Number Systems & Binary Arithmetic

Binary · Hex · Decimal · Two's Complement · Addition · Subtraction · Overflow · Conversion

1 — Positional Notation

Base	Name	Digits	Example	Value
2	Binary	0–1	`1101`	$1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 =$ **13**
8	Octal	0–7	`015`	$1\times8^1 + 5\times8^0 =$ **13**
10	Decimal	0–9	`13`	$1\times10^1 + 3\times10^0 =$ **13**
16	Hexadecimal	0–9,A–F	`0xD` / `x000D`	$13\times16^0 =$ **13**

Hex digit↔nibble: Each hex digit maps to exactly 4 bits. `0=0000` · `1=0001` · `2=0010` · `3=0011` · `4=0100` · `5=0101` · `6=0110` · `7=0111`
`8=1000` · `9=1001` · `A=1010` · `B=1011` · `C=1100` · `D=1101` · `E=1110` · `F=1111`

2 — Powers of 2 (Quick Lookup)

2^n	n=0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	1	2	4	8	16	32	64	128	256	512	1K	2K	4K	8K	16K	32K

Useful: $2^{16} = 65\,536$ (LC-3 address space) · $2^8 = 256$ · $2^{12} = 4\,096$ · $2^{15} = 32\,768$ (most-negative 16-bit signed)

3 — Base Conversion Methods

Decimal → Binary (repeated division by 2)

Divide by 2, record remainders bottom-up:

÷ 2	Quotient	Remainder (bit)
25	12	**1** (LSB)
12	6	**0**
6	3	**0**
3	1	**1**
1	0	**1** (MSB)

Read remainders **upward**: $25_{10} =$ `11001`$_2$

Decimal → Hex (repeated division by 16)

Same idea, divide by 16:

÷ 16	Quotient	Remainder (hex digit)
255	15	**F** (LSD)

Binary → Decimal (sum of place values)

Write out bit positions, sum the 1s:

bit 7	6	5	4	3	2	1	0 (LSB)
0	1	1	0	1	0	0	1
—	64	32	—	8	—	—	1

64 + 32 + 8 + 1 = **105**

Binary ↔ Hex (group into nibbles)

Split binary into groups of 4 from the right, convert each:

```
1010 1111 0011
 A    F    3
  → 0xAF3
```

Pad with leading zeros if the leftmost group has < 4 bits.

15	0	**F** (MSD)

Read upward: 255_{10} = `xFF`

Hex → Decimal

Expand each digit by its power of 16:

`0x2B` $= 2\times16^1 + 11\times16^0 = 32 + 11 =$ **43**

4 — Unsigned vs. Signed Ranges

Bits (n)	Unsigned range	Unsigned max	Signed range (2's comp)	Signed max / min
4	$0 \to 15$	$2^4-1 = 15$	$-8 \to +7$	-2^3 to 2^3-1
8	$0 \to 255$	$2^8-1 = 255$	$-128 \to +127$	-2^7 to 2^7-1
16	$0 \to 65\,535$	$2^{16}-1 = 65\,535$	$-32\,768 \to +32\,767$	-2^{15} to $2^{15}-1$
n	$0 \to 2^n-1$	2^n-1	$-2^{n-1} \to 2^{n-1}-1$	MSB is sign bit

5 — Two's Complement

How to negate a number

Method 1 — Invert & add 1:

Step	Example: negate +5 (4-bit)
Start	`0101` (+5)
Invert all	`1010` (flip every bit)
Add 1	`1011` (−5 ✓)

Method 2 — Copy from right up to and including first 1, then invert the rest:

```
0110 0 1 0 0 → copy rightmost 1 and zeros:
…1 0 0
invert the rest: 1 0 0 1  → 1001 1100
```

Sign extension (SEXT)

To extend an n-bit value to more bits, **copy the MSB** into all new positions:

4-bit	8-bit SEXT	Value
`0101`	`0000 0101`	+5 (MSB=0 → pad with 0s)
`1011`	`1111 1011`	−5 (MSB=1 → pad with 1s)

Sign extension **preserves the value**. Zero extension (`ZEXT`) pads with 0s regardless (for unsigned values).

Reading a negative two's complement value

If MSB = 1, the number is negative. Two ways to read it:

Option A — Negate it, read as positive, add minus sign:
`1011` → invert: `0100` → +1 = `0101` = 5 → value is **−5**

Option B — Weighted MSB: treat MSB as -2^{n-1}, sum the rest normally:
`1011` $= -1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = -8+0+2+1 =$ **−5**

Special cases

Pattern	n-bit value	Note
`0000…0`	0	Only zero
`0111…1`	$+2^{n-1} - 1$	Most positive
`1000…0`	-2^{n-1}	Most negative — **has no positive counterpart!**
`1111…1`	−1	All ones = −1

6 — Binary Addition

Single-bit addition table

A	B	C_{in}	Sum · C_{out}
0	0	0	0 · 0

Column addition example (8-bit): 53 + 78

bit	7	6	5	4	3	2	1	0
carry	0	1	1	0	0	0	1	0

0	1	0	1 · 0
1	0	0	1 · 0
1	1	0	0 · 1
0	0	1	1 · 0
0	1	1	0 · 1
1	0	1	0 · 1
1	1	1	1 · 1

53 =	0	0	1	1	0	1	0	1
78 =	0	1	0	0	1	1	1	0
sum	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**1**

Result: `01111011` = 64+32+16+8+2+1 = **131** ✓

Rules to remember

$0 + 0 =$ **0**, carry 0 $\quad$ $0 + 1 =$ **1**, carry 0
$1 + 1 =$ **0**, carry 1 $\quad$ $1 + 1 + 1 =$ **1**, carry 1

7 — Binary Subtraction

Key insight: A − B is computed as A + (−B). Negate B using two's complement, then add normally. No separate subtraction circuit needed.

Example: 12 − 5 (8-bit)

Step	Value
12 in binary	`0000 1100`
5 in binary	`0000 0101`
Invert 5	`1111 1010`
Add 1 (−5)	`1111 1011`
Add 12 + (−5)	`0000 0111` (carry out discarded)
Result	`0000 0111` = **7** ✓

Example: 5 − 12 (8-bit) — negative result

Step	Value
5 in binary	`0000 0101`
12 in binary	`0000 1100`
Invert 12	`1111 0011`
Add 1 (−12)	`1111 0100`
Add 5 + (−12)	`1111 1001` (no carry out)
Result (MSB=1→neg)	negate: `0000 0111` = **−7** ✓

8 — Overflow Detection

Operand signs	Result sign	Overflow?	Example (4-bit)
1. 1. +	+	No	3 + 4 = 7 `0011+0100=0111` ✓
1. 1. +	−	**YES**	5 + 4 = −7? `0101+0100=1001` ✗
− + −	−	No	−3 + (−4) = −7 ✓
− + −	+	**YES**	−5 + (−4) = +7? `1011+1100=0111` ✗
1. 1. −	either	Never	Mixed signs can never overflow

Overflow rules (signed):
- Adding two **positives** and getting a **negative** → overflow
- Adding two **negatives** and getting a **positive** → overflow
- **Hardware shortcut:** overflow = C_{in} to MSB **XOR** C_{out} from MSB
- **Unsigned overflow** (carry): simply check if carry-out from MSB = 1

9 — Bitwise Operations & Tricks

Truth tables

A	B	AND	OR	XOR	NOT A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Shifts

Shift	Effect	Example (8-bit)
Left $\ll$ n	$\times 2^n$; fill LSBs with 0	`0000 0011` $\ll 2 =$ `0000 1100` (3→12)
Logical $\gg$ n	$\div 2^n$ (unsigned); fill MSBs with 0	`0000 1100` $\gg 2 =$ `0000 0011` (12→3)
Arith. $\gg$ n	$\div 2^n$ (signed); fill MSBs with sign	`1111 0000` $\gg 2 =$ `1111 1100` (−16→−4)

Common bit manipulation patterns

Goal	Operation
Test bit k	`x AND (1 << k)` $\neq 0$
Set bit k	`x OR (1 << k)`
Clear bit k	`x AND NOT(1 << k)`
Toggle bit k	`x XOR (1 << k)`
Clear all	`x AND 0000…0 = 0`
Check if power of 2	`x AND (x-1) == 0`
Isolate LSB	`x AND (-x)`
Check odd/even	`x AND 1` (1=odd, 0=even)

Masking example — extract bits 7–4

Value: `1010 1101`
Mask: `1111 0000` (`AND` with `0xF0`)
Result:`1010 0000` → shift right 4 → `0000 1010` = 10

10 — Decimal / Binary / Hex Quick Reference (0–31)

Dec	Binary	Hex
0	00000	0x00
1	00001	0x01
2	00010	0x02
3	00011	0x03
4	00100	0x04
5	00101	0x05
6	00110	0x06
7	00111	0x07
8	01000	0x08
9	01001	0x09
10	01010	0x0A
11	01011	0x0B
12	01100	0x0C
13	01101	0x0D
14	01110	0x0E
15	01111	0x0F

Dec	Binary	Hex
16	10000	0x10
17	10001	0x11
18	10010	0x12
19	10011	0x13
20	10100	0x14
21	10101	0x15
22	10110	0x16
23	10111	0x17
24	11000	0x18
25	11001	0x19
26	11010	0x1A
27	11011	0x1B
28	11100	0x1C
29	11101	0x1D
30	11110	0x1E
31	11111	0x1F