Cambridge IGCSE · Computer Science 0478 · Notes

Number systems

Read online at **pastopic.com/cambridge/computer-science-0478/notes/number-systems**

Last checked 30 September 2026

What you need to know

By the end of this topic you should be able to:

1. **Explain why computers use binary.** Every kind of data (numbers, text, sound, images) has to be turned into binary before a computer can process it, because the computer is built from logic gates and stores values in registers, and these only work with two states.
2. **Describe the three number systems:** denary is base 10, binary is base 2 and hexadecimal is base 16.
3. **Convert positive whole numbers** in both directions between denary and binary, denary and hexadecimal, and hexadecimal and binary. Binary numbers are at most 16 bits long.
4. **Explain how and why hexadecimal is used:** say where it appears in computer science, and why it helps people (it is a shorter way of writing binary, so it is easier to read).
5. **Add two positive 8-bit binary integers**, and explain **overflow**: what it is and why it happens (for example, a result bigger than 255 in an 8-bit register).
6. **Carry out logical shifts** on a positive 8-bit binary integer, left or right, by one or more places, and say what the shift does to the number (multiplies or divides it) and which bits are lost.
7. **Use two's complement** to store positive and negative whole numbers in 8 bits: convert a positive or negative denary or binary number into 8-bit two's complement, and back again.

Fractions, decimal points and negative numbers in hexadecimal are **not** in the syllabus: every value you convert is a whole number.

Understand it

Why computers use binary

A computer is made of millions of tiny switches (transistors) wired into **logic gates**. Each switch is either on or off, so it can only hold two values, written **1** and **0**. A number system with just two digits, **binary**, matches this exactly. Anything a computer handles, whether a number, a letter, a pixel's colour or a sample of sound, is stored as a pattern of 1s and 0s in **registers** (small stores inside the processor) and in memory. One binary digit is a **bit**; 4 bits are a **nibble**; 8 bits are a **byte**.

Place value in three bases

In **denary** (base 10) each column is worth 10 times the one to its right: 1, 10, 100, 1000. The digits are 0 to 9.

In **binary** (base 2) each column is worth **2 times** the one to its right. The digits are only 0 and 1. For an 8-bit register the column values are:

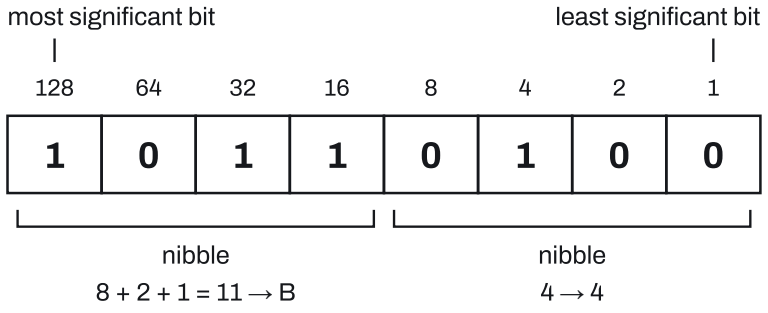
128	64	32	16	8	4	2	1

For 16 bits carry on doubling to the left: 256, 512, 1024, 2048, 4096, 8192, 16384, 32768.

In **hexadecimal** (base 16) each column is worth 16 times the one to its right: 1, 16, 256. It needs sixteen digits, so after 0–9 it uses letters:

Denary	10	11	12	13	14	15
Hexadecimal	A	B	C	D	E	F

One hexadecimal digit (0 to F, that is 0 to 15) holds exactly the same range as **4 bits** (0000 to 1111). That is why hexadecimal and binary convert so easily: each hex digit is one nibble.



Denary: 128 + 32 + 16 + 4 = 180

Hexadecimal: B4

The leftmost bit has the biggest value: it is the **most significant bit** (MSB). The rightmost bit, worth 1, is the **least significant bit** (LSB).

Converting between binary and denary

Binary to denary: write the place values over the bits and add the values that have a 1 under them. For 10110100: $128 + 32 + 16 + 4 = 180$.

Denary to binary: work from the largest place value down. If the place value fits into what is left, write 1 and subtract it; if not, write 0. For 75: 128 doesn't fit (0); 64 fits, leaving 11 (1); 32 and 16 don't fit (0, 0); 8 fits, leaving 3 (1); 4 doesn't fit (0); 2 fits, leaving 1 (1); 1 fits (1). So $75 = 64 + 8 + 2 + 1 = 01001011$.

Another way is to divide by 2 again and again and write down the remainders: 75 gives remainders 1, 1, 0, 1, 0, 0, 1. Read them **from the last to the first** to get 1001011, then add a leading 0 to make 8 bits.

Converting with hexadecimal

Binary to hexadecimal: split the binary into nibbles **starting from the right**, then turn each nibble into one hex digit. 1011 0100 is 11 and 4, so B4. If the number of bits isn't a multiple of 4, add zeros on the left: 101101101 becomes 0001 0110 1101, which is 16D.

Hexadecimal to binary: write each hex digit as its own 4-bit nibble and keep the zeros. 3E is 0011 1110; 16D is 0001 0110 1101.

Hexadecimal to denary: multiply each digit by its place value (256, 16, 1) and add. $2A7 = 2 \times 256 + 10 \times 16 + 7 = 679$. You can also go through binary.

Denary to hexadecimal: divide by 16. The whole-number answer is the "sixteens" digit and the remainder is the "ones" digit. $200 \div 16 = 12$ remainder 8, and 12 is C, so 200 is C8. For numbers of 256 or more, first see how many 256s fit. Going through binary (denary → binary → nibbles → hex) works too.

Why hexadecimal is used

The computer still stores and processes **binary**; hexadecimal is for **people**. A 16-bit value is 16 binary digits but only 4 hex digits, so hex is shorter, easier and quicker to read, remember and type, takes less space on a screen, and leads to fewer mistakes when a programmer copies or checks it. And because each hex digit is one nibble, converting back to binary is quick.

Places you will see hexadecimal:

- **HTML colour codes**, such as #2F15D6 (a red, a green and a blue value, one byte each);
- **MAC addresses**, such as 00:1A:C9:3E:77:B2, and **IPv6 addresses**;
- **memory addresses** and **memory dumps** used when debugging;
- **error codes** and status codes shown to users;
- **assembly language** and machine code listings;
- character codes (ASCII and Unicode values) and special characters in URLs.

Binary addition

Add binary columns from the right, just like denary addition, using four rules:

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 1 = 10$: write 0 and carry 1 into the next column
- $1 + 1 + 1 = 11$: write 1 and carry 1

For example, 00101011 + 00011010 (that is $43 + 26$):

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

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

The answer 01000101 is 69, and $43 + 26 = 69$, which checks it.

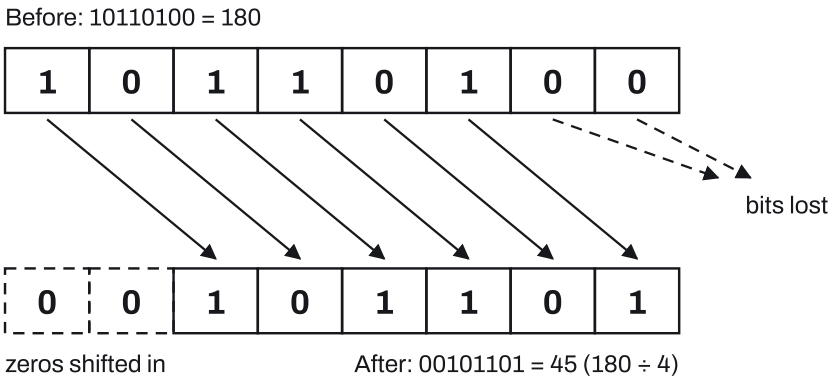
Overflow

A register has a fixed number of bits, so it has a largest value. With n bits the largest value is $2^n - 1$: 255 for 8 bits (11111111), 4095 for 12 bits (FFF in hexadecimal) and 65 535 for 16 bits. The smallest is 0.

If an addition gives a result bigger than the register can hold, the final column produces a carry that has **no ninth bit to go into**. This is an **overflow error**: the result is too large to be stored in the number of bits available, so the value left in the register is wrong.

Logical shifts

A **logical shift** moves every bit in the register the same number of places left or right. Bits pushed off the end are **lost**, and the empty places at the other end are filled with **0s**.



Each place to the **left** doubles the value and each place to the **right** halves it, just as moving denary digits one place multiplies or divides by 10:

- a **left** shift of n places **multiplies** by 2^n (1 place ×2, 2 places ×4, 3 places ×8). 00101101 (45) shifted left 1 is 01011010 (90). The **most significant** bits are lost, so if a 1 falls off the left the answer is wrong: 11000011 (195) shifted left 1 is 10000110 (134), not 390.
- a **right** shift of n places **divides** by 2^n (1 place ÷2, 2 places ÷4, 4 places ÷16). The **least significant** bits are lost, so any remainder is thrown away: 00001011 (11) shifted right 1 is 00000101 (5).

Two's complement

Plain binary can only store positive numbers. **Two's complement** lets an 8-bit register store negative numbers too, by making the most significant bit worth **−128** instead of +128:

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

- If the MSB is **0**, the number is positive and reads exactly like normal binary: +37 is 00100101.
- If the MSB is **1**, the number is negative: 11101100 is $-128 + 64 + 32 + 8 + 4 = -20$.

So 8-bit two's complement stores −128 (10000000) up to +127 (01111111); 11111111 is −1.

Denary to two's complement (negative): write the positive number in 8-bit binary, **flip** every bit ($0 \leftrightarrow 1$), then **add 1**. For −20: 20 is 00010100; flipped, 11101011; add 1, 11101100.

Two's complement to denary: if the MSB is 0, read it as normal binary. If the MSB is 1, either add the place values with −128 for the MSB, or flip every bit, add 1, read the result as a positive number and put a **minus sign** in front.

Key facts and formulas

There is no formula list for this subject, so you must learn all of these.

Formula	Status
Denary base 10 (digits 0–9); binary base 2 (0 and 1); hexadecimal base 16 (0–9 and A–F)	Learn it
8-bit place values: 128, 64, 32, 16, 8, 4, 2, 1 (keep doubling for 12 or 16 bits)	Learn it
Hex place values: 256, 16, 1; A = 10, B = 11, C = 12, D = 13, E = 14, F = 15	Learn it
1 hex digit = 4 bits = 1 nibble; 8 bits = 1 byte = 2 hex digits	Learn it
Largest value in n bits $= 2^n - 1$ (8 bits: 255 = FF; 12 bits: 4095 = FFF; 16 bits: 65 535 = FFFF); smallest = 0	Learn it
Binary addition: $0 + 0 = 0, 0 + 1 = 1, 1 + 1 = 10, 1 + 1 + 1 = 11$	Learn it
Overflow: the result is greater than the largest value the register can hold (above 255 in 8 bits)	Learn it
Logical left shift of n places $= \times 2^n$; logical right shift of n places $= \div 2^n$; lost bits are gone and 0s fill the gap	Learn it

Formula	Status
Two's complement 8-bit: MSB worth -128; range -128 to $+127$	Learn it
Negative denary → two's complement: positive in binary, flip the bits, add 1	Learn it

How to do it

In the Paper 1 questions we have tagged for this topic (39 questions, 2021–2025), the types came up this often. 37 of the 39 questions mix several types, so the counts add up to more than 39.

Question type	Questions
Convert denary to binary	19
Convert binary to hexadecimal	14
Convert binary to denary	12
Convert hexadecimal to binary	12
Add two 8-bit binary integers	11
Perform a logical shift and state its effect	9
Give reasons for, or uses of, hexadecimal	9
State number-system facts	9
Convert denary to hexadecimal	8
Convert a denary integer to 8-bit two's complement	8
Convert hexadecimal to denary	7
Convert 8-bit two's complement to denary	4
Explain why computers use binary	4
Largest value in n bits, or fewest bits for a value	4
Explain or define overflow	3

1. Convert denary to binary (19 questions)

1. Write the place values 128 64 32 16 8 4 2 1 (more to the left for 12 or 16 bits).
2. From the left, write 1 where the place value fits into what is left and subtract it; otherwise write 0.
3. Check by adding the place values of your 1s back up.
4. Give the number of bits asked for. "8-bit" or a register of a given size means **pad with leading 0s** (10 as 00001010); otherwise leading zeros are optional.

Variations you will meet:

- **"Show all your working"**: write the sum of place values (such as $128 + 32 + 8 + 4 + 2 + 1$) as well as the answer; one mark is for that working.
- **Matching** denary values to 8-bit patterns with lines: convert each one; don't guess from the look of the pattern.
- **Context**: an ASCII code, an IP address byte, a price, a count in a 12-bit or 16-bit register. The method is the same.

2. Convert binary to hexadecimal (14 questions)

1. Split the bits into nibbles **from the right**. Pad the leftmost group with 0s if it is short.
2. Turn each nibble into one digit (0–9 or A–F) and write them in the same order.
3. Keep a leading 0 digit if the question's format shows one (a 12-bit register gives 3 hex digits, so 000010100001 is 0A1).

For 12- or 16-bit inputs the marks are usually one per correct digit in the correct order, so work nibble by nibble.

3. Convert binary to denary (12 questions)

1. Write the place values over the bits (for 12 bits start at 2048; for 16 bits at 32768).
2. Add the values that have a 1.
3. For long numbers you can go through hex instead: 000110011011 is 19B, and $1 \times 256 + 9 \times 16 + 11 = 411$.

Variation: two registers that each hold part of a value (for example the dollars and the cents of a price): convert each register separately, then put the parts together as the question says.

4. Convert hexadecimal to binary (12 questions)

1. Write each hex digit as a **4-bit** nibble, keeping its leading zeros: 5 is 0101, not 101.
2. Write the nibbles in the same order. Three hex digits give 12 bits; two give 8 bits.
3. Don't treat a hex number that happens to use only 0–9 (such as 15 or 35) as denary: hex 15 is 0001 0101, not 1111.

5. Add two 8-bit binary integers (11 questions)

1. Write the two numbers one under the other, lined up on the right.
2. Add column by column from the right with the four rules, writing each **carry above** the next column.
3. If the leftmost column produces a carry, write it as a **ninth bit** on the left and say that it is an **overflow**.
4. Check by converting to denary if you have time, but the answer must be done in binary.

Usually 3 marks: one for each correct nibble of the answer and one for the working (the carries). When the sum is over 255 there is a fourth mark for showing the overflow bit.

6. Perform a logical shift and state its effect (9 questions)

1. Write the 8 bits, then move every bit the stated number of places left or right.
2. Bits that go past the end are **lost**. Fill the empty places at the other end with **0s**. The answer is still 8 bits.
3. If asked for the denary value, convert the new 8 bits (show the place-value sum).

Variations you will meet:

- **Describe a logical left shift**: every bit moves one place to the left; the most significant (leftmost) bit is lost; a 0 is put in the least significant (rightmost) place.
- **State the effect**: left shift of n places multiplies by 2^n; right shift of n places divides by 2^n. Give both the operation and the number, for example "divides by 16" for a right shift of 4.
- **Which bits are lost**: the most significant bits on a left shift; the least significant bits on a right shift.

7. Give reasons for, or uses of, hexadecimal (9 questions)

1. **Reasons** must be about **people**: it is shorter than binary; it is easier or quicker to read, understand, remember and type; people make fewer mistakes with it and can spot errors (debug) more easily; it takes less space on a screen. Don't say it makes the computer faster or uses less storage: the computer still stores binary.
2. **Uses**: HTML colour codes, MAC addresses, IPv6 addresses, memory addresses and memory dumps, error codes, assembly language, ASCII/Unicode values, URLs. Give different ones from any the question already mentions.

Variation: "How could long binary error codes be shortened for a small screen?" Convert them to hexadecimal (or denary).

8. State number-system facts (9 questions)

Short recall, often as tick boxes or a paragraph with gaps:

- binary is base 2 and uses only 0 and 1; denary is base 10; hexadecimal is base 16 and uses 0–9 and A–F;
- 8 bits hold 0 to 255; each hex digit is 4 bits; 10 is A and 15 is F;
- the system for negative binary numbers is **two's complement**;
- a similarity between binary and hex: both are number systems (both can represent the same values); differences: base 2 vs base 16, and digits 0–1 vs 0–9 and A–F.

9. Convert denary to hexadecimal (8 questions)

1. Divide by 16: the quotient is the sixteens digit and the remainder is the ones digit. Turn 10–15 into A–F.
2. For 256 or more, first take out the 256s: $415 = 1 \times 256 + 159$, and $159 = 9 \times 16 + 15$, so 415 is 19F.

3. Or go through binary: convert to binary, split into nibbles, convert each.
4. Put the digits in the right order: the sixteens digit comes first (12 is 0C or just C, never "C0").

10. Convert a denary integer to 8-bit two's complement (8 questions)

1. Write the **positive** value in 8-bit binary.
2. If the number is positive, stop: that is the answer (it starts with 0).
3. If it is negative, **flip** every bit and **add 1**.
4. Check: the MSB must be 1, and $-128+$ the other place values must give your number.

Show the steps (the positive binary, the flipped bits and the +1): one of the two marks is for the method.

11. Convert hexadecimal to denary (7 questions)

1. Multiply each digit by its place value, 256, 16 and 1, remembering A = 10 ... F = 15.
2. Add. A leading 0 is worth nothing: 0AC is $10 \times 16 + 12 = 172$.

Don't read hex as denary: hex 2F is 47, not 25 or 215.

12. Convert 8-bit two's complement to denary (4 questions)

1. Look at the MSB. If it is 0, read it as normal binary.
2. If it is 1, either work out $-128+$ the other place values, or flip the bits, add 1, read the positive value and put a minus sign in front.
3. Write the minus sign in the answer. A negative two's complement number read as $+55$ instead of -55 gets no answer mark.

13. Explain why computers use binary (4 questions)

Two linked points: computers are made of **logic gates / switches** that have only **two states** (on and off), and binary also has only two values, **1 and 0**, so every item of data must be converted to binary to be processed.

14. Largest value in n bits, or fewest bits for a value (4 questions)

- **Largest value**: all bits set to 1, which is $2^n - 1$. In hexadecimal it is F repeated once per nibble (12 bits: FFF, which is 4095).
- **Fewest bits for a denary value**: find the smallest power of 2 that is **bigger** than the value; its power is the number of bits. 301 needs 9 bits (256 ≤ 301 < 512); 2000 needs 11 bits. The answer is a **number of bits**, not the binary number itself.
- **Fewest bits for a hex value**: 4 bits per hex digit, so a 3-digit code such as A4D needs 12 bits.

15. Explain or define overflow (3 questions)

Two points: the result is **too large** (greater than 255 for 8 bits), so it **cannot be stored in the number of bits available** in the register. As a tick box, choose the option that says the answer is too large for the bits available; not "too small", not "negative".

Worked examples

The bold codes show where marks are earned, as in Cambridge mark schemes. The number is how many marks.

- **M1** method mark: for a correct method, even if the numbers slip.
- **A1** accuracy mark: for a correct answer; it needs the M mark before it.
- **B1** mark for a correct result on its own, with no method needed.

Example 1

A games console stores values in registers.

(a) A register holds 10011110. Convert it to denary. **[1]**

(b) Convert 10011110 to hexadecimal. **[2]**

(c) A memory address is 2C5 in hexadecimal. Convert it to a 12-bit binary number. **[2]**

(d) Convert the denary value 190 to hexadecimal. **[1]**

(e) Convert the hexadecimal value 3B to denary. **[1]**

(a) $128 + 16 + 8 + 4 + 2 = 158$. **B1**

(b) Nibbles 1001 and 1110 are 9 and 14, so **9E**. **B1** for each correct digit in the correct order.

(c) 2 → 0010, C → 1100, 5 → 0101, so **0010 1100 0101**. **B1** for two correct nibbles, **B1** for all three (the mark scheme may instead give one mark per nibble).

(d) $190 \div 16 = 11$ remainder 14; 11 is B and 14 is E, so **BE**. **B1**

(e) $3 \times 16 + 11 = 59$. **B1**

Example 2

Two 8-bit registers hold 10110110 and 01101101.

(a) Add the two binary numbers using binary addition. Show all your working. **[4]**

(b) Explain why the addition causes an error. **[2]**

(a)

	Overflow	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
Carry	1	1	1	1	1	1			
		1	0	1	1	0	1	1	0
+		0	1	1	0	1	1	0	1
Sum	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**1**

B1 for the carries shown, **B1** for the first nibble 0010, **B1** for the second nibble 0011, **B1** for showing the ninth (overflow) bit.

(b) $182 + 109 = 291$, which is greater than 255, the largest value 8 bits can hold, **B1**, so the result cannot be stored in the 8-bit register: an overflow error. **B1** (The register is left holding 00100011, which is 35, the wrong answer.)

Example 3

A register holds the binary number 11010100.

(a) A logical right shift of two places is performed. Give the 8-bit result and its denary value. **[2]**

(b) State the effect of the shift in part (a) on the original number. **[1]**

(c) Instead, a logical left shift of one place is performed on 11010100. Give the result, and explain why it is not double the original value. **[3]**

(a) Move every bit two places right, lose the last two bits (00) and put two 0s on the left: **00110101**. **B1** In denary $32 + 16 + 4 + 1 = 53$. **B1**

(b) It divides the number by 4 ($212 \div 4 = 53$). **B1**

(c) Move every bit one place left, lose the leftmost 1 and put 0 on the right: **10101000**. **B1** The original is 212 and double would be 424, **B1** but the most significant bit (a 1) was lost off the end, because 424 is too big for 8 bits; the register holds 168. **B1**

Example 4

A freezer stores temperatures as 8-bit two's complement integers.

(a) Give the two's complement integer for –46. Show all your working. **[2]**

(b) Convert the two's complement integer 10101101 to denary. Show all your working. **[2]**

(c) Give the two's complement integer for +100. **[1]**

(a) $46 = 32 + 8 + 4 + 2$, so 00101110. Flip: 11010001. Add 1: **11010010**. **M1** for the method (positive in binary, flip, add 1), **A1** for the answer.

(b) The MSB is 1, so the number is negative. Flip: 01010010; add 1: 01010011, which is $64 + 16 + 2 + 1 = 83$. So the value is **−83**. **M1 A1**

Check with the −128 column: $-128 + 32 + 8 + 4 + 1 = -83$.

(c) Positive, so normal binary with a leading 0: $64 + 32 + 4 = 100$, **01100100**. **B1**

Common mistakes

- **Not showing the carries.** The examiner reports for June 2023, March 2025 and June 2025 note that correct additions lose the working mark because the carries aren't written. Write each carry above its column.
- **Adding in denary instead.** Converting both numbers to denary, adding, and converting back does not show binary addition, so it doesn't answer the question. Do the addition in binary.
- **Forgetting the overflow bit.** When the leftmost column carries, write the ninth bit and say "overflow"; it is a separate mark.
- **Leaving out the minus sign.** In two's complement to denary, many candidates work correctly and then write 55 instead of −55. A number whose MSB is 1 is negative.
- **Giving the positive number.** "Two's complement for −22" needs the flip and add 1; writing the binary for +22 gets no answer mark.
- **Shifting the wrong way, or filling with 1s.** Left means towards the MSB. The gaps are always filled with **0s**.
- **Hex digits in the wrong order.** 12 is 0C (sixteens digit first), not C0. And A is 10, not 11.
- **Reading hexadecimal as denary.** Hex 15 is 0001 0101 (21 in denary), not 1111.
- **Dropping leading zeros when the format shows them.** If a register or display has a fixed number of digits, give all of them (02C, 0010 1101), and don't split a hex value into separate chunks that it wasn't asked for.
- **Answering a different question.** "How many bits are needed?" wants a number such as 9, not the binary value. "Convert to hexadecimal" does not want denary.
- **Reasons for hex about the computer.** "Uses less memory" or "the computer processes it faster" are wrong: the computer stores binary either way. Reasons are about people.

How to get the marks

- **"Give" / "State"**: just the value or fact, no explanation needed. A conversion answer is marked right or wrong, so check it by converting back.
- **"Show all your working"**: part of the marks is for the method. For a conversion, write the place-value sum; for addition, the carries; for two's complement, the positive binary, the flipped bits and the +1.

- **"Convert … to 8-bit" or a register of fixed size**: give exactly that many bits, with leading zeros. Hex answers from a 12-bit register have 3 digits.
- **One mark per nibble or digit**: long conversions and additions are often marked a nibble (or a hex digit) at a time, so a slip in one nibble only costs one mark. Keep the nibbles in the right order.
- **"Explain" (why binary, overflow)**: two linked points, for example "the result is greater than 255" **and** "so it cannot be stored in 8 bits".
- **"Describe" a shift**: say how the bits move, which end loses a bit and what is put in at the other end.
- **Effect of a shift**: name the operation **and** the factor ("divides by 16").
- **"Identify two …"** uses or reasons: two different ones, not a repeat of the example in the question.
- If you do a calculation twice, **cross out** the one you don't want the examiner to mark.

Quick check

1. Convert the denary number 213 to 8-bit binary and to hexadecimal.
2. Convert the hexadecimal number 4F7 to 12-bit binary and to denary.
3. Add 01011011 and 00110110 using binary addition. Does an overflow occur?
4. A logical right shift of three places is performed on 10010110. Give the result, its denary value and the effect of the shift.
5. (a) Give the 8-bit two's complement integer for −61. (b) Convert the two's complement integer 11110000 to denary.

Answers

1. $213 = 128 + 64 + 16 + 4 + 1$, so **11010101**. Nibbles 1101 and 0101 are 13 and 5, so **D5**.
2. 4 → 0100, F → 1111, 7 → 0111, so **0100 1111 0111**. Denary: $4 \times 256 + 15 \times 16 + 7 = 1024 + 240 + 7 = 1271$.
3. Carries go into the 4, 8, 16, 32, 64 and 128 columns; the sum is **10010001** ($91 + 54 = 145$). The 128 column produces no carry, and $145 \leqslant 255$, so **no overflow**.
4. **00010010**, which is $16 + 2 = 18$. A right shift of 3 divides by $2^3 = 8$: $150 \div 8 = 18.75$, and the lost bits (110) take the remainder with them, leaving 18.
5. (a) $61 = 00111101$; flip, 11000010; add 1, **11000011**. (b) MSB is 1, so negative: $-128 + 64 + 32 + 16 = -16$ (or flip, 00001111, add 1, 00010000 = 16, so **−16**).

Past questions to try

- **0478/12/O/N/25 Q1**: hexadecimal codes in a 12-bit register, the largest value and conversions both ways.
- **0478/12/O/N/23 Q2**: denary, hex and a left shift from one register, then −99 in two's complement and an addition with overflow.
- **0478/13/O/N/24 Q1**: showing working in a denary-to-binary conversion, hex with leading zeros, addition with overflow and two's complement to denary.
- **0478/11/M/J/25 Q2**: ticket numbers converted between hexadecimal, binary and denary, and why hexadecimal is displayed.