

MATH 6

NUMERAL SYSTEMS AND BINARY COMPUTATION

The ancient Babylonians counted in base 60 — that’s why an hour still has 60 minutes. The Maya used base 20, fingers *and* toes. And your phone? It thinks in **base 2**, using nothing but 0s and 1s. Every photo, song, and message you send is a giant pattern of 0s and 1s. In this handout we learn to read and write that language, plus its shorthand, **hexadecimal**.

Place Value in Any Base

Positional notation

In base b we use the digits $0, 1, \dots, b-1$, and each place is worth b times the place to its right. The value of a number is the sum of (digit $\times$ place value). We write the base as a subscript: $(1011)_2$, $(2B)_{16}$.

In base 10 the places are 1, 10, 100, 1000, ... In base 2 they are powers of 2:

Place value	128	64	32	16	8	4	2	1
Power of 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Binary $\rightarrow$ decimal: add up the places that hold a 1.

$$(1011)_2 = 8 + 2 + 1 = 11$$

$$(11010110)_2 = 128 + 64 + 16 + 4 + 2 = 214$$

Decimal $\rightarrow$ binary: divide by 2 again and again; read the remainders **from the bottom up**.

$$\begin{array}{rcl} 45 \div 2 = 22 & \text{rem } 1 \\ 22 \div 2 = 11 & \text{rem } 0 \\ 11 \div 2 = 5 & \text{rem } 1 \\ 5 \div 2 = 2 & \text{rem } 1 \\ 2 \div 2 = 1 & \text{rem } 0 \\ 1 \div 2 = 0 & \text{rem } 1 \end{array} \Rightarrow 45 = (101101)_2$$

Check: $32 + 8 + 4 + 1 = 45$. $\checkmark$ The same two methods work in any base — just use that base’s place values, or divide by that base.

Binary Arithmetic

Binary arithmetic works exactly like the arithmetic you already know. The only new fact is that $1 + 1$ makes a full “two,” which is $(10)_2$: write 0 and carry 1.

Adding bits

$$0 + 0 = 0, \quad 0 + 1 = 1, \quad 1 + 1 = (10)_2$$

Three 1s in a column: $1 + 1 + 1 = (11)_2$ — write 1, carry 1.

Example. $(1011)_2 + (1101)_2$:

$$\begin{array}{r} \text{carries } 1 \ 1 \ 1 \ 1 \\ 1 \ 0 \ 1 \ 1 \\ + 1 \ 1 \ 0 \ 1 \\ \hline 1 \ 1 \ 0 \ 0 \ 0 \end{array}$$

Check: $11 + 13 = 24 = 16 + 8$. $\checkmark$

Multiplying is even easier: each row is either a copy of the number (for a 1) or all zeros (for a 0), shifted one place left each time. For example, $(101)_2 \times (11)_2 = (101)_2 + (1010)_2 = (1111)_2$, which is $5 \times 3 = 15$. $\checkmark$

Hexadecimal: Binary’s Shorthand

Binary is perfect for computers but painful for people: 255 is 11111111. Programmers use **base 16** instead, with six extra “digits”:

$$A = 10, \quad B = 11, \quad C = 12, \quad D = 13, \quad E = 14, \quad F = 15.$$

Example. $(2B)_{16} = 2 \times 16 + 11 = 43$, and $(1A3)_{16} = 1 \times 256 + 10 \times 16 + 3 = 419$.

The magic is that $16 = 2^4$, so **one hex digit is exactly four bits**. To convert binary to hex, split the bits into groups of four *starting from the right* and look each group up:

$$\begin{array}{ccccccc} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ \underbrace{\hspace{1.5cm}}_{1011 = B} & \underbrace{\hspace{1.5cm}}_{0100 = 4} \\ (10110100)_2 = (B4)_{16} \end{array}$$

0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

Going back is just as easy: $(3AF)_{16}$ becomes 0011 1010 1111.

Octal and Unix File Permissions

Octal is base 8, with digits 0–7. Since $8 = 2^3$, **one octal digit is exactly three bits**, so we group bits in threes:

$$(1101110)_2 = (1\ 101\ 110)_2 = (156)_8.$$

Octal still lives inside every Mac and Linux computer. A file’s permissions are three octal digits — for the *owner*, the *group*, and *everyone else*. Each digit is three bits that switch on read, write, and execute:

Digit	Bits	Letters	Allowed
7	111	rwX	read, write, execute
6	110	rw-	read, write
5	101	r-X	read, execute
4	100	r--	read only
0	000	---	nothing

Example. Permission 755: the owner gets rwx (7); the group and everyone else get r-x (5) — they can read and run the file but not change it.

Colors in Hex

Your screen makes every color by mixing Red, Green, and Blue light. Each channel runs from 0 (off) to 255 (full) — one byte, or exactly two hex digits.

#	FF	57	33	
	↓	↓	↓	
	Red 255	Green 87	Blue 51	
				■ #FF0000 red ■ #00FF00 green ■ #FFFF00 yellow ■ #FFFFFF white ■ #000000 black

Example. Red = 100, Green = 150, Blue = 200. Since $100 = 6 \cdot 16 + 4$, $150 = 9 \cdot 16 + 6$, and $200 = 12 \cdot 16 + 8$, the code is #6496C8 — a soft, cornflower-like blue.

Negative Numbers: Two’s Complement

A computer has no minus sign — only bits. So how can it store -45 ? Think about what -45 *means*: it’s the number that gives 0 when you add it to 45. Let’s find that number!

First, a surprise. Numbers are kept in exactly 8 bits. Add 1 to 11111111 and the answer 1 00000000 needs 9 bits — so the red 1 falls off the end, leaving 00000000 = 0. It’s like a car’s odometer rolling over from 99999 to 00000. So for the computer, “adds up to 0” means “adds up to 1 00000000.”

Example. What should we add to 45 = 00101101 to get 0? Build the answer X bit by bit from the right, aiming for a 0 in every column.

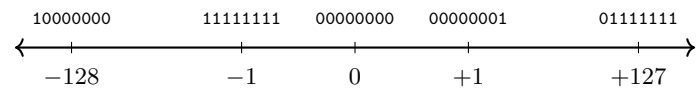
- Keep going to the left. Every column ends up writing 0, and the final carry falls off the end:

carry arriving	1	1	1	1	1	1	1	0
bit of 45	0	0	1	0	1	1	0	1
bit of X	1	1	0	1	0	0	1	1

			0	0	1	0	1	1	0	1
+		1	1	0	1	0	0	1	1	
	1	0	0	0	0	0	0	0	0	0

Spot the pattern. After the first column a carry of 1 always arrives, so each bit of X is the *opposite* of 45's bit. Only the last column needed one extra 1. So X is 45 with all its bits flipped (11010010), plus one: 11010011. That's the rule, called **two's complement**:

To store $-x$: write x in 8 bits, **flip every bit**, then **add 1**. A leading 1 means the number is negative. 8 bits hold the numbers from -128 to 127 .



Computers can also combine two bit-patterns *column by column*, with no carrying at all:

A	B	AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

AND is 1 only if *both* bits are 1.
OR is 1 if *at least one* bit is 1.
XOR (“exclusive or”) is 1 if *exactly one* bit is 1 — so XOR with a 1 *flips* a bit.

Example. Line up the bits and work one column at a time:

$\begin{array}{cccc} & 1 & 0 & 1 & 0 \\ \text{AND} & 1 & 1 & 0 & 0 \\ \hline & 1 & 0 & 0 & 0 \end{array}$	$\begin{array}{cccc} & 1 & 0 & 1 & 0 \\ \text{OR} & 1 & 1 & 0 & 0 \\ \hline & 1 & 1 & 1 & 0 \end{array}$	$\begin{array}{cccc} & 1 & 0 & 1 & 0 \\ \text{XOR} & 1 & 1 & 0 & 0 \\ \hline & 0 & 1 & 1 & 0 \end{array}$
---	--	---

- **Reading remainders top-down.** After repeated division, read the remainders from the *bottom up*.
- **Grouping from the left.** Group bits in 4s (hex) or 3s (octal) starting from the *right*; pad the leftmost group with zeros.
- **Forgetting the hex letters.** $A = 10$, $B = 11$, $C = 12$, $D = 13$, $E = 14$, $F = 15$.
- **Leaving off the base.** Is “10” ten, or two? Write $(10)_2$ or $(10)_{10}$.
- **Dropping bits in two’s complement.** Always keep all 8 bits: 00000101, not 101.

Homework

1. Convert to decimal:
(a) $(1101)_2$ (b) $(10010)_2$ (c) $(101010)_2$
2. Convert to binary:
(a) 23 (b) 64 (c) 100
3. Compute in binary, then check your answer by converting to decimal:
(a) $(11011)_2 + (1110)_2$ (b) $(10110)_2 + (1011)_2$ (c) $(111)_2 \times (11)_2$
4. Convert between binary and hexadecimal:
(a) $(10101100)_2$ to hex (c) $(7B)_{16}$ to binary
(b) $(01101001)_2$ to hex (d) $(C3)_{16}$ to binary
5. Colors in hex:
(a) The color #1E90FF is a bright sky blue. What are its red, green, and blue values in decimal?
(b) Write the hex code for the color with Red = 255, Green = 165, Blue = 0. What color do you think it is?
6. Octal:
(a) Convert $(10111011)_2$ to octal.
(b) Convert $(752)_8$ to binary.
7. A Unix file has permission 644. What does each digit mean? Who can read the file? Who can change it?
8. A byte is 8 bits. What is the largest ordinary (unsigned) number that fits in one byte? Write it in binary, hexadecimal, and decimal. How does it compare with the largest number in 8-bit two's complement?
9. **M** Two's complement (8 bits):
(a) Write -12 in 8-bit two's complement.
(b) Which number is stored as 11111000? *Hint: what do you add to it to get 100000000?*
10. Compute $(1101)_2$ AND $(1011)_2$, then OR, then XOR.
11. The number $(111)_b$ in some base b equals 21 in decimal. Find the base b .
12. A number is **palindromic** if it reads the same forwards and backwards. Find a number greater than 10 that is palindromic in *both* base 10 and base 2. (For instance, 11 doesn't work: $11 = (1011)_2$.)
13. **H** **Binary weighing.** You have a balance scale. You want to weigh any whole number of kilograms from 1 to 15, putting your weights only on one side of the scale. What is the smallest number of weights you need, and how heavy should they be? *Hint: think in binary!*