

Exercise walk-through

3.1

Binary and data encoding

University Computer Science

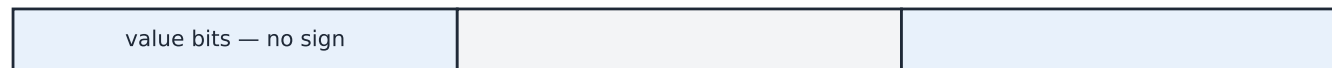
Binary and data encoding: three readings of the same bits

- 1 **Ranges Unsigned** 0 to $2^n - 1$; **two's complement** -2^{n-1} to $2^{n-1} - 1$ — flip all bits, add 1 (computes $2^n - b$).
- 2 **Worked** -73 in 8 bits: $73 = 01001001$, flip 10110110 , $+1 = 10110111$; $96 + 64 = 160$ overflows signed 8-bit (max 127) $\rightarrow$ reads -96 .
- 3 **Float/text IEEE-754** sign-exponent-fraction: 0.1 has no finite binary form, so $0.1 + 0.2 = 0.300000000000000004$ (rounding); ASCII 7 bits, Unicode code points, UTF-8 variable width.

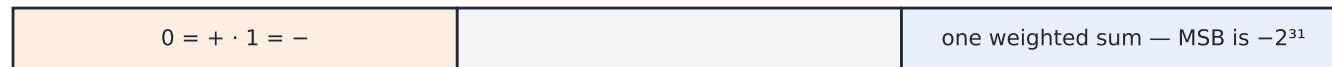
Bit-layout boxes

the same 32 bits, read three ways

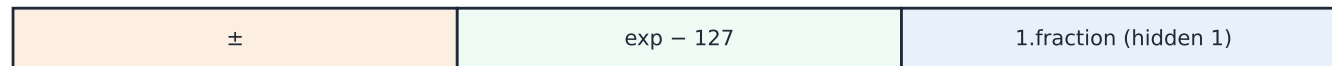
unsigned 32-bit: $0 \dots 2^{32}-1$
bit 31



two's complement 32-bit: $-2^{31} \dots 2^{31}-1$
sign



IEEE-754 float32: sign · biased exponent · fraction
1 8 23



-73 in 8 bits: 73 = 01001001 → flip 10110110 → +1 10110111 ✓
96 + 64 = 160 overflows signed 8-bit (max 127) → reads -96 · 0.1 + 0.2 = 0.30000000000000004 (rounding)

The same 32 bits read three ways: **unsigned, two's complement, IEEE float.**

Exercises 2.1-2.3

2.1 Convert 45 to 8-bit binary.

[3]

2.2 Write -50 as 8-bit two's complement.

[3]

2.3 The character A has ASCII code 65 – what stores the character 1?

[3]

Exercises 2.1-2.3

2.1 Convert 45 to 8-bit binary.

[3]

2.1

- $45 = 32 + 8 + 4 + 1 = 00101101_2$ ✓.

2.2 Write -50 as 8-bit two's complement.

[3]

2.2

- $50 = 00110010_2$; flip 11001101; add 1 $\rightarrow 11001110$ ✓.

2.3 The character A has ASCII code 65 – what stores the character 1?

[3]

2.3

- The code 49 – a code, not a value ✓.

Exercise 3.1

3.1 All arithmetic in 8-bit two's complement.

Compute (i) $60 + 50$, (ii) $-60 - 50$, (iii) $100 + 55$, stating for each whether overflow occurs and what the stored pattern reads. Explain in one line why unsigned and signed overflow look different in the same adder. [8]

Exercise 3.1

3.1 All arithmetic in 8-bit two's complement.

Compute (i) $60 + 50$, (ii) $-60 - 50$, (iii) $100 + 55$, stating for each whether overflow occurs and what the stored pattern reads. Explain in one line why unsigned and signed overflow look different in the same adder. [8]

3.1

- (i) 110, in range ✓.
- (ii) -110 , in range $(-128..127)$ ✓.
- (iii) $155 > 127$: overflow; pattern reads $155 - 256 = -101$ ✓.
- Unsigned overflow is carry out of bit 7; signed overflow is when the carry INTO the sign bit differs from the carry out – same adder, two tests ✓.

Exercise 3.2

3.2 IEEE-754 single precision has 1 sign, 8 exponent, 23 fraction bits with bias 127.

State the largest finite float shape (exponent all-ones excluded) and why 0.1 cannot be stored exactly. Explain why float equality tests after arithmetic are unsafe, and what to do instead. [7]

Exercise 3.2

3.2 IEEE-754 single precision has 1 sign, 8 exponent, 23 fraction bits with bias 127.

State the largest finite float shape (exponent all-ones excluded) and why 0.1 cannot be stored exactly. Explain why float equality tests after arithmetic are unsafe, and what to do instead. [7]

3.2

- Largest: exponent 254 (biased) $\rightarrow 2^{127} \times \sim 1.111 \dots_2 \approx 3.4 \times 10^{38}$ ✓.
- $0.1 = 0.000110011 \dots$ repeating – no finite binary form, so it rounds ✓.
- Equality is unsafe because two differently-rounded paths reach slightly different values; compare $|a - b| < \varepsilon$ instead ✓.