Cambridge IGCSE · Computer Science 0478 · Notes

Methods of error detection

Read online at **pastopic.com/cambridge/computer-science-0478/notes/error-detection**

Last checked 30 September 2026

Read first: **Number systems**, **Types and methods of data transmission**

What you need to know

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

1. **Explain why data must be checked for errors after it has been transmitted, and how errors happen.** Interference on the way can make data **lost**, **gained** or **changed**.
2. **Describe how each of these methods detects errors after transmission:**
   - the **parity check**, with **odd** or **even** parity, both as a **parity byte check** (a parity bit on each byte) and as a **parity block check** (a parity bit on each byte plus a parity byte that checks each column);
   - the **checksum**;
   - the **echo check**.
3. **Describe how a check digit detects errors when data is entered**, and give examples of where check digits are used, including **ISBNs** (book numbers) and **barcodes**.
4. **Describe how an automatic repeat query (ARQ) makes sure data arrives without errors**, using **positive** and **negative acknowledgements** and a **timeout**.

Packets, serial and parallel transmission are in Types and methods of data transmission; encryption is a separate topic.

Understand it

Why check for errors after transmission?

When data travels along a cable or through the air, it is carried as electrical, light or radio signals. Other signals can disturb it: **interference** from nearby equipment, or **crosstalk** between wires lying side by side. A disturbance can flip a 1 into a 0 or a 0 into a 1. Over a network, packets can also collide, time out or go missing. So the data that arrives may not be the data that was sent:

- **data loss**: some bits or packets never arrive;
- **data gain**: extra bits arrive that were never sent;
- **data change**: bits arrive with the wrong value.

The receiver can't tell just by looking at a string of bits whether it is correct. A bank payment of $10couldarriveas$90, or a program could arrive corrupted and crash. So extra checking information is sent with the data, and the receiving device uses it to decide whether an error has happened. If it has, the data is sent again. All of this is done automatically by the devices, not by the user.

Parity check

Before sending, the two devices **agree** to use **even parity** or **odd parity**.

- **Even parity**: every byte must contain an **even number of 1s**.
- **Odd parity**: every byte must contain an **odd number of 1s**.

Each byte carries 7 bits of data and one extra bit, the **parity bit** (here the leftmost bit). The sender counts the 1s in the 7 data bits and sets the parity bit to 0 or 1, whichever makes the total match the agreed parity.

For the data 1011001 there are four 1s.

- With even parity, four is already even, so the parity bit is **0**: the byte sent is **0**1011001.
- With odd parity, the parity bit must be **1** to make five 1s: the byte sent is **1**1011001.

The parity bit doesn't say which parity is used; it is just whichever bit makes the count right.

After transmission, the receiver **counts the 1s in each byte again**. With even parity, a byte with an odd number of 1s must have been changed, so an **error is detected** and the data is sent again. If 01011001 arrives as 01001001, it has three 1s, which is odd, so the error is found.

A parity check has a weakness: it only counts the 1s. If **two** bits change (or any even number of bits), or two bits **swap places** (a transposition), the count can still match the parity. If 01011001 arrives as 01101001, two bits have changed but there are still four 1s, so the error is **not** detected.

Parity block check

A **parity block check** makes parity stronger. The data is sent as a block of bytes, each with its own parity bit (checking its **row**). After the block comes one extra byte, the **parity byte**, whose bits check each **column**: bit 1 of the parity byte makes column 1 match the parity, bit 2 makes column 2 match, and so on.

	bit 1 parity	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7	bit 8	
byte 1	1	1	0	0	0	0	1	1	4 ones
byte 2	1	1	0	0	1	1	1	1	6 ones
byte 3	0	1	0	0	**1**	1	0	0	**3 ones: odd, error**
byte 4	1	1	0	0	0	1	0	1	4 ones
byte 5	0	1	0	1	0	0	1	1	4 ones
parity byte	1	1	0	1	1	1	1	0	

column 5: three 1s, odd

Even parity: every row and every column should have an even number of 1s.

The receiver checks every row **and** every column. If one bit has changed, exactly one row and one column will fail, and the wrong bit is **where they cross**. Here byte 3 and column 5 both have an odd number of 1s, so the error is byte 3, bit 5. Because the receiver knows exactly which bit is wrong, it can even flip it back. A block check also catches some errors that a single byte's parity misses, such as two bits changing in one byte, because the columns they are in will fail.

Checksum

A **checksum** is a value **calculated from the data** using an **algorithm** (a fixed set of rules).

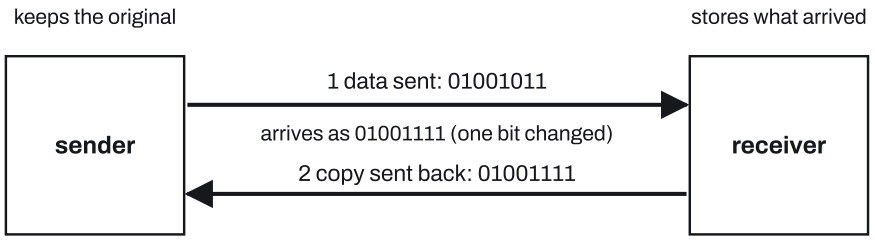
1. Before sending, the sender works out the checksum from the block of data.
2. The checksum is **sent with the data** (in a packet it goes in the trailer).
3. The receiver works out the checksum again from the data it received, **using the same algorithm**.
4. It **compares** the two values. If they **match**, no error is detected. If they **don't match**, the data must have changed on the way, so an error is detected and the data is requested again.

The same algorithm on the same data always gives the same answer. So different answers can only mean the data is different.

For example, one simple algorithm adds up the values of all the bytes and, if the total is more than 255, subtracts 256 to keep it within one byte. For bytes with values 200, 100 and 50 the total is 350, so the checksum is $350 - 256 = 94$. If the 100 arrives as 104, the receiver gets 98, which doesn't match 94, so the error is detected. You won't need to learn a particular checksum algorithm; you need to be able to describe the process.

Echo check

In an **echo check**, the receiver sends a **copy** of the data it received **back to the sender**. The sender compares the copy with the **original** data it kept. If they match, no error is detected; if they don't, an error happened and the data is sent again.



3 The sender compares the copy with the original it kept:

01001011 and 01001111 do not match, so an error is detected and the data is sent again.

If they match, no error is detected.

An echo check is simple, but it doubles the amount of data sent, and a mismatch doesn't show **which** journey the error happened on: the data may have arrived correctly and been damaged on the way back, and it is resent anyway.

Check digit

A **check digit** is different from the other methods: it checks for errors when data is **entered** (typed in or scanned), not after it has been sent between devices. It is used, for example, on **ISBNs** for books, **barcodes** on products, and other long code numbers.

1. When the code number is created, a **check digit is calculated from the other digits** using an algorithm, and **added to the end** of the number.
2. When the number is entered or scanned, the computer **recalculates** the check digit from the other digits, using the same algorithm.
3. If the recalculated digit **matches** the check digit that was entered, the number is accepted. If not, an error was made (for example a digit typed wrongly or two digits swapped), and the number must be entered or scanned again.

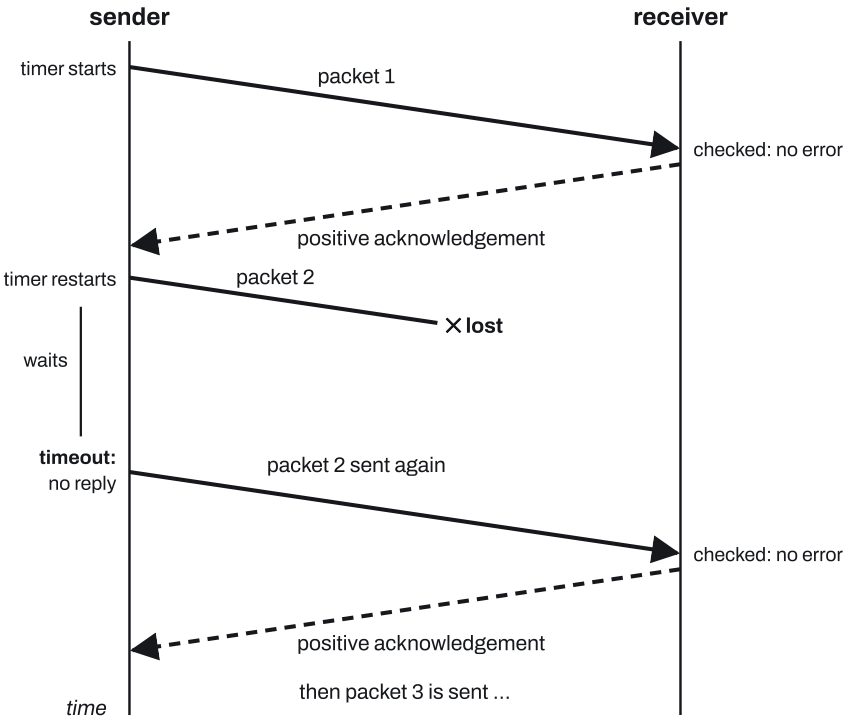
The ISBN-13 algorithm, for example: multiply the first 12 digits alternately by 1 and 3, add the results, and find the remainder when the total is divided by 10. The check digit is 10 minus the remainder (or 0 if the remainder is 0). For the book number 978-0-9934-0612:

$$9 + 3(7) + 8 + 3(0) + 9 + 3(9) + 3 + 3(4) + 0 + 3(6) + 1 + 3(2) = 114$$

The remainder when 114 is divided by 10 is 4, so the check digit is $10 - 4 = 6$, and the full ISBN is 978-0-9934-0612-6. If someone types 978-0-9943-0612-6 (two digits swapped), the check fails, so the error is caught. In an exam you are told which algorithm to use.

Automatic repeat query (ARQ)

An **ARQ** makes sure that each piece of data arrives, and arrives without errors. It uses **acknowledgements** (short messages from the receiver back to the sender) and a **timeout**. The receiver checks each packet with an error detection method such as a parity check or a checksum.



With positive acknowledgement:

1. The sender sends a packet and **starts a timer**.
2. The receiver checks the packet for errors.
3. If there is **no error**, the receiver sends a **positive acknowledgement** back. The sender then sends the next packet and restarts the timer.
4. If there **is** an error, the receiver sends nothing back.
5. If the sender gets no acknowledgement before the timer runs out (a **timeout**), it **sends the packet again**.
6. This repeats until the packet is acknowledged, or until a set number of attempts has been made.

With negative acknowledgement, the receiver sends a **negative acknowledgement** when it finds an error, and this makes the sender resend the packet straight away. The timeout still covers packets that never arrive, since a lost packet produces no reply at all.

Key facts and formulas

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

Formula	Status
Errors in transmission: interference or crosstalk, causing data loss, data gain or data change	Learn it
Even parity: even number of 1s in each byte; odd parity: odd number of 1s	Learn it

Formula	Status
Parity bit: set to 0 or 1 to make the count of 1s match the agreed parity	Learn it
Parity misses an even number of changed bits and transposed bits	Learn it
Parity block: the error is where the failing byte (row) and failing bit (column) cross	Learn it
Checksum: calculated from the data with an algorithm, sent with it, recalculated and compared	Learn it
Echo check: a copy is sent back and compared with the original by the sender	Learn it
Check digit: calculated from the other digits and added to the number; recalculated on data entry (ISBN, barcode)	Learn it
ARQ: acknowledgement (positive or negative) and timeout; resend until acknowledged	Learn it

How to do it

In the Paper 1 questions we have tagged for this topic (20 questions, 2021–2025), the types came up this often. 11 of the 20 questions mix types, so the counts add up to more than 20.

Question type	Questions
Name, match or choose the method	9
Describe how a parity check detects errors	5
Explain how errors occur in transmission	4
Parity with given bytes: type, error location, missed errors	4
Describe the checksum	4
Describe ARQ	4
Echo check: describe, draw or name	3
Describe how a check digit is used	1

1. Name, match or choose the method (9 questions)

1. Match the **key idea** to the method:
   - odd or even, a parity bit, counting 1s → **parity check**;
   - a value calculated from the data with an algorithm, before and after sending → **checksum**;
   - a copy sent back to the sender and compared → **echo check**;

    - acknowledgement and timeout, resending until received → **ARQ**;
    - a value calculated from the digits and added to them, checked on **data entry** → **check digit**.
2. Give the full name, for example "echo check", not "echo".

Variations you will meet:

- **"Give two error detection methods that could check the data after transmission"**: parity check (or parity byte / parity block check), checksum, echo check, ARQ. **Not** check digit, which is for data entry.
- **"Identify one other method"**: don't repeat the one named in the question.
- **Tick the method that does not use a calculated value**: the echo check (it compares a copy; nothing is calculated).
- **A table of statements to tick** against several methods. Tick every method a statement fits: "compares two calculated values" fits checksum **and** check digit; "checks on data entry" fits only check digit; "uses acknowledgement and timeout" and "may resend until received" fit ARQ; "sends extra values with the data" fits checksum and parity.
- **Complete the sentences from a word list**: parity is set to **odd** or **even**; a parity **bit** is added to each byte; an **echo check** compares the data sent with the copy received back; **ARQ** uses acknowledgement and **timeout**; acknowledgements can be **positive** or **negative**. Each word is used once.

2. Describe how a parity check detects errors (5 questions)

1. Odd or even parity is **agreed** by the sender and receiver.
2. The sender **counts the 1s** in each byte (7 data bits).
3. A **parity bit** is added to each byte, 0 or 1, to make the number of 1s match the parity.
4. After transmission the receiver **counts the 1s in each byte again**.
5. If a byte doesn't match the parity (for even parity: an odd number of 1s), an **error is detected**.

Variations you will meet:

- **The question names the parity**, for example "an even parity check". Use it in every step: "the parity bit makes the number of 1s even"; "if a byte has an odd number of 1s, an error is detected". A general answer that never says "even" loses marks.
- **Parity check with ARQ** (6–8 marks): give the parity steps, then the ARQ steps: timer started, acknowledgement if the byte passes, no acknowledgement (or a negative one) if it fails, resent after a timeout. For odd parity: if the number of 1s is odd the parity bit is 0, otherwise 1.
- **Parity block check**: add that a parity byte checks each column, and the error is where the failing row and column cross.

3. Explain how errors occur in transmission (4 questions)

1. Name a **cause**: interference, crosstalk, data collisions, packets timing out, or skewing in parallel transmission.
2. Say what it **does to the data**: bits can be **lost**, **gained** or **changed**.

For example: "Interference on the cable could change some bits, so a 1 arrives as a 0 and the data is changed." That is the cause and the effect, one mark each.

4. Parity with given bytes: type, error location, missed errors (4 questions)

Which parity was used (bytes received correctly): count the 1s in each byte. An even count means even parity, an odd count means odd parity. Write the word "odd" or "even"; don't add a bit.

Find the wrong bit in a parity block:

1. Work out the parity from a row that is clearly fine (most rows will have the same kind of count).
2. Count the 1s in every **byte** (row): the one that doesn't match is the byte number.
3. Count the 1s in every **bit column**, including the parity byte: the one that doesn't match is the bit number.
4. Give both numbers, and explain: "I counted the 1s in each row and column; byte 4 and bit 6 have an even number, but odd parity was used, so the error is where they meet."

Why a parity check missed an error: compare the sent and received bytes. Two bits changed (an even number), or two bits swapped places, so the number of 1s is still the same and still matches the parity. "Multiple bits changed" is not enough, since three changed bits would be caught.

5. Describe the checksum (4 questions)

1. The checksum is **calculated from the data** before it is sent,
2. using an **algorithm**.
3. The checksum is **transmitted with the data**.
4. The receiver **recalculates** the checksum from the data it received, using the **same algorithm**.
5. The two values are **compared**: if they don't match, an error is detected; if they match, no error is detected.
6. If there is an error, the data is **requested again**.

Variations you will meet:

- **"Explain why the values not matching shows an error"**: both values come from the same algorithm, so they are the same if the data is the same; different values mean the data must have changed.
- **Checksum with ARQ, "explain why both are used"**: the checksum **detects** errors using a calculated value; ARQ checks the data was **received**, using acknowledgement and timeout, and asks for the data to be **sent again** if the checksum finds an error or nothing arrives.

6. Describe ARQ (4 questions)

1. The sender sends a packet and **starts a timer**.
2. The receiver **checks** the packet for errors.
3. **Positive acknowledgement**: if there is no error, an acknowledgement is sent back; the sender then sends the **next** packet.

4. **Negative acknowledgement** (if the question uses it): if there is an error, a negative acknowledgement is sent back, which makes the sender resend.
5. If no acknowledgement arrives before the time runs out, a **timeout** happens and the packet is **resent**,
6. **until** it is acknowledged, or a limit on the number of attempts is reached.

Read which kind the question asks for. For positive acknowledgement only, "an error means no acknowledgement is sent, so a timeout makes the sender resend".

7. Echo check: describe, draw or name (3 questions)

1. The data is sent to the receiver.
2. The receiver sends a **copy** of the data it received **back to the sender**.
3. The **sender compares** the copy with the original data.
4. If they don't match, an error is detected (and the data is sent again); if they match, no error.

Drawing it: two labelled boxes (sender and receiver), an arrow labelled "data" from sender to receiver, an arrow labelled "copy of data" back again, and a note at the sender: "compares the original with the copy; if they don't match, an error is detected". The comparison must be at the **sender**, not the receiver.

Naming it: "data sent, then a copy sent back and the two compared" is an **echo check**.

8. Describe how a check digit is used (1 question)

1. The check digit is **calculated from the other digits** of the number (for example a barcode),
2. using an **algorithm** (an example algorithm earns this mark),
3. and **added to the number**.
4. When the barcode is scanned (or the number typed in), the check digit is **recalculated** using the **same algorithm**.
5. If the two check digits **don't match**, an error happened during entry, and the item must be scanned again; if they match, no error.

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

Data is sent between two computers using parity checks.

(a) The four bytes below were all received correctly. Identify whether each was sent using odd or even parity. **[4]**

Byte	Parity
10010110	
01110011	
11111110	
00100100	

(b) The 7-bit value 0110101 is sent using odd parity, with the parity bit added on the left. Write the 8-bit value that is sent. **[1]**

(c) Another computer uses even parity. It sends the byte 10110100, which is received as 10101100. Explain why the parity check does not detect the error. **[2]**

(a) Count the 1s in each byte: 4, 5, 7 and 2. So: even **B1**; odd **B1**; odd **B1**; even **B1**.

(b) 0110101 has four 1s. Odd parity needs an odd total, so the parity bit is 1: **1**0110101 **B1**.

(c) Bits 4 and 5 have swapped places (a transposition) **B1**. The received byte still has four 1s, an even number, so it still matches even parity and looks correct **B1**.

Example 2

Five bytes are sent with a parity block check using **odd** parity. Bit 1 of each byte is its parity bit, and the parity byte checks each column. One bit was changed during transmission. The table shows the data received.

	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8
Byte 1	1	1	0	0	0	0	1	0
Byte 2	1	1	0	1	1	0	0	1
Byte 3	0	1	0	1	0	1	0	0
Byte 4	0	1	0	0	0	0	0	1
Byte 5	1	1	0	1	0	0	1	1
Parity byte	0	0	1	0	0	1	1	0

(a) Identify the byte number and bit number of the error, and explain how you found it. **[4]**

(b) Write the correct value of that byte. **[1]**

(a) Byte 4 **B1**, bit 6 **B1**.

The 1s in each byte were counted: bytes 1, 2, 3 and 5 have 3, 5, 3 and 5 (odd, correct), but byte 4 has only 2, which is even **B1**. The 1s in each column were counted too: column 6 (with the parity byte) has two 1s, an even number, while every other column is odd. The error is where byte 4 and bit 6 cross **B1**.

(b) Flip bit 6 of 01000001 back: **01000101 B1**.

Example 3

A school sends students' exam results from its office computer to a server in another building.

(a) Explain how errors could occur in the data during transmission. **[2]**

(b) A checksum is used to check the results for errors. Describe how the checksum detects errors. **[4]**

(c) The school also uses an automatic repeat query (ARQ) with positive acknowledgement. Describe how the ARQ makes sure the results arrive without errors. **[5]**

(a) There could be interference on the cable between the buildings **B1**, which could change some bits, or cause data to be lost or gained **B1**.

(b) Any four of: the checksum is calculated from the results data **B1**, using an algorithm **B1**; the checksum is sent to the server with the data **B1**; the server recalculates the checksum from the data it received, using the same algorithm **B1**; the two checksums are compared, and if they don't match an error is detected **B1**; the data is then requested again.

(c) Any five of: the office computer sends a packet and starts a timer **B1**; the server checks the packet for errors (with the checksum) **B1**; if there is no error the server sends a positive acknowledgement back **B1**, and the next packet is sent **B1**; if there is an error, no acknowledgement is sent; if the timer runs out before an acknowledgement arrives, a timeout happens **B1** and the packet is sent again **B1**; this repeats until the packet is acknowledged or a set number of attempts is reached.

Example 4

A bookshop scans the barcode on each book it sells. The barcode holds the book's ISBN, whose last digit is a check digit. The sale is then sent to the shop's stock control computer.

(a) Explain how the check digit is used to detect errors when the barcode is scanned. **[4]**

(b) The ISBN check digit is found by multiplying the first 12 digits alternately by 1 and 3, adding the results, finding the remainder when the total is divided by 10, and subtracting that remainder from 10. If the remainder is 0, the check digit is 0. Calculate the check digit for 978-1-84146-208. **[2]**

(c) Give two error detection methods that could check the data after it has been sent to the stock control computer. **[2]**

(a) The check digit was calculated from the other digits of the ISBN **B1**, using an algorithm **B1**. When the barcode is scanned, the check digit is recalculated from the scanned digits using the same algorithm **B1**. If the recalculated digit doesn't match the scanned check digit, an error is detected and the book must be scanned again **B1**.

(b) The 12 digits in order are 9, 7, 8, 1, 8, 4, 1, 4, 6, 2, 0, 8. The 1st, 3rd, 5th, ... are multiplied by 1 and the 2nd, 4th, 6th, ... by 3:

$$(9+8+8+1+6+0)+3\times(7+1+4+4+2+8)=32+3\times 26=110.$$

M1 for multiplying by 1 and 3 and adding. The remainder when 110 is divided by 10 is 0, so the check digit is **0 A1** (not 10).

(c) Any two of: parity check, checksum, echo check, ARQ **B1 B1**. (A check digit is not correct here: it checks data entry, not transmission.)

Common mistakes

- **Giving check digit as a method for checking data after transmission.** A check digit checks data when it is **entered or scanned**; parity, checksum, echo check and ARQ check data after it has been sent (March 2022 report).
- **Thinking the parity bit tells you the parity.** "A parity bit of 1 means even parity" is wrong: the parity bit is whichever of 0 or 1 makes the count of 1s match the agreed parity (March 2022 report).
- **Adding a bit when asked which parity was used.** Just count the 1s and write "odd" or "even" (June 2021 report).
- **Describing parity in general when the question names odd or even.** Say "the parity bit makes the number of 1s even" and "an odd number of 1s means an error" when it's an even parity check (June 2025 report).
- **Vague reasons why parity missed an error.** "More than one bit changed" isn't enough, because three changed bits would be caught. Say **an even number of bits** changed, or bits were **transposed**, so the count still matches (March 2023 report).
- **Describing a checksum as counting bits.** Counting bits gives the size, not a checksum, and sounds like parity. A checksum is a value **calculated from the data with an algorithm**. It isn't a check digit either, and can be more than one digit (March 2021 and June 2025 reports).
- **Saying the user does the checking.** The devices check and request data automatically: "the receiving device recalculates", "the sender resends", not "the user asks for it again" (March 2022 report).
- **Putting the echo check comparison at the receiver.** The copy goes **back**, and the **sender** compares it with the original.
- **Mixing up the ARQ words** in a gap-fill. ARQ uses acknowledgement and **timeout**, and acknowledgements are **positive** or **negative**, not "correct" and "incorrect" (June 2025 report).

How to get the marks

- **"Describe the process"** questions are marked one step per point. Give each step as a separate sentence, in order, and name **who** does it (the sender, the receiver, the receiving device).
- **Use the method's key words**: parity bit, count the 1s, algorithm, recalculate, compare, copy sent back, acknowledgement, timeout, resend. These are what the mark scheme looks for.
- **Use the question's context.** If it says even parity, or positive acknowledgement, or a barcode, use that in your answer, not a general description.
- **"Explain how errors occur"**: a cause (interference, crosstalk, collisions) **and** its effect (data lost, gained or changed).
- **"Give two methods"**: two different methods by name; don't repeat the one in the question, and don't give check digit for transmission.
- **"Identify the byte and bit"**: give both numbers, then say how you found them (counted 1s in rows and columns; the error is where they cross).
- **"Draw and annotate"**: the marks come from the labels. Show both devices, both arrows, and write what the sender does with the copy.
- **Tick one box**: ticking two scores nothing. In a table where a statement can fit more than one method, tick every one that fits.

Quick check

1. Add a parity bit on the left of the 7-bit value 1101100 so that it uses odd parity.
2. Even parity is used. Which of these received bytes contains an error: 10110100, 01110011, 11111111?
3. Name the error detection method: (a) the receiver sends back a copy of the data; (b) it uses acknowledgement and timeout; (c) it checks a number when it is typed in.
4. Even parity is used. The byte 01101100 is received as 01100110. Will the parity check detect the error? Explain why.
5. Using the ISBN algorithm from Example 4, calculate the check digit for 978-1-5098-7412.

Answers

1. 1101100 has four 1s, an even number, so the parity bit is 1: **1**1101100 (five 1s).
2. 01110011: it has five 1s, which is odd. The others have four and eight 1s, which are even.
3. (a) Echo check. (b) ARQ (automatic repeat query). (c) Check digit.
4. No. Two bits have changed (bits 5 and 7), so the byte still has four 1s. That is still even, so it matches the parity and the error is not detected.
5. $(9+8+5+9+7+1)+3\times(7+1+0+8+4+2)=39+3\times22=105$. The remainder is 5, so the check digit is $10-5=5$: 978-1-5098-7412-5.

Past questions to try

- **0478/11/M/J/24 Q9**: odd parity with positive ARQ, described together.
- **0478/12/O/N/22 Q5**: find the wrong bit in a parity block.
- **0478/12/O/N/24 Q5**: how a check digit checks a barcode, then two methods for transmission.
- **0478/13/O/N/23 Q5**: pick the method that doesn't calculate a value, then positive ARQ.