SECTION 3.3.1

Number bases

Decimal, binary and hexadecimal — the three counting systems every computer scientist must read fluently.

Decimal · Binary · Hexadecimal | Why computers use binary | Why hexadecimal is used

BY THE END OF THIS TOPIC YOU WILL BE ABLE TO

- State the base of the decimal, binary and hexadecimal number systems.
- Explain why computers represent all data and instructions in binary.
- Recognise that one bit pattern may stand for text, an image, sound or an integer.
- Explain clearly why hexadecimal is so widely used in computer science.

A **number base** (or radix) is simply how many different digit symbols a counting system uses. Humans count in tens; computers are built to count in twos. Hexadecimal sits between the two as a convenient shorthand. Reading all three fluently is the foundation for the entire rest of this unit.

Decimal — base 10

Decimal is the everyday system you have used since primary school. It has **ten** digit symbols, `0`–`9`, and each column is worth ten times the column to its right. We say decimal is **base 10**. The number 4 `072` means:

1000	100	10	1
4	0	7	2

That is (4 × 1000) + (0 × 100) + (7 × 10) + (2 × 1) = 4072. The column values are the *powers of ten*: 10^0, 10^1, 10^2, 10^3.

Binary — base 2

Binary uses only **two** digit symbols, `0` and `1`. It is **base 2**, so each column is worth twice the column to its right. A single binary digit is called a **bit**. Eight bits make a byte, and the eight column values are the powers of two:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

The 8-bit place values. Memorise these in order — you will use them constantly in 3.3.2 and 3.3.4.

Hexadecimal — base 16

Hexadecimal (often shortened to *hex*) uses **sixteen** digit symbols. The digits `0`–`9` behave as usual, then the letters `A`–`F` stand for the values ten to fifteen. It is **base 16**, so each column is worth sixteen times the one to its right.

WORKED EXAMPLE — CONVERT 156 TO BINARY

1. Does 128 fit into 156? Yes → write **1**; 156 − 128 = 28 remaining.
2. 64 into 28? No → **0**. 32 into 28? No → **0**.
3. 16 into 28? Yes → **1**; 28 − 16 = 12. 8 into 12? Yes → **1**; 12 − 8 = 4.
4. 4 into 4? Yes → **1**; 4 − 4 = 0. 2 → **0**; 1 → **0**.

128	64	32	16	8	4	2	1
1	0	0	1	1	1	0	0

156 = `1001 1100`

EXAM TIP

Always check by converting straight back: 128 + 16 + 8 + 4 = 156. ✔ A 30-second check catches almost every slip. Remember the boundary value — the largest 8-bit number is `1111 1111` = 255, so any decimal answer above 255 means you have made an error.

Binary ↔ hexadecimal

This is the easiest pair because four bits = one hex digit. Split the byte into two **nibbles** (groups of four) and convert each using the table on the previous page.

BINARY → HEX: 1011 0110

1. Left nibble `1011` = 8+2+1 = 11 = **B**
2. Right nibble `0110` = 4+2 = 6 = **6**

1011 0110 = B6

HEX → BINARY: 2F

1. **2** = `0010`
2. **F** = 15 = `1111`

2F = `0010 1111`

Decimal ↔ hexadecimal

You can always route through binary, but the direct method is quicker. To go **decimal → hex**, divide by 16: the quotient is the first hex digit and the remainder is the second. To go **hex → decimal**, multiply the first digit by 16 and add the second.

DECIMAL → HEX: 156

1. 156 ÷ 16 = 9 remainder 12.
2. Quotient 9 → **9**. Remainder 12 → **C**.

156 = 9C

HEX → DECIMAL: B6

1. **B** = 11, so 11 × 16 = 176.
2. Add the second digit: 176 + 6.

B6 = 182

Exam-style questions — 3.3.8

1 State **two** benefits of compressing a file before sending it over the internet. **2 marks**

2 A black-and-white image contains the 16-pixel row `1111000011110000`, where 1 = black and 0 = white.

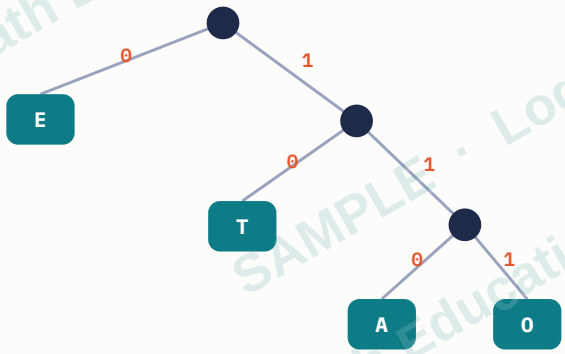
(a) Encode this row using run length encoding, writing your answer as frequency/data pairs. **2 marks**

RLE:

(b) Explain why run length encoding is well suited to images of this kind. **2 marks**

(c) Give an example of a row for which RLE would **not** reduce the size, and explain why. **2 marks**

3 The Huffman tree below is used to encode a message. Left branches are 0 and right branches are 1.



(a) Decode the bit sequence `0 10 110` using the tree. **2 marks**

Decoded:

(b) Using the tree, write the Huffman code for the word TOE. **2 marks**

Code:

(c) The message TEA is to be sent. Calculate how many bits are saved by using this Huffman code instead of 7-bit ASCII. **3 marks**

WORKING

MARK SCHEME · 1 OF 3

Knowledge Checks

Award one mark per correct point up to the maximum shown, unless stated otherwise.

3.3.1 Number bases

Q	Answer	Marks
1	(i) base 10 / ten; (ii) base 2 / two; (iii) base 16 / sixteen.	3
2	A bit. *(accept: binary digit)*	1
3	Any two: electronic components/transistors have two stable states (on/off, high/low voltage); only two states need to be told apart, so circuits are simpler / cheaper / more reliable / less error-prone.	2
4	Four bits.	1
5	Any three: more compact than binary; easier for humans to read/write; fewer errors when copying; quick to convert to/from binary (one digit = four bits); used in memory addresses / colour codes / MAC addresses / machine code.	3

3.3.2 Converting between number bases

Q	Answer	Marks
1	`93` *(64+16+8+4+1)*	1
2	`1100 1001` *(128+64+8+1 = 201)*	1
3	`122` *(7×16 + 10)*	1
4	`F4` *(244 ÷ 16 = 15 r 4)*	1
5	`E5` *(1110 = E, 0101 = 5)*	1

3.3.3 Units of information

Q	Answer	Marks
1	8 bits.	1
2	`b` = bit; `B` = byte.	1
3	1,000,000 bytes *(10^6 B / 1,000 kB)*	1
4	6 MB.	1
5	16,000 bits *(2,000 bytes × 8)*	1

3.3.4 Binary arithmetic

Q	Answer	Marks
1	Write 1 and carry 1.	1
2	`0111 0011` *(45 + 70 = 115)*	1
3	`0010 1000` = 40 *(5×2^3)*	1
4	`0001 1100` = 28 *(57 ÷ 2, remainder lost)*	1