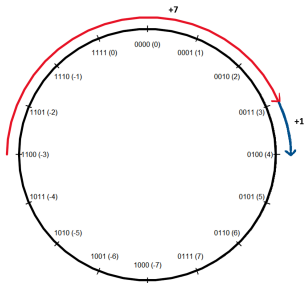
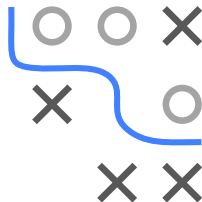


# Algorithms and Data Structures

## Encoding

## Machine numbers for $\mathbb{Z}$



### Learning goals

- Signed magnitude representation
- One's complement
- Two's complement
- Integer overflow

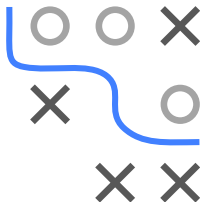
# MACHINE NUMBERS FOR $\mathbb{Z}$

There are different options to represent positive and negative integers ( $\mathbb{Z}$ ) by a computer:

- Signed magnitude representation
- Excess encoding
- One's complement
- Two's complement

Each representation has advantages and disadvantages regarding:

- Symmetry of the representable value range
- Uniqueness of representation
- Execution of arithmetic operations



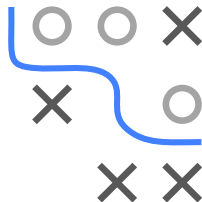
# SIGNED MAGNITUDE REPRESENTATION

If the 32nd bit is reserved for the sign on a 32-bit computer, 31 bits are available for encoding the absolute value of the number.

	sign	31	...	8	7	6	5	4	3	2	1
-1	1	0	...	0	0	0	0	0	0	0	1
0	1/0	0	...	0	0	0	0	0	0	0	0
1	0	0	...	0	0	0	0	0	0	0	1
51	0	0	...	0	0	1	1	0	0	1	1
		$2^{30}$	...	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$

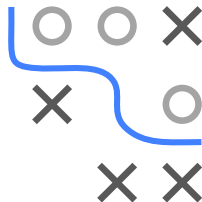
The number is then given by:  $x = (-1)^{u_{32}} \sum_{i=1}^{31} u_i 2^{i-1}$ .

The sign bit  $u_{32} = 1$  indicates negative numbers.



# SIGNED MAGNITUDE REPRESENTATION / 2

- Covered number range in 32-bit:  $-2^{31} + 1$  to  $2^{31} - 1$
- Very good readability
- Representation of zero not unique (e.g. problem with equality check:  $-0 \neq 0$ )
- Addition/subtraction is cumbersome, since the sign bit must be handled separately. You cannot simply write and add two numbers below each other (but this would be desirable!).



Example  $7 - 3$  in 4-bit system:

$$\begin{array}{r} 0111 \quad |(7) \\ + \quad 1011 \quad |(-3) \\ \hline (1)0010 \quad |(2) \end{array}$$

But  $0010_2 \neq 4_{10}$ . Implementation of addition is complicated.





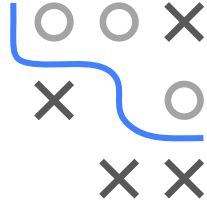
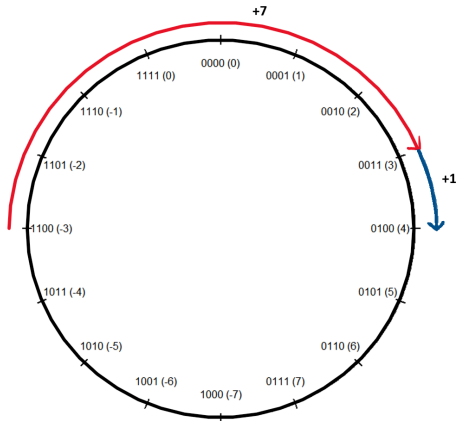






# ONE'S COMPLEMENT / 4

Why adding a 1?



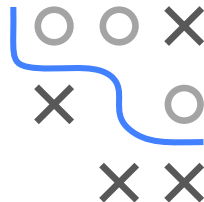
- Because of the 0 being represented twice, a 1 must be added when overflowing the 0, so that the result is correct.



# TWO'S COMPLEMENT / 2

Example: two's complement

	Bit $u_i$										
	32	31	...	8	7	6	5	4	3	2	1
$-2^{31}$	1	0	...	0	0	0	0	0	0	0	0
-51	1	1	...	1	1	0	0	1	1	0	1
-1	1	1	...	1	1	1	1	1	1	1	1
0	0	0	...	0	0	0	0	0	0	0	0
1	0	0	...	0	0	0	0	0	0	0	1
51	0	0	...	0	0	1	1	0	0	1	1
$2^{31} - 1$	0	1	...	1	1	1	1	1	1	1	1
	$-2^{31}$	$2^{30}$	...	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$

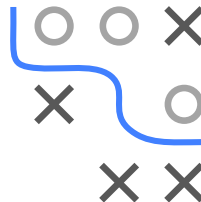
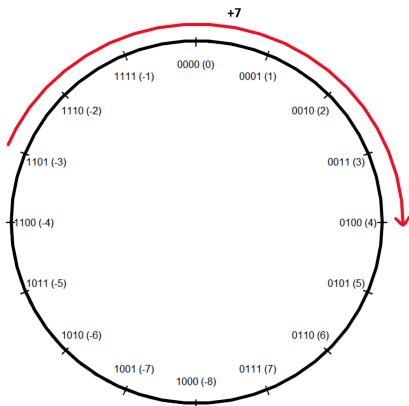


The coded number is then:  $x = \sum_{i=1}^{31} u_i 2^{i-1} - u_{32} 2^{31}$ .



# TWO'S COMPLEMENT / 4

The carry bit can simply be ignored here, since the representation of the 0 is unique:



- Caution when leaving the number range:
- Example:  $0011 (3) + 0101 (5) = 1000 (-8)$

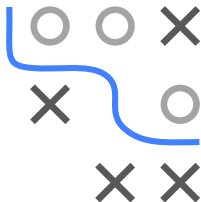
# INTEGER OVERFLOW

**Caution:** Arithmetic operations can cause an **overflow**. This is a common programming error in languages like C and can lead to undefined behavior (e.g. wrap around).

**Example:**  $(2^{31} - 1) + 1$ .

In a 32-bits two's complement representation  $(2^{31} - 1) + 1$  would be outside of the covered number range, since  $(2^{31} - 1)$  is the largest possible number that can be represented. Adding 1 results in  $-2^{31}$  due to an integer overflow.

$(2^{31} - 1)$	01111111	11111111	11111111	11111111
+1	00000000	00000000	00000000	00000001
$(-2^{31})$	10000000	00000000	00000000	00000000

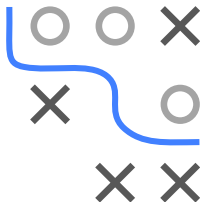


# INTEGER OVERFLOW / 2

Excerpt from Wikipedia "Integer Overflow":

*On 30 April 2015, the Federal Aviation Authority announced it will order Boeing 787 operators to reset its electrical system periodically, to avoid an integer overflow which could lead to loss of electrical power and ram air turbine deployment, and Boeing is going to deploy a software update in the fourth quarter.*

*When Donkey Kong breaks on level 22 it is because of an integer overflow in its time/bonus. Donkey Kong takes the level number you're on, multiplies it by 10 and adds 40. When you reach level 22 the time/bonus number is 260 which is too large for its 8-bit 256 value register so it resets itself to 0 and gives the remaining 4 as the time/bonus - not long enough to complete the level.*

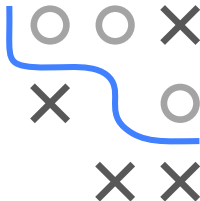


# MACHINE NUMBERS FOR $\mathbb{Z}$ IN $\mathbb{R}$

In  $\mathbb{R}$ , integers are encoded in 32-bit (also in 64-bit  $\mathbb{R}$ !) by the two's complement.

```
.Machine$integer.max  
## [1] 2147483647
```

```
2^31 - 1  
## [1] 2147483647
```



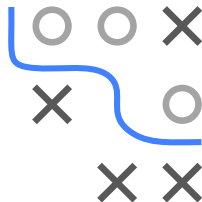


# MACHINE NUMBERS FOR $\mathbb{Z}$ IN $\mathbb{R} / 2$

```
intToBits(-51L)
## [1] 01 00 01 01 00 00 01 01 01 01 01 01 01 01 01 01 01 01
## [19] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
```

```
intToBits(51L)
## [1] 01 01 00 00 01 01 00 00 00 00 00 00 00 00 00 00 00 00
## [19] 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
# Caution: In R the operation x^y always results in the type 'numeric'!
intToBits(2L^31L - 1L)
## [1] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
## [19] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 00
```



# MACHINE NUMBERS FOR $\mathbb{Z}$ IN $\mathbb{R}$ / 3

In  $\mathbb{R}$ , integer overflows are caught and set to NA.

```
.Machine$integer.max + 1  
## [1] 2147483648
```

```
str(.Machine$integer.max + 1)  
## num 2.15e+09
```

```
str(.Machine$integer.max + 1L)  
## Warning in .Machine$integer.max + 1L: NAs produced by  
integer overflow  
## int NA
```

