

CS 208

Wed, 18 Jan 2023

C
int k = 15;

In memory

0000 0000 0000 0000 0000 0000 0000 1111
byte byte

Printing (paper, screen, whiteboard*)

Base 10: 15

Base 16: F, 7

Base 8: 17

Logical operations

AND

and 

OR

or 

NOT

not 

XOR

T	XOR	T	T	T
T	XOR	F	T	F
F	XOR	T	T	F
F	XOR	F	T	T

Bitwise operations

0101 | 0110

bitwise
OR

int k = -4;
int j = 3;

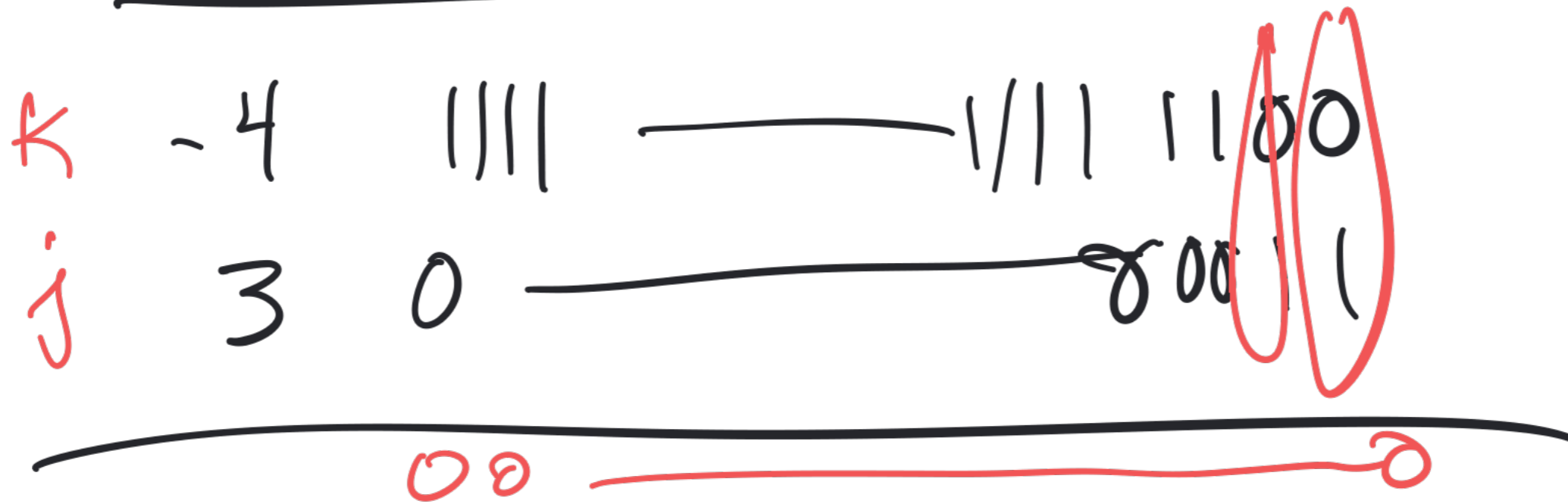
0101
0110

k 1111...1100
j 0000...0011

0111

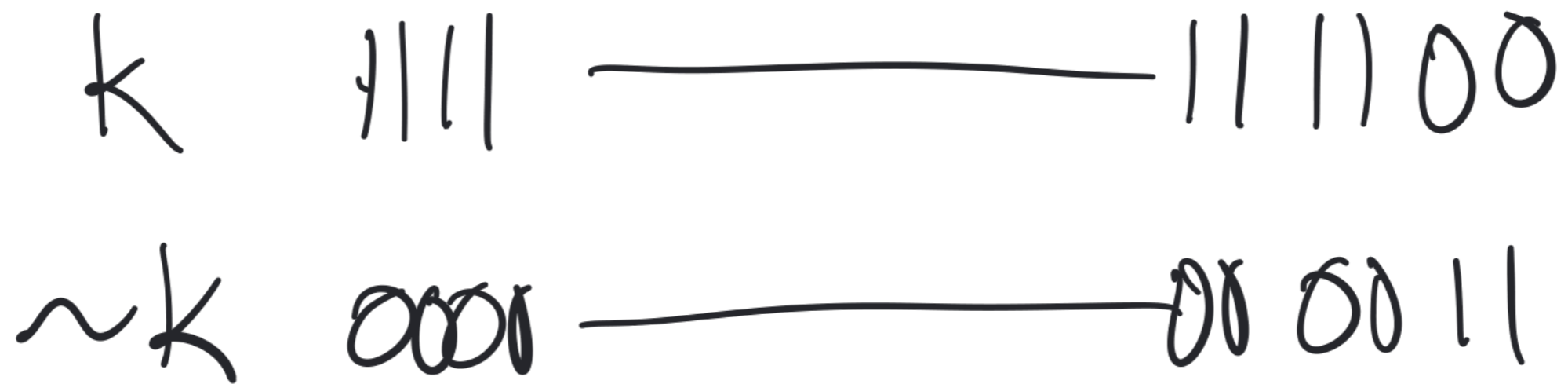
→ 1111...1111
-1

Bitwise AND



$$k \& j \rightarrow 0$$

Bitwise NOT



// Boolean context // in C

if ()

while ()

for (; ;)

Integers are "true" if non-zero
"false" if zero

Why do we want bitwise operators?

A	0x41	01 <u>000001</u>
a	0x61	01 <u>100001</u>

B		01 <u>000010</u>
b		01 <u>100010</u>

Bitwise op to do to lower to upper

Mult. * (slow)

Division / (even slower)

Mod % (")

Addition + (fast)

Sub. - (fast)

(8 ~) → (super fast)

Bitwise ops
in construction
of network
packets

Character encoding

Codepoint: an integer
that represents a character

Unicode:

international agreement
which codepoints go w/ which
characters

U+0041

↔ A

U+03B1

↔ α

(Greek alpha)

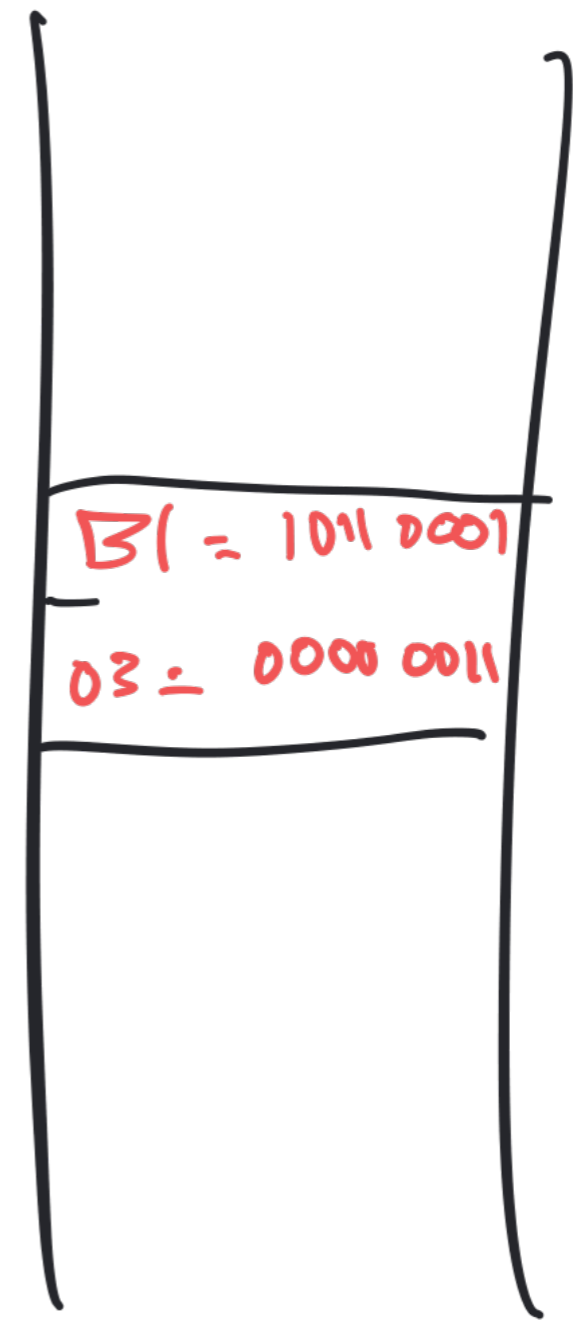
Character coding

A scheme for storing codepoints as sequences of bytes

UTF-16 LE ← char encoding
Memory

always 2 bytes

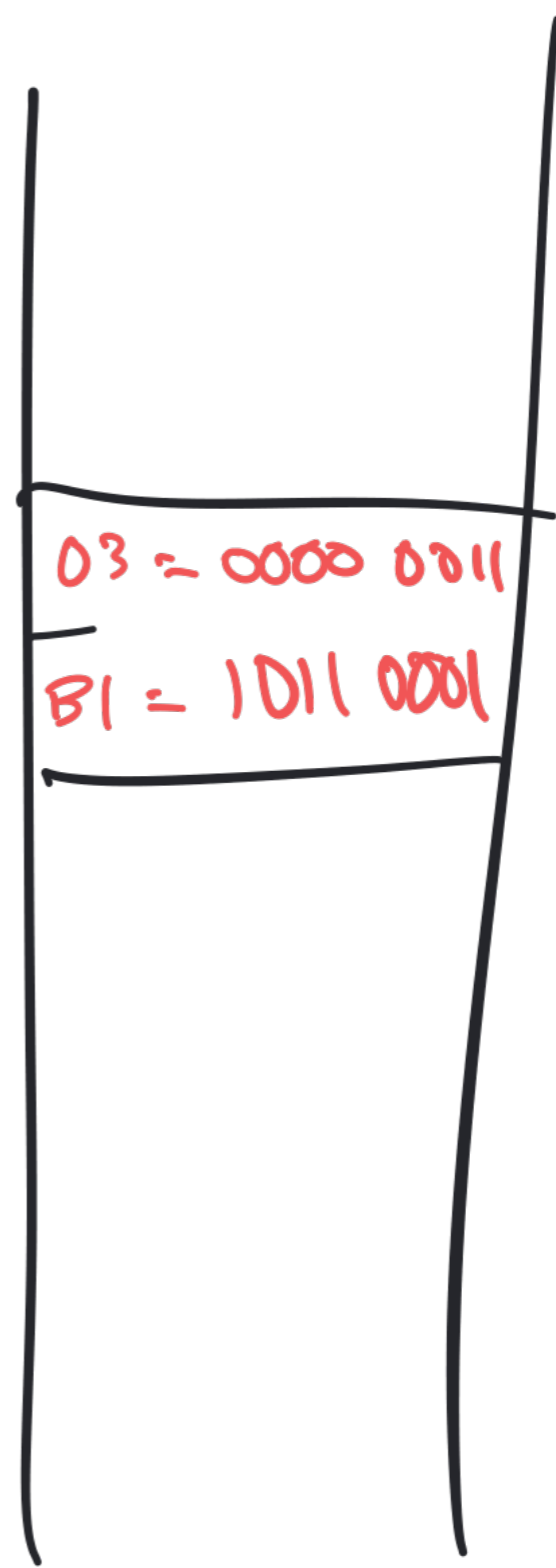
$$\alpha$$
$$U + 03B1$$



UTF-16 BE ← char encoding
Memory

always 2 bytes

α
U+03B1



UTF-16 disadvantages

- If you're programming or writing in Latin alphabet these take up two bytes per char, — a lot.
- There aren't enough codepoints that fit in two bytes. (2^{16} isn't enough)

UTF-8

Ken Thompson

- UNIX (1969)
- Turing Award (1983)
- OS: Plan 9
- "Reflections on Trusting Trust"

~~- UTF-8~~

address UTF-16
disadvantages

UTF-8

\propto

U+03B1

2nd row
of UTF-8 chart

11 1011 0001

10 x's

110 0 - - - - 10 - - - -