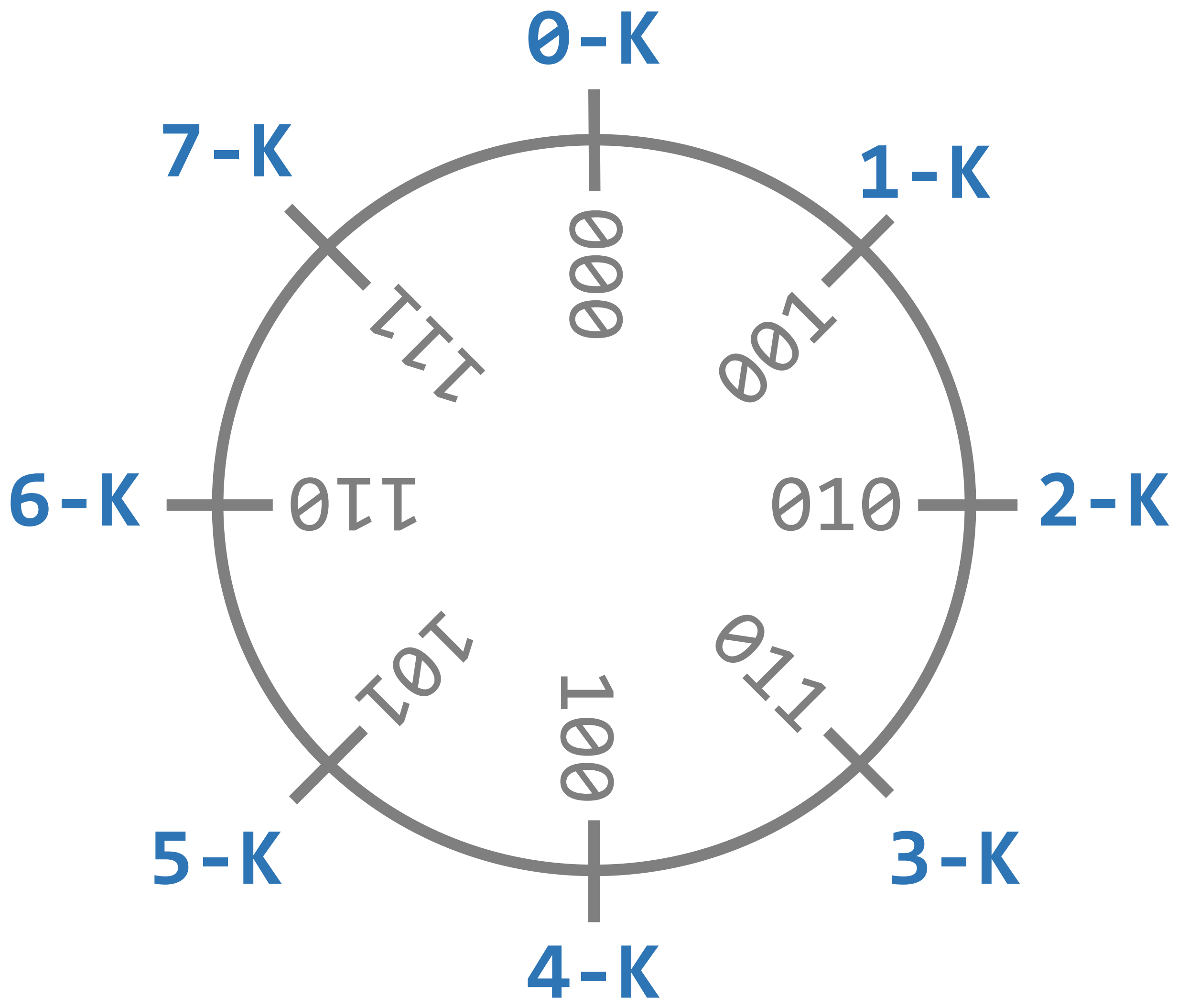
Sometimes useful

Agenda

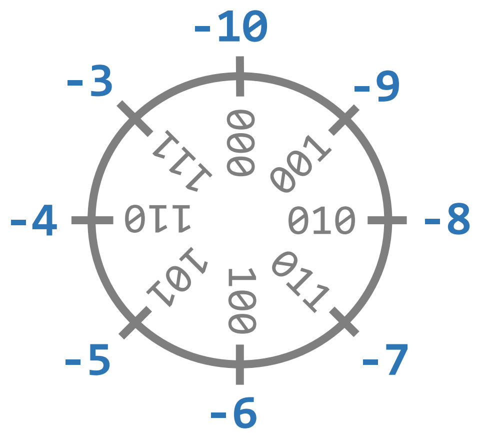
- Endianness
- Two's Complement
  - Useful properties
- Other signed representations
  - **Bias-K Representation**
  - Sign Magnitude
  - One's Complement

Bias-K Representation

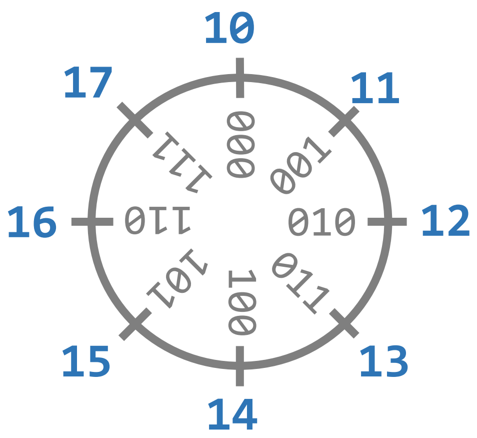
- **Key idea**: shift all values by some integer K



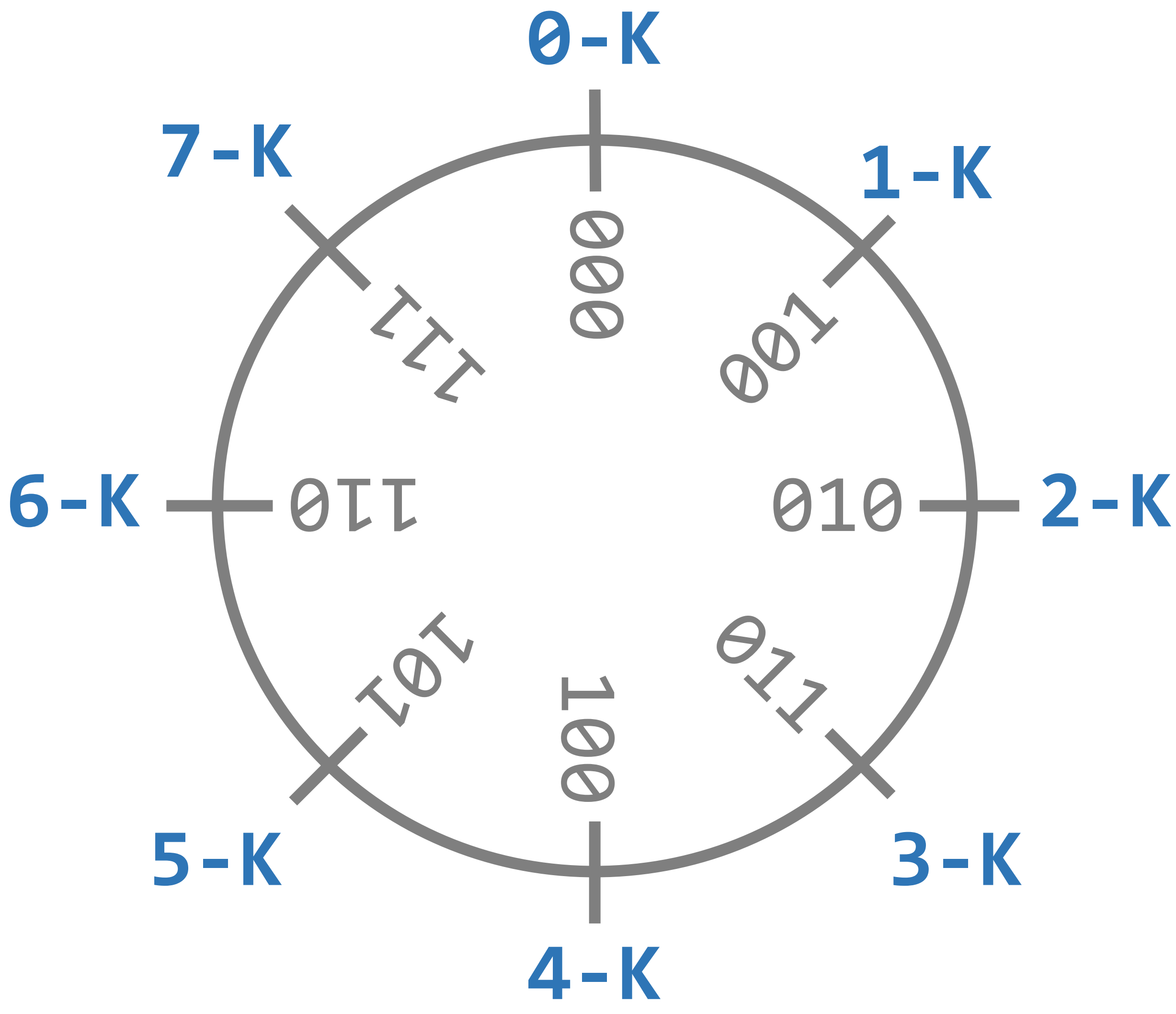
K = 10



K = -10



Bias-K Arithmetic



- Arithmetic is weird for **K≠0**

Arithmetic with **one** value is OK

```
(0-K) +   1 = (1-K)
  000 + 001 =   001
```

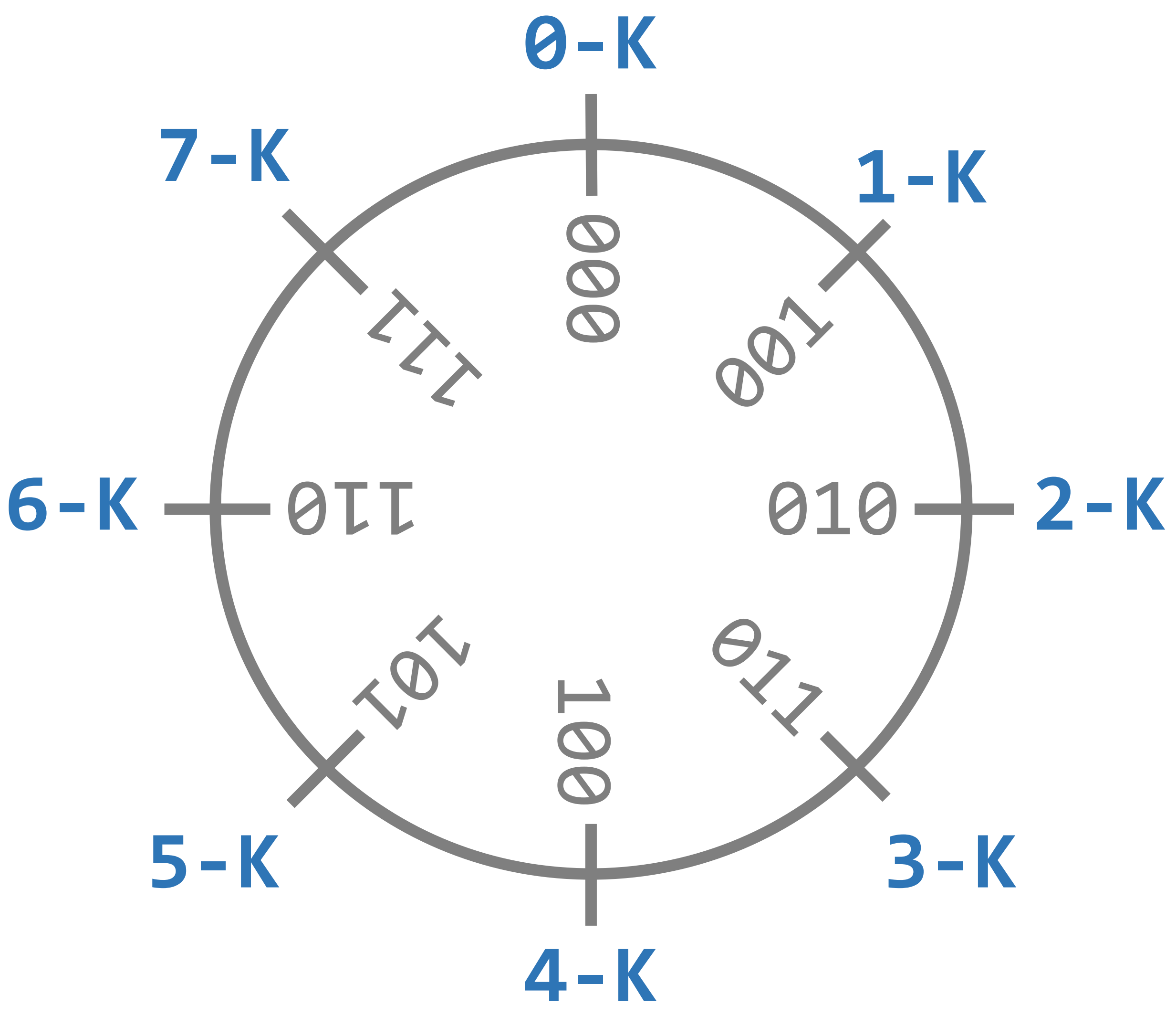
✓

Arithmetic with **two** values is NOT

```
(0-K) + (1-K) = (1-2K)
  000 +   001 =    001
```

✗

Bias-K Representation



✓ Easy to convert from value to representation

- R = $(V-K)_2$

✗ C(0) = 0

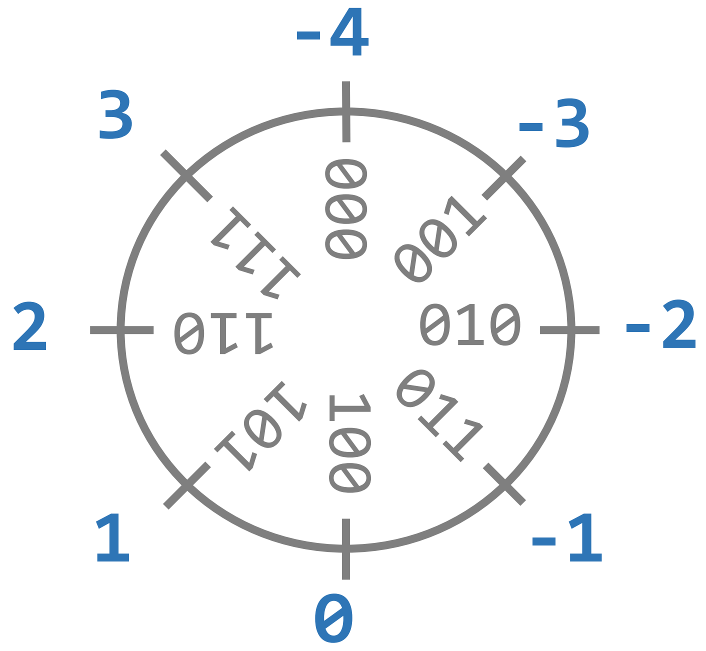
✗ Sensible arithmetic (mod 2^3)

- Add/subtract are weird
- Might not even have a zero

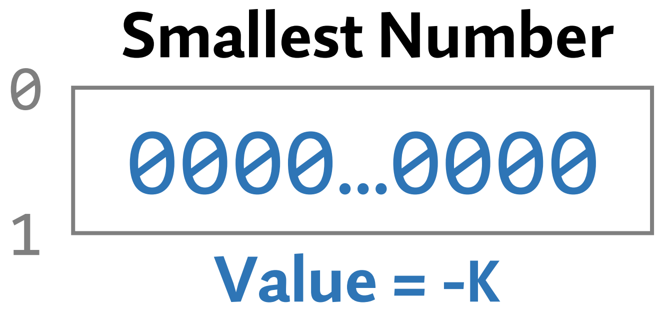
✓ Negative numbers

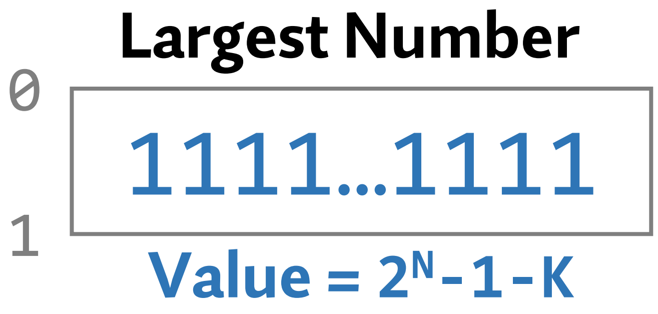
Bias-K with "Standard Bias"

$K = 2^{N-1}$



- Same **values** as 2C, different **representation**
  - Wheel is rotated 180 degrees
  - MSB is inverted
- Bias-K preserves total order over representations
  - Useful for logical comparisons on the representations





Integer Representations

Unsigned

- ✓ Simple code
- ✓ `C(0) = 0`
- ✓ Sensible arithmetic
- ✗ Negative numbers

Two's Complement

- ✗ Simple code
- ✓ `C(0) = 0`
- ✓ Sensible arithmetic
- ✓ Negative numbers

Bias-K

- ✓ Simple code
- ✗ `C(0) = 0`
- ✗ Sensible arithmetic
- ✓ Negative numbers

Sign Magnitude

One's Complement

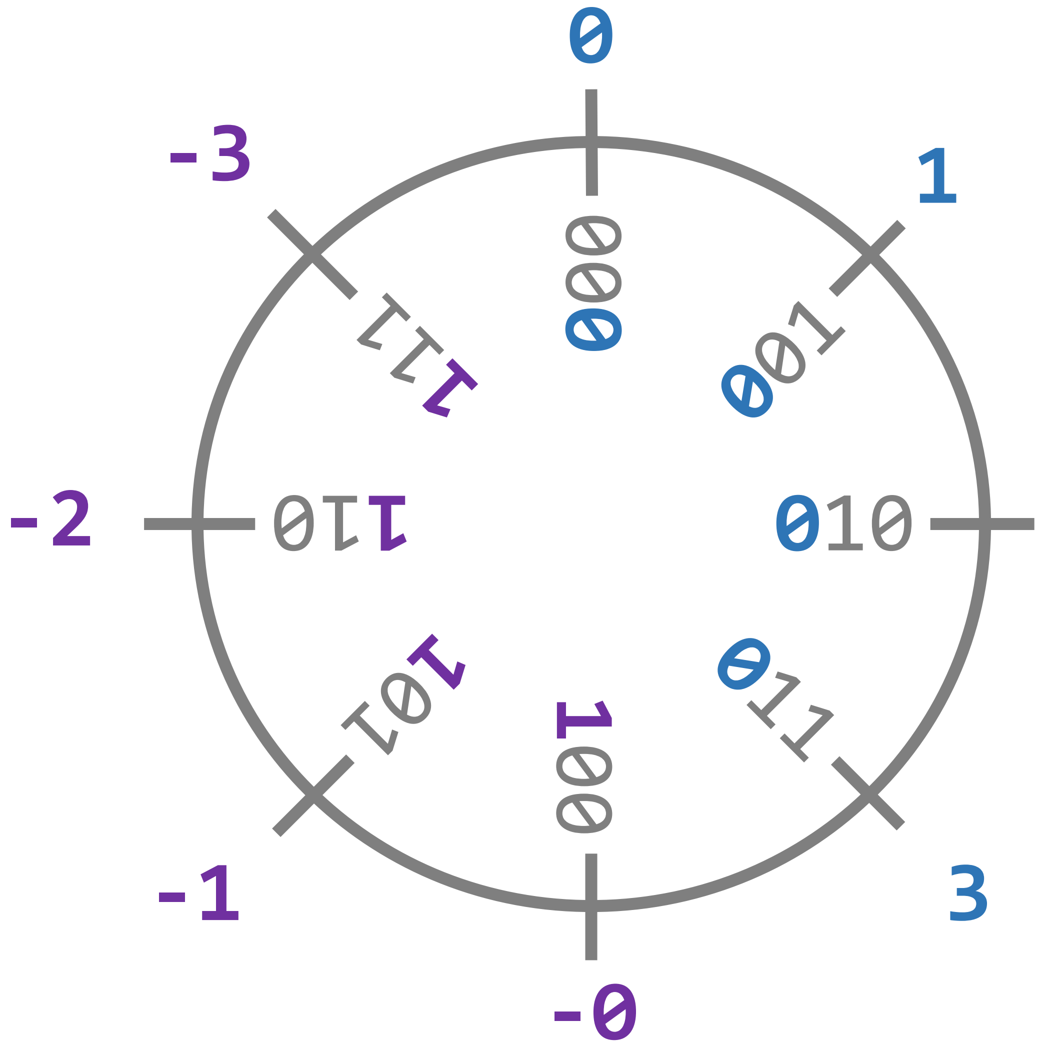
Agenda

- Endianness
- Two's Complement
  - Useful properties
- Other signed representations
  - Bias-K Representation
  - **Sign Magnitude**
  - One's Complement

Sign Magnitude Representation

- **Key idea**: MSB indicates positive (0) or negative (1)

Data value	8-Bit Representation
$(-27)_{10}$	1 | 001_1011
$(27)_{10}$	0 | 001_1011
$(-X)_{10}$	1 | $(X)_2$
$(X)_{10}$	0 | $(X)_2$



Sign Magnitude Representation

- Comparable to human notation (e.g., -27 vs +27)

Data value	Data Representation
$-(X)_{10}$	0 | 1 | $(X)_2$ | 1
$+(X)_{10}$	0 | 0 | $(X)_2$ | 1

Smallest Number

0 | 1 | 11…11 | 1

Value = $-(2^{N-1}-1)$

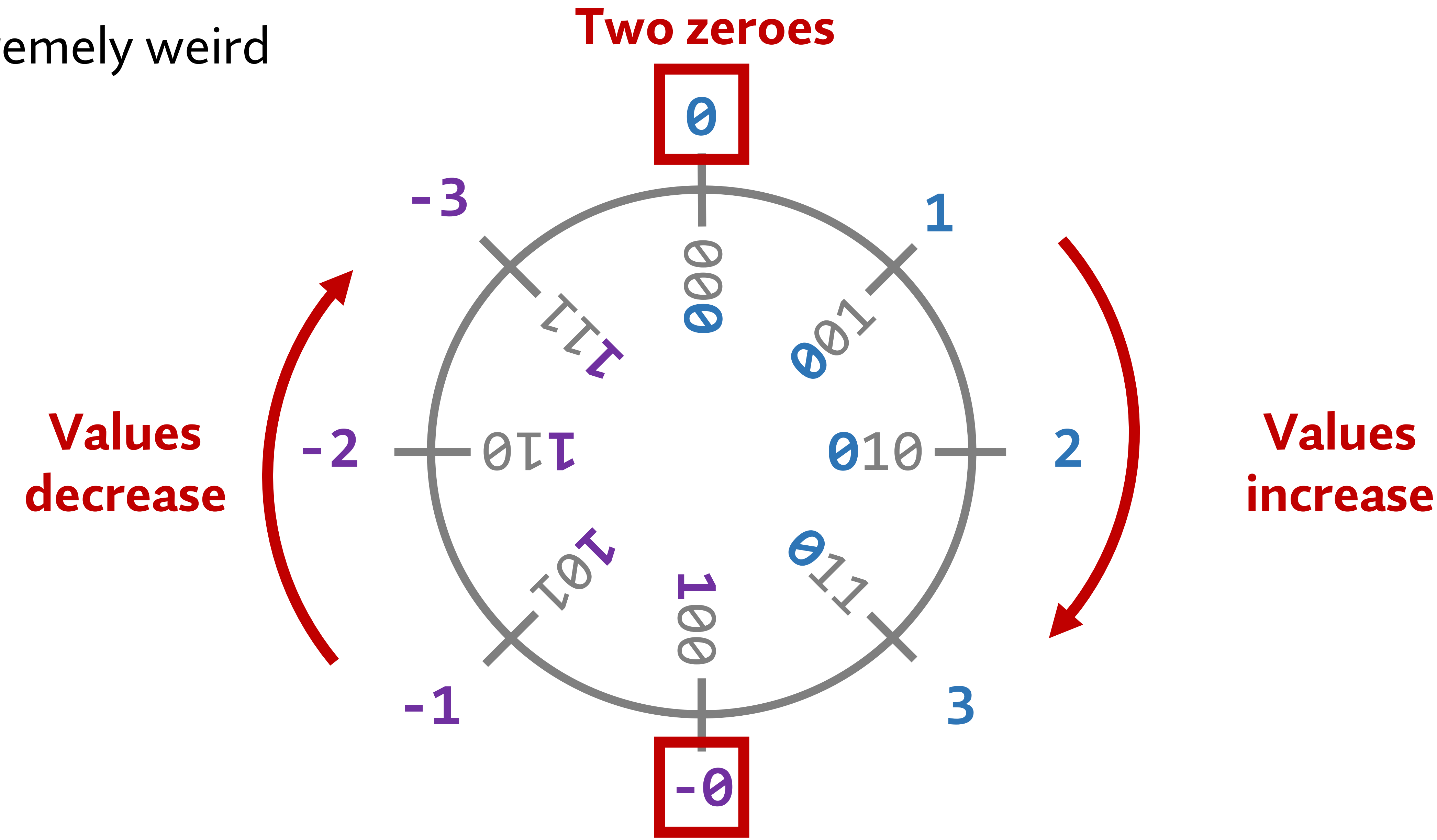
Largest Number

0 | 0 | 11…11 | 1

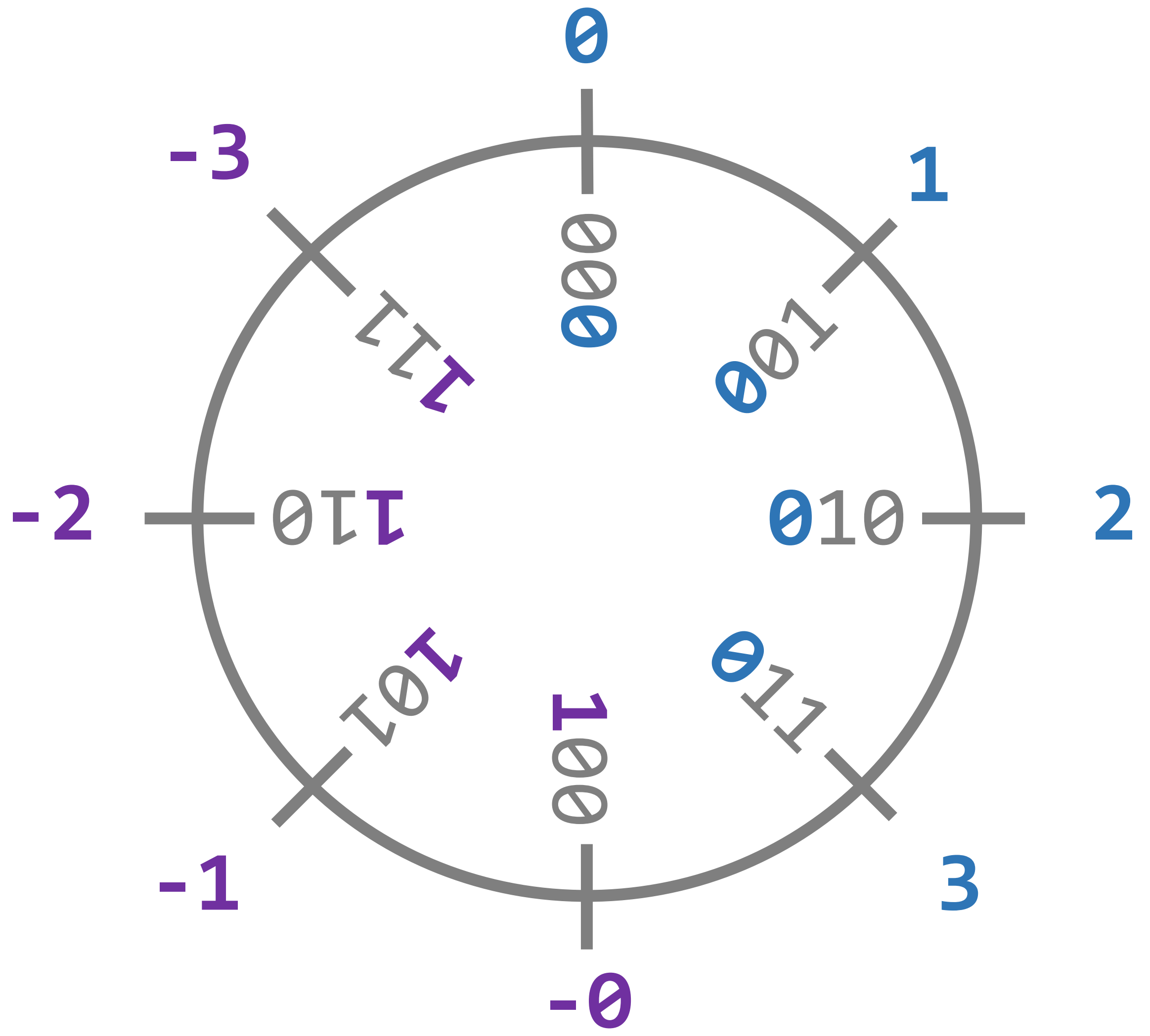
Value = $+(2^{N-1}-1)$

Sign Magnitude Arithmetic

- Extremely weird



Sign Magnitude Properties



✓ Easy to convert from value to representation

- R = [sign | $(|V|)_2$]

✓ C(0) = 0

✗ Sensible arithmetic (mod 2^3)

- Two zeroes
- Values are non-monotonic
- Add/subtract don't work

✓ Negative numbers

Integer Representations

Unsigned

- ✓ Simple code
- ✓ `C(0) = 0`
- ✓ Sensible arithmetic
- ✗ Negative numbers

Two's Complement

- ✗ Simple code
- ✓ `C(0) = 0`
- ✓ Sensible arithmetic
- ✓ Negative numbers

Bias-K

- ✓ Simple code
- ✗ `C(0) = 0`
- ✗ Sensible arithmetic
- ✓ Negative numbers

Sign Magnitude

- ✓ Simple code
- ✓ `C(0) = 0`
- ✗ Sensible arithmetic
- ✓ Negative numbers

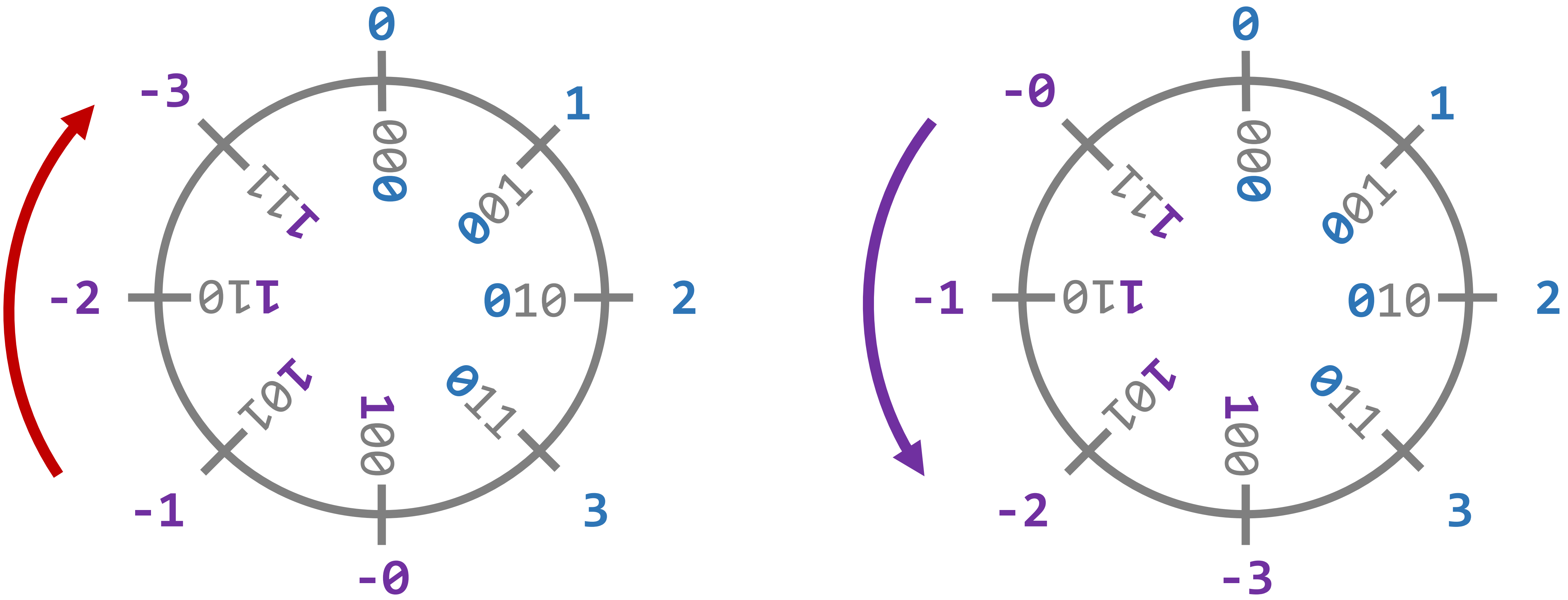
One's Complement

Agenda

- Endianness
- Two's Complement
  - Useful properties
- Other signed representations
  - Bias-K Representation
  - Sign Magnitude
  - **One's Complement**

One's Complement Representation

- **Key idea**: correct the total order on sign magnitude representation



$$\mathbf{R} = \begin{cases} 0 \mid (V)_2 & V \geq 0 \\ 1 \mid (\sim|V|)_2 & V < 0 \end{cases}$$

One's Complement Representation

Data value	8-Bit Representation
$(-27)_{10}$	1 | 110_0100
$(27)_{10}$	0 | 001_1011

1 111_1111

Smallest Number

1 | 00…00

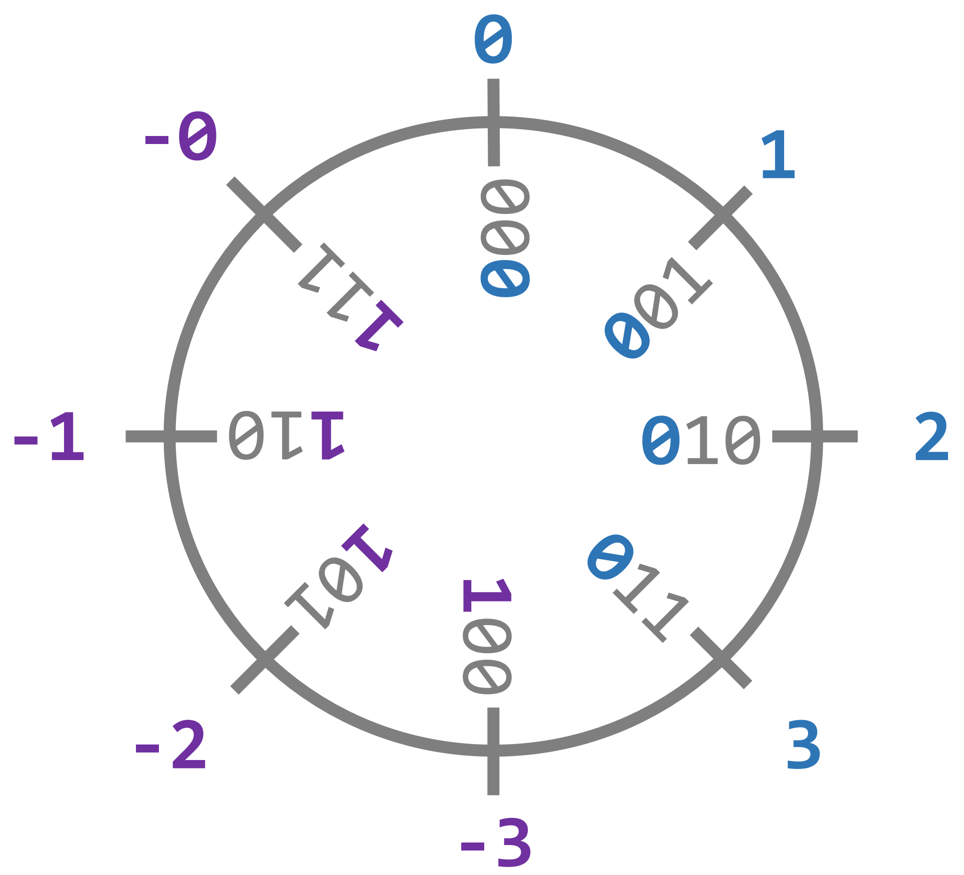
Value = $-(2^{N-1}-1)$

Largest Number

0 | 11…11

Value = $+(2^{N-1}-1)$

One's Complement Properties



✗ Strange conversion from value to representation

$$R = \begin{cases} 0 \mid (V)_2 & V \geq 0 \\ 1 \mid (\sim|V|)_2 & V < 0 \end{cases}$$

✓ $C(0) = 0$

✗ Sensible arithmetic (mod 2^3)

- Two zeroes
- Add/subtract make no sense

✓ Negative numbers

Integer Representations

Unsigned

- ✓ Simple code
- ✓ `C(0) = 0`
- ✓ Sensible arithmetic
- ✗ Negative numbers

Two's Complement

- ✗ Simple code
- ✓ `C(0) = 0`
- ✓ Sensible arithmetic
- ✓ Negative numbers

Bias-K

- ✓ Simple code
- ✗ `C(0) = 0`
- ✗ Sensible arithmetic
- ✓ Negative numbers

Sign Magnitude

- ✓ Simple code
- ✓ `C(0) = 0`
- ✗ Sensible arithmetic
- ✓ Negative numbers

One's Complement

- ✗ Simple code
- ✓ `C(0) = 0`
- ✗ Sensible arithmetic
- ✓ Negative numbers

Thursday: Floating-Point

Data value

$(-6.022 \times 10^{23})_{10}$

Significand
Sign Magnitude

Exponent
Bias-K (K=127)

Data Representation

0 | 1 | e_6-e_0 | s_{23}-s_0 | 1

CS 211: Intro to Computer Architecture

3.1: Character and Integer Representations Cntd.

Minesh Patel

Spring 2025 – Thursday 30 January