A mark scheme shows the answer and leaves the route to you. These are written the way you should write them in the exam: say what rule applies, show the working line by line, and finish with a check that would catch the mistake. Where a question asks for code, the pseudocode is complete enough to be marked as it stands. 官方答案只给结果；这里把“用什么规则 – 怎么算 – 如何检查”完整写出。

Number and data representation

Convert the denary number 217 to binary and to hexadecimal, showing your working. [3]

Repeatedly divide by 2 and read the remainders upwards: 217, 108 r1, 54 r0, 27 r0, 13 r1, 6 r1, 3 r0, 1 r1, 0 r1, giving `11011001`. For hexadecimal, split that into nibbles from the right: `1101` and `1001`, which are D and 9, so `D9`.

Read the remainders from the LAST division upwards. Reading them downwards reverses every bit and scores nothing.

Check by adding place values: 128+64+16+8+1 = 217.

A two's complement 8-bit register holds 11010110. State its denary value, and give the result of an arithmetic right shift of 2 places. [4]

The leading bit is 1, so the number is negative. Its magnitude is the two's complement: invert to `00101001` and add 1, giving `00101010` = 42, so the value is −42. An arithmetic right shift copies the sign bit into the vacated places: `11110101`, which is −11.

A logical shift would put zeros in and give `00110101` = +53, turning a negative number positive.

Check the size: −42 halved twice is −10.5, and integer shifting rounds towards negative infinity, so −11 is right.

In a floating-point system the mantissa is 8 bits and the exponent 4 bits, both two's complement. Write +7.25 in normalised form. [4]

In binary 7.25 = `111.01`. Normalised form for a positive number needs `0.1` at the front, so shift the point three places left: $\texttt{0.11101} \times 2^{3}$. Pad the mantissa to eight bits: `01110100`. The exponent is 3, which in 4-bit two's complement is `0011`.

Normalised means the first two bits differ: 01 for a positive number, 10 for a negative one.

Check by converting back: `0.11101` = 0.90625, and 0.90625 × 8 = 7.25.

A sound file is recorded for 30 seconds at a sampling rate of 44.1 kHz with a 16-bit sample resolution, in stereo. Calculate the file size in mebibytes. [3]

Bits = $44\,100 \times 16 \times 2 \times 30 = 4.23 \times 10^{7}$. Bytes = $4.23 \times 10^{7}/8 = 5.29 \times 10^{6}$. Mebibytes = $5.29 \times 10^{6}/1\,048\,576 = 5.0$ MiB.

Stereo doubles the data: two channels are recorded, not one.

A mebibyte is 2^{20} bytes, not a million. Dividing by 10^{6} gives 5.3 and loses the mark.

Explain why run-length encoding compresses a two-colour logo well but can enlarge a pho-

tograph. [3]

Run-length encoding replaces each run of identical pixels with one colour value and a count. A logo has long runs of the same colour, so a row of 200 white pixels becomes a single short record. In a photograph almost every pixel differs slightly from its neighbour, so each run is one pixel long and the file stores a count beside every pixel – more data than the original.

Say what it stores, not just that it 'removes repetition'. The count is the whole idea.

The failure case earns the third mark: naming when a compression method makes things worse shows you understand it.

Processor, logic and assembly

The accumulator holds 01101100. Write the single instruction that sets bit 0 to 1 without changing any other bit, and give the resulting contents. [2]

Setting a bit needs OR with a mask that is 1 only in that position: `OR B00000001`. ORing leaves every other bit unchanged, because x OR $0 = x$. The accumulator becomes `01101101`.

AND clears bits and OR sets them. Choosing AND here would wipe the register instead.

`OR #1` and `OR &01` are the same instruction with the operand in denary and hexadecimal.

Write assembly instructions to test whether bit 3 of the accumulator is set, and explain how the program decides. [3]

`AND B00001000` leaves every bit zero except bit 3, which keeps its value. The accumulator is therefore non-zero if and only if bit 3 was 1. The program then branches on that: a conditional jump on zero takes the 'not set' path.

The mask must be 1 in the bit being tested and 0 everywhere else – the opposite of a clear mask.

The answer is not the mask alone: the marks include what is done with the result.

Simplify $X = \bar{A}B + AB\bar{C} + ABC$ using a Karnaugh map, and state how many two-input gates the result needs. [4]

The three terms fill: the whole $AB = 01$ row, and both cells of the $AB = 11$ row. Group the $\bar{A}B$ row as one pair, giving $\bar{A}B$; group the AB row as another, giving AB. Those two groups combine vertically into the single column pair B covering all four cells, so $X = B$. That needs no gates at all.

Always look for the largest legal group. Two pairs would give $\bar{A}B + AB$, correct but not simplified.

Check against the truth table: X is 1 exactly when B is 1, whatever A and C do.

State the row order of a four-variable Karnaugh map and explain why it is not 00, 01, 10, 11. [2]

The order is 00, 01, 11, 10 – Gray code, in which consecutive rows differ in exactly one variable. Grouping works because adjacent cells differ in one variable only, so a group of two eliminates that variable. Between 01 and 10 two variables change, so a group spanning them would eliminate nothing and the simplification would be wrong.

The same applies to the columns, and to the wrap-around: the top row is adjacent to the bottom one.

Describe the fetch stage of the fetch-execute cycle, naming the registers used. [4]

The contents of the Program Counter are copied to the Memory Address Register. The address bus carries that address to memory, and the instruction at it is returned along the data bus into the Memory Data Register. The Program Counter is incremented so it points at the next instruction. The instruction is then copied from the MDR into the Current Instruction Register, ready to be decoded.

Increment the PC during the fetch, not after the execute: a jump instruction overwrites it later, and that only works if it already points past the current instruction.

Databases and files

A CONTAINER table has ContainerID as its primary key and ShipID as a foreign key to SHIP. Write SQL to list each ship's name with its total container weight, for ships carrying more than 500 tonnes, heaviest first. [5]

SELECT ShipName, SUM(Weight) FROM SHIP, CONTAINER WHERE SHIP.ShipID = CONTAINER.ShipID GROUP BY ShipName HAVING SUM(Weight) > 500 ORDER BY SUM(Weight) DESC;

The condition is on an aggregate, so it belongs in HAVING. WHERE is applied before the grouping and cannot see SUM.

Without the join condition every ship pairs with every container and the totals are nonsense.

Explain why the foreign key in CONTAINER must match an existing primary key in SHIP. [3]

The foreign key is what links a container to its ship. If it held a ShipID that no ship has, the container would refer to nothing and any query joining the tables would silently lose that row. The DBMS enforces this as referential integrity, rejecting an insert or a delete that would leave an orphaned reference.

Name the rule: referential integrity is the mark, not just 'it would be wrong'.

A text file stores one member per line as number, surname and class separated by |. Explain why the separator must be a character that cannot appear in the data, and what changes when a field is encrypted. [4]

The three fields have different lengths, so the program finds each one by counting separators. A separator inside the data would split one field in two and shift every later field on that line. Encrypted data is arbitrary bytes and can contain any character, including the separator, so the encrypted values must be re-encoded – as hexadecimal or Base64 – or the file must use fixed-length fields instead.

The problem is not that encryption is unreadable; it is that it can produce the separator byte itself.

Algorithms and programming

An array of 1000 sorted values is searched with a binary search. State the maximum number of comparisons and justify it. [3]

Each comparison halves the range: 1000, 500, 250, 125, 63, 32, 16, 8, 4, 2, 1. That is 10 halvings, so at most 10 comparisons. In general the maximum is $\lceil \log_2 n \rceil$, and $\log_2 1000 = 9.97$.

A linear search would need up to 1000. Quoting both makes the comparison concrete.

The list must be sorted first, or a binary search can miss a value that is present.

Explain why a WHILE loop is the right choice for finding the first negative value in an array, and a FOR loop is not. [3]

The number of iterations is not known in advance: it depends on where the first negative value appears. A FOR loop is count-controlled and always runs its full range, so it either finds the last match rather than the first or wastes every remaining iteration. A WHILE loop tests a condition before each pass, so it can stop as soon as the value is found.

'It stops early' is the mark. Saying only that WHILE is condition-controlled describes it without answering the question.

State the three essential features of a recursive routine, and explain why a stack is used to implement one. [5]

A recursive routine must call itself, must have a base case that does not, and must move towards that base case on every call. Each call needs its own parameters, local variables and return address, and the calls must be unwound in the reverse of the order they were made. A stack is last-in first-out, so pushing a frame for each call and popping it on return gives exactly that behaviour.

Without a base case the routine recurses until the stack is full: stack overflow.

The reason for the stack is the ORDER of unwinding, not merely that it stores things.

Trace the output of a bubble sort's first complete pass over [5, 2, 8, 1]. [3]

Compare 5 and 2: swap, giving [2, 5, 8, 1]. Compare 5 and 8: no swap. Compare 8 and 1: swap, giving [2, 5, 1, 8]. After one pass the largest value, 8, has bubbled to the end.

One pass is one sweep, not the whole sort. The array is not yet in order.

Every pass places one more value in its final position, which is why the next pass can be one comparison shorter.

Describe an algorithm, without using pseudocode, that finds the position of the first negative value in an array of 20 integers and reports when there is none. [5]

Set a found flag to false and a position variable to -1, then start at the first element. While the flag is still false and the index is within the array, examine the current element: if it is negative, record its index in the position variable and set the flag to true; otherwise move to the next element. When the loop ends, output the position if the flag is true, and a not-found message if it is still false.

Describing an algorithm means naming the variables, the test and the loop condition – not just saying "search through the array".

The loop must end on either condition. Testing only the index makes it run to the end and report the LAST negative value, which is the classic error.

-1 is chosen because every valid position is positive, so no real answer can be confused with it.

Networks and security

An IPv4 address is 192.168.10.37 with subnet mask 255.255.255.0. State the network address and explain how it is obtained. [3]

Bitwise AND the address with the mask, octet by octet. The first three octets of the mask are all ones, so they pass through unchanged; the last is all zeros, so it clears the host part. The network address is 192.168.10.0.

The mask says which bits identify the network, not how many devices there are.

192.168 is a private range, so this address is not routable on the internet.

Describe how a digital signature is created and checked, and state what it proves. [5]

The sender hashes the message to produce a digest, then encrypts the digest with their own private key: that is the signature. The receiver decrypts the signature with the sender's public key to recover the digest, and hashes the received message themselves. If the two digests match, the message came from the holder of that private key and has not been altered in transit.

Signing uses the SENDER's private key. Encrypting a message for someone uses the RECIPIENT's public key – the two are constantly confused.

A signature proves origin and integrity, not secrecy: the message itself may still be readable.

Explain why a TLS session uses asymmetric encryption first and symmetric encryption afterwards. [4]

Asymmetric encryption solves the key-distribution problem: the client can send a secret to the server without any prior shared key, because only the server's private key can decrypt it. But it is slow and demanding on the processor. So it is used only to agree a session key, after which the bulk of the traffic is encrypted symmetrically, which is far faster for large amounts of data.

Both halves are needed: what asymmetric encryption is for, and why it is abandoned once its job is done.

Two packets of the same message arrive by different routes and out of order. Describe how the message is still reassembled correctly. [4]

Each packet carries a sequence number in its header, so the receiver can place them in the right order however they arrive. Each also carries an error check; a packet that fails it, or that never arrives, is requested again from the sender. Only when every numbered packet is present and correct is the message passed up to the application.

Routing decisions are made per packet, so different routes are normal, not a fault.