Name: _____

Class: _____

Mr. Rawson • GCSE Computer Science

Year 10 2026

Data Representation

Number Bases & Units of Information

OCR J277 • 1.2.3 Units • 1.2.4 Data Storage — Numbers

```
01000011 01010011
0x43 0x53     67 83
```

In this booklet: bits, nibbles and bytes
kB • MB • GB • TB • PB
denary, binary and hexadecimal
converting between all three bases

Why must computers use binary?

A computer is built from millions of tiny electrical switches. A switch has no in-between: it is either **on** or **off**. We label on as **1** and off as **0**, and every one of these single on/off values is called a **bit** — short for *binary digit*.

Two states are easy to tell apart, even when a signal is weak or noisy, which is why binary is so reliable. With ten different voltage levels (one per denary digit) a slightly faded signal could easily be misread as its neighbour; with two levels it almost never is.

On its own, one bit says very little. Joined together, bits form a **bit pattern**, and bit patterns can represent *anything*: numbers, text, images, sound and program instructions are all stored the same way — as patterns of 1s and 0s.

Key concept

Everything in a computer — this booklet included — is stored as patterns of 1s and 0s. Data must be converted into binary before a computer can process it.

What you will learn

- why data must be stored in a binary format
- the units of data storage: bit, nibble, byte, kB, MB, GB, TB, PB
- how to calculate data capacity and storage requirements
- the three number bases: **denary** (base 10), **binary** (base 2) and **hexadecimal** (base 16)
- how to convert between all three, in every direction

Exam limits

In the exam the largest values you will be asked to convert are: denary **255** • binary **1111 1111** • hexadecimal **FF**. *No calculators allowed* — every method in this booklet works on paper.

Warm-up. Which of these could be binary numbers? Circle them.

`1011` `2001` `111111` `10A1` `0000` `1234`

Bits, nibbles and bytes

A **bit** is the smallest unit of data a computer can use — a single 1 or 0. A **byte** is a group of **8 bits**, and it is the smallest bit pattern normally used to represent data. Half a byte — **4 bits** — is called a **nibble**.

$\underbrace{1\,0\,1\,0}_{\text{nibble}}\ \underbrace{0\,0\,1\,0}_{\text{nibble}}$ 1 byte = 8 bits = 2 nibbles

Key facts

- Abbreviations: **b** = bit, **B** = byte. Case matters!
- 1 nibble gives **16** possible values — 0 to 15
- 1 byte gives **256** possible values — 0 to 255

Worked example

8 b = 8 bits = 1 B = 1 byte 3 B = 3 bytes = 24 bits 5 b = 5 bits

Practice 3.1 — How many *bits* are there in:

a) 2 bytes _____ b) 3 bytes _____ c) 7 bytes _____ d) 10 bytes _____

Practice 3.2 — How many *bytes* are there in:

a) 32 bits _____ b) 72 bits _____ c) 96 bits _____ d) 160 bits _____

Practice 3.3 — How many *nibbles* are there in 3 bytes? _____

Practice 3.4 — A byte holds 256 different values but a nibble only 16. Why does adding 4 more bits multiply the number of values by 16?

Quantities of bytes

Files are far bigger than single bytes, so we measure them with the International System of Units, known as **SI** units. Each step up the ladder multiplies by 1,000:

1 **kilo**byte =	1 kB =	1,000 B =	1,000 bytes =	10^3 bytes
1 **mega**byte =	1 MB =	1,000 kB =	1,000,000 bytes =	10^6 bytes
1 **giga**byte =	1 GB =	1,000 MB =	1,000,000,000 bytes =	10^9 bytes
1 **tera**byte =	1 TB =	1,000 GB =	1,000,000,000,000 bytes =	10^{12} bytes
1 **peta**byte =	1 PB =	1,000 TB =	1,000,000,000,000,000 bytes =	10^{15} bytes

Maths Link – kilos

The prefixes are the same ones you already use in maths and science:

1 **kilo**gram = 1 kg = 1,000 grams

1 **kilo**metre = 1 km = 1,000 metres

Practice 4.1 — Convert the following:

a) 8 kB = _____ bytes

b) 23 MB = _____ bytes

c) 2 MB = _____ kB

d) 8 GB = _____ kB

e) 42 TB = _____ GB

f) 0.5 TB = _____ GB

Practice 4.2 — Write these units in order, smallest to largest:

`MB` `nibble` `GB` `bit` `PB` `byte` `kB` `TB`

Practice 4.3 — A 64 GB memory card stores photos of 4 MB each. How many photos will it hold? Show your working.

Denary — base 10

A **number base** says how many different digits a number system uses. **Denary** (base 10, also called decimal) is the system you have used all your life. It has ten digits:

`0 1 2 3 4 5 6 7 8 9`

Once we run out of digits we add a new column to the left, and each column is worth **ten times** the one before — units, tens, hundreds, thousands:

Thousands	Hundreds	Tens	Units
10^3	10^2	10^1	10^0
1000	100	10	1

Worked example

Take **3,512**. Put each digit under its column heading and add up what each is worth:

Thousands	Hundreds	Tens	Units
3	**5**	**1**	**2**

$3{,}000 + 500 + 10 + 2 = 3{,}512$

You never think about this — that is the point. Binary and hexadecimal work by *exactly the same rule*; only the size of the columns changes.

Practice 5.1 — Break these denary numbers into their column values, as in the worked example:

	Thousands	Hundreds	Tens	Units
a) 4,807				
b) 926				
c) 5,237				

a) $4{,}807 =$ _____

b) $926 =$ _____

Binary — base 2

Binary (base 2) is the same idea with only **two** digits:

`0 1`

Because there are only two digits, each column is worth **twice** the one before — units, twos, fours, eights:

Eights	Fours	Twos	Units
2^3	2^2	2^1	2^0
8	4	2	1

Worked example

Take the binary number **1011**. Put each bit under its column and add up the columns that contain a 1:

Eights	Fours	Twos	Units
1	**0**	**1**	**1**

$8 + 0 + 2 + 1 = 11$ so `1011` in binary = **11** in denary.

Maths Link – powers of 2

The column values are the powers of 2: $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $2^5 = 32$, $2^6 = 64$, $2^7 = 128$. Doubling each time — learn this line and you can rebuild any conversion table from memory.

Practice 6.1 — Convert these nibbles to denary using the 8–4–2–1 columns:

a) `1001` = _____ b) `1111` = _____ c) `0110` = _____ d) `1100` = _____

Practice 6.2 — What is the largest denary value a nibble can hold? _____ And the smallest? _____

Binary → denary

For a full byte we extend the column headings to eight places. This is the **binary line** — write it out as soon as your exam starts:

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 method: write the binary number under the line, one bit per column, then add up the value of every column that holds a 1.

Worked example

Convert `1101 1011` to denary.

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

$128 + 64 + 16 + 8 + 2 + 1 = \mathbf{219}$

Practice 7.1 — Convert to denary. Use the empty rows to write the binary line and the bits underneath.

Binary	Working	Denary
a) `1110`		
b) `101010`		
c) `1100 1100`		
d) `0101 0101`		
e) `1000 0001`		
f) `1111 1111`		

Practice 7.2 — Explain why `1111 1111` is the largest number the exam will ever ask you to convert.

Denary → binary

To go the other way, **repeatedly divide by 2** and keep the remainders. Each remainder is one bit of the answer.

Worked example

Convert **156** to binary.

$$\begin{aligned} 156 \div 2 &= 78 \quad \text{remainder } \mathbf{0} \\ 78 \div 2 &= 39 \quad \text{remainder } \mathbf{0} \\ 39 \div 2 &= 19 \quad \text{remainder } \mathbf{1} \\ 19 \div 2 &= 9 \quad \text{remainder } \mathbf{1} \\ 9 \div 2 &= 4 \quad \text{remainder } \mathbf{1} \\ 4 \div 2 &= 2 \quad \text{remainder } \mathbf{0} \\ 2 \div 2 &= 1 \quad \text{remainder } \mathbf{0} \\ 1 \div 2 &= 0 \quad \text{remainder } \mathbf{1} \end{aligned}$$

Stop when the answer reaches 0, then read the remainders **from bottom to top**: $156 =$ `1001 1100`

Check your answer

Convert back the other way: $128 + 16 + 8 + 4 = 156$. ✓ A conversion you can check both ways is a conversion you cannot get wrong.

Practice 8.1 — Convert to binary. Show the division ladder.

a) 21

b) 45

c) 164

d) 230

Hexadecimal — base 16

Hexadecimal (base 16, or just *hex*) uses **sixteen** digits. After 9 we run out of ordinary digits, so the letters A to F stand for the values ten to fifteen:

`0 1 2 3 4 5 6 7 8 9 A B C D E F`

Why does hex exist? Long bit patterns are hard for humans to read, remember and copy without mistakes. Hexadecimal is a compact, human-friendly shorthand for binary:

The nibble link

One hex digit represents exactly **one nibble** (4 bits), because both have 16 possible values. Two hex digits represent exactly **one byte**. That is the entire reason hexadecimal is used in computing.

Denary	Binary	Hex	Denary	Binary	Hex
0	`0000`	0	8	`1000`	8
1	`0001`	1	9	`1001`	9
2	`0010`	2	10	`1010`	A
3	`0011`	3	11	`1011`	B
4	`0100`	4	12	`1100`	C
5	`0101`	5	13	`1101`	D
6	`0110`	6	14	`1110`	E
7	`0111`	7	15	`1111`	F

You have already met hex in the wild: colour codes like `#FF6600` are three bytes — red, green and blue — each written as two hex digits.

Practice 9.1 — Fill in the missing values:

a) denary 11 = hex _____ b) hex E = denary _____ c) binary `0111` = hex _____

d) hex C = binary _____ e) denary 15 = hex _____ f) binary `1010` = hex _____

Hex place value

Hexadecimal follows the same place-value rule as denary and binary — but each column is worth **sixteen times** the one before:

Four-thousand-and-ninety-sixes	Two-hundred-and-fifty-sixes	Sixteens	Units
16^3	16^2	16^1	16^0
4,096	256	16	1

In the exam you only meet **two-digit** hex numbers (up to FF), so the columns you will actually use are the **sixteens** and the **units**.

Worked example

Convert **8B** to denary.

	Sixteens	Units
Hex digit	**8**	**B**
As denary	8	11

$(8 \times 16) + (11 \times 1) = 128 + 11 =$ **139**

Maths Link – your 16 times table

You already know it — it is the 8 times table, doubled: $4 \times 8 = 32$ so $4 \times 16 = 64$.

1×16	2×16	3×16	4×16	5×16	6×16	7×16	8×16
16	32	48	64	80	96	112	128
9×16	10×16	11×16	12×16	13×16	14×16	15×16	16×16
144	160	176	192	208	224	240	256

Practice 10.1 — Use the sixteens-and-units columns to convert:

a) 4F = _____ b) 26 = _____ c) 3C = _____

Binary ↔ hexadecimal — the nibble trick

Because one hex digit *is* one nibble, converting between binary and hex never needs any arithmetic bigger than 8 + 4 + 2 + 1.

Binary → hex: split the bit pattern into nibbles (from the right — add 0s to the front if needed), convert each nibble to denary with 8–4–2–1, then write each as a hex digit.

Worked example

Convert `1101 1001` to hexadecimal.
First nibble: `1101` $= 8 + 4 + 1 = 13 =$ **D** Second nibble: `1001` $= 8 + 1 = 9 =$ **9**
`1101 1001` = **D9**

Hex → binary: the same trick in reverse — turn each hex digit into its 4-bit nibble.

Worked example

Convert **F2** to binary.
F $= 15 = 8 + 4 + 2 + 1 =$ `1111` 2 $= 0 + 0 + 2 + 0 =$ `0010`
F2 = `1111 0010`

Practice 11.1 — Binary to hexadecimal:

a) `1001` = _____ b) `1111` = _____ c) `0110 1110` = _____ d) `0101 0111` = _____

Practice 11.2 — Hexadecimal to binary:

a) 5C = _____ b) 3E = _____ c) A4 = _____ d) E7 = _____

Practice 11.3 — A pixel's red value is stored as the byte `1010 0010`. Write it as two hex digits.

_____

Denary → hexadecimal

Divide by powers of 16, largest first, keeping whole numbers only. Each answer is one hex digit; each remainder feeds the next step.

Worked example

Convert **495** to hexadecimal.
Step 1 — the largest power of 16 below 495 is 256.
$495 \div 256 = 1$ (ignore the decimals) → first digit **1**
remainder: $495 - (1 \times 256) = 239$
Step 2 — divide the remainder by 16.
$239 \div 16 = 14$ → second digit 14 = **E**
remainder: $239 - (14 \times 16) = 239 - 224 = 15$
Step 3 — divide the remainder by 1.
$15 \div 1 = 15$ → third digit 15 = **F**
495 = **1EF**

For exam-sized numbers (up to 255) there is only one division:

Worked example

Convert **235** to hexadecimal.
$235 \div 16 = 14$ remainder 11 14 = E and 11 = B so 235 = **EB**

Practice 12.1 — Convert to hexadecimal. Show your divisions.

Denary	Working	Hex
a) 35		
b) 176		
c) 201		
d) 255		

Stretch — beyond the exam limit: convert 300 to hex. _____

Hexadecimal → denary

Multiply each digit by its column value and add. Letters go through their denary values first: A = 10, B = 11, C = 12, D = 13, E = 14, F = 15.

Worked example

Convert **B7** to denary.

$$\begin{aligned} &\text{units digit: } 7 \times 16^0 = 7 \times 1 = 7 \\ &\text{sixteens digit: } \mathrm{B} \times 16^1 = 11 \times 16 = 176 \\ \hline &\text{total: } 176 + 7 = \mathbf{183} \end{aligned}$$

Check both ways

Reverse it: $183 \div 16 = 11$ remainder 7, and $11 = \mathrm{B}$. ✓

Practice 13.1 — Convert to denary. Show your working.

Hex	Working	Denary
a) D		
b) 3C		
c) 5D		
d) E2		
e) F3		

Stretch — convert 1EF back to denary. Do you get 495?

Mixed practice

No calculators — just like the exam. Write the binary line before you start.

Exam question [3 marks]

Convert the following binary numbers to denary:

a) `1111` = _____ b) `0101 0101` = _____ c) `1010 1010` = _____

Exam question [3 marks]

Convert the following denary numbers to binary:

a) 7 = _____ b) 35 = _____ c) 182 = _____

Exam question [2 marks]

Convert the following hexadecimal numbers to denary:

a) C = _____ b) 5D = _____

Exam question [2 marks]

Convert the following denary numbers to hexadecimal:

a) 47 = _____ b) 199 = _____

Exam question [2 marks]

a) Convert `0111 1101` to hexadecimal: _____

b) Convert 4F to binary: _____

Exam question [2 marks]

Explain why computers store data in binary.

_____

_____

Exam question [2 marks]

Explain why programmers often use hexadecimal to represent binary values.

_____

_____

Adding binary numbers

You can add two binary numbers straight in binary, without converting to denary first. It works exactly like column addition in denary — start at the right, carry to the left — but there are only **four rules** to learn:

Key concept

`0 + 0 = 0` `0 + 1 = 1` `1 + 1 = 10` `1 + 1 + 1 = 11`

The last two are the only ones that carry. $1+1$ writes **0** and carries **1**; $1+1+1$ writes **1** and carries **1**.

Worked example

Add `0011 0101` and `1000 0010`. Work right to left, writing any carry above the next column.

```
  0 0 1 1 0 1 0 1
+ 1 0 0 0 0 0 1 0
-----------------
  1 0 1 1 0 1 1 1
```

Check: $53 + 130 = 183$, and `1011 0111` is 183. ✓

Practice 15.1 — Add these binary numbers. Show your carries.

```
  0 0 1 1        0 0 0 1        1 1 0 1 0 0 1 0
+ 0 1 0 0      + 1 0 0 0      + 0 0 1 0 1 1 0 1
---------      ---------      -----------------
```

Overflow

A byte holds eight bits, and the largest value eight bits can hold is **255**. If an addition needs a ninth bit, that bit has nowhere to go — it is simply lost. This is an **overflow error**.

Worked example

Add `0111 0001` and `1001 1110`. $113 + 158 = 271$.

```
  0 1 1 1 0 0 0 1
+ 1 0 0 1 1 1 1 0
-----------------
1 0 0 0 0 1 1 1 1
```

The answer needs **nine** bits. Only eight are stored, so the computer keeps `0000 1111` — which is 15, not 271. The bold ninth bit has fallen off the end. **That is an overflow error.**

Binary shifts

Sliding every bit one place to the **left** doubles the number. Sliding every bit one place to the **right** halves it. That is all a **binary shift** is — and it is how computers multiply and divide quickly.

Key concept

Left shift multiplies by 2 for every place moved
Right shift divides by 2 for every place moved
So 1 place $= \times 2$, 2 places $= \times 4$, 3 places $= \times 8$, 4 places $= \times 16$. Gaps left behind are filled with **0**.

Worked example

`0001 0010` $\times 8$. Multiplying by 8 is a **left shift of 3 places**.

`0001 0010` $\longrightarrow$ `1001 0000`

Check: $18 \times 8 = 144$, and `1001 0000` is 144. ✓

Practice 16.1 — Use a left shift. Give each answer in 8 bits.

a) `0101` $\times 2$ = _____ b) `1100` $\times 8$ = _____

c) `0101 1110` $\times 2$ = _____

Practice 16.2 — Use a right shift. Give each answer in 8 bits.

a) `0110` $\div 2$ = _____ b) `0011 1000` $\div 8$ = _____

c) `0101 1110` $\div 2$ = _____

Key concept

What falls off the end is gone. Shift `0111 1101` (125) left by 2 to multiply by 4 and you need 500 — far past 255, so bits drop off the left and the answer is wrong: that is **overflow** again. Shift `0001 0011` (19) right by 1 to halve it and you get `0000 1001` (9), not 9.5 — the bit that fell off the right was the half, and binary shifts cannot keep it. **Shifting loses information at both ends.**

Practice 16.3 — `1011 1011` $\times 8$. Explain what goes wrong.

Reference — the cheat sheet

The three place-value lines. Same rule every time: multiply each digit by its column, add them up.

Binary (base 2)	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
Denary (base 10)					10^3	10^2	10^1	10^0
					1,000	100	10	1
Hex (base 16)					16^3	16^2	16^1	16^0
					4,096	256	16	1

The sixteen hex digits and their nibbles.

Denary	Binary	Hex	Denary	Binary	Hex
0	`0000`	0	8	`1000`	8
1	`0001`	1	9	`1001`	9
2	`0010`	2	10	`1010`	A
3	`0011`	3	11	`1011`	B
4	`0100`	4	12	`1100`	C
5	`0101`	5	13	`1101`	D
6	`0110`	6	14	`1110`	E
7	`0111`	7	15	`1111`	F

Units of storage. *Kids Must Go To Parties* — kB, MB, GB, TB, PB.

8 bits	= 1 byte	1,000 GB	= 1 TB
4 bits	= 1 nibble	1,000 MB	= 1 GB
1,000 bytes	= 1 kB	1,000 kB	= 1 MB
1,000 TB	= 1 PB	1 byte	= 256 values (0–255)

The four methods.

- **binary → denary:** binary line, add the 1-columns
- **denary → binary:** divide by 2, read remainders bottom-up
- **binary ↔ hex:** one nibble = one hex digit
- **denary ↔ hex:** multiply by 16s, or divide by 16 and keep the remainder

Glossary

bit	a single binary digit, 1 or 0 — the smallest unit of data
bit pattern	a group of bits joined together to represent data or instructions
nibble	4 bits — half a byte; holds 16 values, and matches one hex digit
byte	8 bits — the smallest bit pattern normally used to represent data; holds 256 values
number base	how many different digits a number system uses
denary	base 10 — digits 0–9; also called decimal
binary	base 2 — digits 0 and 1; the form all computer data takes
hexadecimal	base 16 — digits 0–9 and A–F; shorthand for binary, one digit per nibble
SI units	the international system of measurement prefixes: kilo, mega, giga, tera, peta
kilobyte (kB)	1,000 bytes (10^3); then MB, GB, TB, PB — each 1,000 times the last

Can I …?

Tick a column for each skill. Be honest — the point is to find what to practise before the end-of-unit test, not to collect ticks.

	Not yet	Getting there	Confident
Explain **why** data must be stored in binary			
Name the units of storage in order: bit, nibble, byte, kB, MB, GB, TB, PB			
Calculate data capacity and storage requirements			
Convert **binary** → **denary** (add the 1-columns)			
Convert **denary** → **binary** (divide by 2, remainders bottom-up)			
Convert **binary** ↔ **hex** (one nibble = one hex digit)			
Convert **denary** ↔ **hex** (multiply by 16s, or divide by 16)			
Add two 8-bit binary numbers, and say what **overflow** means			
Perform a **binary shift** and say what it does to the value			

The one thing I most need to practise:

Notes