Negative Numbers

CS143: lecture 15

Byron Zhong, July 20

What is this?

`1110 1001`

- 233
- Ascii é
- A pointer pointing to memory location 233 (0xE9)
- Sound wave sample with some magnitude
- A pixel intensity of 91.31%
- ...
- It's one of the 256 choices, whatever that is.

Negative Numbers

- If we have 32 bits, we have 2^{32} choices
  - unsigned: 2^{32} choices of non-negative numbers -- 0 through 2^{32} - 1
  - int: 2^{32} choices of both negative and positive numbers *about* -2^{31} through 2^{31}

Negative Numbers

Attempt 1

- The first bit is the sign, 0 -> positive, 1 -> negative

```
 5 = 0000 0000 0000 0000 0000 0000 0000 0011
-5 = 1000 0000 0000 0000 0000 0000 0000 0011
```

- Range?
  - $[-(2^{31} - 1), 2^{31} - 1]$

Negative Numbers

Attempt 1

- The first bit is the sign, 0 -> positive, 1 -> negative

```
 5 = 0000 0000 0000 0000 0000 0000 0000 0011
-5 = 1000 0000 0000 0000 0000 0000 0000 0011
```

- What's wrong with this?

```
   0000 0000 0000 0000 0000 0000 0000 0000
   1000 0000 0000 0000 0000 0000 0000 0000
```

Negative Numbers

Attempt 1

- The first bit is the sign, 0 -> positive, 1 -> negative

```
 5 = 0000 0000 0000 0000 0000 0000 0000 0011
-5 = 1000 0000 0000 0000 0000 0000 0000 0011
```

- What's wrong with this?

```
0000 0000 0000 0000 0000 0000 0000 0000
1000 0000 0000 0000 0000 0000 0000 0000
```

- We have +0 and -0 !?
  - Waste a combination, equality check is complicated +0 == -0

Negative Numbers

Attempt 1

- The first bit is the sign, 0 -> positive, 1 -> negative

```
 5 = 0000 0000 0000 0000 0000 0000 0000 0011
-5 = 1000 0000 0000 0000 0000 0000 0000 0011
```

- What's wrong with this?

```
   0000 0000 0000 0000 0000 0000 0000 0011
 + 1000 0000 0000 0000 0000 0000 0000 0011
```

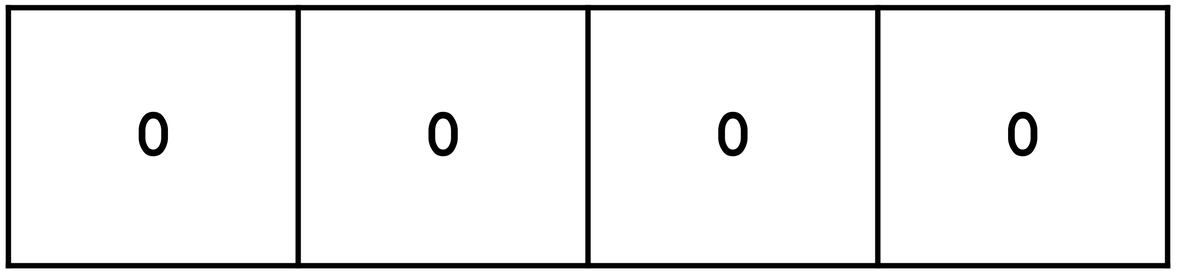
- 5 + (-5) is not 0 :(((
  - Need special circuit for adding signed numbers and unsigned numbers

Negative Numbers

Better Idea: Two's Complement

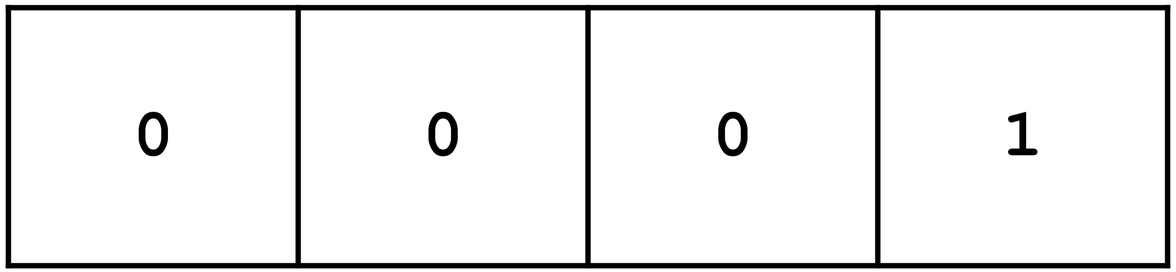
Negative Numbers

Better Idea: Two's Complement



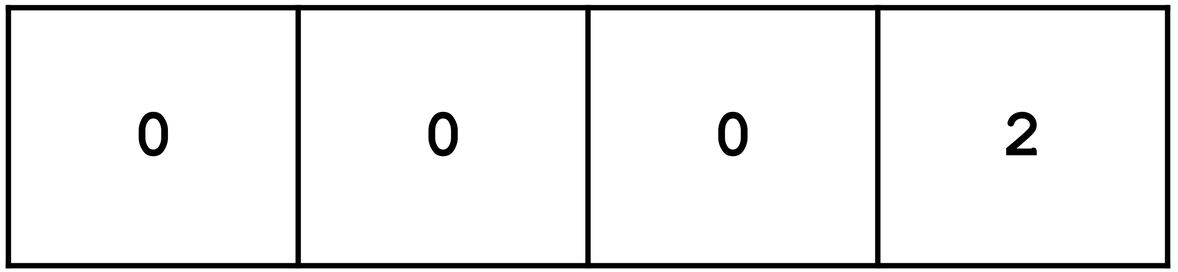
Negative Numbers

Better Idea: Two's Complement



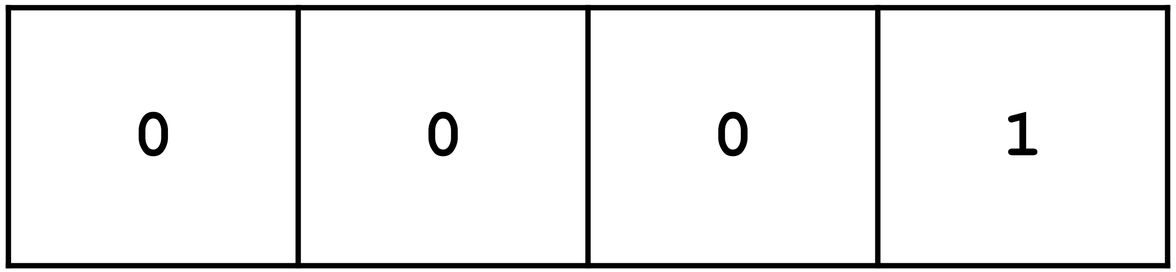
Negative Numbers

Better Idea: Two's Complement



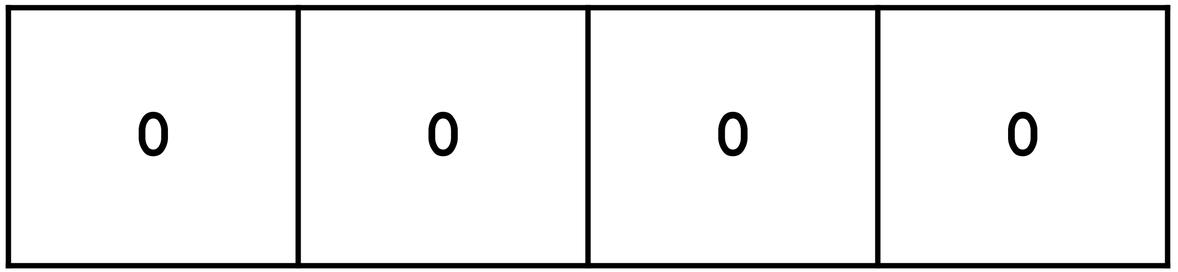
Negative Numbers

Better Idea: Two's Complement



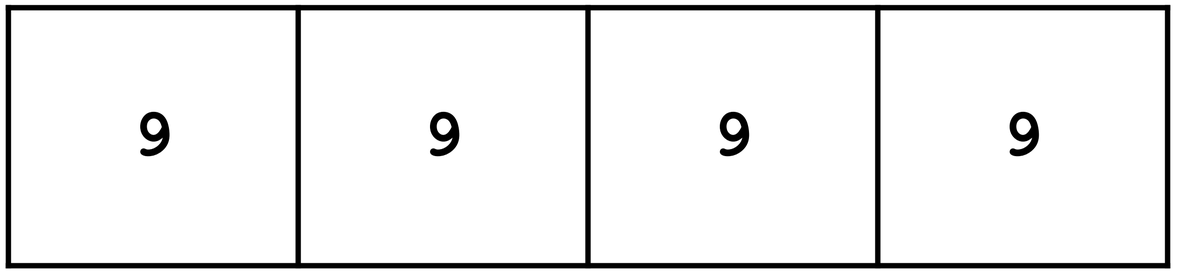
Negative Numbers

Better Idea: Two's Complement



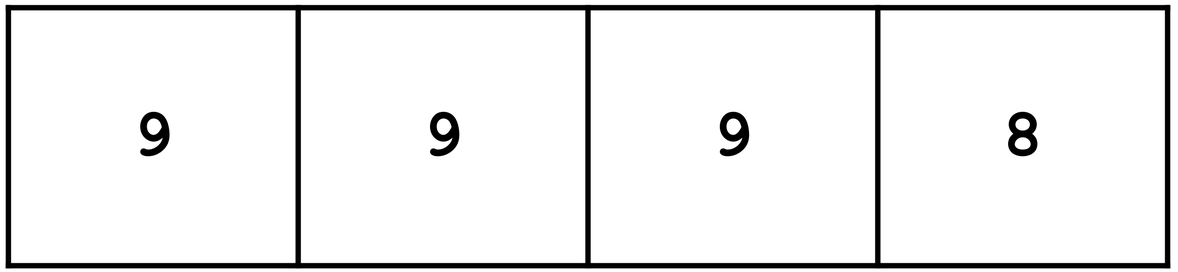
Negative Numbers

Better Idea: Two's Complement



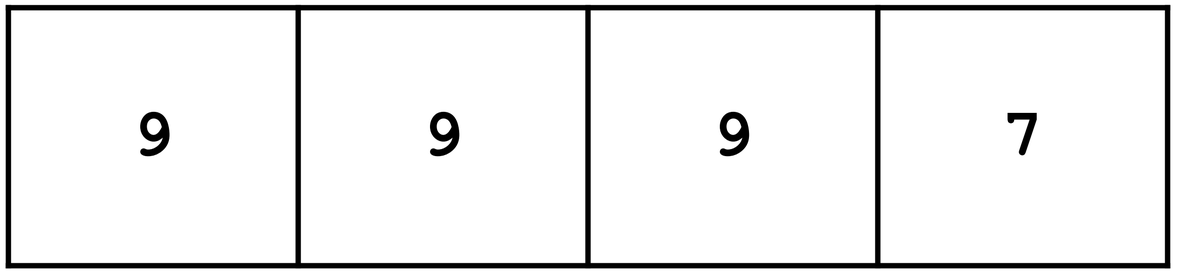
Negative Numbers

Better Idea: Two's Complement



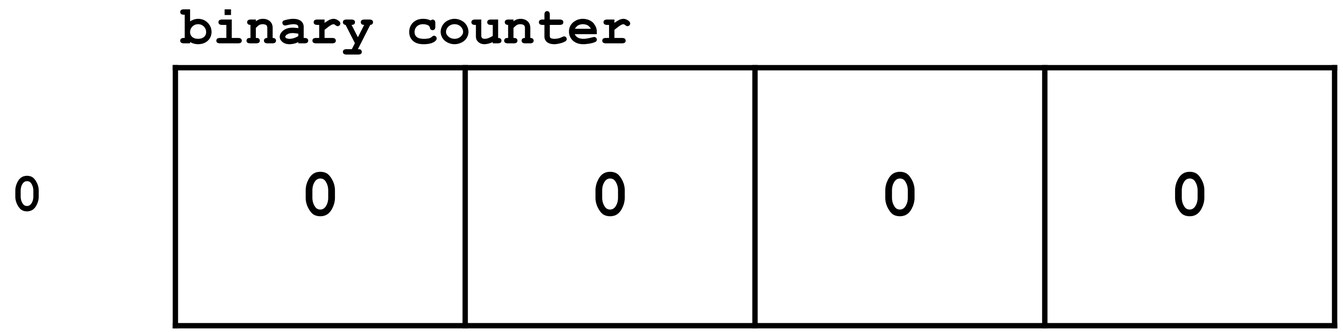
Negative Numbers

Better Idea: Two's Complement



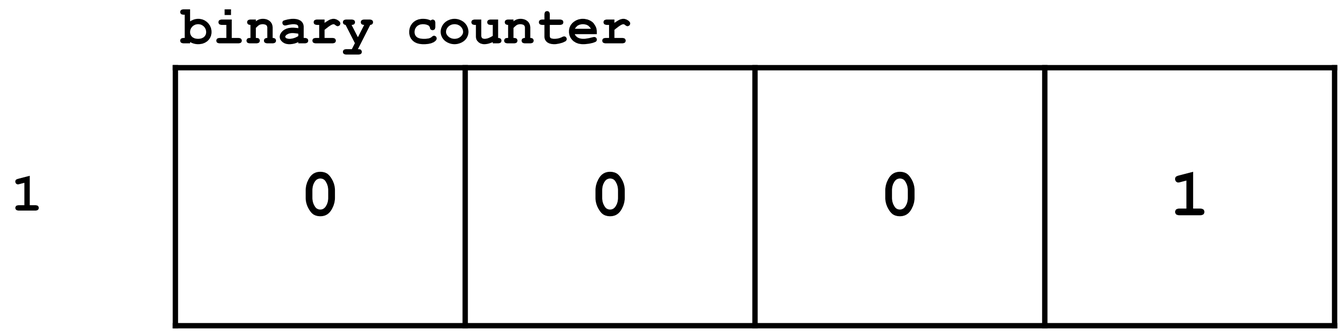
Negative Numbers

Better Idea: Two's Complement



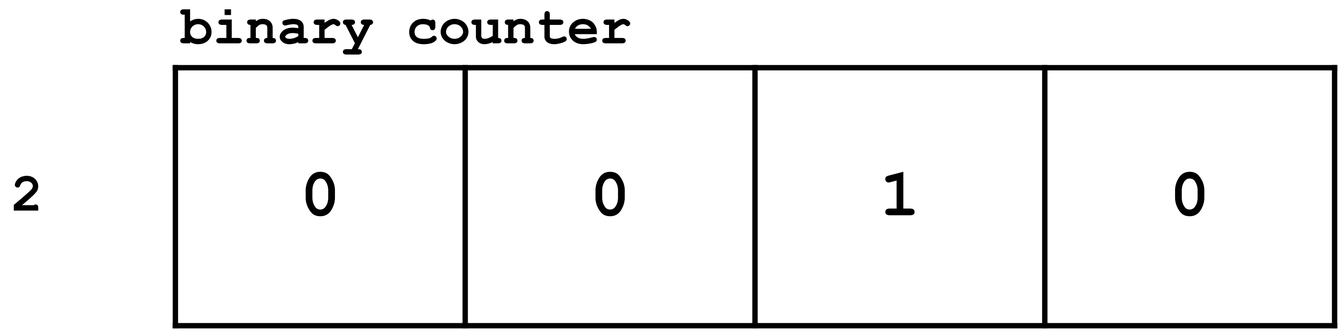
Negative Numbers

Better Idea: Two's Complement



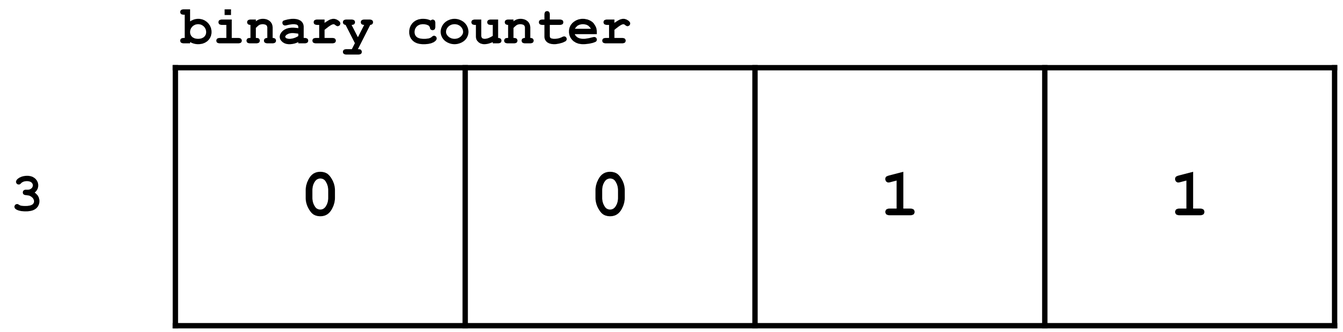
Negative Numbers

Better Idea: Two's Complement



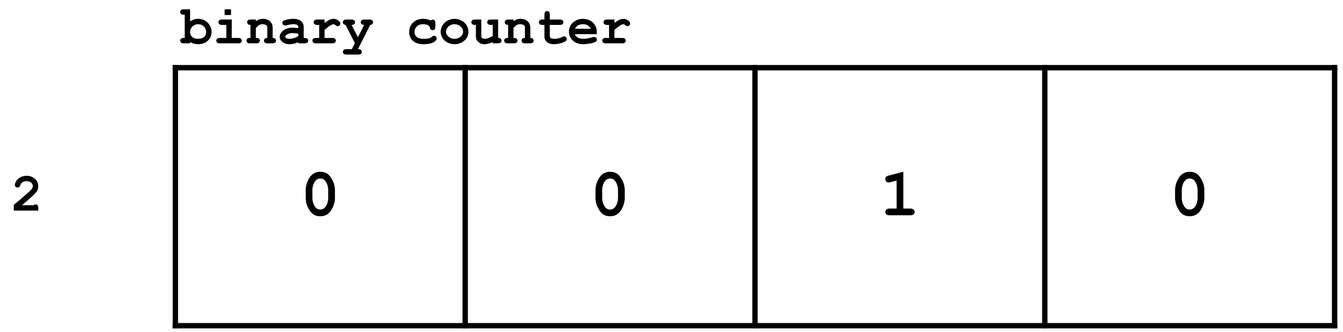
Negative Numbers

Better Idea: Two's Complement



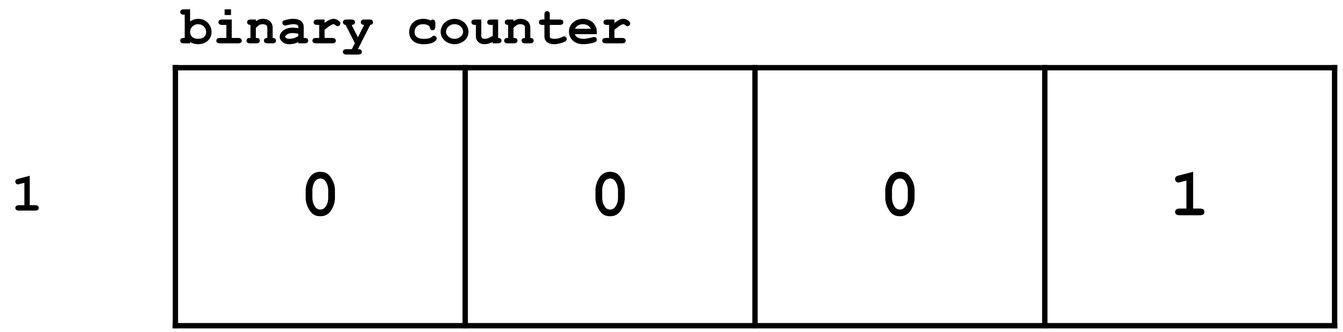
Negative Numbers

Better Idea: Two's Complement



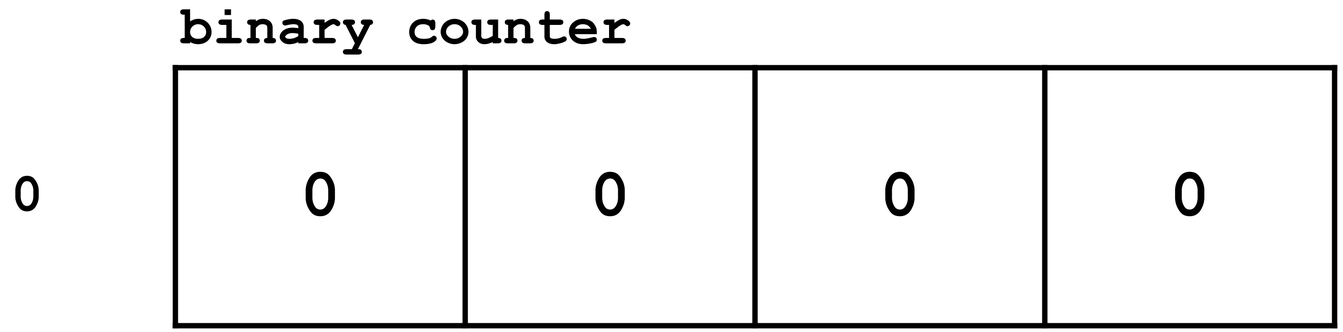
Negative Numbers

Better Idea: Two's Complement



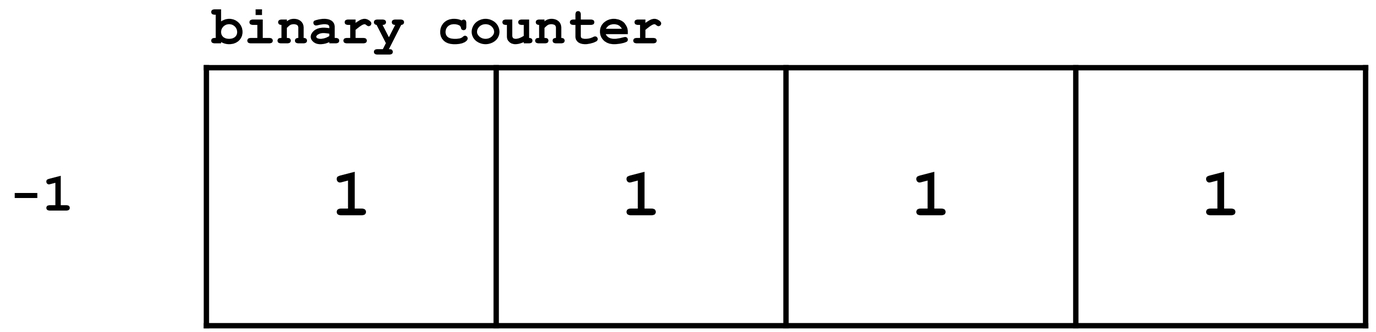
Negative Numbers

Better Idea: Two's Complement



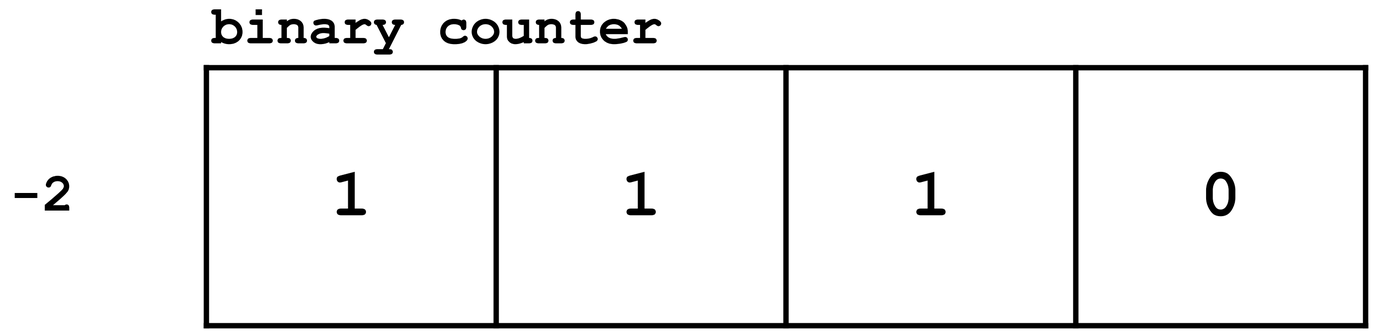
Negative Numbers

Better Idea: Two's Complement



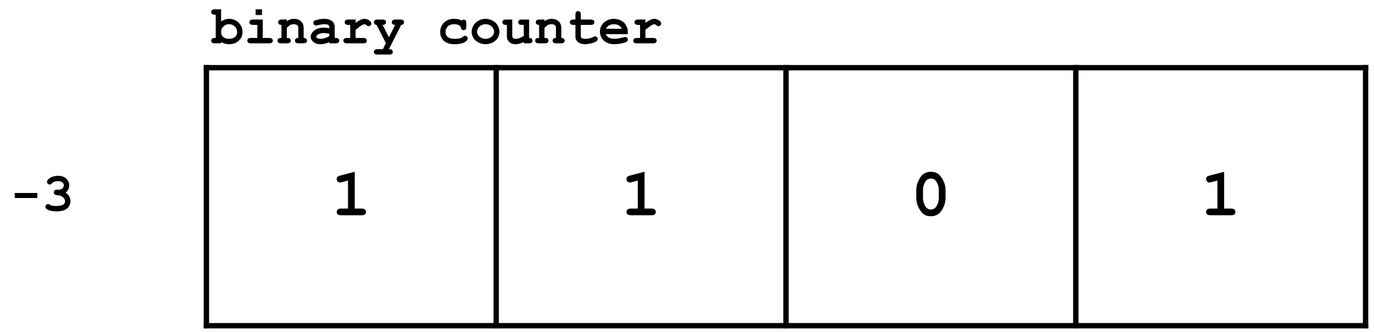
Negative Numbers

Better Idea: Two's Complement



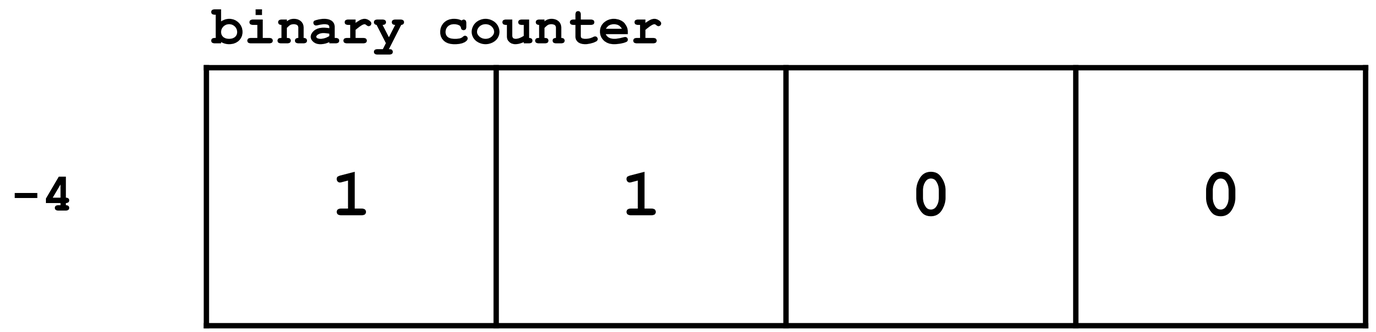
Negative Numbers

Better Idea: Two's Complement



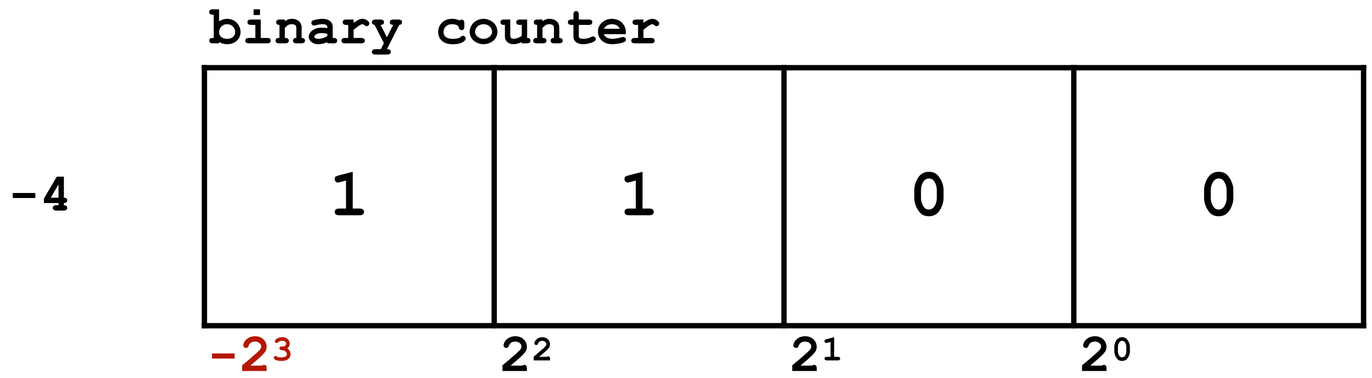
Negative Numbers

Better Idea: Two's Complement



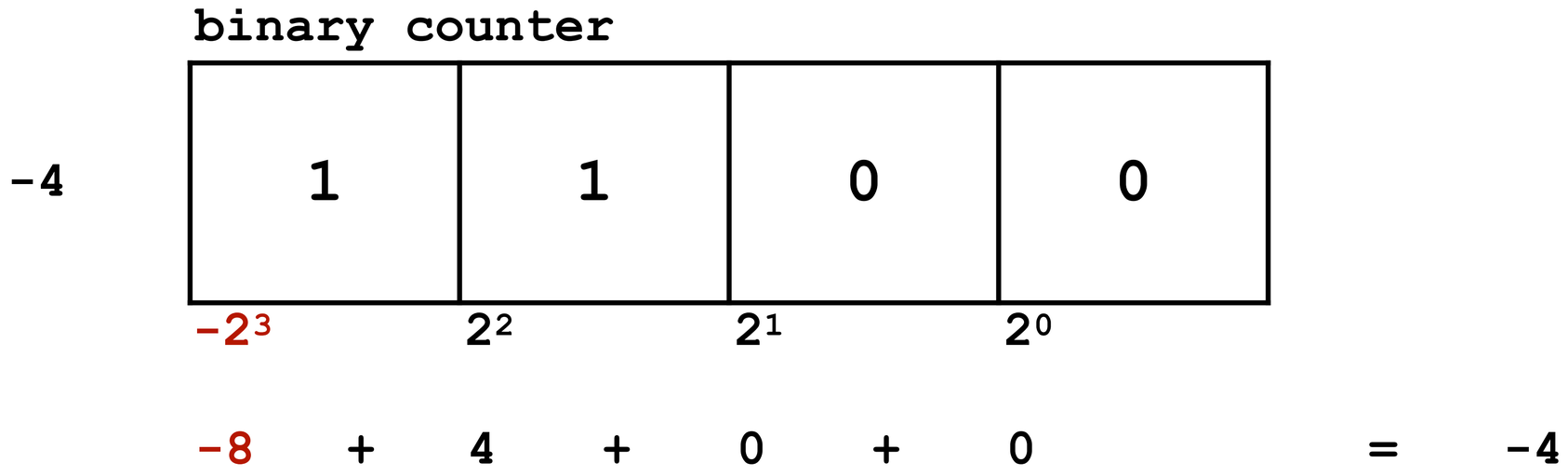
Negative Numbers

Two's complement: two ways



Negative Numbers

Two's complement: two ways



- The most significant bit is negative.

Negative Numbers

Two's complement: two ways

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

```
1110 1001
```

- Unsigned: 233
- What does this represent if it is a signed integer?
- -128 + 64 + 32 + 8 + 1 = -23

Negative Numbers

Two's complement: two ways

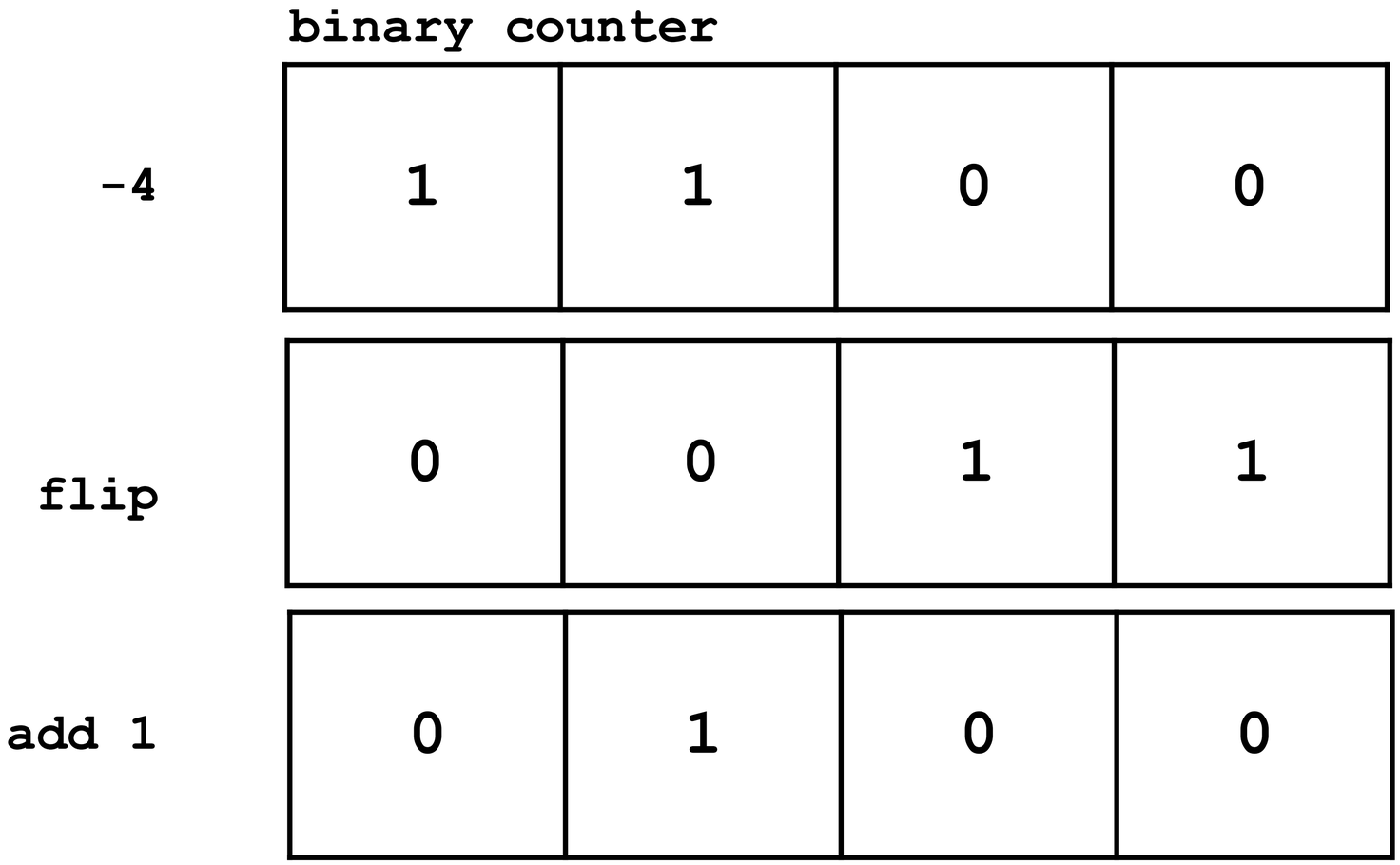
n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

```
1110 1001
```

- How do hardwares negate the sign?
  - Flip all the bits and add one
- Flip all the bits: 0001 0110
- add one: 0001 0111
- : 16 + 4 + 2 + 1 = 23

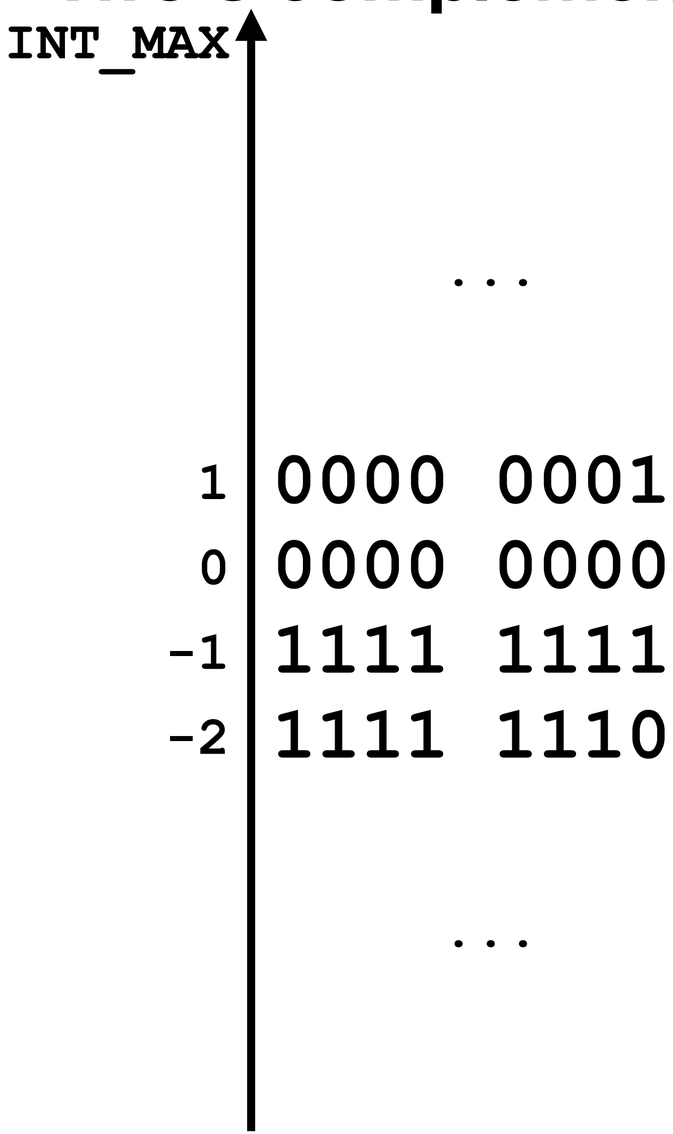
Negative Numbers

Two's complement: two ways



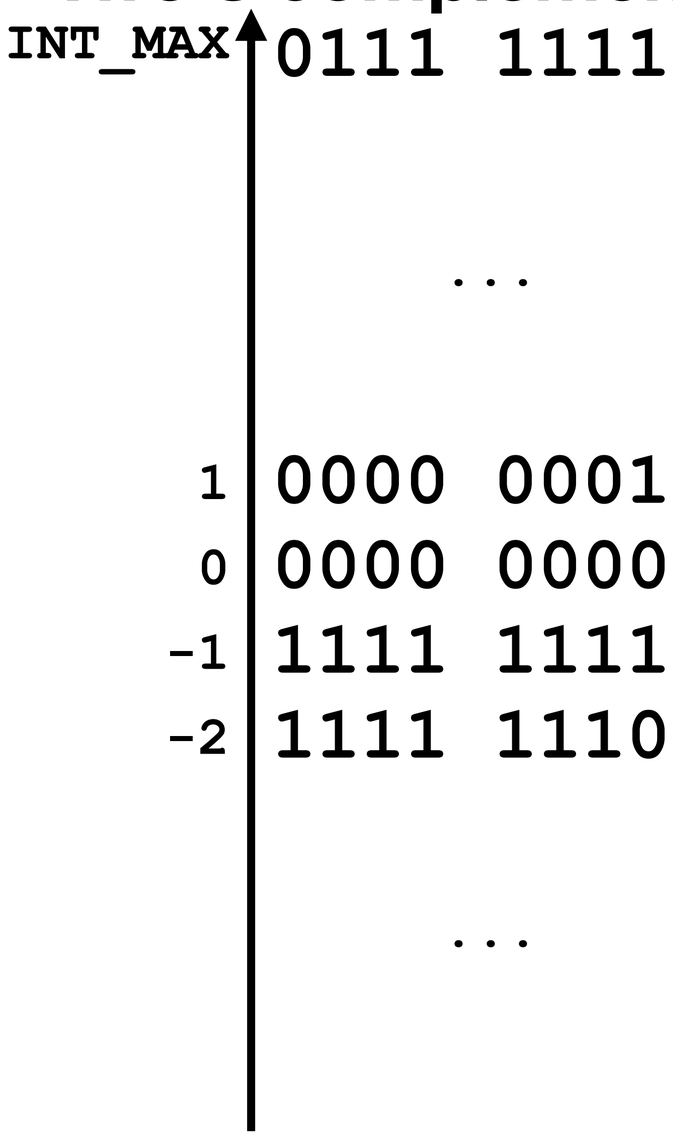
Negative Numbers

Two's complement: range



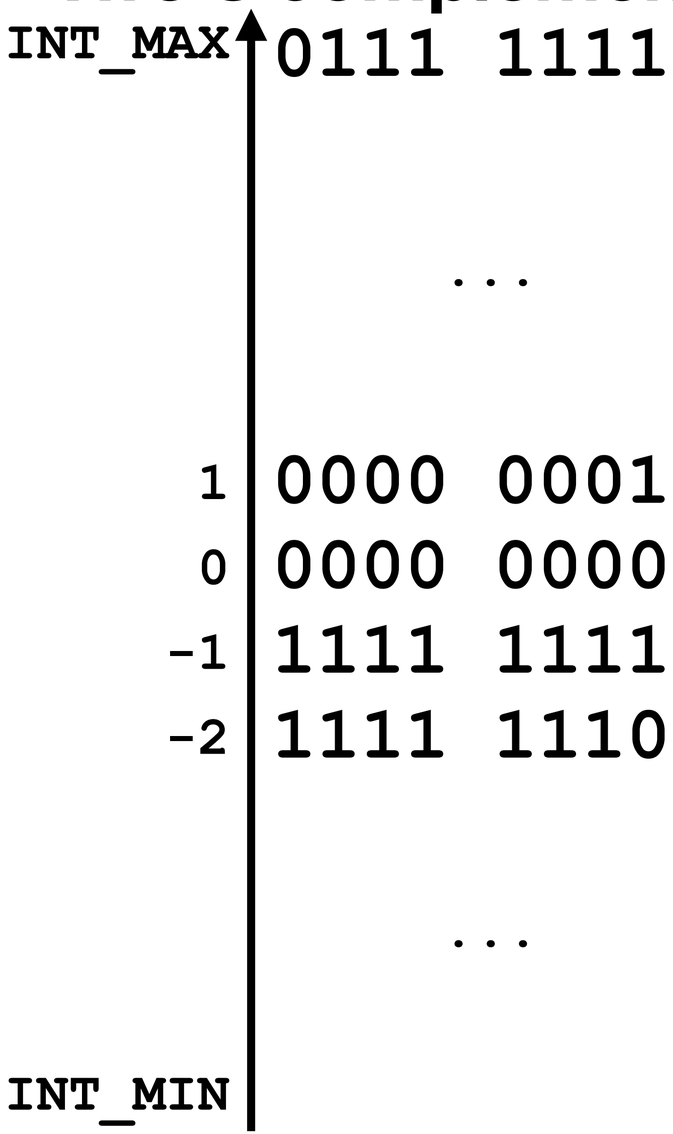
Negative Numbers

Two's complement: range



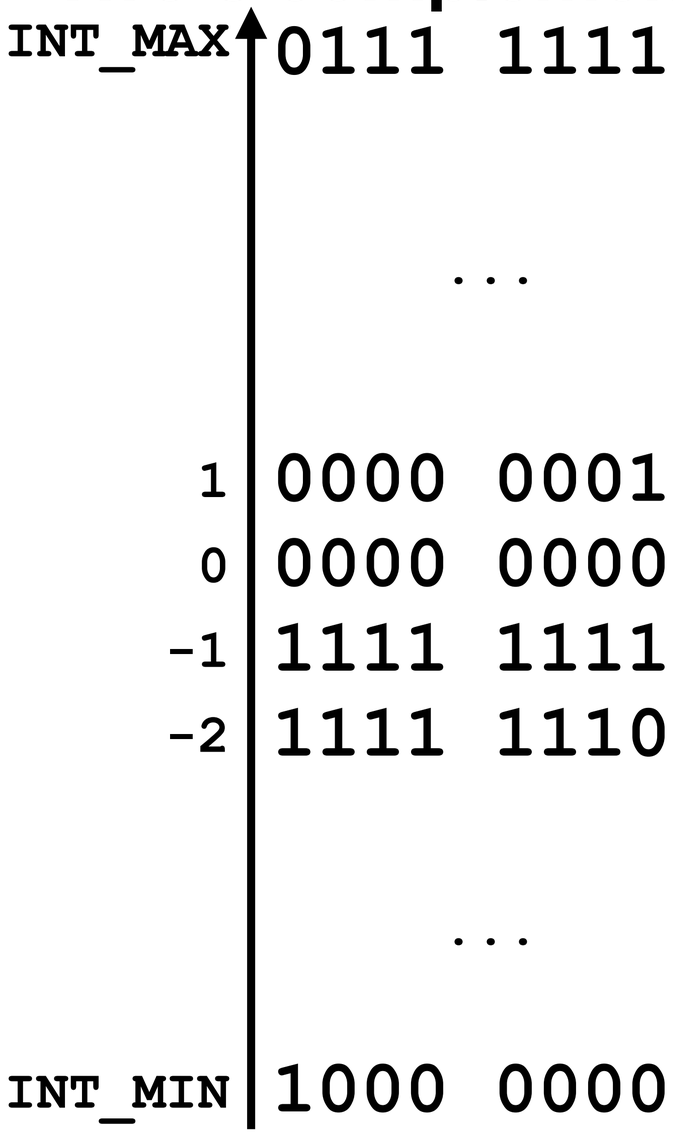
Negative Numbers

Two's complement: range



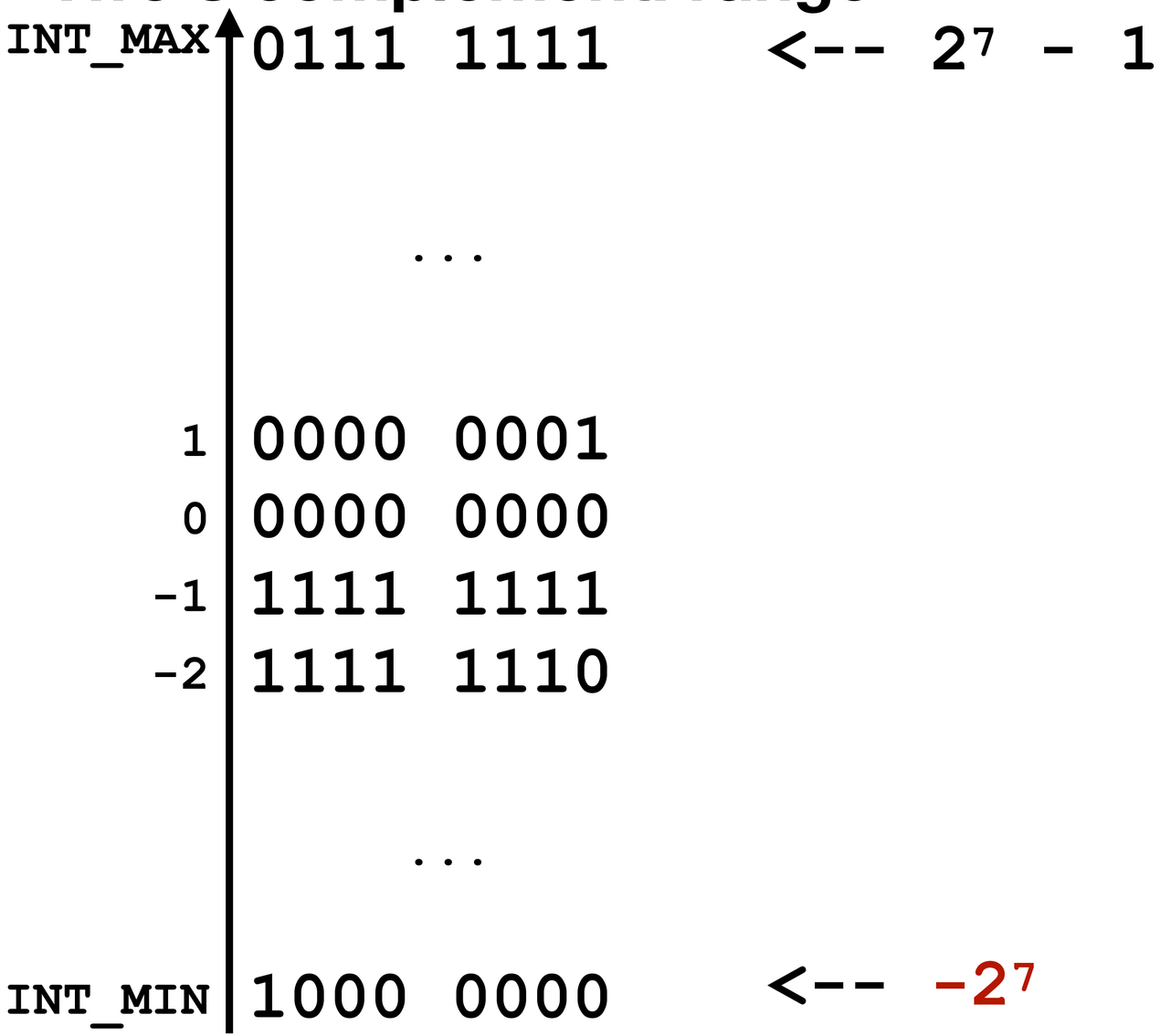
Negative Numbers

Two's complement: range



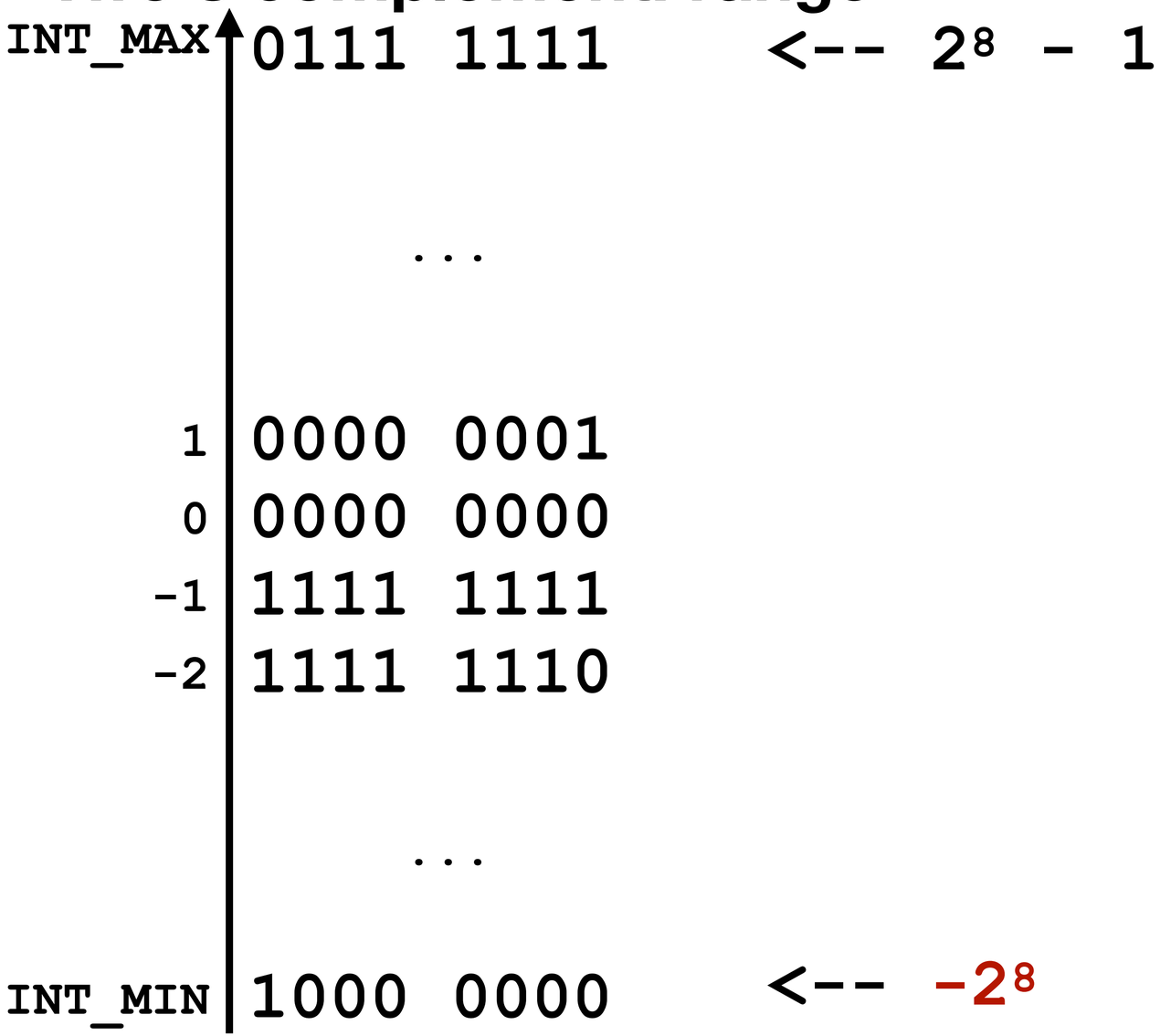
Negative Numbers

Two's complement: range

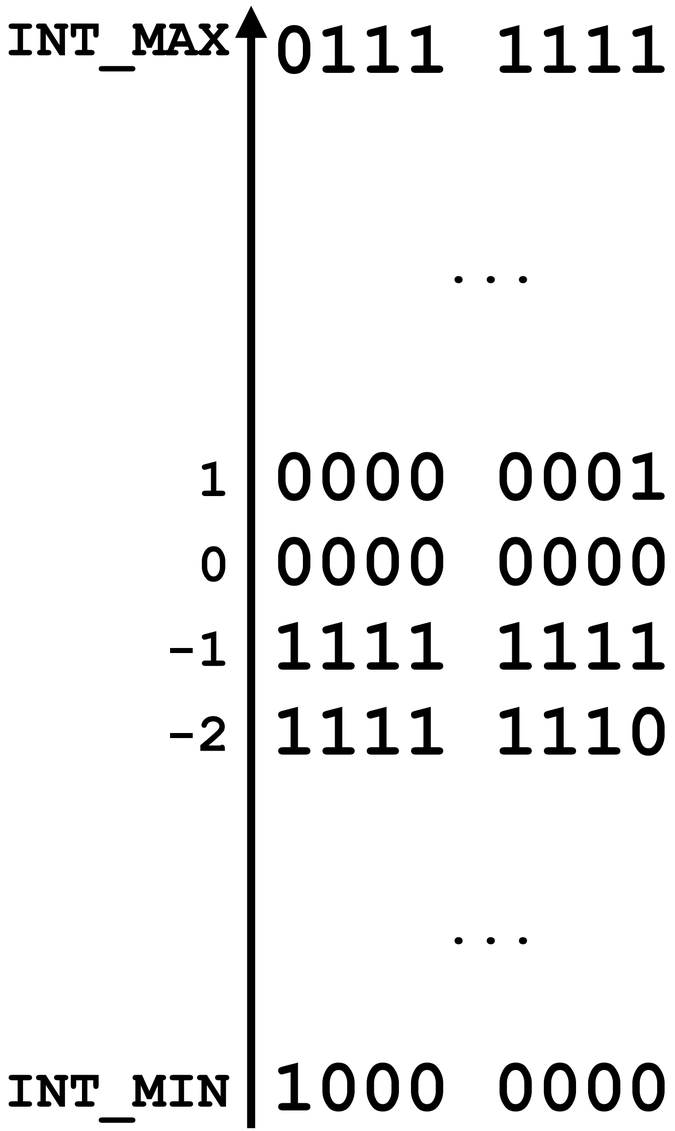


Negative Numbers

Two's complement: range

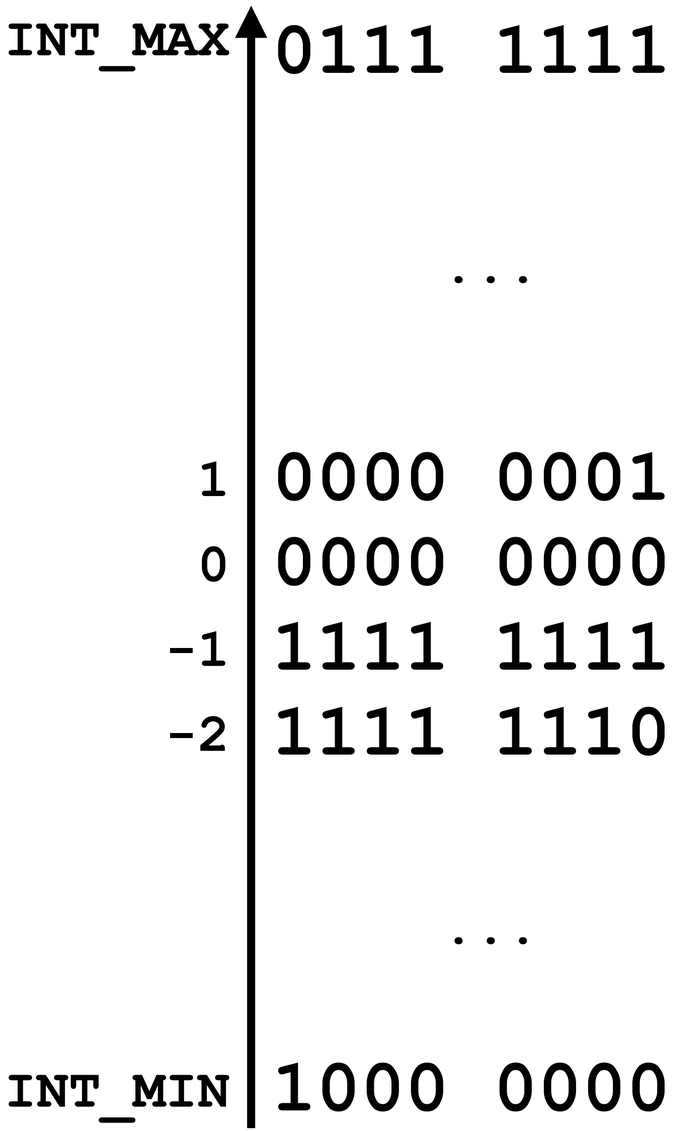


Signed Arithmetics



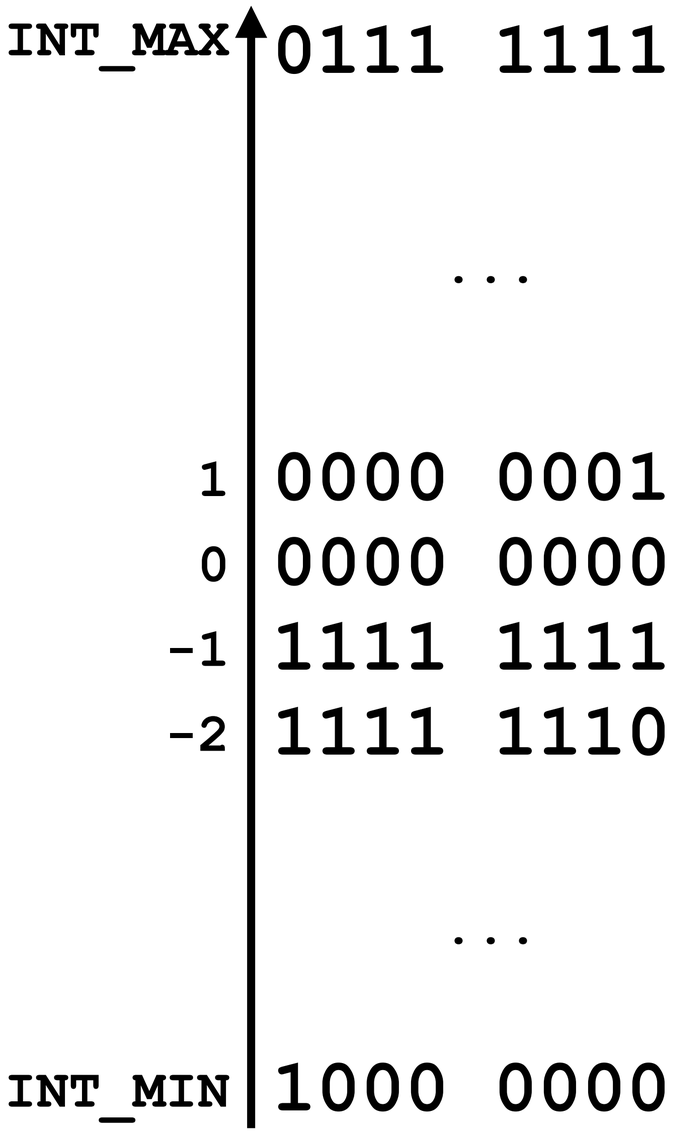

```
  1111 1110
+ 0000 0011
-----------
```

Signed Arithmetics



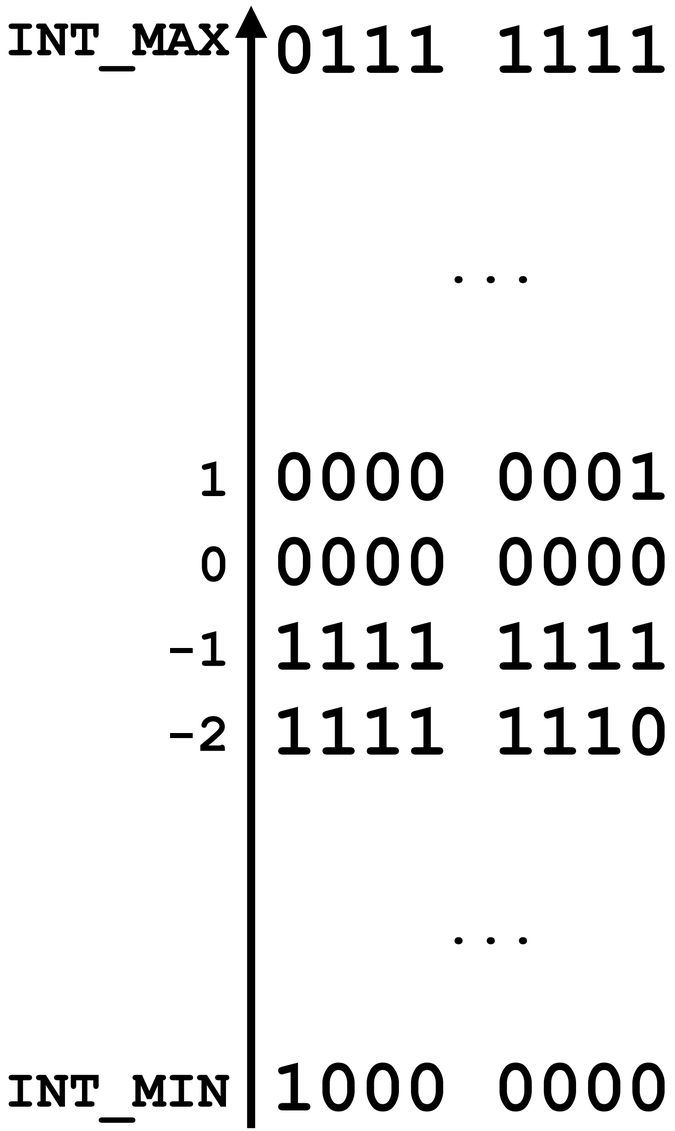

```
  1111 1110          -2
+ 0000 0011           3
-------------
```

Signed Arithmetics



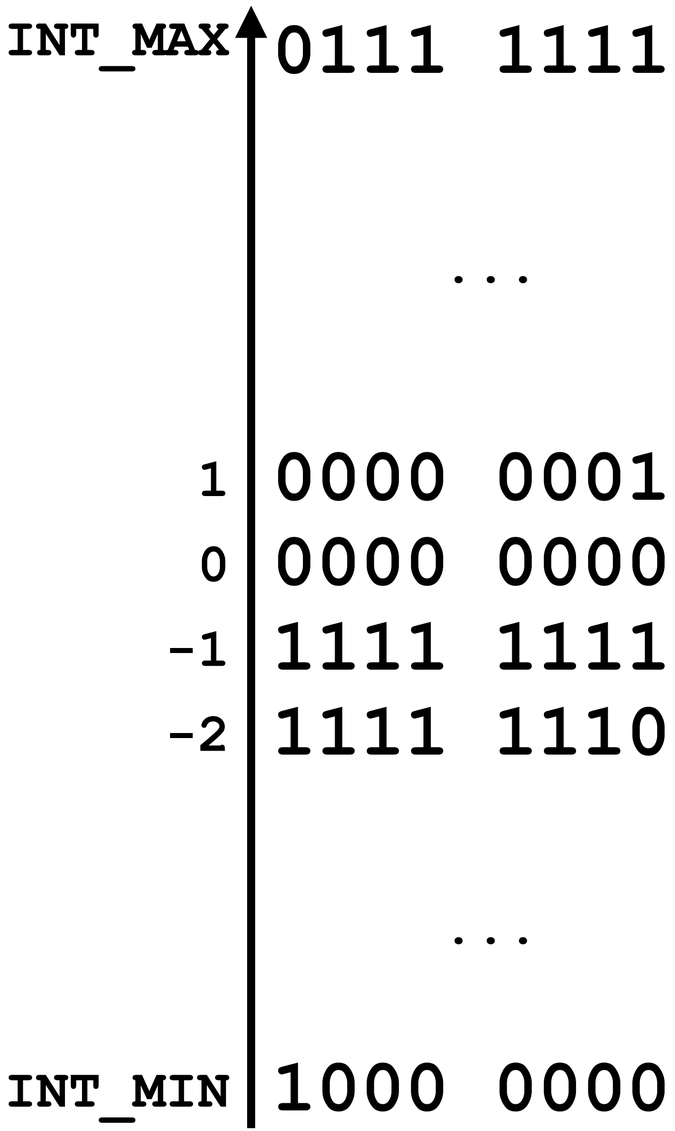

```
  1111 1110          -2
+ 0000 0011           3
-------------------------
             1
```

Signed Arithmetics




```
  1111 1110          -2
+ 0000 0011           3
-------------------------
           01
```

Signed Arithmetics



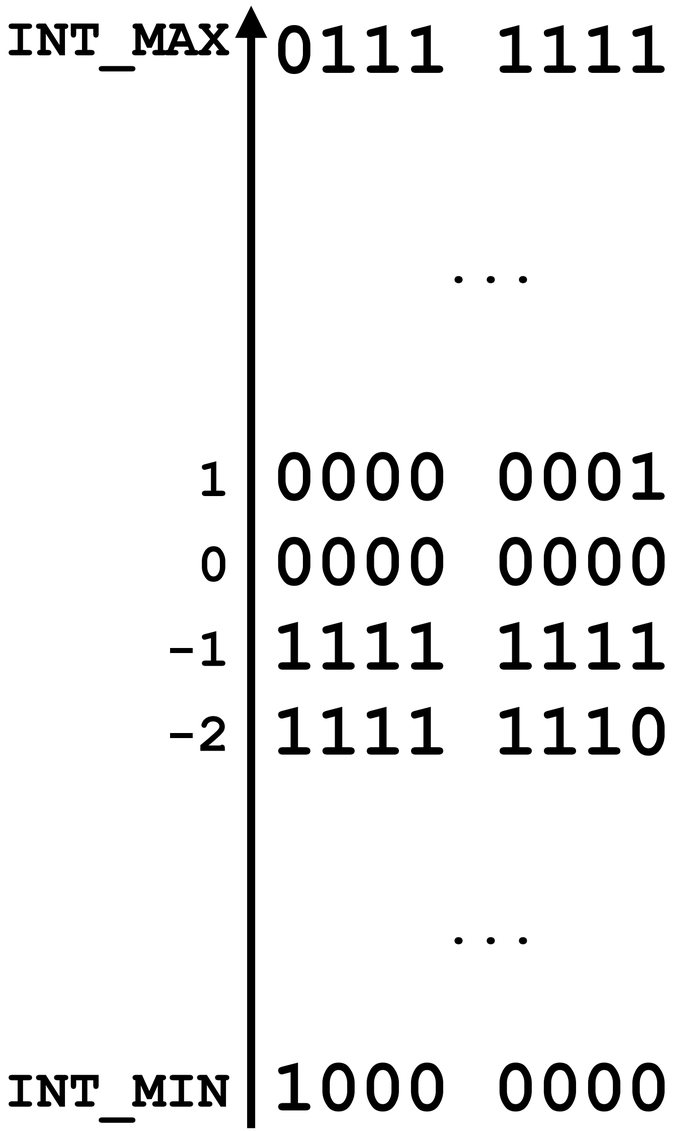

```
  1111 1110          -2
+ 0000 0011           3
-------------------------
           001
```

Signed Arithmetics

INT_MAX	0111 1111
	...
1	0000 0001
0	0000 0000
-1	1111 1111
-2	1111 1110
	...
INT_MIN	1000 0000

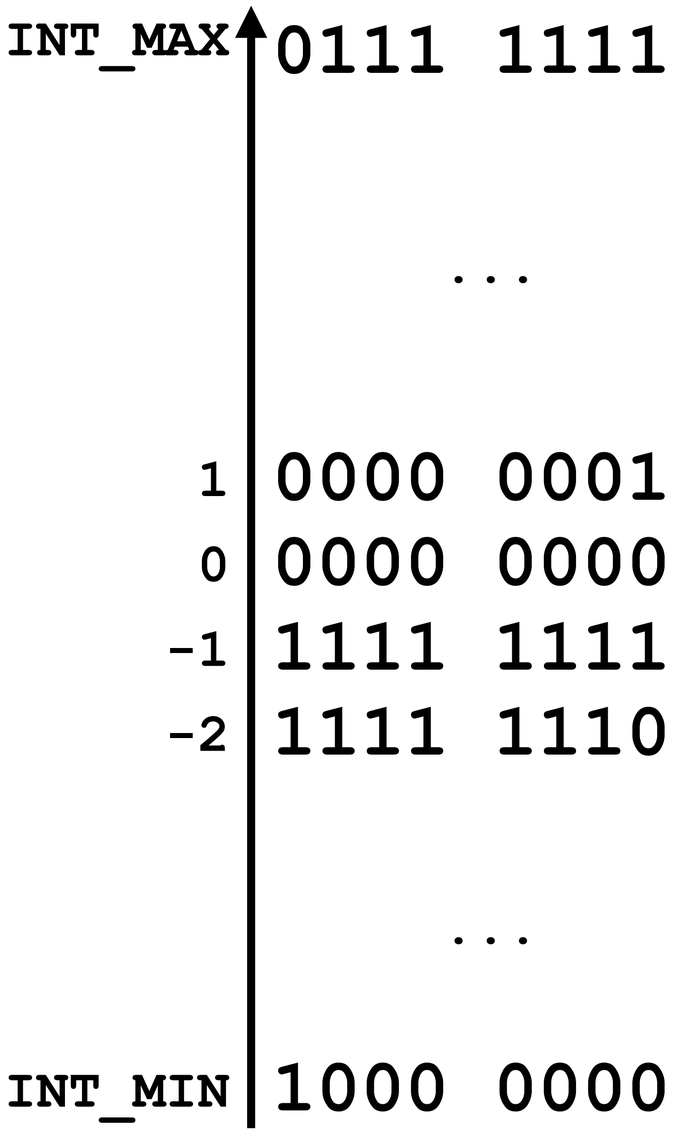
```
  1111 1110        -2
+ 0000 0011         3
      1  1 1
-----------
       0001
```

Signed Arithmetics



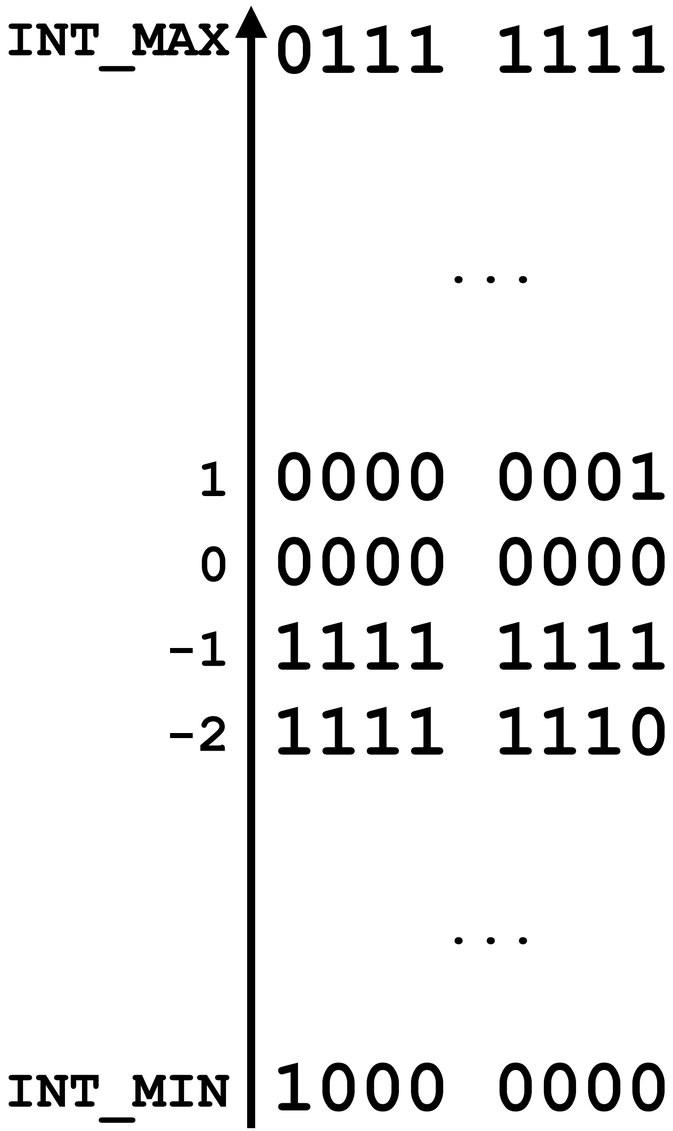

```
  1111 1110          -2
+ 0000 0011           3
  ¹¹  ¹¹
-------------------------
       0 0001
```

Signed Arithmetics



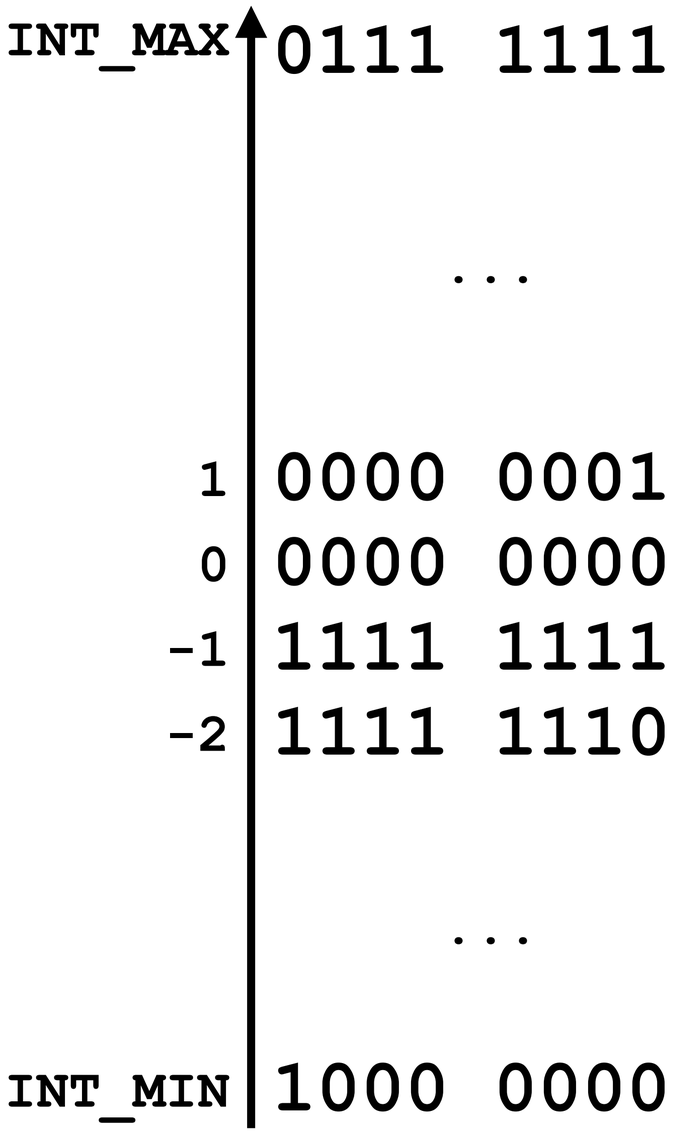

```
  1111 1110          -2
+ 0000 0011           3
  ^^^^ ^^^^
  0000 0001
```

Signed Arithmetics




```
  1111 1110          -2
+ 0000 0011           3
  1111 1111
-------------
  1 000 0001
```

Signed Arithmetics




```
  1111 1110          -2
+ 0000 0011           3
  ---------
  0000 0001
```

- The hardware does not need to know/ care whether the arithmetics is signed or unsigned
- **`INT_MAX + 1 == INT_MIN`**

Signed Arithmetics

Unsigned >>

```
  1111 1111 1111 1111 1111 1111 1111  1100          -4
>>                                       1
```

Signed Arithmetics

Unsigned >>

```
  1111 1111 1111 1111 1111 1111 1111  1100          -4
>>                                       1
-------------------------------------------------------
  0111 1111 1111 1111 1111 1111 1111  1110   2147483646
```

- If we just add 0 in right shift, shifting a negative number results in a positive number.
- Ideally, we would like to keep the sign / shifting is dividing by 2.
- We repeat the most significant bit when doing a *signed* right shift.

Signed Arithmetics

Signed >>

```
    1111 1111 1111 1111 1111 1111 1111 1100          -4
>>                                        1
-----------------------------------------------------
    1111 1111 1111 1111 1111 1111 1111 1110
```

Signed Arithmetics

Signed >>

```
    1111 1111 1111 1111 1111 1111 1111 1100          -4
>>                                      1
-------------------------------------------------------
    1111 1111 1111 1111 1111 1111 1111 1110          -2

    0000 0000 0000 0000 0000 0000 0000 0100           4
>>                                      1
-------------------------------------------------------
```

Signed Arithmetics

Signed >>

```
    1111 1111 1111 1111 1111 1111 1111 1100          -4
>>                                      1
-------------------------------------------------------
    1111 1111 1111 1111 1111 1111 1111 1110          -2

    0000 0000 0000 0000 0000 0000 0000 0100           4
>>                                      1
-------------------------------------------------------
    0000 0000 0000 0000 0000 0000 0000 0010           2
```

- If the first bit is 0, right shifting fills with 0.
- If the first bit is 1, right shifting fills with 1.

Signed Arithmetics

Signed >>

- In C, *signed* right shift and *unsigned* right shift looks exactly the same
- C will use *signed* shift if the variable being shifted is a *signed* integer

```
int x = ~0;
x = x >> 4;
printf("0x%x\n", x);
```

```
0xffffffff
```

```
unsigned int x = ~0;
x = x >> 4;
printf("0x%x\n", x);
```

```
0x0fffffff
```

Signed Arithmetics

- Signed extension works the same way:

```
short x = ~0;
int y = x;
printf("0x%x\n", y);
```

```
0xffffffff
```

```
unsigned short x = ~0;
unsigned int y = x;
printf("0x%x\n", y);
```

```
0x0000ffff
```

Machine Arithmetics

Summary

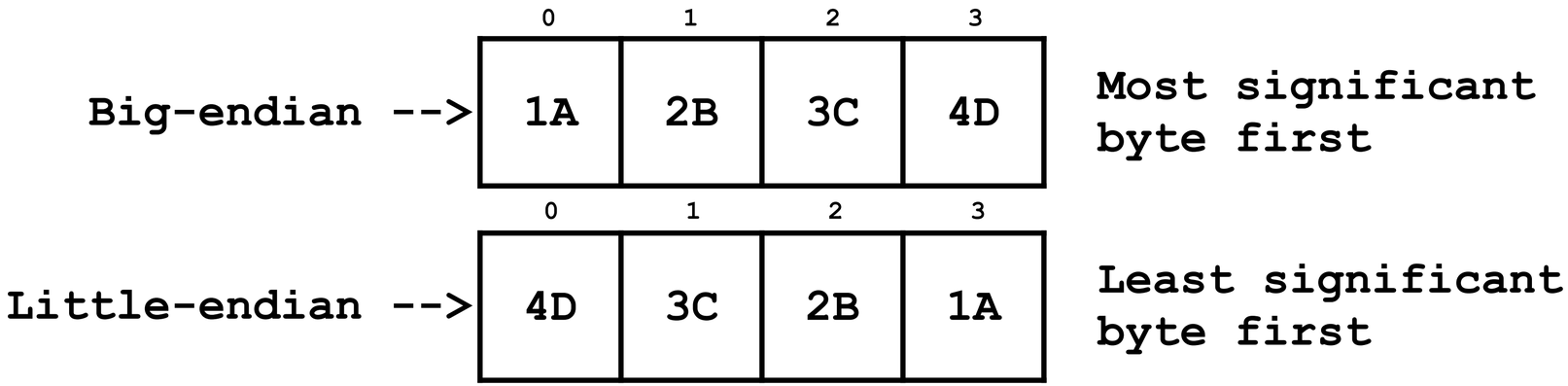
- For a 32-bit integer:
  - Range: unsigned $[0, 2^{32} - 1]$; signed $[-2^{31}, 2^{31} - 1]$.
  - Meaning of the highest bit: unsigned 2^{31}, signed -2^{31}.
- Negate a number: flips all bits and add 1 (-n == ~n + 1).
- Most arithmetic operators work regardless of the signedness of the integer except right shift:
  - Signed right shift (arithmetic right shift): fills with the highest bit.
  - Unsigned right shift (logical right shift): fills with 0.
  - Left shift is the same.

Enum

Demo

Endian

- We think of an integer as one atomic value:
  - `int x = 0x1A2B3C4D;`
- But if an integer has 4 bytes and each byte is addressable, which of the 4 bytes is stored first?



Endian

- Is my machine little-endian or big-endian?
- Let's find out!

Endian

- Our machine is little-endian?????
- We usually write numbers in big-endian: 345 is three hundred and forty-five
- But there are some advantages for little-endian:
  - comparing two numbers of different length (long and int e.g.)
    - `4E3C2B1A`
    - `4E3C2B1A00000000`
  - addition, subtraction circuits work from low to high
  - etc.

Endian

Does it matter?

- Mostly we don't care. Unless you do memory trickery, variables work as you would expect
- However, when we serialize data into byte sequences, you need to pay extra attention:
  - Writing a number to a file
  - Sending a number over a network
- You and the reader must agree on byte order
  - For this purpose, *network byte order* is defined for TCP/IP

Octal Number

Octal Number

- Base-8 numbers, 0-7
- Corresponds to 3 bits
- In C, prefix by 0, e.g. `0123`, is 83
- Not very common

Bits

- A bit answers a yes/no question
  - Bits have no inherent meanings; to read encoding, need to know the intended interpretation
  - Any device that can be one of two states can encode a bit.
- Binary and hexadecimal numbers
  - $xyz_m = x \cdot m^2 + y \cdot m^1 + z \cdot m^0$
  - One hex digit corresponds two 4 binary digits (bits) -- easy conversion between the two
- Positive and negative numbers
- Arithmetics
  - `+`, `-`, `<<`, `&`, `|`, `^`, `~`, two `>>`, negate
  - Bit-packing
- Endianness