

MATH 6

NUMERAL SYSTEMS AND BINARY COMPUTATION — FULL 2026–27

Introduction

When ancient Babylonians studied the sky, they used base 60—which is why we still divide an hour into 60 minutes and a circle into 360 degrees. The Maya used base 20, perhaps inspired by counting fingers and toes. Binary patterns are ancient too, but in the 20th century **binary** (base 2) became the language of computers.

In a 1937 master’s thesis, a young mathematician named Claude Shannon showed how the rules of logic could control networks of electrical switches. That idea became one of the foundations of digital circuits. Today, every photo, song, and message you send is encoded using enormous patterns of 0s and 1s.

This handout reviews binary and hexadecimal (base 16) from Math 5, then uses them in new ways. Hexadecimal is a compact way to write binary, and color codes such as #FF5733 give us one vivid application. We will also explore octal, signed binary numbers, and bitwise operations.

Review: Place Value Systems

Positional Notation

In a **positional** system with an integer base $b \geq 2$:

- We use digits $0, 1, 2, \dots, b - 1$
- Each position represents a power of b
- The value equals the sum of (digit $\times$ place value)

In decimal (base 10): $347 = 3 \times 100 + 4 \times 10 + 7 \times 1 = 3 \times 10^2 + 4 \times 10^1 + 7 \times 10^0$

Binary (Base 2)

Why do computers use binary instead of decimal? The answer lies in electronics. The voltages inside a computer can vary continuously, but digital circuits sort a whole range of low voltages into one state and a whole range of high voltages into another. We call these states 0 and 1, or OFF and ON. Having only two states makes circuits fast and reliable: a tiny voltage wobble usually does not change which state the circuit reads. That is why **binary** (base 2) became the language of digital computing.

Binary Place Values

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

Example 1. Convert $(1011)_2$ to decimal.

Solution. Multiply each digit by its place value and add:

$$(1011)_2 = 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 8 + 2 + 1 = 11_{10}$$

Example 2. Convert $(11010110)_2$ to decimal.

Solution. $(11010110)_2 = 128 + 64 + 16 + 4 + 2 = 214_{10}$

Example 3. Convert 45_{10} to binary.

Solution. Divide repeatedly by 2, recording remainders:

$$\begin{array}{rcl} 45 \div 2 & = & 22 \text{ remainder } 1 \\ 22 \div 2 & = & 11 \text{ remainder } 0 \\ 11 \div 2 & = & 5 \text{ remainder } 1 \\ 5 \div 2 & = & 2 \text{ remainder } 1 \\ 2 \div 2 & = & 1 \text{ remainder } 0 \\ 1 \div 2 & = & 0 \text{ remainder } 1 \end{array}$$

Reading remainders from bottom to top: $45_{10} = (101101)_2$.

Check: $32 + 8 + 4 + 1 = 45$. ✓

Binary Counting (like an odometer)

Decimal	Binary	Decimal	Binary
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

Notice: the rightmost bit alternates every number; the next bit alternates every 2 numbers; etc.

Quick Check

1. Convert $(1100)_2$ to decimal.
2. Convert $(10101)_2$ to decimal.
3. Convert 19_{10} to binary.
4. Convert 100_{10} to binary.

Binary Arithmetic

Computers don't just store information in binary—their digital circuits also work with binary-valued logic. Binary arithmetic follows the same place-value rules as decimal arithmetic, but it has only two digits. The one new habit is remembering that $1 + 1 = 10$ (read that as “one-zero” in binary, not “ten”)!

Binary Addition Rules

$$0 + 0 = 0, \quad 0 + 1 = 1, \quad 1 + 0 = 1, \quad 1 + 1 = 10 \text{ (write 0, carry 1)}$$

When three 1s meet: $1 + 1 + 1 = 11$ (write 1, carry 1)

Example 4. Compute $(1011)_2 + (1101)_2$.

Solution. Add column by column from right to left, carrying as needed:

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

Check: $11 + 13 = 24$, and $(11000)_2 = 16 + 8 = 24$. ✓

Binary Multiplication Rules

$$0 \times 0 = 0, \quad 0 \times 1 = 0, \quad 1 \times 0 = 0, \quad 1 \times 1 = 1$$

Multiplying by 0 gives 0; multiplying by 1 copies the number. Then add the partial products (shifted).

Example 5. Compute $(101)_2 \times (11)_2$.

Solution. Multiply and shift like in decimal:

$$\begin{array}{r} 101 \\ \times 11 \\ \hline 101 \\ +1010 \\ \hline 1111 \end{array}$$

Check: $5 \times 3 = 15$, and $(1111)_2 = 8 + 4 + 2 + 1 = 15$. ✓

Quick Check

5. Add in binary: $(1001)_2 + (0110)_2$
6. Multiply in binary: $(110)_2 \times (10)_2$

Hexadecimal (Base 16)

Binary is perfect for computers, but it's terrible for humans. The number 255 becomes 11111111 in binary—eight digits that are hard to read and easy to mistype. Programmers needed a more compact notation, and **hexadecimal** (base 16) turned out to be ideal. Since $16 = 2^4$, each hex digit represents exactly four binary digits, making conversion effortless. The number 255 becomes simply FF—much easier to remember!

Hexadecimal uses 16 digits: 0, 1, 2, ..., 9, A, B, C, D, E, F
where $A = 10$, $B = 11$, $C = 12$, $D = 13$, $E = 14$, $F = 15$.

Why Hexadecimal?

- Each hex digit represents exactly 4 binary digits (bits)
- Much shorter than binary: $(11111111)_2 = FF_{16}$
- Colors use hex codes: #FF5733 means Red=255, Green=87, Blue=51

Example 6. Convert $(2B)_{16}$ to decimal.

Solution. $(2B)_{16} = 2 \times 16 + 11 \times 1 = 32 + 11 = 43_{10}$

Example 7. Convert $(1A3)_{16}$ to decimal.

Solution. $(1A3)_{16} = 1 \times 256 + 10 \times 16 + 3 \times 1 = 256 + 160 + 3 = 419_{10}$

Example 8. Convert 500_{10} to hexadecimal.

Solution. Divide repeatedly by 16:

$$\begin{array}{rcl} 500 \div 16 & = & 31 \text{ remainder } 4 \\ 31 \div 16 & = & 1 \text{ remainder } 15 = F \\ 1 \div 16 & = & 0 \text{ remainder } 1 \end{array}$$

Reading from bottom to top: $500_{10} = (1F4)_{16}$.

Check: $256 + 15 \times 16 + 4 = 256 + 240 + 4 = 500$. ✓

Quick Check

7. Convert $(3C)_{16}$ to decimal.
8. Convert 255_{10} to hexadecimal.

Binary–Hexadecimal Conversion

The magic of hexadecimal is that converting to and from binary requires no calculation—just pattern matching. Since one hex digit equals exactly four binary digits, you simply group the binary digits and look up each group. This is why programmers love hex: it's a human-friendly way to write binary.

Each hex digit corresponds to exactly 4 binary digits:

Hex	Binary	Hex	Binary	Hex	Binary	Hex	Binary
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

Example 9. Convert $(11010110)_2$ to hexadecimal.

Solution. Group into sets of 4 bits from the right:

$$(11010110)_2 = (1101\ 0110)_2 = (D6)_{16}$$

Example 10. Convert $(3AF)_{16}$ to binary.

Solution. Replace each hex digit with its 4-bit equivalent:

$$3 = 0011, \quad A = 1010, \quad F = 1111$$

$$(3AF)_{16} = (001110101111)_2$$

Binary to Hexadecimal: Group by 4 bits

Binary: 1 0 1 1 0 1 0 0

$\underbrace{1011}_{1011 = B} \quad \underbrace{0100}_{0100 = 4}$

Hex: B 4

$(10110100)_2 = (B4)_{16}$

Quick Check

9. Convert $(10110011)_2$ to hexadecimal.
10. Convert $(A5)_{16}$ to binary.

A Bit of History

Binary is far older than computers!

- **Over 2000 years ago**, the Indian scholar **Pingala** studied patterns of long and short syllables in Sanskrit poetry. These are two-choice patterns, much like strings of 0s and 1s—although Pingala was not writing ordinary numbers in our modern binary notation.
- In **1703**, the German mathematician **Gottfried Leibniz** described the modern binary system in a paper presented to the French Academy. He loved that every whole number can be built from just 0 and 1. He also connected binary patterns with the 64 hexagrams of the ancient Chinese *I Ching*: $64 = 2^6$, the number of possible strings of six binary choices.
- The word “**bit**” is short for *binary digit*. A group of 8 bits is a “**byte**”. The spelling was changed from “bite” so it would not be mistaken for “bit.” Half a byte (4 bits, exactly one hex digit) is playfully called a “**nibble**”!
- **Today**, Unicode code points are conventionally written in hexadecimal. For example, the grinning-face emoji has code point U+1F600. Some emoji are made from more than one code point, and text is stored and sent as bytes rather than as these printed hexadecimal labels.

Octal (Base 8)

Before hexadecimal became standard, many early computers used **octal** (base 8). Since $8 = 2^3$, each octal digit represents exactly three binary digits—convenient for machines with 12-bit, 24-bit, or 36-bit words. While hexadecimal has largely replaced octal in modern computing, octal survives in one important place: Unix and Linux file permissions, where the three bits for read/write/execute map perfectly to a single octal digit.

Octal uses 8 digits: 0, 1, 2, 3, 4, 5, 6, 7.

Binary–Octal Conversion

Since $8 = 2^3$, each octal digit corresponds to exactly **3 binary digits**:

Oct	Bin	Oct	Bin	Oct	Bin	Oct	Bin
0	000	2	010	4	100	6	110
1	001	3	011	5	101	7	111

Example 11. Convert $(1101110)_2$ to octal.

Solution. Group into sets of 3 bits from the right:

$$(1101110)_2 = (1\ 101\ 110)_2 = (001\ 101\ 110)_2 = (156)_8$$

Check: $(156)_8 = 1 \times 64 + 5 \times 8 + 6 = 110$. And $64 + 32 + 8 + 4 + 2 = 110$. ✓

Unix File Permissions

In Unix/Linux, file permissions use octal. Each digit represents:

Digit	Binary	Meaning	Permissions
7	111	rwX	read, write, execute
6	110	rw-	read, write
5	101	r-X	read, execute
4	100	r-	read only
0	000	—	no permissions

Permission 755 means: owner=rwx (7), group=r-x (5), others=r-x (5)

Quick Check

- Convert $(110101)_2$ to octal.
- Convert $(72)_8$ to binary.

RGB Color Codes

One of the most visible uses of hexadecimal is in color codes. A common way to describe a screen color is with a code like #FF5733. You may spot one in a graphics program or a CSS stylesheet. (Hex is not the only way computers describe color, but it is a wonderfully compact one.)

Computer screens create colors by mixing **Red**, **Green**, and **Blue** light. In the common 8-bit RGB system, each color channel has an intensity from 0 (off) to 255 (maximum)—exactly one byte (8 bits).

Hexadecimal Color Codes

A color code like #FF5733 has three parts:

- FF = Red intensity (255 in decimal = maximum red)
- 57 = Green intensity (87 in decimal)
- 33 = Blue intensity (51 in decimal)

Each component ranges from 00 (0) to FF (255).

FF **57** **33**
 ↓ ↓ ↓
 Red: 255 Green: 87 Blue: 51

Common Colors:

	#FF0000 (Red)
	#00FF00 (Green)
	#0000FF (Blue)
	#FFFFFF (White)
	#000000 (Black)
	#FFFF00 (Yellow)

Example 12. What color is #008080?

Solution. Red = 00 (0), Green = 80 (128), Blue = 80 (128). Equal amounts of green and blue with no red creates **teal** (a blue-green color).

Example 13. Create a hex code for a color with Red=100, Green=150, Blue=200.

Solution. Convert each to hex:

- $100 = 6 \times 16 + 4 = (64)_{16}$

- $150 = 9 \times 16 + 6 = (96)_{16}$
- $200 = 12 \times 16 + 8 = (C8)_{16}$

The color code is #6496C8 (a muted blue).

Quick Check

13. What color is #FFFF00?
14. Write the hex code for a color with Red = 0, Green = 0, Blue = 255.

Two's Complement: Negative Numbers

So far we've represented only nonnegative numbers. But computers need negative numbers too—for bank balances, temperatures below zero, and game characters moving backward. A row of bits has no printed minus sign, so how can it tell us that a number is negative?

Here is a good way to think about it. What is -45 , really? It is the number that gives 0 when you add it to 45. So let's go looking for an 8-bit pattern that does exactly that.

Numbers That Roll Over

First, a surprise. A computer keeps each number in a fixed number of bits — here, 8. Watch what happens when we add 1 to the largest 8-bit pattern:

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

The answer needs 9 bits, but there are only 8 places, so the red 1 simply falls off the end. The computer is left with 00000000, which is 0. It's just like a car's odometer rolling over from 99999 to 00000.

So in an 8-bit computer, “adding up to 0” means “adding up to $\textcolor{red}{1}$ 00000000.”

Example 14. Earlier we found that $45 = (101101)_2$, which is 00101101 in 8 bits. Find an 8-bit pattern X such that $00101101 + X = \textcolor{red}{1} 00000000$. This X will play the role of -45 .

Solution. What should we add to 45 to get 0? Let's build X one bit at a time, starting from the right — the same way we add. Our goal is to write a 0 in every column.

- **Column 1.** 45 ends in 1, so X must also end in 1: $1 + 1 = 10$. Write 0, carry 1.
- **Column 2.** Now we have the carry 1, plus 45's bit 0, plus ?. For $1 + 0 + ?$ to end in 0, we need $? = 1$: $1 + 0 + 1 = 10$. Write 0, carry 1.
- **Column 3.** The carry 1, plus 45's bit 1, plus ?. Now $1 + 1$ is already 10, so $? = 0$: $1 + 1 + 0 = 10$. Write 0, carry 1.

Every column works the same way. Here is the full record, right to left:

Column	8	7	6	5	4	3	2	1
Carry arriving	1	1	1	1	1	1	1	0
Bit of 45	0	0	1	0	1	1	0	1
Bit of X we need	1	1	0	1	0	0	1	1
Column total	10	10	10	10	10	10	10	10

So $X = 11010011$. Let's check the whole addition:

$$\begin{array}{r}
 \text{carries } 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \\
 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \\
 + \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \\
 \hline
 \textcolor{red}{1} \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0
 \end{array}
 \quad \checkmark$$

The carry out of Column 8 is the red 1, and it falls off the end. We are left with 00000000. So in an 8-bit computer, -45 is stored as 11010011.

Spot the pattern. Put 45 and X side by side:

$$\begin{array}{r}
 45 : 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \\
 X : 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1
 \end{array}$$

From Column 2 on, a carry of 1 always arrives, so each bit of X must be the *opposite* of the bit of 45 (carry + bit + ?: $1 + 0 + 1$ and $1 + 1 + 0$ both make 10). So X is simply 45 with its bits **flipped** — except in Column 1, where no carry arrives and we needed one extra 1. Indeed, flipping 45 gives 11010010, and $X = 11010011$ is exactly **one more**. That's the rule, and it has a name — **two's complement**:

Two's Complement (8 bits)

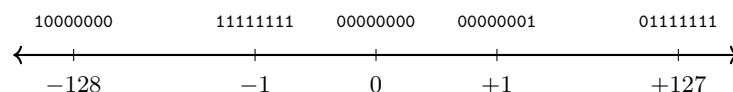
To store $-x$: write x in 8 bits, **flip every bit**, then **add 1**.

- If the leftmost bit is 0, the number is **nonnegative** (read it normally).
- If the leftmost bit is 1, the number is **negative**.
- 8 bits can store the numbers from -128 to $+127$.

Why does the rule always work? Flipping the bits of *any* number gives a partner that adds up to 11111111 with no carries at all — every column is $0 + 1$ or $1 + 0$. Adding one more 1 rolls 11111111 over to 1 00000000, which is 0. So “flip and add 1” always produces the negative.

The rule works in both directions. To find out which negative number a pattern like 11010011 stands for, flip and add 1 again: you get 00101101 = 45, so the pattern was -45 . And the best part: the computer never needs a separate “subtraction machine.” To compute $50 - 45$, it just adds $50 = 00110010$ and the pattern 11010011 for -45 . The sum is 1 00000101; the red 1 falls off, leaving 00000101 = 5. The same adding circuit does subtraction too!

8-bit Two's Complement (schematic)



Quick Check

15. What is the 8-bit two's complement representation of -3 ?
16. In 8-bit two's complement, what decimal number does $(11111110)_2$ represent?

Bitwise Operations

Beyond arithmetic, computers can manipulate individual bits directly. These **bitwise operations** are surprisingly powerful. They can check whether a number is odd (look at the last bit!), store many true/false flags in one number, and combine data with a key in some encryption systems.

AND, OR, XOR Operations

A	B	$A \text{ AND } B$	$A \text{ OR } B$	$A \text{ XOR } B$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- **AND**: Both bits must be 1 to get 1
- **OR**: At least one bit must be 1 to get 1
- **XOR** (exclusive or): Exactly one bit must be 1 to get 1

Example 15. Compute $(1010)_2$ AND $(1100)_2$, OR, and XOR.

Solution. Apply the operation bit by bit:

$$\begin{array}{r} 1\ 0\ 1\ 0 \\ \text{AND } 1\ 1\ 0\ 0 \\ \hline 1\ 0\ 0\ 0 \end{array} \quad \begin{array}{r} 1\ 0\ 1\ 0 \\ \text{OR } 1\ 1\ 0\ 0 \\ \hline 1\ 1\ 1\ 0 \end{array} \quad \begin{array}{r} 1\ 0\ 1\ 0 \\ \text{XOR } 1\ 1\ 0\ 0 \\ \hline 0\ 1\ 1\ 0 \end{array}$$

Applications:

- **AND** with a “mask” extracts specific bits (e.g., get the last 4 bits)
- **OR** sets specific bits to 1
- **XOR** flips selected bits (used with a key in some encryption systems, and also in error detection)

Quick Check

17. Compute $(1111)_2$ AND $(0101)_2$.
18. Compute $(1010)_2$ XOR $(1100)_2$.

Key Takeaways

- **Binary (base 2):** Uses digits 0 and 1; place values are powers of 2
- **Hexadecimal (base 16):** Uses 0–9 and A–F; each hex digit = 4 binary digits
- **Octal (base 8):** Uses 0–7; each octal digit = 3 binary digits (used for file permissions)
- **To convert to decimal:** Multiply each digit by its place value and add
- **To convert from decimal:** Divide repeatedly by the base, read remainders bottom-up
- **Binary arithmetic:** Same rules as decimal, but $1 + 1 = 10$ (carry the 1)
- **Two’s complement:** For a fixed number of bits, flip bits and add 1 to negate; the leftmost bit indicates sign
- **Bitwise ops:** AND (both 1), OR (either 1), XOR (exactly one 1)

Common Mistakes

- **Forgetting hex letters:** A=10, B=11, C=12, D=13, E=14, F=15
- **Reading remainders wrong way:** Read from bottom to top, not top to bottom
- **Binary grouping from wrong end:** Group from the right when converting to hex
- **Confusing bases:** Always write subscripts like $(1010)_2$ or $(1010)_{10}$ to avoid ambiguity

Practice Problems

1. Convert $(1110)_2$ to decimal.
2. Convert 37_{10} to binary.
3. Convert $(4D)_{16}$ to decimal.
4. Convert 200_{10} to hexadecimal.
5. Convert $(11110000)_2$ to hexadecimal.
6. Add in binary: $(1101)_2 + (0111)_2$

7. Convert $(101110)_2$ to octal.
8. What color does $\text{\#FF00FF}$ represent? (Hint: which components are on?)
9. Compute $(1100)_2$ AND $(1010)_2$.
10. Compute $(1100)_2$ XOR $(1010)_2$.
11. Convert $(100110)_2$ to decimal.
12. Convert 88_{10} to binary.
13. Convert $(2F)_{16}$ to decimal.
14. Convert 300_{10} to hexadecimal.
15. Convert $(10011010)_2$ to hexadecimal.
16. Convert $(5C)_{16}$ to binary.
17. Multiply in binary: $(101)_2 \times (101)_2$.
18. Convert $(11011)_2$ to octal.
19. Represent -7 in 8-bit two's complement binary.
20. The number $(144)_b$ in some base b equals 100 in decimal. Find b .

Practice Problems: Solutions

1. $(1110)_2 = 8 + 4 + 2 = \mathbf{14}$
2. $37 = 32 + 4 + 1 = \mathbf{(100101)}_2$
3. $(4D)_{16} = 4 \times 16 + 13 = 64 + 13 = \mathbf{77}$
4. $200 \div 16 = 12$ remainder 8; $12 \div 16 = 0$ remainder 12 = C
So $200_{10} = \mathbf{(C8)}_{16}$
5. $(11110000)_2 = (1111\ 0000)_2 = \mathbf{(F0)}_{16}$
6. $(1101)_2 + (0111)_2$: Adding column by column with carries: $\mathbf{(10100)}_2 = 20$
Check: $13 + 7 = 20 \checkmark$
7. $(101110)_2 = (101\ 110)_2 = \mathbf{(56)}_8$
Check: $5 \times 8 + 6 = 46$, and $32 + 8 + 4 + 2 = 46 \checkmark$
8. Red = FF (max), Green = 00 (off), Blue = FF (max). This is **magenta** (purple/pink).
9. $(1100)_2 \text{ AND } (1010)_2 = \mathbf{(1000)}_2 = 8$
(Only position 3 has both bits = 1)
10. $(1100)_2 \text{ XOR } (1010)_2 = \mathbf{(0110)}_2 = 6$
(Positions where exactly one bit = 1)
11. $(100110)_2 = 32 + 4 + 2 = \mathbf{38}$
12. $88 = 64 + 16 + 8 = \mathbf{(1011000)}_2$
13. $(2F)_{16} = 2 \times 16 + 15 = \mathbf{47}$
14. $300 \div 16 = 18$ remainder 12 = C ; $18 \div 16 = 1$ remainder 2; $1 \div 16 = 0$ remainder 1. So $300_{10} = \mathbf{(12C)}_{16}$. Check: $256 + 32 + 12 = 300 \checkmark$
15. $(10011010)_2 = (1001\ 1010)_2 = \mathbf{(9A)}_{16}$
16. $(5C)_{16} = (0101\ 1100)_2 = \mathbf{(01011100)}_2$
17. $(101)_2 \times (101)_2 = \mathbf{(11001)}_2 = 25$. Check: $5 \times 5 = 25 \checkmark$
18. $(11011)_2 = (011\ 011)_2 = \mathbf{(33)}_8$. Check: $3 \times 8 + 3 = 27$, and $16 + 8 + 2 + 1 = 27 \checkmark$
19. Start with $+7 = (00000111)_2$. Flip: $(11111000)_2$. Add 1: $-7 = \mathbf{(11111001)}_2$
20. $(144)_b = b^2 + 4b + 4 = (b + 2)^2 = 100$. A base is an integer at least 2, and the digit 4 requires $b \geq 5$.
Thus $b + 2 = 10$, so $\mathbf{b = 8}$. Check: $(144)_8 = 64 + 32 + 4 = 100 \checkmark$

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 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
 - (b) $(01101001)_2$ to hex
 - (c) $(7B)_{16}$ to binary
 - (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 *unsigned* integer that can be stored in one byte? Express it in binary, hexadecimal, and decimal. How does this compare with the largest 8-bit two's-complement integer?
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 1 00000000?*
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 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 does not work: it is a palindrome in base 10, but $11 = (1011)_2$, which is not.)

13. **H Binary Weighing.** You have a balance scale and need to weigh any integer weight from 1 to 15 kg. If you can only place weights on one side of the scale (with the object on the other), what is the minimum number of weights you need, and what should they weigh? *Hint: Think about binary representation!*

Quick Check Answers

1. $(1100)_2 = 8 + 4 = \mathbf{12}$
2. $(10101)_2 = 16 + 4 + 1 = \mathbf{21}$
3. $19 = 16 + 2 + 1 = \mathbf{(10011)}_2$
4. $100 = 64 + 32 + 4 = \mathbf{(1100100)}_2$
5. $(1001)_2 + (0110)_2 = \mathbf{(1111)}_2 = 15$ Check: $9 + 6 = 15$ ✓
6. $(110)_2 \times (10)_2 = \mathbf{(1100)}_2 = 12$ Check: $6 \times 2 = 12$ ✓
7. $(3C)_{16} = 3 \times 16 + 12 = 48 + 12 = \mathbf{60}$
8. $255 = 15 \times 16 + 15 = \mathbf{(FF)}_{16}$
9. $(10110011)_2 = (1011\ 0011)_2 = \mathbf{(B3)}_{16}$
10. $(A5)_{16} = (1010\ 0101)_2 = \mathbf{(10100101)}_2$
11. $(110101)_2 = (110\ 101)_2 = \mathbf{(65)}_8$ Check: $6 \times 8 + 5 = 53$, and $32 + 16 + 4 + 1 = 53$ ✓
12. $(72)_8 = (111\ 010)_2 = \mathbf{(111010)}_2$
13. Red and green at maximum, no blue $\Rightarrow$ **yellow**.
14. $\#0000FF$ — pure **blue**.
15. $+3 = (00000011)_2$. Flip: $(11111100)_2$. Add 1: $-3 = \mathbf{(11111101)}_2$
16. The leftmost bit is 1, so it's negative. Flip $(11111110)_2 \rightarrow (00000001)_2$, add 1: $(00000010)_2 = 2$. So the answer is **-2**.
17. $(1111)_2$ AND $(0101)_2 = \mathbf{(0101)}_2 = 5$
18. $(1010)_2$ XOR $(1100)_2 = \mathbf{(0110)}_2 = 6$