INTRODUCTION TO COMPUTER SCIENCE · WEEK 3 · STATION I

Everything is bits.

Numbers, negative numbers, overflow, characters, text, files, images, code — one notation, and what the machine makes of it.

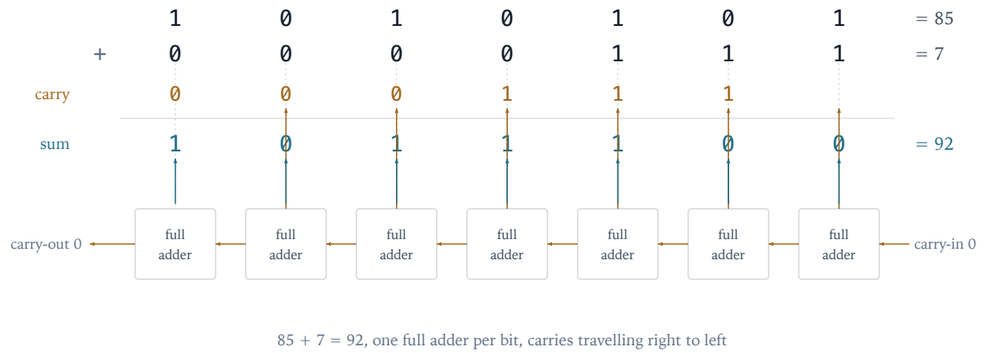
BINARY

85 is 1010101. Nothing else changed.

Decimal: 85 = 8·10 + 5·1. Binary: 1010101 = 1·64 + 0·32 + 1·16 + 0·8 + 1·4 + 0·2 + 1·1. Same number, a different base. The value is the meaning; the digits are the notation.

Adding works the same way in any base — carry when a column overflows its base. 1010101 + 111 = 1011100, which is 85 + 7 = 92.

Hexadecimal groups four bits into one digit: 1010101 is 0x55. Octal groups three. Both are just shorter ways to write the same bits; the machine only ever sees the bits.

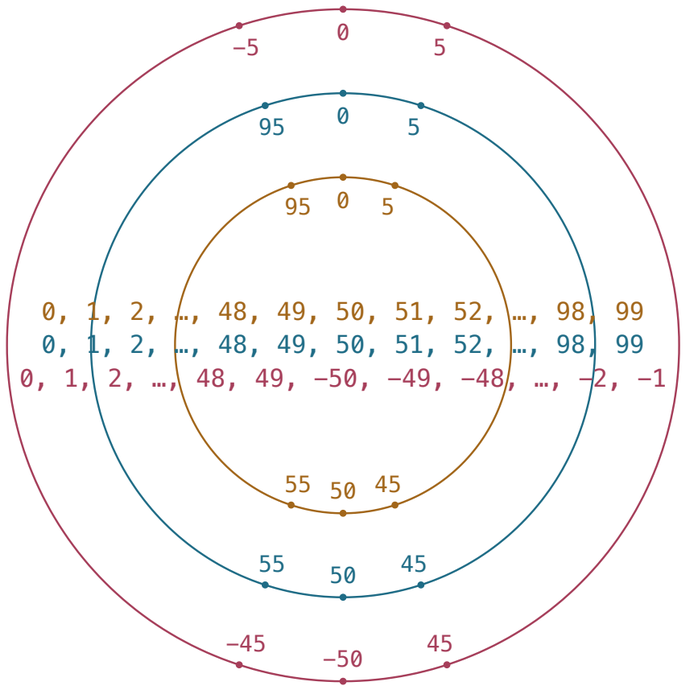


85 + 7 = 92, one full adder per bit, carries travelling right to left

85 + 7 = 92, one full adder per bit

NEGATIVE NUMBERS

Two digits, read three ways.



With two decimal digits you can encode a hundred things. Read them as 0 to 99, or read 50 to 99 as -50 to -1. Same digits. Subtraction becomes addition: $7 - 8$ is $7 + 92 = 99$, which is -1.

The machine does exactly this in base two: two's complement. With 64 bits, 2^{64} encodings, read as 0 to $2^{64}-1$ or as -2^{63} to $2^{63}-1$. The bits do not say which.

1010101 with seven bits is 85 unsigned — and -43 signed. Same seven bits.

OVERFLOW

Finite means the numbers wrap around.

With 64 bits, $2^{64} - 1$ plus 1 is 0. Nothing crashes; the carry out of the top bit is simply lost. That is an overflow, and the machine does not tell you.

Signed, the same wrap makes $2^{63} - 1$ plus 1 equal to -2^{63}: the largest positive number plus one is the most negative one.

Every integer in every program you will ever run lives in a finite box. Most of the time nobody notices. The times somebody does are famous.

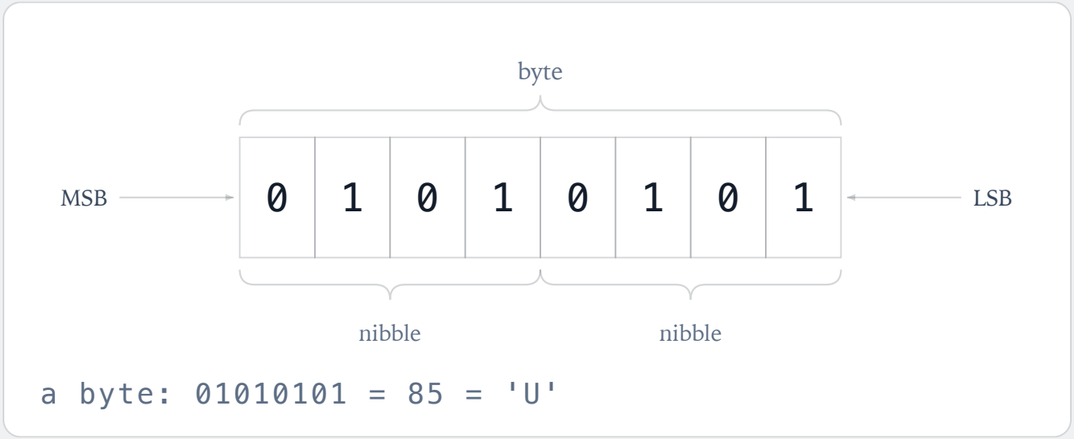
```
$ cat examples/overflows.c
// … prints UINT64_MAX + 1, INT64_MAX + 1, and
$ ./selfie -c examples/overflows.c -m 1
…
```

CHARACTERS

1010101 is also the letter U. People in the 1960s sat down and agreed.

ASCII: seven bits, 128 characters, a table agreed in 1963. 85 is U. 48 is the digit 0, 65 is A, 97 is a, 32 is a space. The digit '0' and the number 0 are different things with different codes.

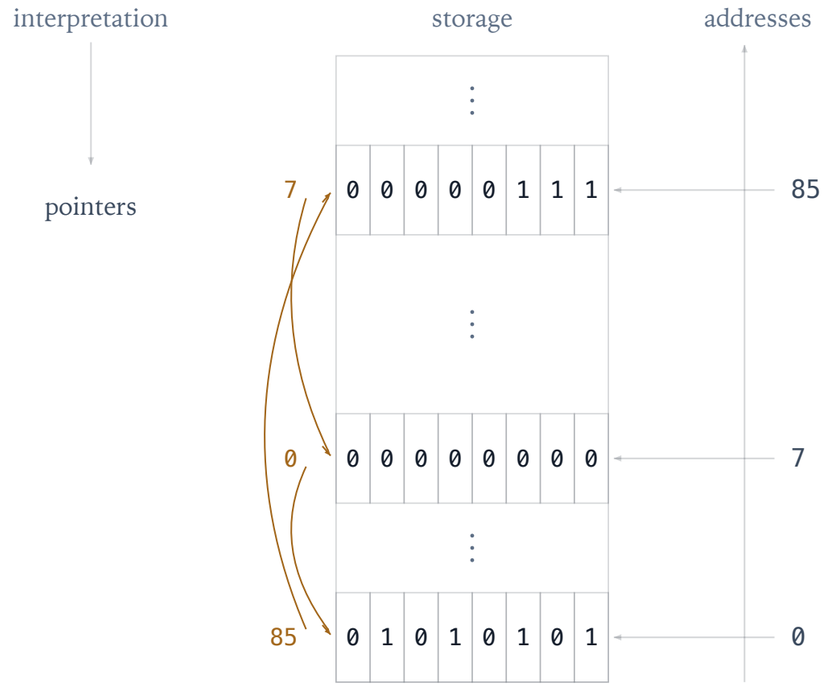
Unicode extends the table to every script, and UTF-8 encodes it in one to four bytes so that ASCII is unchanged. A byte is eight bits, a nibble four, and the figure names its ends.



MEMORY

Memory is bytes with addresses. An address is a number. So a byte can hold an address.

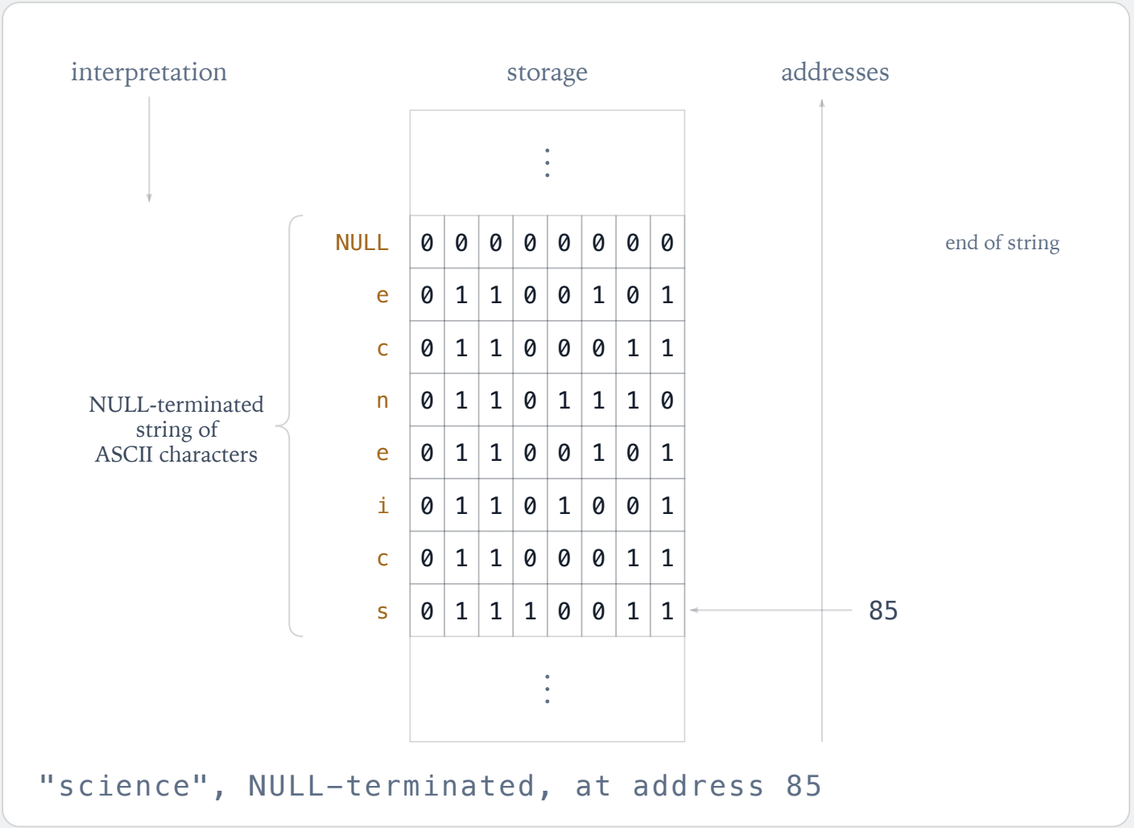
storage and addresses



a value read as an address: a pointer

TEXT AND FILES

A string is bytes in a row, ended by a zero. A file is a string with a name.



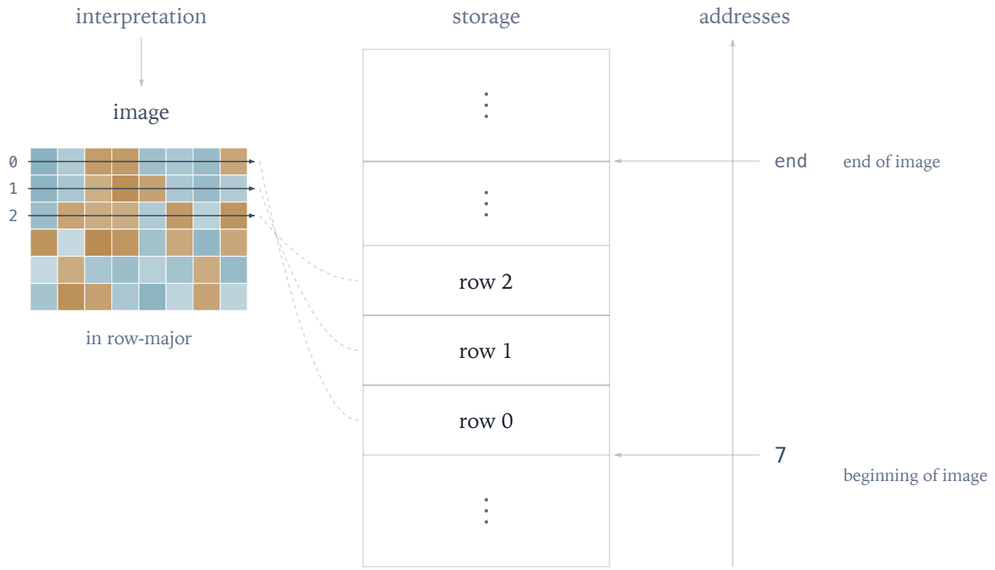
Contiguous: the letters of a word one after the other. The zero byte says where it ends, which is how a machine that only sees bytes knows a word is over.

Non-contiguous: a text is paragraphs, each contiguous, reached through pointers. A directory is names and pointers to files. A file system is a tree of directories, and a pathname is the route from the root.

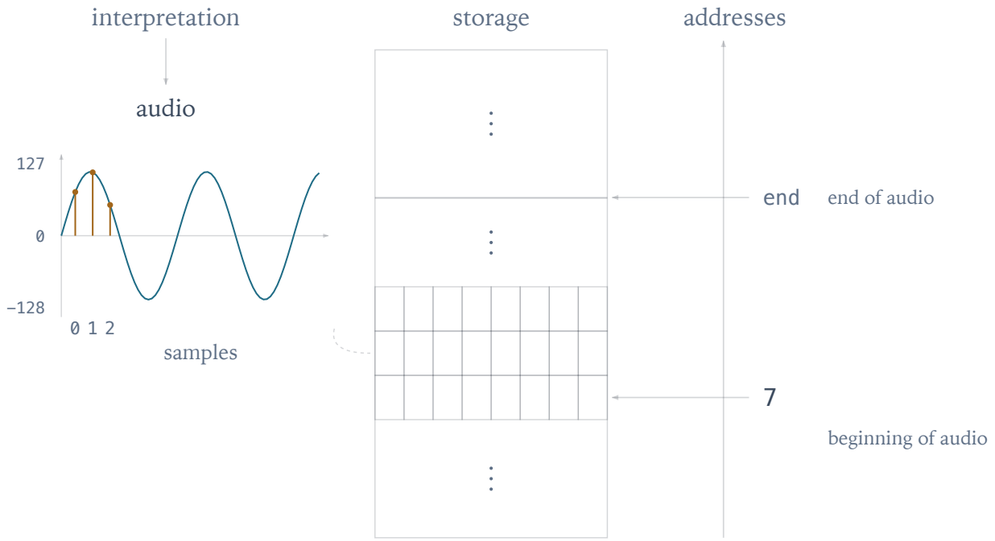
All of it is bytes at addresses. The structure is in how they are read.

IMAGES, VIDEO, AUDIO

A picture is numbers in rows. A film is pictures in a row. A sound is numbers in a row.



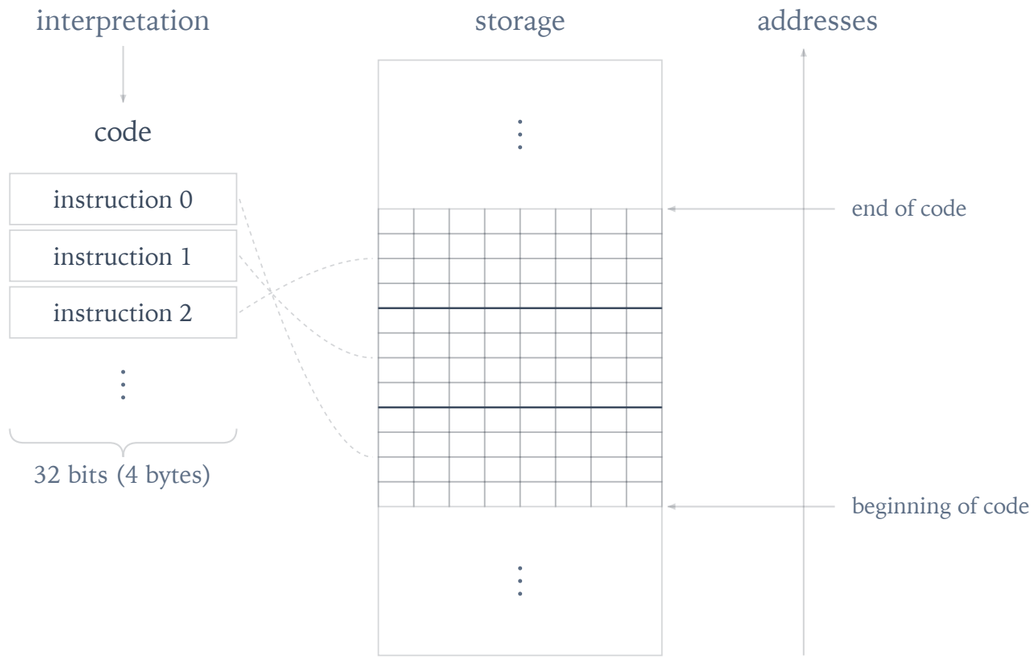
an image, row-major



audio, 8-bit samples

CODE

And an instruction is 32 bits. Code is bytes too.



machine code, four bytes per instruction

Selfie compiles its source to 43,492 instructions of 32 bits each: 173,968 bytes of code and 14,424 bytes of data, laid out one after the other in a file.

That file is bytes at addresses, like a text or an image. The machine reads it and does what it says. Nothing in the bytes tells you whether they are code or data; the machine chapter shows what does.

```
$ ./selfie -c examples/hello-world.c -m 1
./selfie: selfie compiling examples/hello-world.c to 64-b
…
./selfie: 64-bit mipster executing 64-bit RISC-U binary e
Hello World!
```

ONE NOTATION, MANY MEANINGS

1010101 is 85, and U, and −43, and an address, and half an instruction. The bits do not say which.

Everything a machine stores is a number. Everything a number means is decided by whoever reads it, and by nothing else.

Last week: one prefix, two meanings. This week: one byte, five. The gap between the notation and its meaning is not a defect of computers. It is what this class is about, and next week we start on the notations that pin meaning down.

> *Whatever you see on a screen could come from anywhere and mean anything.*
>
> THE BOOK, LIFE 4

BEFORE NEXT WEEK

Recommended exercises.

1. Read the Size chapter from *Numbers* to *Code*.
2. Write 42, 255, and 1000 in binary and hexadecimal. Add 85 and 42 in binary with carries.
3. What is 1010101 as a signed 7-bit number? What is 11111111 as an unsigned and as a signed byte?
4. Write your first name in ASCII, in binary, and count the bits. Then in UTF-8 if it has an umlaut.
5. Run `examples/overflows.c` and explain each line of output.
6. In the pointers figure, what happens if the byte at address 0 held 7 instead of 85?

NEXT WEEK

Notation: formal languages. EBNF, and the two languages selfie is made of — C* with seven keywords, RISC-U with fourteen instructions — read exactly.