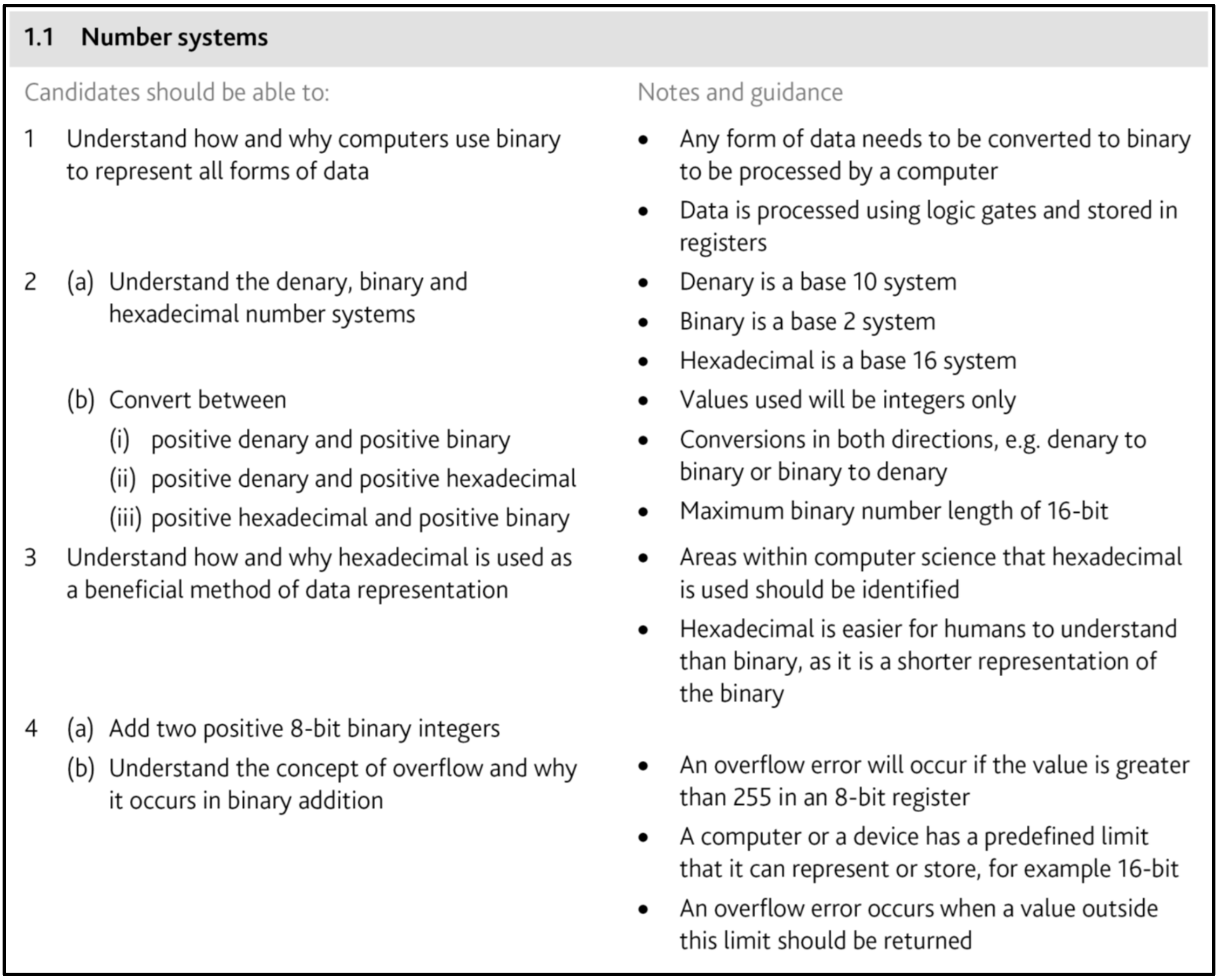
Data Representation

(Chapter 1)

Syllabus Content:



1.1 Number systems

Candidates should be able to:	Notes and guidance
1 Understand how and why computers use binary to represent all forms of data	• Any form of data needs to be converted to binary to be processed by a computer • Data is processed using logic gates and stored in registers
2 (a) Understand the denary, binary and hexadecimal number systems	• Denary is a base 10 system • Binary is a base 2 system • Hexadecimal is a base 16 system
(b) Convert between (i) positive denary and positive binary (ii) positive denary and positive hexadecimal (iii) positive hexadecimal and positive binary	• Values used will be integers only • Conversions in both directions, e.g. denary to binary or binary to denary • Maximum binary number length of 16-bit
3 Understand how and why hexadecimal is used as a beneficial method of data representation	• Areas within computer science that hexadecimal is used should be identified • Hexadecimal is easier for humans to understand than binary, as it is a shorter representation of the binary
4 (a) Add two positive 8-bit binary integers	
(b) Understand the concept of overflow and why it occurs in binary addition	• An overflow error will occur if the value is greater than 255 in an 8-bit register • A computer or a device has a predefined limit that it can represent or store, for example 16-bit • An overflow error occurs when a value outside this limit should be returned

Syllabus Content:

1.1 Number systems continued

Candidates should be able to:	Notes and guidance
5 Perform a logical binary shift on a positive 8-bit binary integer and understand the effect this has on the positive binary integer	• Perform logical left shifts • Perform logical right shifts • Perform multiple shifts • Bits shifted from the end of the register are lost and zeros are shifted in at the opposite end of the register • The positive binary integer is multiplied or divided according to the shift performed • The most significant bit(s) or least significant bit(s) are lost
6 Use two's complement to represent positive and negative 8-bit binary integers	• Convert a positive binary or denary integer to a two's complement 8-bit integer and vice versa • Convert a negative binary or denary integer to a two's complement 8-bit integer and vice versa

1.2 Text, sound and images

Candidates should be able to:	Notes and guidance
1 Understand how and why a computer represents text and the use of character sets, including American standard code for information interchange (ASCII) and Unicode	• Text is converted to binary to be processed by a computer • Unicode allows for a greater range of characters and symbols than ASCII, including different languages and emojis • Unicode requires more bits per character than ASCII
2 Understand how and why a computer represents sound, including the effects of the sample rate and sample resolution	• A sound wave is sampled for sound to be converted to binary, which is processed by a computer • The sample rate is the number of samples taken in a second • The sample resolution is the number of bits per sample • The accuracy of the recording and the file size increases as the sample rate and resolution increase

Syllabus Content:

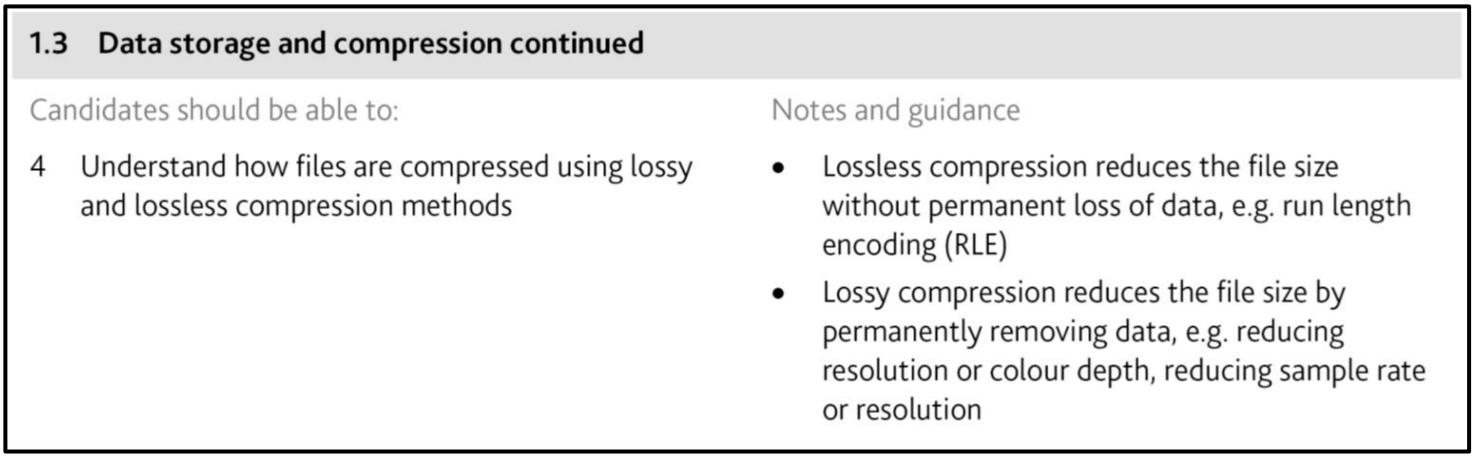
1.2 Text, sound and images continued

Candidates should be able to:	Notes and guidance
3 Understand how and why a computer represents an image, including the effects of the resolution and colour depth	• An image is a series of pixels that are converted to binary, which is processed by a computer • The resolution is the number of pixels in the image • The colour depth is the number of bits used to represent each colour • The file size and quality of the image increases as the resolution and colour depth increase

1.3 Data storage and compression

Candidates should be able to:	Notes and guidance
1 Understand how data storage is measured	• Including: – bit – nibble – byte – kibibyte (KiB) – mebibyte (MiB) – gibibyte (GiB) – tebibyte (TiB) – pebibyte (PiB) – exbibyte (EiB) • The amount of the previous denomination present in the data storage size, e.g.: – 8 bits in a byte – 1024 mebibytes in a gibibyte
2 Calculate the file size of an image file and a sound file, using information given	• Answers must be given in the units specified in the question • Information given may include: – image resolution and colour depth – sound sample rate, resolution and length of track
3 Understand the purpose of and need for data compression	• Compression exists to reduce the size of the file • The impact of this is, e.g.: – less bandwidth required – less storage space required

Syllabus Content:



1.3 Data storage and compression continued

Candidates should be able to:	Notes and guidance
4 Understand how files are compressed using lossy and lossless compression methods	• Lossless compression reduces the file size without permanent loss of data, e.g. run length encoding (RLE) • Lossy compression reduces the file size by permanently removing data, e.g. reducing resolution or colour depth, reducing sample rate or resolution

1 Data Representation

1.1 | Number Systems

1.1.1 Use of Binary in Computers:

- Any form of data needs to be converted to binary to be processed by a computer.
- The data is processed using logic gates and stored in registers.
- The basic building block in all computers is the binary number system which consists of 1's and 0's only.
- A computer contains millions of tiny switches, which must be in ON or OFF position.
- These switches make use of logic gates and are used to store & process data.
- A switch in the ON position is represented by 1 and a switch in the OFF position is represented by 0 in binary.

1.1.2 Binary, Denary & Hexadecimal Systems:

In our syllabus, there are three types of number systems:

1. The binary system; $(0,1)_2$
2. The denary system; $(0\text{-}9)_{10}$
3. The hexadecimal system; $(0\text{-}9, A\text{-}F)_{16}$

Base means number of components e.g., binary has two values 0 and 1 hence 2 components only. It means a base of 2.

1) The Binary System:

- The binary number system is a base 2 number system.
- The numbers in this form are said to be in base 2 as only two values 0 and 1 are used to represent each digit; they are called binary numbers.
- Any positive integer (whole number) can be easily represented in binary by a sequence of 0's and 1's.
- This means that it counts in multiples of 2 giving us the headings such as 2^0, 2^1, 2^2, 2^3 and so on.

Example:

The following is an 8-bit register which means it contains 8 digits. The registers are always filled from right to left.

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

2) The Denary System:

- The denary number system is a base 10 number system.
- The numbers in this form are said to be in base 10 as ten values 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 are used to represent each digit; they are called denary numbers.
- They are also called the decimal number system.
- This means that it counts in multiples of 10 giving us the headings such as 10^0, 10^1, 10^2, 10^3 and so on.

Example:

10^4	10^3	10^2	10^1	10^0
10*10*10*10	10*10*10	10*10	10	1
10 000	1000	100	10	1

Understanding Bits:

For a better understanding, binary numbers can be broken down into their smallest representation called **bits** (the term bit is short for **b**inary dig**it**). The easiest way to understand bits is to think of them as digits. Digit is simply a single place that can hold numerical values between 0 and 9.

Digits are normally combined together in groups to create larger numbers.

For example, **6357** has four digits. It is understood that in this number:

- The 7 is filling the 1's place.
- The 5 is filling the 10's place.
- The 3 is filling the 100's place.
- The 6 is filling the 1000's place.

So you could express things this way if you wanted to be clear:

(6 x 1000) + (3 x 100) + (5 x 10) + (7 x 1) = 6000 + 300 + 50 + 7 = **6357**

Another way to represent it would be in denary (decimal) number system by using powers of 10 like we saw in the table above. The power increases from right-hand side to the left-hand side.

(6 x 10^3) + (3 x 10^2) + (5 x 10^1) + (7 x 10^0) = 6000 + 300 + 50 + 7 = **6357**

It can be seen from this that each digit is a placeholder for the next higher power of 10, starting from the first digit with 10 raised to the power of zero and so on.

The binary number system works exactly the same way as the denary system, except that it contains only two digits, 0 and 1.

For example, to figure out the value of binary number **1101**, you use the similar method described above but with a base of 2 instead of 10 (which means that increasing powers of 2 from right-hand side to the left-hand side instead of powers of 10).

(1 x 2^3) + (1 x 2^2) + (0 x 2^1) + (1 x 2^0) = (1 x 8) + (1 x 4) + (0 x 2) + (1 x 1) = 8 + 4 + 0 + 1 = 13

To further simplify this, we can use the table method as well, by drawing a 4 bit register as **1101** contains only 4 digits.

$\mathbf{2^3}$	$\mathbf{2^2}$	$\mathbf{2^1}$	$\mathbf{2^0}$
8	4	2	1
?	**?**	**?**	**?**

Now fill **1101** in the table and remember that **1** means **YES/ON** and **0** means **NO/OFF**. So accordingly, the powers of 2 above **1's** would be **counted** whereas the powers above **0** would be **ignored**.

$\mathbf{2^3}$	$\mathbf{2^2}$	$\mathbf{2^1}$	$\mathbf{2^0}$
8	4	2	1
1	**1**	**0**	**1**

We get 8 (2^3), 4 (2^2) and 1 (2^0) from the table and 2 (2^1) is ignored as the binary value 0 means OFF or more simply, 2 x 0 = 0.

Add together all the numbers obtained: 8 + 4 + 1 = **13**.

Conversions:

(i) binary to denary:

1. First, draw a table like the ones above.
2. On top of the table, simply write the powers of 2 starting from right to left in an increasing order.
3. The binary number given to you is written underneath the powers.
4. The powers of 2 above binary number "1" are taken into consideration and those above "0" are simply ignored.
5. The powers of 2 above "1" are all written and added together to obtain the denary number.

Examples:

- The maximum binary number length for conversion to be used in this syllabus is 16-bit.

The process of binary to denary conversion is best shown by four examples below which use the 6-bit, 8-bit, 12-bit & 16-bit binary numbers:

1) Convert $(11101110)_2$ to denary [8-bit]:

Since the total digits are 8, we'll use an 8-bit register as given below.

You can either write it as powers of 2 from right to left or straight away the results of the powers. For example, you can either use 2^3 or its result 8.

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

Numbers "16" and "1" are ignored since the binary value underneath them is 0. We simply take 128, 64, 32, 8, 4, 2 and add them all together.

128 + 64+ 32 + 8 + 4 + 2 = **$(238)_{10}$**

$(238)_{10}$ represents that this is a denary number as 10 is written outside the bracket in sub-script indicating this is a base 10 number.

2) Convert $(111001)_2$ to denary [6-bit]:

Since the total digits are 6, we'll still use an 8-bit register but in a slightly different way.

We will write this binary number in the table starting from right-hand side to the left hand-side. Only 6 places will be filled and leftmost 2 will be left empty.

In a register, all the places have to be filled so we will write 0 in the empty spaces instead of leaving them blank to fulfil the pattern of 8-bit register.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
0 (blank)	0 (blank)	1	1	1	0	0	1

Numbers "128", "64", "4" and "2" are ignored since the binary value underneath them is 0. We simply take 32, 16, 8, 1 and add them all together.

32 + 16+ 8 + 1 = **$(57)_{10}$**

3) Convert $(011110001011)_2$ to denary [12-bit]:

Since the total digits are 12, we'll use an 12-bit register as given below.

You can either write it as powers of 2 from right to left or straight away the results of the powers. For example, you can either use 2^3 or its result 8.

2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
2048	**1024**	**512**	**256**	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
0	1	1	1	1	0	0	0	1	0	1	1

Numbers "2048", "64", "32", "16" and "4" are ignored since the binary value underneath them is 0. We simply take 1024, 512, 256, 128, 8, 2, 1 and add them all together.

1024 + 512 + 256 + 128 + 8 + 2 + 1 = **$(1931)_{10}$**

4) Convert $(0011000111100110)_2$ to denary [16-bit]:

Since the total digits are 16, we'll use a 16-bit register as given below.

You can either write it as powers of 2 from right to left or straight away the results of the powers. For example, you can either use 2^3 or its result 8.

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
32768	**16384**	**8192**	**4096**	**2048**	**1024**	**512**	**256**	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
0	0	1	1	0	0	0	1	1	1	1	0	0	1	1	0

Numbers "32768", "16384", "2048", "1024", "512", "16", "8" and "1" are ignored since the binary value underneath them is 0. We simply take 8192, 4096, 256, 128, 64, 32, 4, 2 and add them all together.

8192 + 4096 + 256 + 128 + 64 + 32 + 4 + 2 = $\mathbf{(12774)_{10}}$

This same method can be used for a binary number of any size.

(ii) denary to binary:

To convert from denary to binary, place all 1s in appropriate position under the suitable column headings so that it equates to the denary number.

Examples:

The process of denary to binary conversion is best shown by three examples below:

1) Convert $(142)_{10}$ to binary [8-bit]:

Step by step subtraction approach:

1. Does the number go into 128? Yes, so it becomes a '1' and (142 – 128 = 14).
2. Does the remaining 14 go into 64? No, so it becomes a '0'.
3. Does the remaining 14 go into 32? No, so it also becomes a '0'.
4. Does the remaining 14 go into 16? No, so it also becomes a '0'.
5. Does the remaining 14 go into 8? Yes, so it becomes a '1' and (14 – 8 = 6).
6. Does the remaining 6 go into 4? Yes, so it also becomes a '1' and (6 – 4 = 2).
7. Does the remaining 2 go into 2? Yes, so it also becomes a '1' and (2 – 2 = 0).
8. Since there is nothing left to compare with 1, it becomes a '0' by default.

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

Hence the answer becomes: **$(10001110)_2$**

You can now double check your answer by the same method used in binary to denary conversion:

Numbers "64", "32", "16" and "1" are ignored since the binary value underneath them is 0. We simply take 128, 8, 4, 2 and add them all together.

128 + 8 + 4 + 2 = **$(142)_{10}$**

Hence this confirms that the answer of conversion from denary to binary $(10001110)_2$ is correct.

2) Convert $(59)_{10}$ to binary [8-bit]:

Step by step subtraction approach:

1. Does the number go into 128? No, so it becomes a '0'.
2. Does the number go into 64? No, so it also becomes a '0'.
3. Does the number go into 32? Yes, so it becomes a '1' and (59 – 32 = 27).
4. Does the remaining 27 go into 16? Yes, so it also becomes a '1' and (27 – 16 = 11).
5. Does the remaining 11 go into 8? Yes, so it also becomes a '1' and (11 – 8 = 3).
6. Does the remaining 3 go into 4? No, so it becomes a '0'.
7. Does the remaining 3 go into 2? Yes, so it becomes a '1' and (3 – 2 = 1).
8. Does the remaining 1 go into 1? Yes, so it becomes a '1' and (1 – 1 = 0).

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

Hence the answer becomes: **$(00111011)_2$**

You can now double check your answer by the same method used in binary to denary conversion:

Numbers "128", "64" and "4" are ignored since the binary value underneath them is 0. We simply take 32, 16, 8, 2, 1 and add them all together.

32 + 16 + 8 + 2 + 1 = **$(59)_{10}$**

Hence this confirms that the answer of conversion from denary to binary $(00111011)_2$ is correct.

The third example shows how this method can be used for any size of binary number, which in this case is a 16-bit binary number.

3) Convert $(35000)_{10}$ to binary [16-bit]:

Step by step subtraction approach:

1. Does the number go into 32768? Yes, so it becomes a '1' and (35000 – 32768 = 2232).
2. Does the remaining 2232 go into 16384? No, so it becomes a '0'.
3. Does the remaining 2232 go into 8192? No, so it also becomes a '0'.
4. Does the remaining 2232 go into 4096? No, so it also becomes a '0'.
5. Does the remaining 2232 go into 2048? Yes, so it becomes a '1' and (2232 – 2048 = 184)
6. Does the remaining 184 go into 1024? No, so it becomes a '0'.
7. Does the remaining 184 go into 512? No, so it also becomes a '0'.
8. Does the remaining 184 go into 256? No, so it also becomes a '0'.
9. Does the remaining 184 go into 128? Yes, so it becomes a '1' and (184 – 128 = 56).
10. Does the remaining 56 go into 64? No, so it also becomes a '0'.
11. Does the remaining 56 go into 32? Yes, so it becomes a '1' and (56 – 32 = 24).
12. Does the remaining 24 go into 16? Yes, so it also becomes a '1' and (24 – 16 = 8).
13. Does the remaining 8 go into 8? Yes, so it also becomes a '1' and (8 – 8 = 0).
14. Since there is nothing left to compare with 4, 2 & 1, they all become '0' by default.

32768	**16384**	**8192**	**4096**	**2048**	**1024**	**512**	**256**	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
1	0	0	0	1	0	0	0	1	0	1	1	1	0	0	0

Hence the answer becomes: **$(1000100010111000)_2$**

You can now double check your answer by the same method used in binary to denary conversion:

Numbers "16384", "8192", "4096", "1024", "512", "256", "64", "4", "2" and "1" are ignored since the binary value underneath them is 0. We simply take 32768, 2048, 128, 32, 16, 8 and add them all together.

32768 + 2048 + 128 + 32 + 16 + 8 = **$(35000)_{10}$**

Hence this confirms that the answer of conversion from denary to binary $(1000100010111000)_2$ is correct.

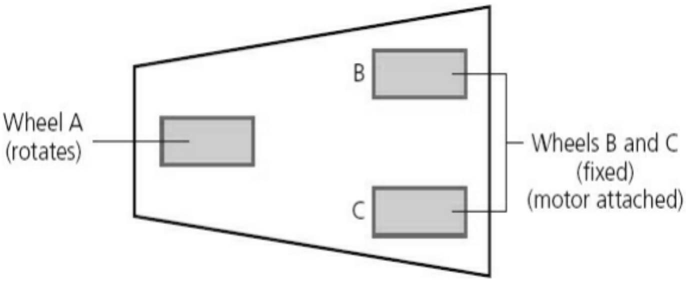
Why is data always stored as binary in computers?

- The computer uses transistors/logic circuits.
- As a result, they can only store data in 2 states that are ON and OFF which are represented by binary digits 1 and 0.

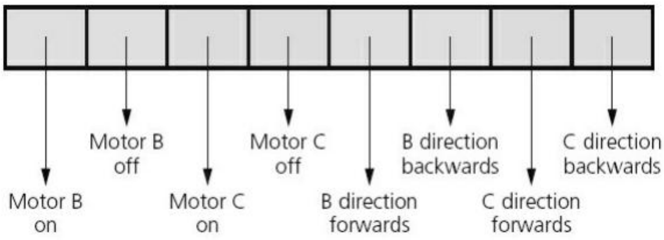
Example Use of Binary:

- Binary is used in registers; a register is made up of a group of binary digits (bits).
- When computers are used to control devices e.g., robots, registers are used as part of the control system.
- They store the instructions and allow for the movement and functioning of devices, e.g. robots.

For example, the following is a robot vacuum cleaner with the shown structure and register:



An 8-bit register is used to control the movement of the robot vacuum cleaner:



(a) What would be the effect if the register contains:

(Instructions: Always remember that binary value 1 means True/Yes that shows it follows the condition above it whereas binary value 0 means False/No that shows it does not follow the condition above it).

i) 10101010:

- motor B is ON and motor C is ON and both motors are turning to produce forward motion.

ii) 10011000:

- motor B is ON and motor C is OFF and motor B is turning to produce forward motion.

iii) 10100110:

- motor B is ON and motor C is ON and motor B is moving backwards while motor C is moving forward.

(b) What would the register contain if only motor C was ON and the motors were turning in a backwards direction?

- 01100101

c) What would the register contain if motor B and motor C were both ON but B was turning in a backward direction and C was turning in a forward direction?

- 10100110

Uses for Binary Numbers (stored in a register):

1. Address in main memory
2. Data
3. Number
4. Instruction
5. ASCII value
6. A sound
7. Part of image

Exam Style Questions:

Question 1:

Explain the difference between binary number system and denary number system. (4)

Exam Tip:

- One statement about either binary or denary earns you 1 mark while a statement involving both binary and denary with an appropriate comparison using words such as "whereas or while" earns you 2 marks for each proper comparison.
- To obtain 4/4 marks in such questions, you need to write two proper comparisons which means a total of 4 statements (2 about binary + 2 about denary).

Possible answers:

- A binary number system is a base-2 system whereas a denary number system is a base-10 system.
- A binary number system uses 0 and 1 values whereas a denary number system uses 0 to 9 values.
- Binary has units that increase by power of 2 whereas denary has units that increase by power of 10.

Question 2:

A hockey club records the number of people that watch each match. An 8-bit binary register is used to store this value.

(a) 46 people watch the first match and 171 people watch the second match.

Show how the registers would store these denary values as 8-bit binary.

Denary value	**8-bit binary**							
46								
171								

[2]

Working space

....

....

....

....

(b) Give the largest denary value that can be stored in the 8-bit binary register.

.... [1]

(c) The hockey club wants to increase the number of people that can watch each match to 2000. The 8-bit binary register may no longer be able to store the value.

Give the smallest number of bits that can be used to store the denary value 2000.

.... [1]

Answer:

<table>
<tr><td>1(a)</td><td>1 mark each
<table>
<tr><th>Denary Value</th><th colspan="8">8-bit binary</th></tr>
<tr><td>46</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>171</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr>
</table>
</td><td>2</td></tr>
<tr><td>1(b)</td><td>– 255</td><td>1</td></tr>
<tr><td>1(c)</td><td>– 11</td><td>1</td></tr>
</table>

3) The Hexadecimal System:

- The hexadecimal number system is a base 16 number system.
- It uses 16 different values to represent each digit and the values used are 0-9 and A-F.
- It can be written in sequence as 0,1,2,3,4,5,6,7,8,9 and 10,11,12,13,14,15 which becomes letters such as:

Digit		Letter
10	=	A
11	=	B
12	=	C
13	=	D
14	=	E
15	=	F

- This means that it counts in multiples of 16 giving us the headings such as 16^0, 16^1, 16^2, 16^3 and so on.

Since $16 = 2^4$ this means that four binary digits are equivalent to each hexadecimal digit.

The table on the next page summarizes the link between binary, hexadecimal and denary. It is mostly used for direct conversions from binary to hexadecimal, and hexadecimal to binary.

It should be memorized at all costs.

Conversions (continued):

Binary value				Hexadecimal value	Denary value
0	0	0	0	**0**	0
0	0	0	1	**1**	1
0	0	1	0	**2**	2
0	0	1	1	**3**	3
0	1	0	0	**4**	4
0	1	0	1	**5**	5
0	1	1	0	**6**	6
0	1	1	1	**7**	7
1	0	0	0	**8**	8
1	0	0	1	**9**	9
1	0	1	0	**A**	10
1	0	1	1	**B**	11
1	1	0	0	**C**	12
1	1	0	1	**D**	13
1	1	1	0	**E**	14
1	1	1	1	**F**	15

(i) binary to hexadecimal:

Examples:

1) Convert $(101111100001)_2$ to hexadecimal:

First split this up into groups of 4 bits starting from right-hand side to left-hand side like this:

1011 1110 0001

Then using the table on left, find the equivalent hexadecimal digits for all three parts.

1011 = B, 1110 = E, 0001 = 1

Hence the hexadecimal number is **$(BE1)_{16}$**

(ii) hexadecimal to binary:

Examples:

1) Convert $(21FD)_{16}$ to binary:

Use the table on the left and for each digit find equivalent binary digits.

2 = 0010, 1 = 0001, F = 1111, D = 1101

Hence the binary number is:

$(0010000111111101)_2$

Note: In case of a binary number with 14 digits, it will be split into 4 groups (from right to left always) and each must contain four bits. 4 bits x 4 groups = 16 digits total. Hence, we must add 0's to fill the empty space of 2 digits.

Example: 10000111111101 becomes 10 0001 1111 1101

To complete the space of 2 digits on left, we will add 2 zeroes and it will become:

0010 0001 1111 1101

Following the same method, we will use the table above and find the equivalent hexadecimal digits for all four parts.

0010 = 2, 0001 = 1, 1111 = F, 1101 = D, hence the hexadecimal number is **$(21FD)_{16}$**

(iii) hexadecimal to denary:

Examples:

1) Convert $(45A)_{16}$ to denary:

First multiply each digit by its value (power of 16 written above it) and then add the results together:

$16^2 = 256$	$16^1 = 16$	$16^0 = 1$
4	**5**	**A = 10**

(4 x 256) + (5 x 16) + (10 x 1) = 1024 + 80 + 10 = 1114

Hence the denary number is **$(1114)_{10}$**

2) Convert $(C8F)_{16}$ to denary:

First multiply each digit by its value (power of 16 written above it) and then add the results together:

$16^2 = 256$	$16^1 = 16$	$16^0 = 1$
C = 12	**8**	**F = 15**

(12 x 256) + (8 x 16) + (15 x 1) = 3072 + 128 + 15 = 3215

Hence the denary number is **$(3215)_{10}$**

(iv) denary to hexadecimal:

Examples:

1) Convert $(2004)_{10}$ to hexadecimal:

Place hexadecimal digits in appropriate positions (under appropriate powers of 16) so that the total equates to 2004:

256	16	1
7	**13 = D**	**4**

(<u>7</u> x 256) + (<u>13</u> x 16) + (<u>4</u> x 1) = 1792 + 208 + 4 = 2004

Hence the hexadecimal number is **$(7D4)_{16}$**

2) Convert $(3743)_{10}$ to hexadecimal:

Place hexadecimal digits in appropriate positions (under appropriate powers of 16) so that the total equates to 3743:

256	16	1
14 = E	9	15 = F

(14 x 256) + (9 x 16) + (15 x 1) = 3584 + 144 + 15 = 3743

Hence the hexadecimal number is **$(E9F)_{16}$**

Another method involves successive division by 16. The remainders are then read from bottom to top to give hexadecimal value. Using 3743, we get:

Divisor	Dividend	Remainder
16	3743	
16	233	**15**
16	14	**9**
	0	**14**

Reading the remainder from bottom to top gives:

- 14 = **E**
- **9**
- 15 = **F**

Hence the hexadecimal number is **$(E9F)_{16}$**

1.1.3 Uses of Hexadecimal System:

- Hexadecimal is easier for humans to understand than binary, as it is a shorter representation of the binary.
- A computer can only work with binary data whilst computer scientists find hexadecimal to be more convenient to use.
- This is because one hex digit represents four binary digits so a complex binary number, such as 1101001010101111 can be written in hex in a shorter form as D2AF.

There are four main uses of the hexadecimal system in our syllabus:

1. Error codes
2. Media Access Control (addresses) / MAC addresses
3. Internet Protocol addresses / IPv6 addresses
4. HTML color codes

1) Error Codes:

- The error codes are often shown as hexadecimal values.
- These numbers refer to the memory location of the error and are usually automatically generated by the computer.
- The programmer needs to know how to interpret the hexadecimal error codes.

2) Media Access Control (MAC) addresses:

It is represented using six groups of hexadecimal digits:

NN–NN–NN–DD–DD–DD **OR** NN:NN:NN:DD:DD:DD

Explain what is meant by MAC address?

NOTE: Any of the following points written according to number of marks will earn you full marks in exam question such as:

- For a 3 mark question, write any 3 points.
- For a 4 mark question, write 4 points

Therefore, write answers accordingly based on number of marks

- It is media access control (address).
- It is a unique number/address that identifies a device connected to the internet.
- It is a static address which means it does not change.
- The first part of the address is manufacturer ID, and second part is the serial number of device.
- It is set by the manufacturer.

State why MAC addresses are used.

- It allows all devices to be uniquely identified.

Explain what the hexadecimal code in a MAC address represents.

NOTE: The hexadecimal code in a MAC address is the NN's and DD's.

- It is a unique physical address/number associated with network interface card in a device.
- It is usually made up of 48 bits which are shown as six groups of hexadecimal digits.
- The first part or 6 digits (NN-NN-NN) is the manufacturer ID of device.
- The second part or last 6 digits (DD-DD-DD) is the serial number of device.

3) Internet Protocol (IP) addresses:

- Each device on the internet is given a unique address known as the Internet Protocol (IP) address.
- IP addresses can be IPv4 (32 bit) or IPv6 (128 bit).

IPv4 address:

- IPv4 address is a 32-bit number written in denary or hexadecimal form.
- For example: 109.108.158.1 in denary or 77.76.9e.01 in hex.

IPv6 address:

- IPv4 has recently been improved upon by adoption of IPv6.
- IPv6 address is a 128-bit number broken down into 16-bit chunks, represented by a hexadecimal number.
- For example: a8fb:7a88:fff0:0fff:3d21:2085:66fb:f0fa

Note: IPv6 uses a colon (:) rather than a decimal point (.) as used in IPv4.

4) HyperText Mark-up Language (HTML) color codes:

What is HyperText Mark-up Language?

- It is used to create and develop webpages/websites (web authoring language).
- It is not a programming language but is simply a mark-up language.
- It is used in processing, definition, and presentation of text.

HTML Color Codes:

- HTML is used in specifying the color of the text using tags which bracket a piece of code.
- It is used to represent different colors of text on computer screens and all colors can be made up of different combinations of three primary colors: red, green & blue.
- The different intensity of each primary color is determined by its hexadecimal value.
- This shows different hexadecimal values represent different colors.

- The color codes are always six hexadecimal digits representing the red, green & blue components.
- There are a possible 256 values for red, 256 values for green and 256 values for blue.
- This proves 256 x 256 x 256 = 16 177 216 possible colors.

Example:

1. **# FF 00 00** represents primary color red.
2. **# 00 FF 00** represents primary color green.
3. **# 00 00 FF** represents primary color blue.

Exam Style Questions:

Question 1:

(a) Convert the hexadecimal number **B5** into binary:

....

Convert the binary number **1 1 1 1 0 1 1 0** into hexadecimal:

.... [2]

(b) Give **two** examples where hexadecimal numbers are used in computer science.

1 ....

....

2 ....

.... [2]

(c) State **two** benefits of using hexadecimal numbers in computer science.

1 ....

....

2 ....

.... [2]

Answer:

(a) 1 0 1 1 0 1 0 1

F 6 [2]

(b) Any **two** from:

- HTML
- MAC address
- used in assembly language/machine code
- debugging (displays bytes in hex when using memory dumps)

[2]

(c)
- Can represent 16 bit words as only 4 hexadecimal digits
- It is easy to convert hex digits back to binary if necessary [2]

Question 2:

A computer stores data in binary form. Binary numbers can be represented as hexadecimal and denary numbers.

(a) Convert the 8-bit binary number 01010101 to denary.

.... [1]

Working space

....

....

....

(b) Convert the binary number 11000000 to hexadecimal.

.... [1]

Working space

....

....

....

(c) Convert the hexadecimal number 1A to denary.

.... [1]

Working space

....

....

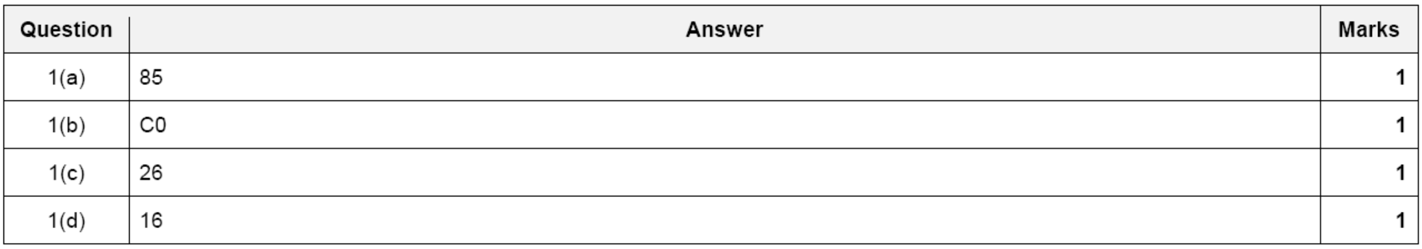
....

(d) Binary numbers can be stored as bytes.

State how many bits are in **two** bytes.

.... [1]

Answer:



Question	Answer	Marks
1(a)	85	**1**
1(b)	C0	**1**
1(c)	26	**1**
1(d)	16	**1**

Question 3:

Binary is a number system that is used by computers.

(a) Tick (✓) **one** box to show whether binary is a base-2, base-10 or base-16 number system.

Tick (✓)

□ Base-2

□ Base-10

□ Base-16

[1]

(b) Hexadecimal and denary are number systems that can be used by programmers.

Convert these **four** hexadecimal values into denary values.

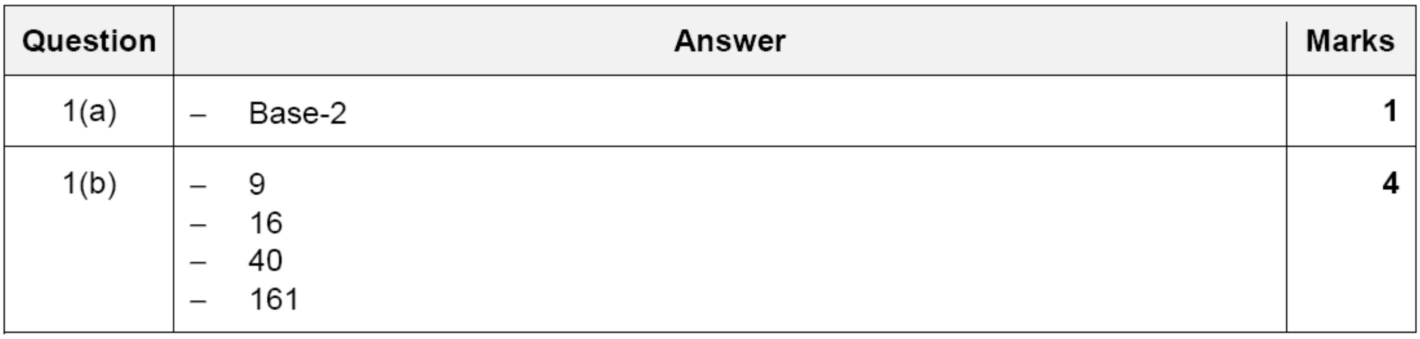
09 ....

10 ....

28 ....

A1 ....

Answer: [4]



Question	Answer	Marks
1(a)	– Base-2	**1**
1(b)	– 9 – 16 – 40 – 161	**4**

Question 4:

A school network has several computers.

Each computer in the network has a media access control (MAC) address.

Hexadecimal is used for MAC addresses.

Part of a MAC address is given.

97–5C–E1

Each pair of digits is stored as binary in an 8-bit register.

(a) Complete the binary register for these two pairs of digits.

97								
5C								

[4]

(b) Describe what is meant by a MAC address.

....

....

....

....

....

....

....

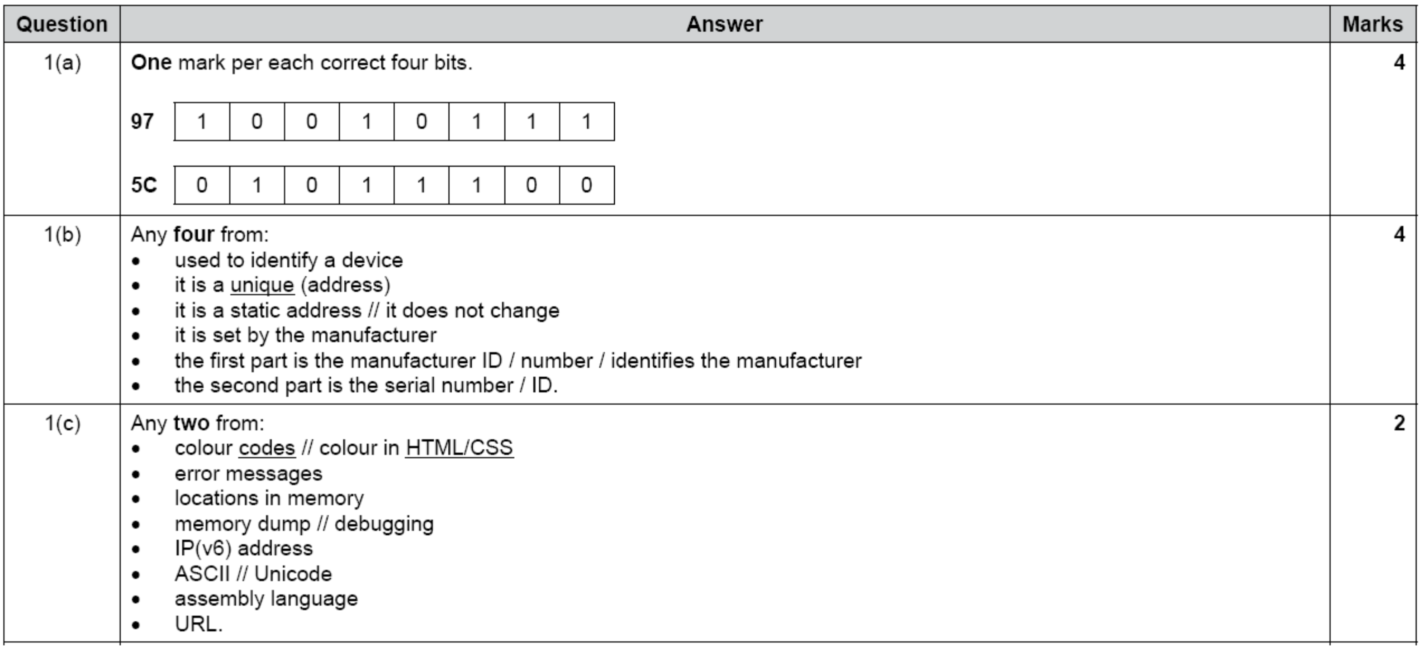
.... [4]

(c) Give **two other** uses of hexadecimal in computer science.

1 ....

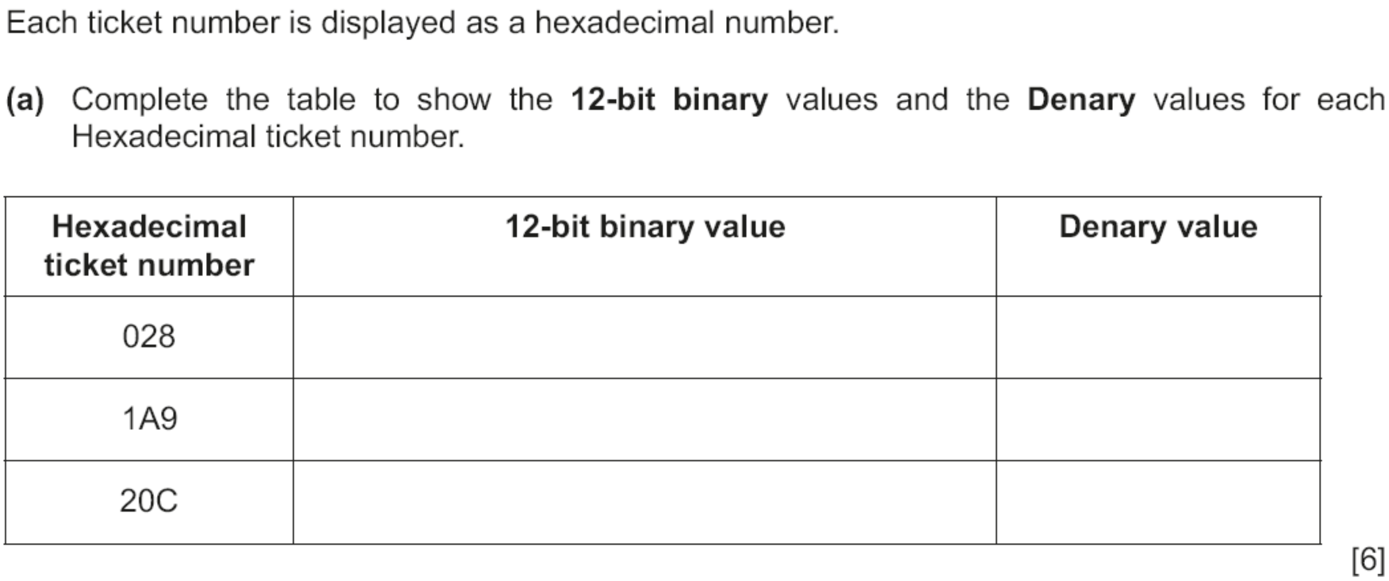
2 .... [2]

Answer:



Question	Answer	Marks
1(a)	**One** mark per each correct four bits. **97** 1 0 0 1 0 1 1 1 **5C** 0 1 0 1 1 1 0 0	**4**
1(b)	Any **four** from: • used to identify a device • it is a unique (address) • it is a static address // it does not change • it is set by the manufacturer • the first part is the manufacturer ID / number / identifies the manufacturer • the second part is the serial number / ID.	**4**
1(c)	Any **two** from: • colour codes // colour in HTML/CSS • error messages • locations in memory • memory dump // debugging • IP(v6) address • ASCII // Unicode • assembly language • URL.	**2**

Question 5:



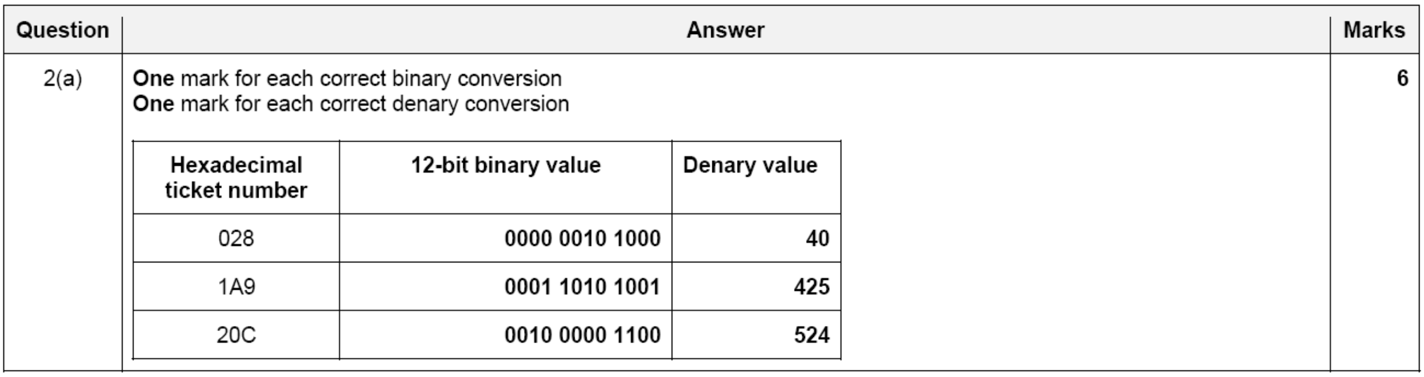
Each ticket number is displayed as a hexadecimal number.

(a) Complete the table to show the **12-bit binary** values and the **Denary** values for each Hexadecimal ticket number.

Hexadecimal ticket number	12-bit binary value	Denary value
028		
1A9		
20C		

[6]

Answer:



Question	Answer	Marks
2(a)	**One** mark for each correct binary conversion **One** mark for each correct denary conversion	**6**

Hexadecimal ticket number	12-bit binary value	Denary value
028	**0000 0010 1000**	**40**
1A9	**0001 1010 1001**	**425**
20C	**0010 0000 1100**	**524**

Question 6:

(a) **Six** binary or hexadecimal numbers and **six** denary conversions are given.

Draw a line to connect each binary or hexadecimal number to the correct denary conversion.

Binary or hexadecimal	Denary
01001011	75
4E	78
11011010	157
10011101	167
A7	25
19	218

[5]

(b) Hexadecimal is often used by computer programmers to represent binary values.

Explain why computer programmers may choose to use hexadecimal.

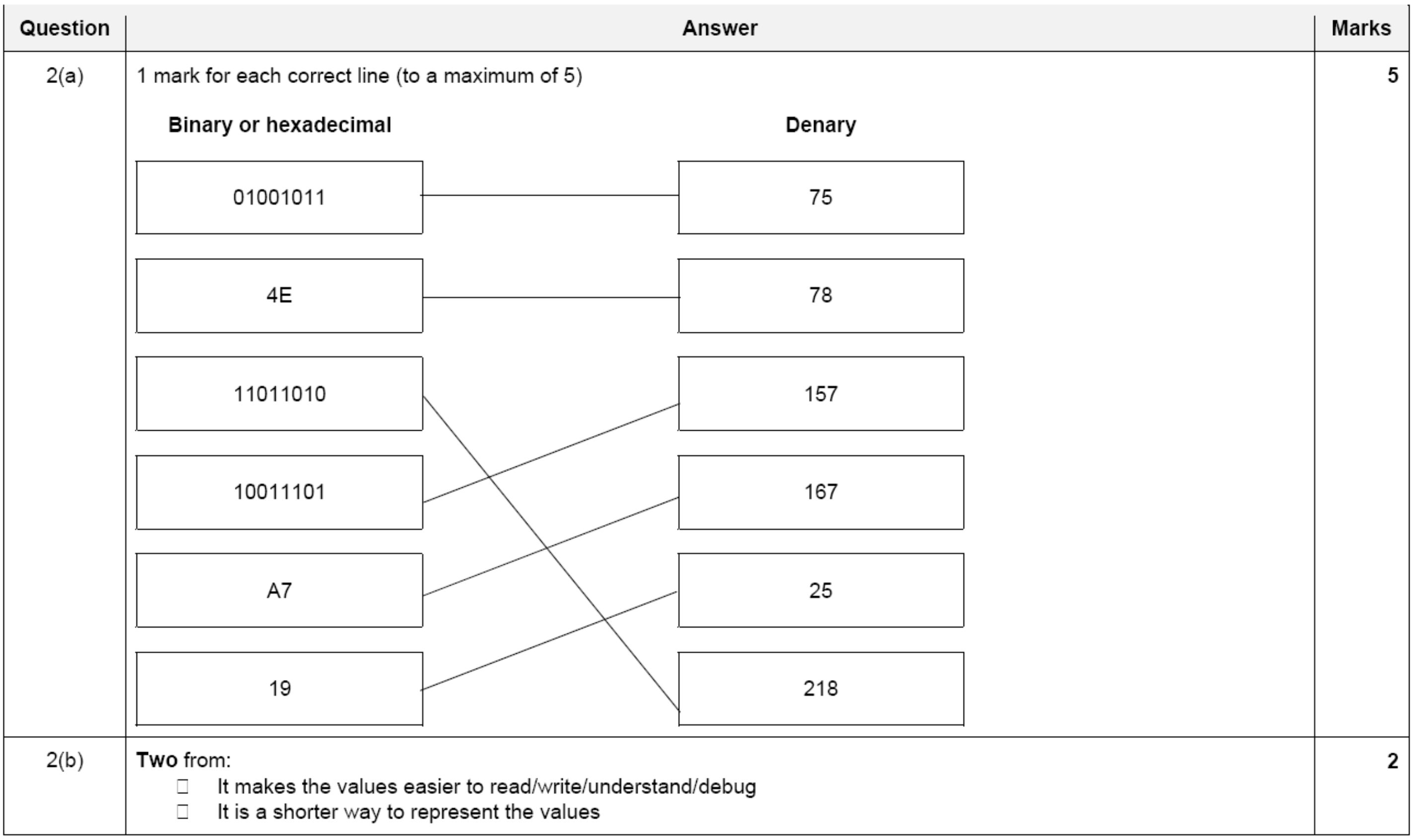
....

....

....

.... [2]

Answer:



Question	Answer	Marks
2(a)	1 mark for each correct line (to a maximum of 5)	5
	Binary or hexadecimal — **Denary** 01001011 — 75 4E — 78 11011010 — 218 10011101 — 157 A7 — 167 19 — 25	
2(b)	**Two** from: - It makes the values easier to read/write/understand/debug - It is a shorter way to represent the values	2

1.1.4 Binary Addition & Overflow:

NOTE: Binary Addition & Overflow are newly added topics in the Computer Science (2210) syllabus for the session 2023–2025.

According to the new syllabus, we will look at the method of adding two positive 8-bit binary integers.

The given key facts must be memorized at all costs because they will help in solving all types of addition questions easily.

Remember the following rules when carrying out addition of two binary digits:

Binary Addition	Carry	Sum
0 + 0	0	0
0 + 1	0	1
1 + 0	0	1
1 + 1	1	0

Remember the following extension of rules when carrying out addition of three binary digits:

Binary Addition	Carry	Sum
0 + 0 + 0	0	0
0 + 0 + 1	0	1
0 + 1 + 0	0	1
0 + 1 + 1	1	0
1 + 0 + 0	0	1
1 + 0 + 1	1	0
1 + 1 + 0	1	0
1 + 1 + 1	1	1

Examples:

- The labeling below represents column numbers, such as c1 is column 1 & c5 is column 5.

Since binary addition is a newly added topic, the following addition example is solved step-by-step so you can understand it better first and then practice a lot afterwards.

Addition of 00100111 + 01001010:

	c8	**c7**	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	
	0	0	1	0	0	1	1	1	
+	0	1	0	0	1	0	1	0	
									← carry values
									← sum values

1) Solving c1 first (according to rules of tables for two-digit addition):

- 1 + 0 = 1 (sum) & 0 (carry); therefore, carry column will be left blank for next column (c2).

	c8	**c7**	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	
	0	0	1	0	0	1	1	1	
+	0	1	0	0	1	0	1	0	
									← carry values
								1	← sum values

2) Solving c2 (according to rules of tables for two-digit addition):

- 1 + 1 = 0 (sum) & 1 (carry); the carry value will be added to next column (c3).

	c8	**c7**	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	
	0	0	1	0	0	1	1	1	
+	0	1	0	0	1	0	1	0	
						1			← carry values
							0	1	← sum values

3) Solving c3 (according to rules of tables for three-digit addition):

- 1 + 0 + 1 (from carry) = 0 (sum) & 1 (carry); the carry value will be added to next column (c4).

	c8	c7	c6	c5	c4	c3	c2	c1	
	0	0	1	0	0	1	1	1	
+	0	1	0	0	1	0	1	0	
					1	1			← carry values
						0	0	1	← sum values

4) Solving c4 (according to rules of tables for three-digit addition):

- 0 + 1 + 1 = 0 (sum) & 1 (carry); the carry value will be added to the next column (c5).

	c8	c7	c6	c5	c4	c3	c2	c1	
	0	0	1	0	0	1	1	1	
+	0	1	0	0	1	0	1	0	
				1	1	1			← carry values
					0	0	0	1	← sum values

5) Solving c5 (according to rules of tables for three-digit addition):

- 0 + 0 + 1 = 1 (sum) & 0 (carry); therefore, carry column will be left blank for next column (c6).

	c8	c7	c6	c5	c4	c3	c2	c1	
	0	0	1	0	0	1	1	1	
+	0	1	0	0	1	0	1	0	
				1	1	1			← carry values
				1	0	0	0	1	← sum values

6) Solving c6 (according to rules of tables for two-digit addition):

- 1 + 0 = 1 (sum) & 0 (carry); therefore, carry column will be left blank for next column (c7).

	c8	c7	c6	c5	c4	c3	c2	c1	
	0	0	1	0	0	1	1	1	
+	0	1	0	0	1	0	1	0	
				1	1	1			← carry values
			1	1	0	0	0	1	← sum values

7) Solving c7 (according to rules of tables for two-digit addition):

- 0 + 1 = 1 (sum) & 0 (carry); therefore, carry column will be left blank for last column (c8).

	c8	c7	c6	c5	c4	c3	c2	c1	
	0	0	1	0	0	1	1	1	
+	0	1	0	0	1	0	1	0	
				1	1	1			← carry values
		1	1	1	0	0	0	1	← sum values

8) Solving c8 (according to rules of tables for two-digit addition):

- 0 + 0 = 0 (sum) & 0 (carry); **therefore, no additional 9th bit would be generated and so no overflow will occur (overflow discussed later).**

	c8	c7	c6	c5	c4	c3	c2	c1	
	0	0	1	0	0	1	1	1	
+	0	1	0	0	1	0	1	0	
				1	1	1			← carry values
	0	1	1	1	0	0	0	1	← sum values

Answer of binary addition = 0 1 1 1 0 0 0 1 (sum values)

The second addition example is more advanced and solved step-by-step so you can understand it better first and then practice a lot afterwards.

Convert 126 and 62 into binary:

- 126 = 0 1 1 1 1 1 1 0
- 62 = 0 0 1 1 1 1 1 0

Add these two binary values obtained above and check that the result matches the addition of these two denary numbers above:

Addition of 00100111 + 01001010:

	c8	c7	c6	c5	c4	c3	c2	c1	
	0	1	1	1	1	1	1	0	
+	0	0	1	1	1	1	1	0	
									← carry values
									← sum values

1) Solving c1 first (according to rules of tables for two-digit addition):

- 0 + 0 = 0 (sum) & 0 (carry); therefore, carry column will be left blank for next column (c2).

	c8	c7	c6	c5	c4	c3	c2	**c1**	
	0	1	1	1	1	1	1	0	
+	0	0	1	1	1	1	1	0	
									← carry values
								0	← sum values

2) Solving c2 (according to rules of tables for two-digit addition):

- 1 + 1 = 0 (sum) & 1 (carry); the carry value will be added to the next column (c3).

	c8	**c7**	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	
	0	1	1	1	1	1	1	0	
+	0	0	1	1	1	1	1	0	
						1			← carry values
							0	0	← sum values

3) Solving c3 (according to rules of tables for three-digit addition):

- 1 + 1 + 1 = 1 (sum) & 1 (carry); the carry value will be added to the next column (c4).

	c8	**c7**	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	
	0	1	1	1	1	1	1	0	
+	0	0	1	1	1	1	1	0	
					1	1			← carry values
						1	0	0	← sum values

4) Solving c4 (according to rules of tables for three-digit addition):

- 1 + 1 + 1 = 1 (sum) & 1 (carry); the carry value will be added to the next column (c5).

	c8	**c7**	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	
	0	1	1	1	1	1	1	0	
+	0	0	1	1	1	1	1	0	
				1	1	1			← carry values
					1	1	0	0	← sum values

5) Solving c5 (according to rules of tables for three-digit addition):

- 1 + 1 + 1 = 1 (sum) & 1 (carry); the carry value will be added to the next column (c6).

	c8	**c7**	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	
	0	1	1	1	1	1	1	0	
+	0	0	1	1	1	1	1	0	
			1	1	1	1			← carry values
				1	1	1	0	0	← sum values

6) Solving c6 (according to rules of tables for three-digit addition):

- 1 + 1 + 1 = 1 (sum) & 1 (carry); the carry value will be added to the next column (c7).

	c8	**c7**	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	
	0	1	1	1	1	1	1	0	
+	0	0	1	1	1	1	1	0	
		1	1	1	1	1			← carry values
			1	1	1	1	0	0	← sum values

7) Solving c7 (according to rules of tables for three-digit addition):

- 1 + 0 + 1 = 0 (sum) & 1 (carry); the carry value will be added to the next column (c8).

	c8	**c7**	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	
	0	1	1	1	1	1	1	0	
+	0	0	1	1	1	1	1	0	
	1	1	1	1	1	1			← carry values
		0	1	1	1	1	0	0	← sum values

8) Solving c8 (according to rules of tables for three-digit addition):

- 0 + 0 + 1 = 1 (sum) & 0 (carry); **therefore, no additional 9th bit would be generated and so no overflow will occur (overflow discussed later)**

	c8	c7	c6	c5	c4	c3	c2	c1	
	0	1	1	1	1	1	1	0	
+	0	0	1	1	1	1	1	0	
	1	1	1	1	1	1			← carry values
	1	0	1	1	1	1	0	0	← sum values

Answer of binary addition = 1 0 1 1 1 1 0 0 (sum values)

The answer of binary addition **1 0 1 1 1 1 0 0** has equivalent denary value of:

- 128 + 32 + 16 + 8 + 4 = **188**

The answer of addition of both denary values is:

- 126 + 62 = **188**

Hence the result of two binary numbers matches the addition of their two denary numbers.

Concept & Reason of Overflow:

- An overflow error will occur if the value of addition (result) is greater than 255 in an 8-bit register.
- A computer or a device has a predefined limit that it can represent or store e.g., 16-bit.
- An overflow error occurs when a value outside this limit should be returned.

Overflow in 8-bit register:

- An overflow error will occur if the result of addition is greater than 255 in an 8-bit register.
- An 8-bit register would allow a maximum denary value of 255 (which is $2^8 - 1$) to be stored.
- The generation of a 9th bit in addition shows sum has exceeded this value causing overflow error.
- This indicates that a number is too big to be stored in the computer using 8 bits hence it requires more bits for storage.

However, the greater the number of bits which can be used to represent a number then the larger the number that can be stored. The examples are given on the next page.

Overflow in 16-bit register:

- An overflow error will occur if the result of addition is greater than 65 535 in a 16-bit register.
- A 16-bit register would allow a maximum denary value of 65 535 (which is $2^{16} - 1$) to be stored.
- The generation of a 9th bit in addition shows sum has exceeded this value causing overflow error.
- This indicates that a number is too big to be stored in the computer using 16 bits hence it requires more bits for storage.

Overflow in 32-bit register:

- An overflow error will occur if the result of addition is greater than 4 294 967 295 in a 32-bit register.
- A 32-bit register would allow a maximum denary value of 4 294 967 295 (which is $2^{32} - 1$) to be stored.
- The generation of a 9th bit in addition shows sum has exceeded this value causing overflow error.
- This indicates that a number is too big to be stored in the computer using 32 bits hence it requires more bits for storage.

This process of overflow goes on for 64-bit register, 128-bit register and so on in a similar way.

Example of Overflow:

- The labeling below represents column numbers, such as c1 is column 1 & c5 is column 5.

Since overflow & binary addition are newly added topics, the following overflow in addition example is solved step-by-step so you can understand it better first and then practice a lot afterwards.

Addition of 01101110 + 11011110 (using 8 bits):

	c8	**c7**	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	
	0	1	1	0	1	1	1	0	
+	1	1	0	1	1	1	1	0	
									← carry values
									← sum values

Note: The denary number of **01101110 is 110** & denary number of **11011110 is 222.**

1) Solving c1 & c2 first (according to rules of tables for two-digit addition):

- **c1 →** 0 + 0 = 0 (sum) & 0 (carry); the carry column will be left blank for next column (c2).
- **c2 →** 1 + 1 = 0 (sum) & 1 (carry); the carry value will be added to the next column (c3).

	c8	**c7**	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	
	0	1	1	0	1	1	1	0	
+	1	1	0	1	1	1	1	0	
						1			← carry values
							0	**0**	← sum values

2) Solving c3 & c4 (according to rules of tables for three-digit addition):

- **c3 →** 1 + 1 + 1 = 1 (sum) & 1 (carry); the carry value will be added to the next column (c4).
- **c4 →** 1 + 1 + 1 = 1 (sum) & 1 (carry); the carry value will be added to the next column (c5).

	c8	**c7**	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	
	0	1	1	0	1	1	1	0	
+	1	1	0	1	1	1	1	0	
				1	**1**	1			← carry values
					1	**1**	0	0	← sum values

3) Solving c5 & c6 (according to rules of tables for three-digit addition):

- **c5 →** 0 + 1 + 1 = 0 (sum) & 1 (carry); the carry value will be added to the next column (c6).
- **c6 →** 1 + 0 + 1 = 0 (sum) & 1 (carry); the carry value will be added to the next column (c7).

	c8	**c7**	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	
	0	1	1	0	1	1	1	0	
+	1	1	0	1	1	1	1	0	
		1	**1**	1	1	1			← carry values
			0	**0**	1	1	0	0	← sum values

4) Solving c7 (according to rules of tables for three-digit addition):

- **c7** → 1 + 1 + 1 = 1 (sum) & 1 (carry); the carry value will be added to the next column (c8).

	c8	c7	c6	c5	c4	c3	c2	c1	
	0	1	1	0	1	1	1	0	
+	1	1	0	1	1	1	1	0	
	1	1	1	1	1	1			← carry values
		1	0	0	1	1	0	0	← sum values

5) Solving c8 (according to rules of tables for three-digit addition):

- **c8** → 0 + 1 + 1 = 0 (sum) & 1 (carry); **therefore, an additional 9th bit would be generated and so overflow will occur**

	c8	c7	c6	c5	c4	c3	c2	c1	
	0	1	1	0	1	1	1	0	
+	1	1	0	1	1	1	1	0	
1	1	1	1	1	1	1			← carry values
1	**0**	1	0	0	1	1	0	0	← sum values

9th bit generated

- This addition has generated a 9th bit.
- The 8 bits of the answer (sum values) are **0 1 0 0 1 1 0 0**
- These bits give the denary value 64 + 8 + 4 = 76.
- This is incorrect because the denary value of the addition of both denary numbers (whose binary numbers are used in example) is 110 + 222 = 332.
- Therefore, generation of a 9th bit indicates an overflow error because an 8-bit register would allow a maximum denary value of 255 to be stored.
- The correct value 332 is greater than 255 (which is maximum capacity of 8-bit register) and hence it cannot be stored in 8-bit register causing an overflow error.

Exam Tip:

The description of overflow given on previous page is for your conceptual understanding only.

However, in the exam you will only write a specific, to the point and short answer such as sample answers written for sample questions given below:

Sample Question 1: Describe an error that may occur during addition of binary numbers. (2)

Sample Answer:

- The answer cannot be represented in the current number of bits.
- This error is called overflow.

Sample Question 2: State how an overflow can occur when adding two binary integers. (1)

Sample Answer:

- The result is a larger number than can be stored in the given number of bits OR
  The result is greater than 255.

Exam Style Questions:

Question 1:

(c) **(i)** Complete the following binary addition. Show your working.

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

[2]

(ii) Describe the error that occurred when you added the binary numbers in **part (c)(i)**.

....

....

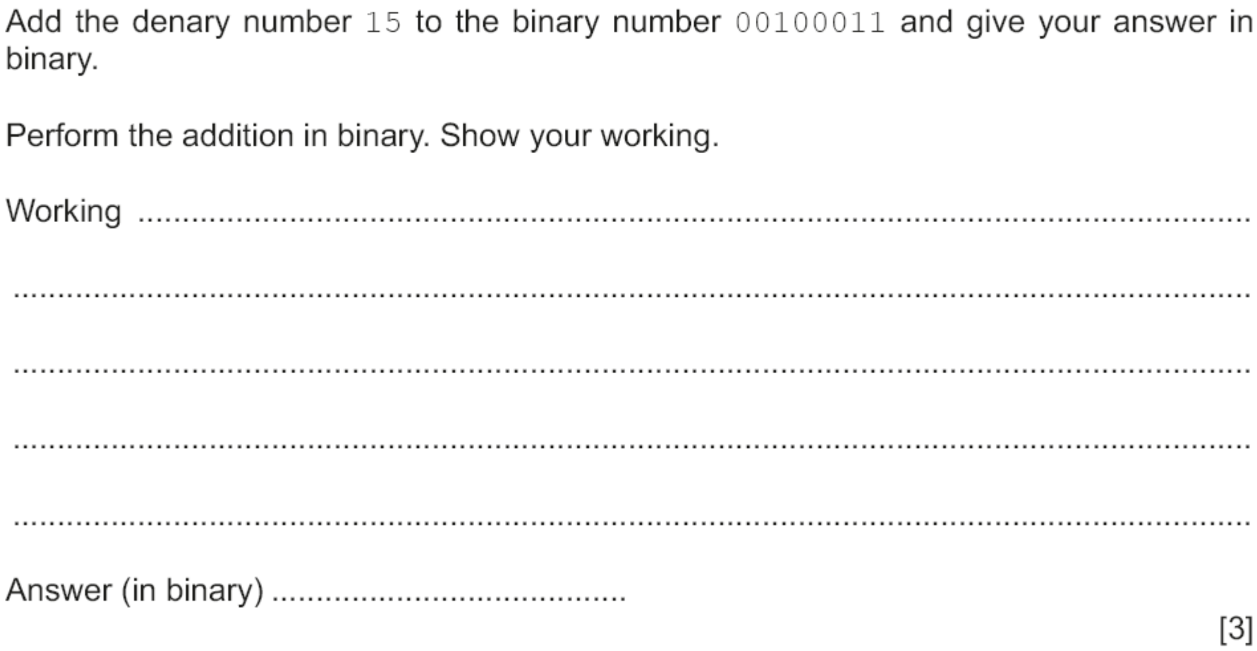
....

.... [2]

Answer:

1(c)(i)	1 mark for answer, 1 mark for working (e.g. carries) 1 0 0 1 1 0 1 0 + 1 1 1 1 0 1 1 1 1 1 0 0 1 0 0 0 1 1 1 1 1 1 1	**2**	
1(c)(ii)	1 mark per bullet • overflow • the answer cannot be represented in the current number of bits	**2**	

Question 2:



Add the denary number `15` to the binary number `00100011` and give your answer in binary.

Perform the addition in binary. Show your working.

Working ....

....

....

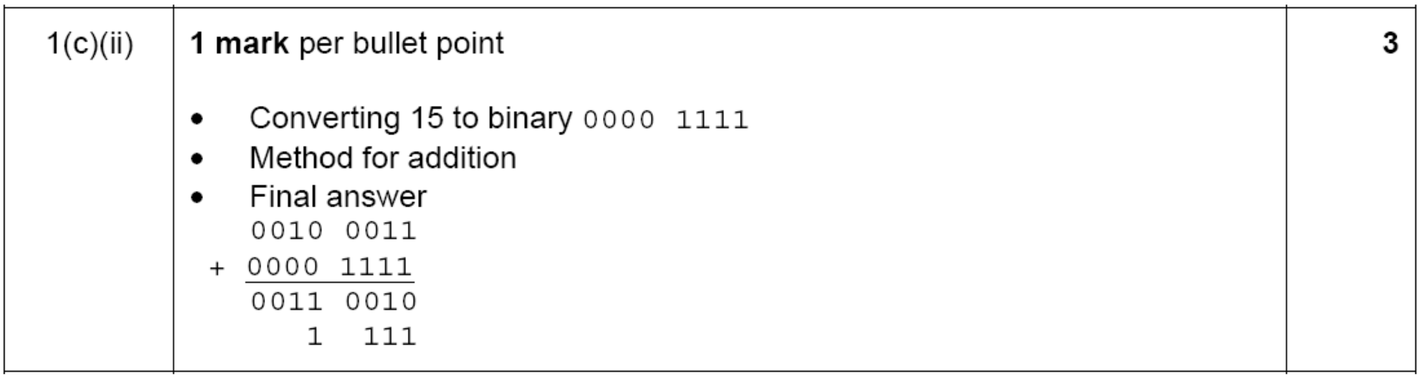
....

....

Answer (in binary) ....

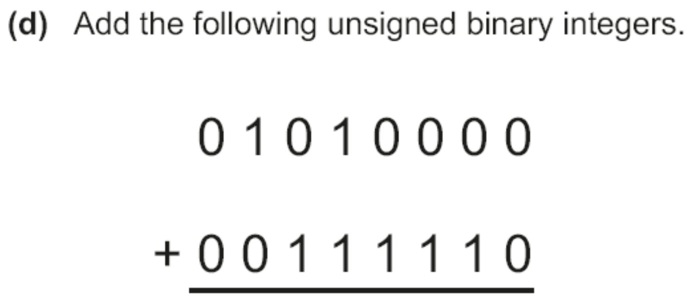
[3]

Answer:



1(c)(ii)	**1 mark** per bullet point
	• Converting 15 to binary `0000 1111`
	• Method for addition
	• Final answer
	`0010 0011`
	`+ 0000 1111`
	`0011 0010`
	`1 111`
		3

Question 3:

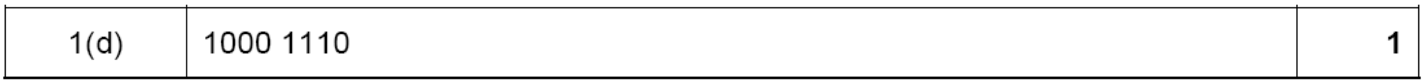


(d) Add the following unsigned binary integers.

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

[1]

Answer:



1(d)	1000 1110	**1**

Question 4:

(b) **(i)** Perform the following binary addition. Show your working.

$$\begin{array}{r} 10101010 \\ \underline{+\ 00110111} \end{array}$$

[2]

(ii) State how an overflow can occur when adding two binary integers.

....

.... [1]

Answer:

1(b)(i)	**1 mark** for answer **1 mark** for working e.g. $\begin{array}{l} 1010\ 1010 \\ \underline{0011\ 0111} \\ \underline{\mathbf{1110\ 0001}} \\ \quad 1\ 1\ 1\ \ 1\ 1\ 1 \end{array}$	**2**
1(b)(ii)	The result is a larger number than can be stored in the given number of bits. // The result is greater than <u>255</u>	**1**

Question 5:

(e) The binary contents of **two** registers are:

Register 1	0	0	1	1	1	1	0	1
Register 2	0	0	1	0	1	1	0	1

(i) Add the contents of **Register 1** and **Register 2**. Show your working.

Answer ....

[2]

Answer:

4(e)(i)	**1 mark** for working, **1 mark** for answer 0011 1101 +0010 1101 **0110 1010** 111 1 1	**2**

1.1.5 Logical Binary Shifts:

NOTE: Logical Binary Shifts is a newly added topic in the Computer Science (2210) syllabus for the session 2023–2025.

According to the new syllabus, we will look at the method of performing a logical binary shift on a positive 8-bit binary integer and understanding the effect this has on the positive binary integer.

- Computers can carry out a logical shift on a sequence of binary numbers.
- Logical shift means moving/shifting the binary number either to the left or to the right.
- As bits are shifted/moved, any empty positions of bits are replaced with a zero.

Throughout this section, as per syllabus requirement, we will only use 8-bit registers.

Logical Left Shift:

- Each shift left is equivalent to multiplying the binary number by 2.

Logical Right Shift:

- Each shift right is equivalent to dividing the binary number by 2.

Multiple Logical Shifts & Limit:

- Multiple right & left shifts can be performed on the same 8-bit binary integer.
- However, there is clearly a limit to the number of shifts that can be carried out if the binary number is stored in an 8-bit register.
- Eventually after a number of shifts the register would only contain zeroes.

For Example:

- If we shift 0 1 1 1 0 0 0 0 (denary 112) five places left in an 8-bit register:

Each shift left is equivalent to multiplying the binary number by 2.
Five shifts left would be equivalent to multiplying the binary number by 2^5 = 32
So, 0 1 1 1 0 0 0 0 (denary 112) would be multiplied by 2^5 / 32.

- The register would end up with 0 0 0 0 0 0 0 0 if we shift five places left.

This makes it seem as denary 112 x 32 (2^5) = 0! which is illogical because register contains 0 values and multiplication of 112 x 32 (2^5) cannot be equal to 0.

This results in the generation of an error message.

Most Significant & Least Significant Bit:

- The left-most bit is referred to as the Most Significant Bit.
- The right-most bit is referred to as the Least Significant Bit.

Process of Logical Left Shift:

- The bits shifted from the left end of the register are lost and zeros are shifted in at the opposite right end of the register.
- The positive binary integer is multiplied by 2 according to the left shift performed.
- In this case, the most significant bit(s) are lost.

Process of Logical Right Shift:

- The bits shifted from the right end of the register are lost and zeros are shifted in at the opposite left end of the register.
- The positive binary integer is divided by 2 according to the right shift performed.
- In this case, the least significant bit(s) are lost.

Examples of Logical Left Shift:

(i) Performing a single logical left shift (1 time):

The original binary number **0 0 0 1 0 1 0 1** is shifted one place to the left.

The denary number **21** is **0 0 0 1 0 1 0 1** in binary that is put into an 8-bit register:

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

- A logical left shift is performed on the register.
- As a result, the left-most bit (most significant bit) is lost following a logical left shift:

Left-most bit is lost

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

- This means that the left-most bit ‘0’ is lost & the remaining original 7 bits of binary number 0 0 1 0 1 0 1 are moved 1 place to the left:

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

The remaining 7 bits are moved 1 place to left each

- This results in an empty right-most bit (least significant bit) position:

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

Right-most bit position is empty

- This empty right-most bit position is filled with a ‘0’:

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

Right-most bit position is filled with 0

- Therefore, left-most bit is lost, and right-most bit position is now filled with a '0' following the logical left shift. This binary number is obtained:

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

- The value of the binary bits is now denary number **21 x 2^1 = 42**
- Therefore, the positive binary integer is multiplied by 2 as a result of the logical left shift performed.

You can now double check your answer by:

- Calculating the denary value of new binary number **0 0 1 0 1 0 1 0** that is **32 + 8 + 2 = 42**.
- The answer of original denary value **21 x 2^1 (according to one left shift)** = **42**

Hence this confirms that the logical left shift is performed correctly.

(ii) Performing a double logical left shift (2 times):

The original binary number **0 0 0 1 0 1 0 1** is shifted two places to the left.

The denary number **21** is **0 0 0 1 0 1 0 1** in binary. Putting this into an 8-bit register:

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

- Double logical left shift is performed on the register.
- As a result, the two left-most bits (most significant bits) are lost following the shift:

Left-most bits are lost

128	64	32	16	8	4	2	1
◌	◌	0	1	0	1	0	1

- This means that the left-most bits '0' & '0' are lost & the remaining original 6 bits of binary number 0 1 0 1 0 1 are moved 1 place to the left each:

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

←

The remaining 6 bits are moved 1 place to left each

- This results in two empty right-most bits (least significant bits) position:

128	64	32	16	8	4	2	1
0	1	0	1	0	1	○	○

Right-most bits position is empty

- These empty right-most bits positions are filled with '0''s both:

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

Right-most bits position is filled with 0's

- Therefore, two left-most bits are lost, and two right-most bits position is now filled with '0's following the double logical left shift. This binary number is obtained:

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

- The value of the binary bits is now denary number **21 x 2^2 = 84**
- Therefore, the positive binary integer is multiplied by 2^2 as a result of the double logical left shift performed.

You can now double check your answer by:

- Calculating the denary value of new binary number **0 1 0 1 0 1 0 0** that is **64 + 16 + 4 = 84**.
- The answer of original denary value **21 x 2^2 (according to two left shifts) = 84**

Hence this confirms that the double logical left shift is performed correctly.

(iii) Performing a triple logical left shift (3 times):

The original binary number **0 0 0 1 0 1 0 1** is shifted three places to the left.

The denary number **21** is **0 0 0 1 0 1 0 1** in binary. Putting this into an 8-bit register:

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

- Triple logical left shift is performed on the register.
- As a result, the three left-most bits (most significant bits) are lost following the shift:

Left-most bits are lost

128	64	32	16	8	4	2	1
○	○	○	1	0	1	0	1

- This means that the left-most bits ‘0’, ‘0’ & ‘0’ are lost & the remaining original 5 bits of binary number 1 0 1 0 1 are moved 1 place to the left each:

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

←

The remaining 5 bits are moved 1 place to left each

- This results in three empty right-most bits (least significant bits) position:

128	64	32	16	8	4	2	1
1	0	1	0	1	○	○	○

Right-most bits position is empty

- These empty right-most bits positions are filled with ‘0’s:

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

Right-most bits position is filled with 0’s

- Therefore, three left-most bits are lost, and three right-most bits position is now filled with '0's following the triple logical left shift. This binary number is obtained:

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

- The value of the binary bits is now denary number **21 x 2^3 = 168**
- Therefore, the positive binary integer is multiplied by 2^3 as a result of the triple logical left shift performed.

You can now double check your answer by:

- Calculating the denary value of new binary number **1 0 1 0 1 0 0 0** that is **128 + 32 + 8 = 168**.
- The answer of original denary value **21 x 2^3 (according to three left shifts) = 168**

Hence this confirms that the triple logical left shift is performed correctly.

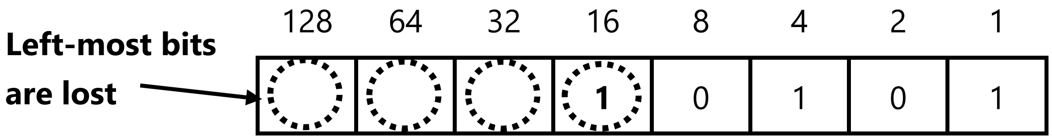
(iv) Performing a tetra logical left shift (4 times):

The original binary number **0 0 0 1 0 1 0 1** is shifted four places to the left.

The denary number **21** is **0 0 0 1 0 1 0 1** in binary. Putting this into an 8-bit register:

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

- Tetra logical left shift is performed on the register.
- As a result, the four left-most bits (most significant bits) are lost following the shift:



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

- This means that the left-most bits '0', '0', '0' & '1' are lost & the remaining original 4 bits of binary number 0 1 0 1 are moved 1 place to the left each.

This loss of left-most 1-bit following a shift operation causes an error.

The loss of left-most 0-bits does not cause an error but as soon as a left-most 1-bit is lost, an error occurs.

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

←

The remaining 4 bits are moved 1 place to left each

- This results in four empty right-most bits (least significant bits) position:

128	64	32	16	8	4	2	1
0	1	0	1	○	○	○	○

Right-most bits position is empty

- These empty right-most bits positions are filled with ‘0’s:

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

Right-most bits position is filled with 0’s

- Therefore, four left-most bits are lost, and four right-most bits position is now filled with ‘0’s following the tetra logical left shift. This binary number is obtained:

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

- The value of the binary bits is now denary number **21 x 2^4 = 336**
- Therefore, the positive binary integer is multiplied by 2^4 as a result of the tetra logical left shift performed.
- However, this is incorrect and an error has occurred.

You can now double check the error by:

- Calculating the denary value of new binary number **0 1 0 1 0 0 0 0** that is **64 + 16 = 80**.
- The answer of original denary value **21 x 2^4 (according to four left shifts) = 336**

Hence this confirms that the tetra logical left shift has caused an error.

This error is because we have exceeded the maximum number of left shifts possible using this 8-bit register.

Error in Logical Left Shift:

- The logical left shifts can be performed as many times as it does not cause loss of left-most 1-bit in the shift operation.
- The loss of left-most 1-bit following a shift operation causes an error.
- Therefore, shifting process can continue (error-free) as long as only left-most 0-bits are being lost.

So, the loss of left-most 0-bits does not cause an error but as soon as a left-most 1-bit is lost, an error occurs.

Examples of Errors in Logical Left Shift:

1) 0 0 0 0 0 0 0 1:

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

- This binary number can be shifted left 7 times because there are 7 left-most 0-bits in this number.

2) 0 0 0 0 0 0 1 1:

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

- This binary number can be shifted left 6 times because there are 6 left-most 0-bits in this number.

3) 0 0 0 0 0 1 1 1:

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

- This binary number can be shifted left 5 times because there are 5 left-most 0-bits in this number.

4) 0 0 0 0 1 1 1 1:

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

- This binary number can be shifted left 4 times because there are 4 left-most 0-bits in this number.

5) 0 0 0 1 1 1 1 1:

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

- This binary number can be shifted left 3 times because there are 3 left-most 0-bits in this number.

6) 0 0 1 1 1 1 1 1:

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

- This binary number can be shifted left 2 times because there are 2 left-most 0-bits in this number.

7) 0 1 1 1 1 1 1 1:

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

- This binary number can be shifted left 1 time because there is 1 left-most 0-bit in this number.

8) 1 1 1 1 1 1 1 1:

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

- This binary number cannot be shifted left even once because there is 0 left-most 0-bit in this number.

Examples of Logical Right Shift:

(i) Performing a single logical right shift (1 time):

The original binary number **1 1 0 0 1 0 0 0** is shifted one place to the right.

The denary number **200** is **1 1 0 0 1 0 0 0** in binary that is put into an 8-bit register:

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

- A logical right shift is performed on the register.
- As a result, the right-most bit (least significant bit) is lost following a logical right shift:

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

Right-most bit is lost

- This means that the right-most bit '0' is lost & the remaining original 7 bits of binary number 1 1 0 0 1 0 0 are moved 1 place to the right:

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

The remaining 7 bits are moved 1 place to right each

- This results in an empty left-most bit (most significant bit) position:

Left-most bit position is empty

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

- This empty left-most bit position is filled with a '0':

Left-most bit position is filled with 0

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

- Therefore, right-most bit is lost, and left-most bit position is now filled with a '0' following the logical right shift. This binary number is obtained:

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

- The value of the binary bits is now denary number **200 ÷ 2^1 = 100**
- Therefore, the positive binary integer is divided by 2 as a result of the logical right shift performed.

You can now double check your answer by:

- Calculating denary value of new binary number **0 1 1 0 0 1 0 0** that is **64 + 32 + 4 = 100**.
- The answer of original denary value **200 ÷ 2^1 (according to one right shift) = 100**

Hence this confirms that the logical right shift is performed correctly.

(ii) Performing a double logical right shift (2 times):

The original binary number **1 1 0 0 1 0 0 0** is shifted two places to the right.

The denary number **200** is **1 1 0 0 1 0 0 0** in binary. Putting this into an 8-bit register:

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

- Double logical right shift is performed on the register.
- As a result, the two right-most bits (least significant bits) are lost following the shift:

128	64	32	16	8	4	2	1
1	1	0	0	1	0	◌	◌

Right-most bits are lost

- This means that the right-most bits '0' & '0' are lost & the remaining original 6 bits of binary number 1 1 0 0 1 0 are moved 1 place to the right each:

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

The remaining 6 bits are moved 1 place to right each

- This results in two empty left-most bits (most significant bits) position:

Left-most bits position is empty

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

- These empty left-most bits positions are filled with ‘0’’s both:

Left-most bits position is filled with 0’s

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

- Therefore, two right-most bits are lost, and two left-most bits position is now filled with ‘0’s following the double logical right shift. This binary number is obtained:

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

- The value of the binary bits is now denary number **200 ÷ 2^2 = 50**
- Therefore, the positive binary integer is divided by 2^2 as a result of the double logical right shift performed.

You can now double check your answer by:

- Calculating denary value of new binary number **0 0 1 1 0 0 1 0** that is **32 + 16 + 2 = 50**.
- The answer of original denary value **200 ÷ 2^2 (according to two right shifts) = 50**

Hence this confirms that the double logical right shift is performed correctly.

(iii) Performing a triple logical right shift (3 times):

The original binary number **1 1 0 0 1 0 0 0** is shifted three places to the right.

The denary number **200** is **1 1 0 0 1 0 0 0** in binary. Putting this into an 8-bit register:

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

- Triple logical right shift is performed on the register.
- As a result, the three right-most bits (least significant bits) are lost following the shift:

128	64	32	16	8	4	2	1
1	1	0	0	1	◌	◌	◌

Right-most bits are lost

- This means that the right-most bits ‘0’, ‘0’ & ‘0’ are lost & the remaining original 5 bits of binary number 1 1 0 0 1 are moved 1 place to the right each:

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

The remaining 5 bits are moved 1 place to right each

- This results in three empty left-most bits (most significant bits) position:

Left-most bits position is empty

128	64	32	16	8	4	2	1
◌	◌	◌	1	1	0	0	1

- These empty left-most bits positions are filled with ‘0’’s:

Left-most bits position is filled with 0’s

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

- Therefore, three right-most bits are lost, and three left-most bits position is now filled with '0's following the triple logical right shift. This binary number is obtained:

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

- The value of the binary bits is now denary number **200 ÷ 2^3 = 25**
- Therefore, the positive binary integer is divided by 2^3 as a result of the triple logical right shift performed.

You can now double check your answer by:

- Calculating denary value of new binary number **0 0 0 1 1 0 0 1** that is **16 + 8 + 1 = 25**.
- The answer of original denary value **200 ÷ 2^3 (according to three right shifts) = 25**

Hence this confirms that the triple logical right shift is performed correctly.

(iii) Performing a tetra logical right shift (4 times):

The original binary number **1 1 0 0 1 0 0 0** is shifted four places to the right.

The denary number **200** is **1 1 0 0 1 0 0 0** in binary. Putting this into an 8-bit register:

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

- Tetra logical right shift is performed on the register.
- As a result, the four right-most bits (least significant bits) are lost following the shift:

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

Right-most bits are lost

- This means that the right-most bits '0', '0', '0' & '1' are lost & the remaining original 4 bits of binary number 1 1 0 0 are moved 1 place to the right each.

This loss of right-most 1-bit following a shift operation causes an error.

The loss of right-most 0-bits does not cause an error but as soon as a right-most 1-bit is lost, an error occurs.

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

The remaining 4 bits are moved 1 place to right each

- This results in four empty left-most bits (most significant bits) position:

Left-most bits position is empty

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

- These empty left-most bits positions are filled with ‘0’’s:

Left-most bits position is filled with 0’s

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

- Therefore, four right-most bits are lost, and four left-most bits position is now filled with ‘0’s following the tetra logical right shift. This binary number is obtained:

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

- The value of the binary bits is now denary number **200 ÷ 2^4 = 12.5**
- Therefore, the positive binary integer is divided by 2^4 as a result of the tetra logical right shift performed.
- However, this is incorrect and an error has occurred.

You can now double check the error by:

- Calculating denary value of new binary number **0 0 0 0 1 1 0 0** that is **8 + 4 = 12**.
- The answer of original denary value **200** ÷ **2^4 (according to four right shifts) = 12.5**

Hence this confirms that the tetra logical right shift has caused an error.

This error is because we have exceeded the maximum number of right shifts possible using this 8-bit register.

Error in Logical Right Shift:

- The logical right shifts can be performed as many times as it does not cause loss of right-most 1-bit in the shift operation.
- The loss of right-most 1-bit following a shift operation causes an error.
- Therefore, shifting process can continue (error-free) as long as only right-most 0-bits are being lost.

So, the loss of right-most 0-bits does not cause an error but as soon as a right-most 1-bit is lost, an error occurs.

Examples of Errors in Logical Right Shift:

1) 1 0 0 0 0 0 0 0:

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

- This binary number can be shifted right 7 times because there are 7 right-most 0-bits in this number.

2) 1 1 0 0 0 0 0 0:

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

- This binary number can be shifted right 6 times because there are 6 right-most 0-bits in this number.

3) 1 1 1 0 0 0 0 0:

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

- This binary number can be shifted right 5 times because there are 5 right-most 0-bits in this number.

4) 1 1 1 1 0 0 0 0:

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

- This binary number can be shifted right 4 times because there are 4 right-most 0-bits in this number.

5) 1 1 1 1 1 0 0 0:

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

- This binary number can be shifted right 3 times because there are 3 right-most 0-bits in this number.

6) 1 1 1 1 1 1 0 0:

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

- This binary number can be shifted right 2 times because there are 2 right-most 0-bits in this number.

7) 1 1 1 1 1 1 1 0:

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

- This binary number can be shifted right 1 time because there is 1 right-most 0-bit in this number.

8) 1 1 1 1 1 1 1 1:

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

- This binary number cannot be shifted right even once because there is 0 right-most 0-bit in this number.

Exam Style Questions:

Question 1:

(d) A **three-place logical shift** to the **left** is performed on the following positive binary integer.

Show the result of this logical shift.

0 1 1 1 1 1 1 0

.... [1]

Answer:

3(d)	11110000	**1**

Question 2:

(d) The machines have a counter to record the number of cake tins filled. Each time a cake tin is filled, the counter is increased by 1. The value is stored in an 8-bit register, the current value is shown.

0	0	0	0	1	0	0	1

(i) Show the value of the binary number after another five cake tins have been filled.

[1]

(iii) A **two-place logical shift** to the **left** is performed on the binary number shown in **part (d)**.

Show the result of this logical shift.

[1]

(iv) State the mathematical result of a **one-place logical shift** to the **right** on a binary number.

....

.... [1]

Answer:

4(d)(i)	0 0 0 0 1 1 1 0	1	
4(d)(iii)	0 0 1 0 0 1 0 0	1	
4(d)(iv)	Division by 4	1	

Question 3:

A computer uses an 8-bit register.

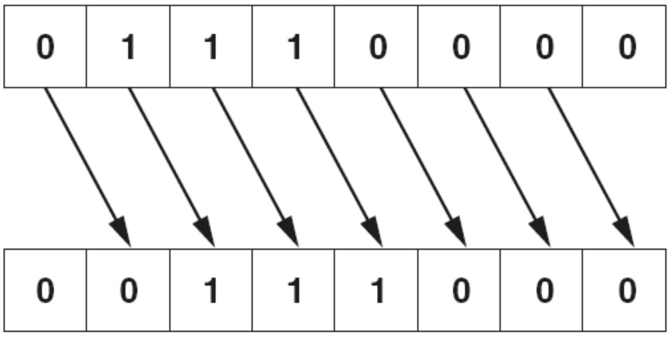
The 8-bit register contains binary integers.

(a) Write the denary (base 10) value represented by:

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

....[1]

(b) All the bits in the register are shifted **one** place to the **right** as shown below.



0	1	1	1	0	0	0	0
0	0	1	1	1	0	0	0

Write the denary number that is represented after this shift.

....[1]

(c) State the effect the shift to the right had on the original denary number from **part (a)**.

....[1]

(d) The original number in **part (a)** is shifted **three** places to the **right**.

(i) Show the new binary number:

[1]

(ii) Write the equivalent denary number.

....[1]

(e) Describe the problems that could be caused if the original binary number in **part (a)** is shifted **five** places to the **right**.

....

....

....

....

....[2]

Answer:

(a) 112 [1]

(b) 56 [1]

(c) divided by 2 // value 112 was halved // multiplied by 0.5 [1]

(d) (i)

0	**0**	**0**	**0**	**1**	**1**	**1**	**0**

[1]

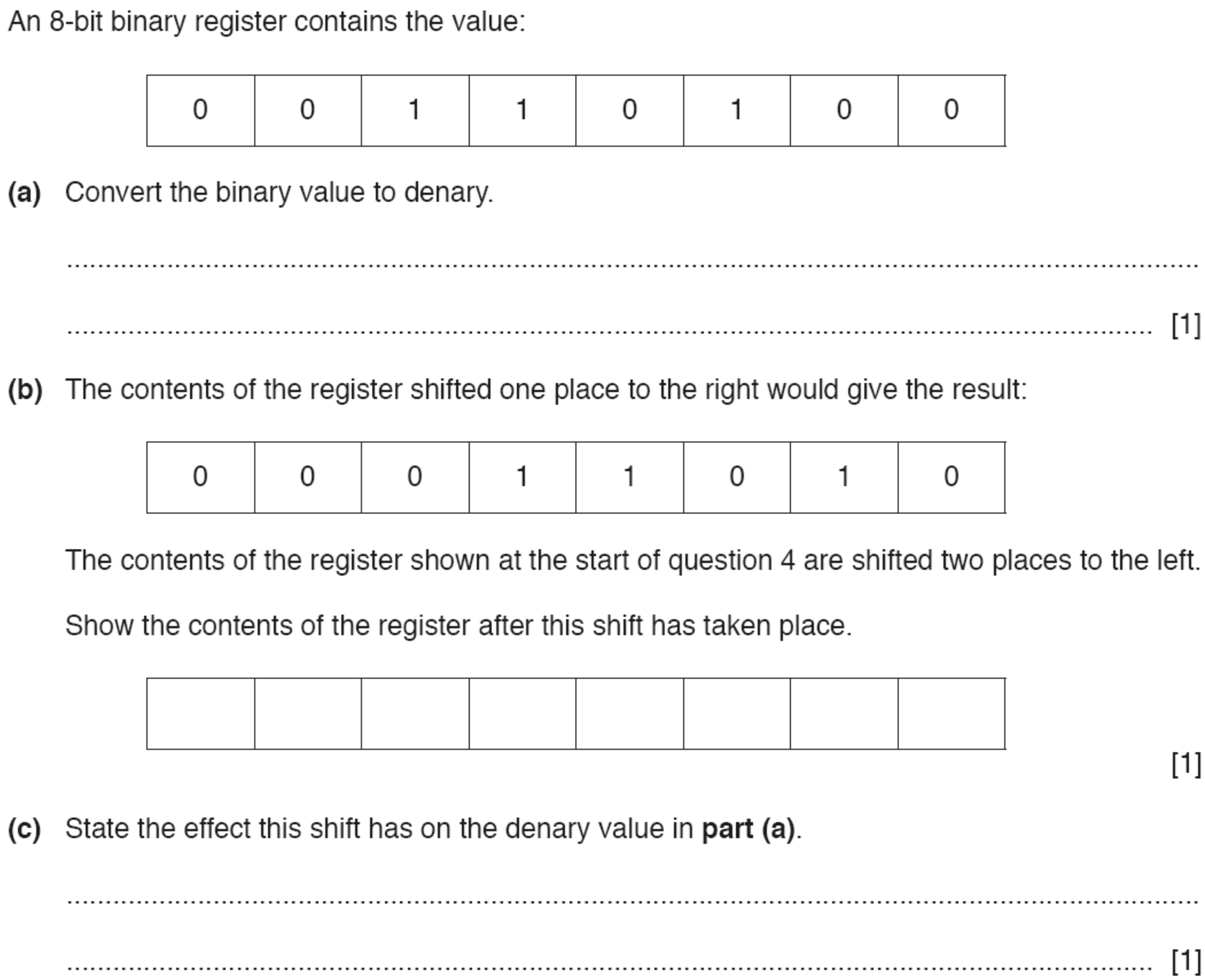
(ii) 14 [1]

(e) Any **two** from:

- run out of places to the right of register / at the end of register
- right-most 1 would be lost
- number would become 3 instead of 3.5
- loss of precision

[2]

Question 4:



An 8-bit binary register contains the value:

0	0	1	1	0	1	0	0

(a) Convert the binary value to denary.

....

.... [1]

(b) The contents of the register shifted one place to the right would give the result:

0	0	0	1	1	0	1	0

The contents of the register shown at the start of question 4 are shifted two places to the left.

Show the contents of the register after this shift has taken place.

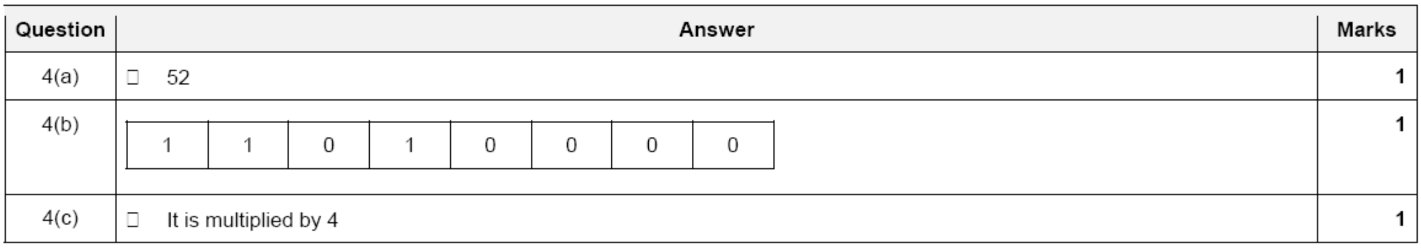
[1]

(c) State the effect this shift has on the denary value in **part (a)**.

....

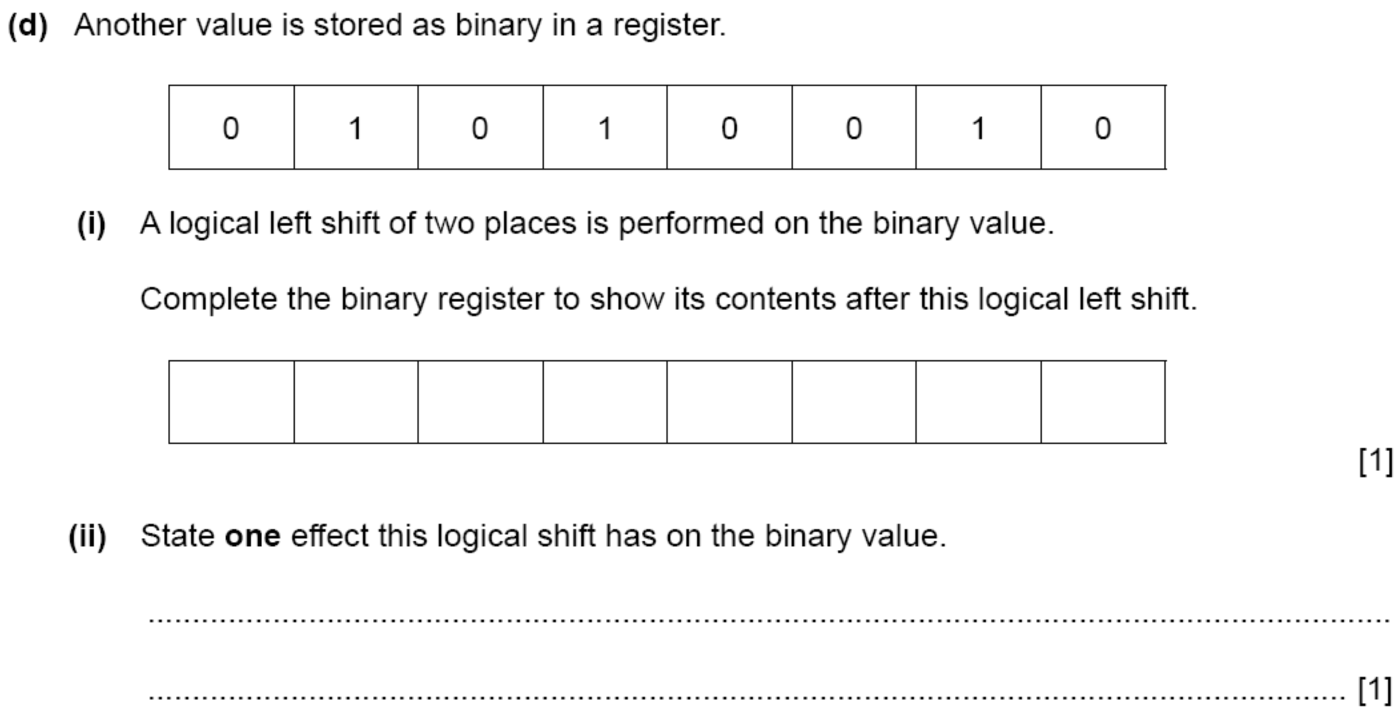
.... [1]

Answer:



Question	Answer	Marks
4(a)	□ 52	1
4(b)	1 1 0 1 0 0 0 0	1
4(c)	□ It is multiplied by 4	1

Question 5:



(d) Another value is stored as binary in a register.

0	1	0	1	0	0	1	0

(i) A logical left shift of two places is performed on the binary value.

Complete the binary register to show its contents after this logical left shift.

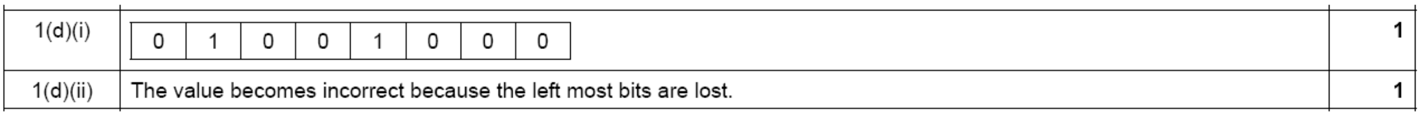
[1]

(ii) State **one** effect this logical shift has on the binary value.

....

.... [1]

Answer:



1(d)(i)	0 | 1 | 0 | 0 | 1 | 0 | 0 | 0	**1**
1(d)(ii)	The value becomes incorrect because the left most bits are lost.	**1**

1.1.6 Two’s Complement (binary numbers):

NOTE: Two’s Complement is a newly added topic in the Computer Science (2210) syllabus for the session 2023–2025.

According to the new syllabus, we will look at the method of using two’s complement to represent positive and negative 8-bit binary integers.

This section will also discuss method of:

- Converting a positive binary or denary integer to a two’s complement 8-bit integer and vice versa.
- Converting a negative binary or denary integer to a two’s complement 8-bit integer and vice versa.

Up until now, we have assumed all binary numbers are positive integers. To allow the possibility of representing negative integers we make use of **two’s complement** method.

Throughout this section, as per syllabus requirement, we will only use 8-bit registers.

In the two’s complement, the left-most bit is changed to a negative value in the binary headings.

In case of an 8-bit binary number, the left-most bit value 128 is changed to negative –128 whilst all the other binary headings remain same as shown below:

–128	64	32	16	8	4	2	1

- The 2^7 in the 8-bit register which is +128 (positive) is changed to –128 (negative).

Unsigned 8-Bit Binary Integer (normal integer):

- In an unsigned 8-bit binary integer, left-most bit heading = 128
- The range of possible numbers = 0 (00000000) to 255 (11111111)
- The smallest/lowest denary value that an 8-bit unsigned integer can represent = 0
- The largest/highest denary value that an 8-bit unsigned integer can represent = 255
- The number of different values that an 8-bit unsigned binary integer can represent = 256

Signed 8-Bit Binary Integer/8-Bit Two's Complement Integer:

- In an 8-bit two's complement integer, left-most bit heading = –128
- The range of possible numbers = –128 (10000000) to +127 (01111111)
- The smallest/lowest denary value an 8-bit two's complement integer can represent = –128
- The largest/highest denary value an 8-bit two's complement integer can represent = +127
- The number of different values that an 8-bit two's complement integer can represent = 256

Basics of Two's Complement Method:

When applying two's complement to a binary number:

- The left-most bit always determines the sign of the binary number.
- The 1-value in the left-most bit indicates a negative number.
- The 0-value in the left-most bit indicates a positive number.

Example:

The binary number **0 0 1 1 0 0 1 1** contains 0-value in the left most-bit indicating a positive number.

–128	64	32	16	8	4	2	1
0	0	1	1	0	0	1	1

- **32 + 16 + 2 + 1 = 51 denary (positive)**

The binary number **1 1 0 0 1 1 1 1** contains 1-value in the left most-bit indicating a negative number.

–128	64	32	16	8	4	2	1
1	1	0	0	1	1	1	1

- **(–128) + (64 + 8 + 4 + 2 + 1) = (–128) + (79)**
- **–128 + 79 = –49 denary (negative)**

Using Two's Complement to represent/write positive 8-bit binary integers:

Steps:

It is very simple and straightforward:

1) Since the left-most bit determines the sign of the binary number; the 0-value in the left-most bit indicates a positive number.
2) Write a '0' in the left-most bit under the heading of –128.
3) Fill the rest of columns by writing '1's in appropriate places according to the binary number you want to represent.

Examples:

1) The binary number **0 0 0 1 0 0 1 1** contains 0-value in the left most-bit indicating a positive number.

–128	64	32	16	8	4	2	1
0	0	0	1	0	0	1	1

- This is a positive binary number because the left-most bit contains a '0'.

2) The binary number **0 1 1 1 0 0 0 1** contains 0-value in the left most-bit indicating a positive number.

–128	64	32	16	8	4	2	1
0	1	1	1	0	0	0	1

- This is also a positive binary number because the left-most bit contains a '0'.

3) The binary number **0 1 1 1 1 1 1 1** contains 0-value in the left most-bit indicating a positive number.

–128	64	32	16	8	4	2	1
0	1	1	1	1	1	1	1

- This is also a positive binary number because the left-most bit contains a '0'.

Converting a positive denary integer to a two’s complement 8-bit integer:

Steps:

1) Since the left-most bit determines the sign of the binary number; the 0-value in the left-most bit indicates a positive number.
2) Write a ‘0’ in the left-most bit under the heading of –128.
3) Fill the rest of columns by writing ‘1’s in appropriate places so that when the values of headings with ‘1’s are added, the total is equal to the denary number you want to represent.

Example:

1) Converting 38 to 8-bit binary number using two’s complement format:

Since the denary number 38 is positive, we will write a ‘0’ in the left-most bit under the heading of –128. Then we will write ‘1’s in appropriate places so when they are added, the total is equal to 38.

–128	64	32	16	8	4	2	1
0	0	1	0	0	1	1	0

- **32 + 4 + 2 = 38 denary (positive)**

2) Converting 125 to 8-bit binary number using two’s complement format:

Since the denary number 125 is positive, we will write a ‘0’ in the left-most bit under the heading of –128. Then we will write ‘1’s in appropriate places so when they are added, total is equal to 125.

–128	64	32	16	8	4	2	1
0	1	1	1	1	1	0	1

- **64 + 32 + 16 + 8 + 4 + 1 = 125 denary (positive)**

Converting a positive two's complement 8-bit integer to a positive denary integer:

Steps:

1) Since the left-most bit determines the sign of the binary number; the 0-value in the left-most bit indicates a positive number.
2) Simply add all the heading values filled with '1's to obtain the positive denary integer.

Example:

1) Converting 0 1 1 0 1 1 1 0 in two's complement binary number to denary:

Since the left-most bit contains a '0', this two's complement binary number is positive. We will simply add all the heading values which contains '1's to obtain the positive denary integer.

–128	64	32	16	8	4	2	1
0	1	1	0	1	1	1	0

- **64 + 32 + 8 + 4 + 2 = 110 denary (positive)**

2) Converting 0 0 1 1 1 1 1 1 in two's complement binary number to denary:

Since the left-most bit contains a '0', this two's complement binary number is positive. We will simply add all the heading values which contains '1's to obtain the positive denary integer.

–128	64	32	16	8	4	2	1
0	0	1	1	1	1	1	1

- **32 + 16 + 8 + 4 + 2 + 1 = 63 denary (positive)**

Using Two's Complement to represent/write negative 8-bit binary integers:

Steps:

1) Since the left-most bit determines the sign of the binary number; the 1-value in the left-most bit indicates a positive number.
2) Write a '1' in the left-most bit under the heading of –128.
3) Fill the rest of columns by writing '1's in appropriate places according to the binary number you want to represent.

NOTE: A two's complement number with a 1-value in the –128 column must represent a negative binary number.

Examples:

1) The binary number **1 0 0 1 0 0 1 1** contains 1-value in the left most-bit indicating a negative number.

–128	64	32	16	8	4	2	1
1	0	0	1	0	0	1	1

- ■ This is a negative binary number because the left-most bit contains a '1'.

2) The binary number **1 1 1 1 0 0 0 1** contains 1-value in the left most-bit indicating a negative number.

–128	64	32	16	8	4	2	1
1	1	1	1	0	0	0	1

- ■ This is also a negative binary number because the left-most bit contains a '1'.

3) The binary number **1 0 0 0 0 0 0 0** contains 1-value in the left most-bit indicating a negative number.

–128	64	32	16	8	4	2	1
1	0	0	0	0	0	0	0

- ■ This is also a negative binary number because the left-most bit contains a '1'.

Converting a negative two's complement 8-bit integer to a negative denary integer:

Steps:

1) Since the left-most bit determines the sign of the binary number; the 1-value in the left-most bit indicates a negative number.
2) Simply add all the heading values filled with '1's to obtain the negative denary integer.

Example:

1) Converting 1 0 0 1 0 0 1 1 in two's complement binary number to denary:

Since the left-most bit contains a '1', this two's complement binary number is negative. We will simply add all the heading values which contain '1's to obtain the negative denary integer.

–128	64	32	16	8	4	2	1
1	0	0	1	0	0	1	1

- **–128 + 16 + 2 + 1 = –109 denary (negative)**

2) Converting 1 1 1 0 0 1 0 0 in two's complement binary number to denary:

Since the left-most bit contains a '1', this two's complement binary number is negative. We will simply add all the heading values which contain '1's to obtain the negative denary integer.

–128	64	32	16	8	4	2	1
1	1	1	0	0	1	0	0

- **–128 + 64 + 32 + 4 = –28 denary (negative)**

3) Converting 1 1 1 1 0 1 0 1 in two's complement binary number to denary:

Since the left-most bit contains a '1', this two's complement binary number is negative. We will simply add all the heading values which contain '1's to obtain the negative denary integer.

–128	64	32	16	8	4	2	1
1	1	1	1	0	1	0	1

- **–128 + 64 + 32 + 16 + 4 + 1 = –11 denary (negative)**

Converting a negative denary integer to a two's complement 8-bit integer:

Method 1:

1) Since the left-most bit determines the sign of the binary number; the 1-value in the left-most bit indicates a negative number.
2) Write a '1' in the left-most bit under the heading of –128.
3) Use the mathematical formulas:

– 128 + X = – Denary Number (negative)

X = + 128 – Denary Number (negative)

4) Fill the rest of columns by writing '1's in appropriate places so that when the values of headings with '1's are added (excluding '1' in the left-most bit), the total is equal to the value of X.
5) When you add all the values of headings with '1's (including '1' in the left-most bit), the total is equal to the negative denary integer

Example:

1) Converting –67 to 8-bit binary number using two's complement format (Method 1):

Since the denary number –67 is negative, we will write a '1' in the left-most bit under the heading of –128. Apply the formulas:

- –128 + X = – Denary Number
- –128 + X = – 67
- X = 128 – 67
- X = 61

How can 61 be achieved? By adding:

- 32 + 16 + 8 + 4 + 1 = 61

Therefore, we will write '1's in appropriate places so when they are added, the total is equal to 61.

–128	64	32	16	8	4	2	1
	0	1	1	1	1	0	1

- **32 + 16 + 8 + 4 + 1 = 61**

Now for the final step, we will simply write a ‘1’ in the left-most bit:

–128	64	32	16	8	4	2	1
1	0	1	1	1	1	0	1

- **–128 + 32 + 16 + 8 + 4 + 1 = –67 denary (negative)**

2) Converting –79 to 8-bit binary number using two’s complement format (Method 1):

Since the denary number –79 is negative, we will write a ‘1’ in the left-most bit under the heading of –128. Apply the formulas:

- –128 + X = – Denary Number
- –128 + X = – 79
- X = 128 – 79
- X = 49

How can 49 be achieved? By adding:

- 32 + 16 + 1 = 49

Therefore, we will write ‘1’s in appropriate places so when they are added, the total is equal to 49.

–128	64	32	16	8	4	2	1
	0	1	1	0	0	0	1

- **32 + 16 + 1 = 49**

Now for the final step, we will simply write a ‘1’ in the left-most bit:

–128	64	32	16	8	4	2	1
1	0	1	1	0	0	0	1

- **–128 + 32 + 16 + 1 = –79 denary (negative)**

Method 2:

1) First, we will convert the given denary number to a positive binary value.
2) We will then invert each binary value, which means swap the 1s with 0s & swap the 0s with 1s for each bit.
3) Then we will add 1 to that binary number we inverted (addition of binary numbers concept).
4) This will give us the negative binary (two's complement) integer of the negative denary integer.
5) When you add all the values of headings with '1's, the total is equal to the negative denary integer.

Example:

1) Converting –67 to 8-bit binary number using two's complement format (Method 2):

We will convert –67 to a positive binary value which would be +67 by writing '1's in appropriate places so that when you add all the values of headings with '1's, the total is equal to 67:

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

- **64 + 2 + 1 = 67**

Inverting each binary value by swapping the 1s with 0s & swapping the 0s with 1s, we get:

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

Adding '1' to the inverted binary number:

	c8	**c7**	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	
	1	0	1	1	1	1	0	0	
+								1	
									← carry values
	1	**0**	**1**	**1**	**1**	**1**	**0**	**1**	← sum values

This gives us the final answer of the conversion of negative denary integer to a two's complement 8-bit integer:

–128	64	32	16	8	4	2	1
1	0	1	1	1	1	0	1

- **–128 + 32 + 16 + 8 + 4 + 1 = –67 denary (negative)**

2) Converting –79 to 8-bit binary number using two's complement format (Method 2):

We will convert –79 to a positive binary value which would be +79 by writing '1's in appropriate places so that when you add all the values of headings with '1's, the total is equal to 79:

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

- **64 + 8 + 4 + 2 + 1 = 79**

Inverting each binary value by swapping the 1s with 0s & swapping the 0s with 1s, we get:

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

Adding '1' to the inverted binary number:

	c8	**c7**	**c6**	**c5**	**c4**	**c3**	**c2**	**c1**	
	1	0	1	1	0	0	0	0	
+								1	
									← carry values
	1	**0**	**1**	**1**	**0**	**0**	**0**	**1**	← sum values

This gives us the final answer of the conversion of negative denary integer to a two's complement 8-bit integer:

–128	64	32	16	8	4	2	1
1	0	1	1	0	0	0	1

- **–128 + 32 + 16 + 1 = –79 denary (negative)**

Exam Style Questions:

Question 1:

(e) Negative **denary** numbers can also be represented as binary using two's complement.

Complete the binary register for the denary value −54.

You must show all your working.

Working space ....

....

....

....

Register:								

[2]

Answer:

1(e)	**One** mark for method, e.g. conversion to binary then flipping and adding 1. **One** mark for correct answer. 1 1 0 0 1 0 1 0	2

Question 2:

(a) (i) Using two's complement, show how the following denary numbers could be stored in an 8-bit register:

124								
–77								

[2]

Answer:

124	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**0**
–77	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**1**

Question 3:

(c) Convert the following two's complement integer number into denary.

11001011

....

....

....[2]

Answer:

(c) −53 **[2]**

One mark for '**53**' and **one** mark for '**−**'

Question 4:

(a) **(i)** Convert the denary number 46 to an 8-bit binary integer.

....

....[1]

(ii) Convert the denary integer −46 to an 8-bit two's complement form.

....

....[1]

(iii) Convert the denary number 46 into hexadecimal.

....

....[1]

Answer:

(a) **(i)** 0 0 1 0 1 1 1 0 [1]

(ii) 1 1 0 1 0 0 1 0 [1]

(iii) 2 E [1]

Question 5:

(a) Each of the following bytes represents an integer in two's complement form.

State the denary value.

(i) `0111 0111` Denary .... [1]

(ii) `1000 1000` Denary .... [1]

(iii) Express the following integer in two's complement form.

`-17`

[1]

(iv) State in denary, the range of integer values that it is possible to represent in two's complement integers using a single byte.

Lowest value ....

Highest value .... [1]

Answer:

Question	Answer	Marks
1(a)(i)	119	**1**
1(a)(ii)	–120	**1**
1(a)(iii)	1 1 1 0 1 1 1 1	**1**
1(a)(iv)	Lowest value: −128 Highest value: +127	**1**

Question 6:

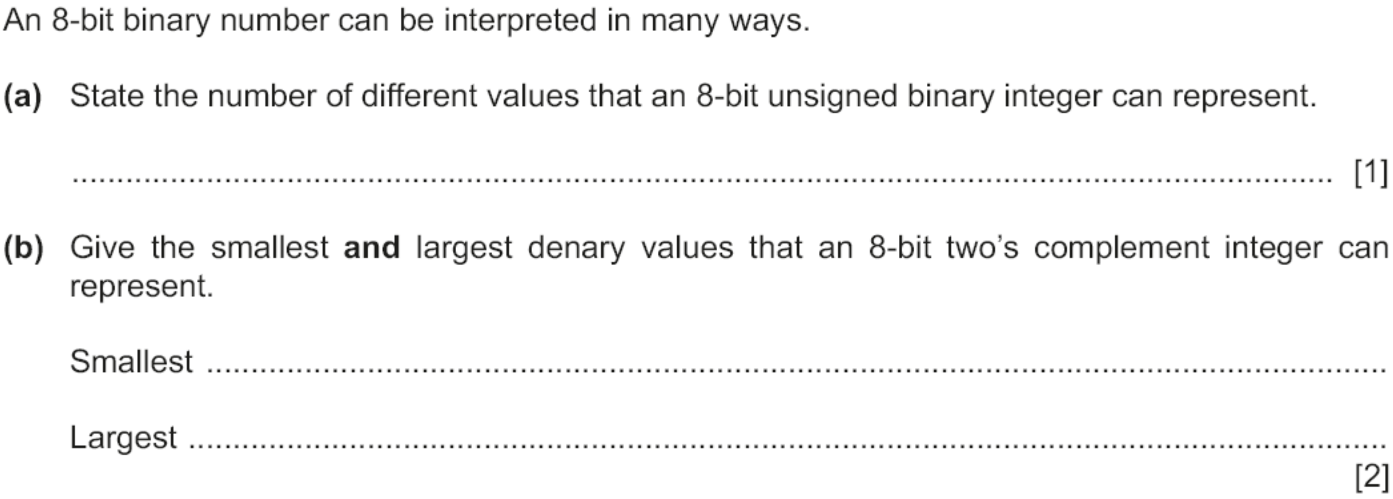
(c) Convert the two's complement number 0110 0101 into denary.

.... [1]

Answer:

4(c)	101	**1**

Question 7:



An 8-bit binary number can be interpreted in many ways.

(a) State the number of different values that an 8-bit unsigned binary integer can represent.

.... [1]

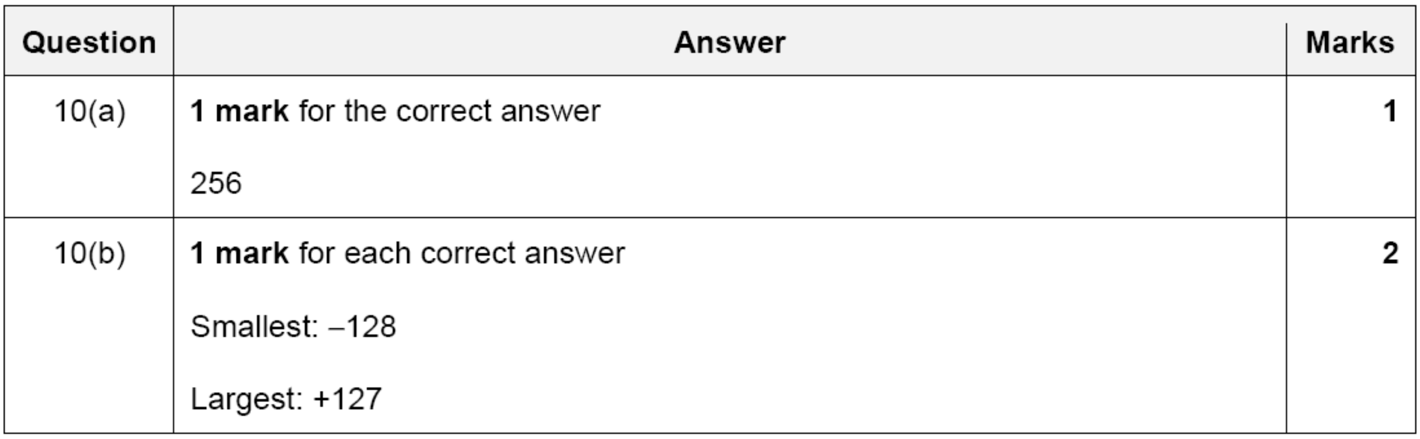
(b) Give the smallest **and** largest denary values that an 8-bit two's complement integer can represent.

Smallest ....

Largest ....

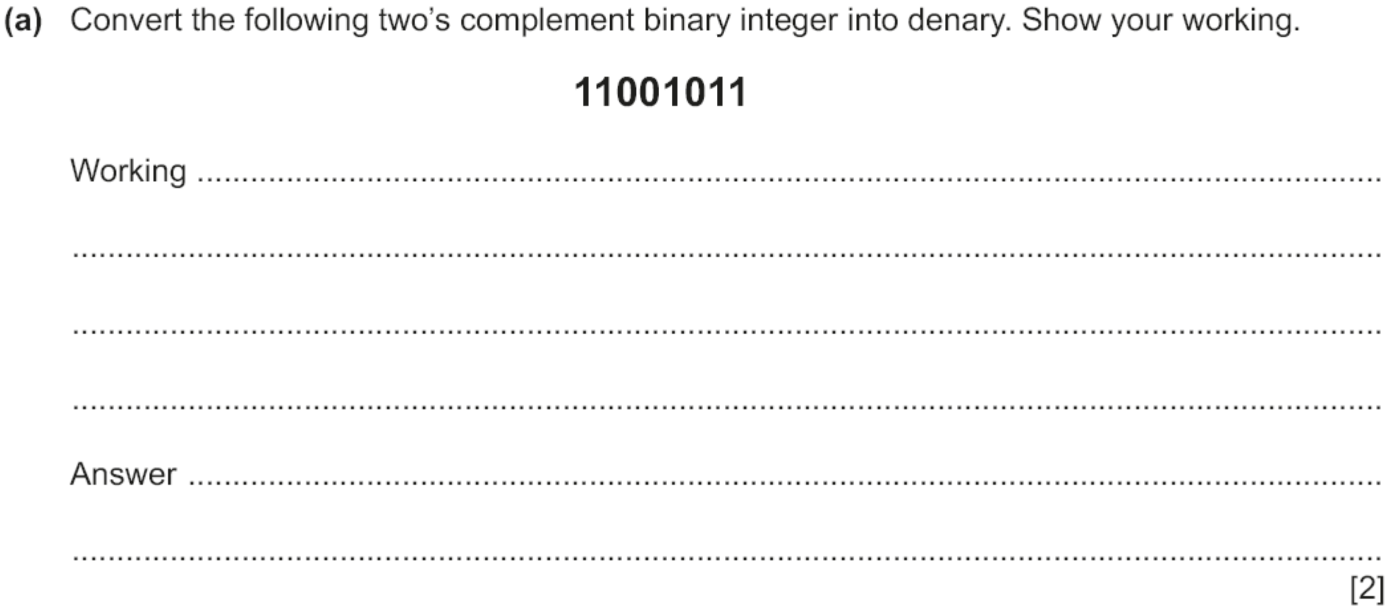
[2]

Answer:



Question	Answer	Marks
10(a)	**1 mark** for the correct answer 256	**1**
10(b)	**1 mark** for each correct answer Smallest: –128 Largest: +127	**2**

Question 8:



(a) Convert the following two's complement binary integer into denary. Show your working.

11001011

Working ....

....

....

....

Answer ....

....

[2]

Answer:

Question	Answer	Marks
1(a)	**1 mark** for working **1 mark** for correct answer 11001011 00110100 flip bits 00110101 add 1 32 + 16 + 4 + 1 Answer: −53	**2**

Question 9:

(b) Convert the signed denary value –100 into an 8-bit two's complement binary integer.

Working ....

....

Answer .... [1]

Answer:

1(b)	1001 1100	**1**

Question 10:

A register stores the following binary number:

1	1	0	0	1	1	0	1

(b) The binary value in the register represents a two's complement binary integer.

Convert the two's complement binary integer into denary.

.... [1]

Answer:

4(b)	–51	**1**

1.2 | Text, Sound & Images

NOTE: Text, Sound & Images are newly added topics in the Computer Science (2210) syllabus for the session 2023–2025.

1.2.1 Representation of Text & Use of Character Sets:

- The text is converted to binary to be processed by a computer.
- A character set is used to represent characters in a computer. The two main character sets used are American Standard Code for Information Interchange (ASCII) and Unicode.

Character Set:

- These are all the characters and symbols that can be represented/used by a computer system.
- Each character and symbol have a unique number/binary number/hexadecimal number.

Examples of Character Sets:

1. ASCII
2. Extended ASCII
3. Unicode

1) ASCII & Extended ASCII Character Set:

- ASCII only allows 128 characters to be represented (256 characters if extended ASCII).
- It uses values 0 to 127 (or 0 to 255 if extended form).
- ASCII is used for English only and many characters used in other languages cannot be represented.
- ASCII is not standardized.
- ASCII uses 7 bits per character whereas extended ASCII uses 8 bits or one byte per character.
- In extended ASCII, the characters from 128 to 255 may be coded differently in different systems.

How one character is represented in a character set such as ASCII:

- Each character is represented by a unique denary, hexadecimal or a binary number.

How the computer uses ASCII codes to represent characters:

- Each character has a unique character code.
- The binary value for each character in the string is stored in sequence.
- ASCII codes may be 7 or 8 bits long (8 bits long if extended ASCII).

How a word such as 'HOUSE' is represented by the ASCII character set:

- Each character has its own unique code.
- Each character in the word is replaced by its code.
- The codes are stored in the order in the word.

Number of characters that can be represented by the ASCII character set:

- Standard ASCII character set has 7 bits.
- 128 characters // 2^7 characters (0 to 127 in denary OR 00 to 7F in hexadecimal) can be represented using standard ASCII.

Number of characters that can be represented by the extended ASCII character set:

- Extended ASCII character set has 8 bits.
- 256 characters // 2^8 characters (0 to 255 in denary OR 0 to FF in hexadecimal) can be represented using extended ASCII.

2) Unicode Character Set:

- Unicode can represent more characters than ASCII/Extended ASCII and has greater range of characters.
- Unicode represents the most written languages in the world.
- Unicode is standardized.
- Unicode requires more bits per character than ASCII and it uses up to 32 bits or 4 bytes per character.
- Unicode is designed to be a superset of ASCII and so most characters of other languages can be represented.
- The text stored using Unicode character set takes up more storage space than ASCII because each character is encoded using more bits (16, 24 or up to 32 bits).

Number of characters that can be represented by the Unicode character set:

- Unicode can represent more characters than ASCII & Extended ASCII both.
- It has 16 bits, 24 bits or 32 bits.
- It allows for a greater range of characters and symbols than ASCII, including different languages and emojis.
- Unicode represents most written language in the world while ASCII is used for English only.

One disadvantage of using the Unicode character set, instead of the ASCII character set, is that the text stored takes up more storage space.

Why does it take up more storage space?

- Each character is encoded using more bits.

Comparisons between Character Sets:

Differences between the ASCII and Unicode character sets:

1. Unicode has greater range of characters than ASCII.
2. Unicode represents most written languages in the world while ASCII is used for English only.
3. ASCII uses 7 or 8 bits or one byte whereas Unicode uses up to 4 bytes per character.
4. Unicode is standardized while ASCII is not.

Disadvantages of using ASCII code:

1. It only allows 128/256 characters to be represented.
2. It uses values 0 to 127 (or 255 if extended form).
3. Many characters used in other languages cannot be represented.
4. In extended ASCII the characters from 128 to 255 may be coded differently in different systems.

How Unicode is designed to overcome the disadvantages of ASCII:

1. It uses 16, 24 or 32 bits / two, three or four bytes.
2. Unicode is designed to be a superset of ASCII.
3. It is designed so that most characters of other languages can be represented.

Key differences between all character sets:

1. ASCII has 7 bits whereas Unicode has 16 bits.
2. Extended ASCII has 8 bits whereas Unicode has 16 bits.
3. ASCII has 7 bits whereas extended ASCII has 8 bits.
4. Unicode can represent more characters than ASCII/Extended ASCII.
5. Extended ASCII can represent more characters than ASCII.

Calculation Based Questions:

Q1. The Unicode character code for 'G' is 0047 in hexadecimal. State, in hexadecimal, the Unicode character code for 'D'.

Answer: 0044

Working:

- We will write the alphabets in reverse order from 'G' to 'D' and keep subtracting 1 from the value 0047 every time we move 1 letter towards 'D' as we're going backwards in alphabets.
- D → E → F → G, so reversing it we get:
- G → F → E → D

Letter	Unicode character code
G	0047
F	0046
E	0045
D	**0044**

Q2. The table shows the ASCII denary values for five characters.

Character	Unicode character code
a	97
b	98
c	99
d	100
e	101

Complete the table by writing ASCII denary value for character 't' and its hexadecimal equivalent.

Character	t
ASCII denary value	
Hexadecimal value	

Answer:

Character	t
ASCII denary value	116
Hexadecimal value	74

Working:

- We will write the alphabets in increasing order from ‘e’ to ‘t’ and keep adding 1 in the value 101 every time we move 1 letter towards ‘t’ as we’re going forward in alphabets.
- e → f → g → h → i → j → k → l → m → n → o → p → q → r → s → t

Character	ASCII denary value
e	101
f	102
g	103
h	104
i	105
j	106
k	107
l	108
m	109
n	110
o	111
p	112
q	113
r	114
s	115
t	**116**

For the hexadecimal part, we will simply convert $(116)_{10}$ denary to hexadecimal:

Place hexadecimal digits in appropriate positions (under appropriate powers of 16) so that the total equates to 116:

256	**16**	**1**
0	**7**	**4**

($\underline{0}$ x 256) + ($\underline{7}$ x 16) + ($\underline{4}$ x 1) = 0 + 112 + 4 = 116

Hence the hexadecimal number is **$(74)_{16}$**

Q3. The following tables show part of the ASCII code character set.

Character	**Denary value**
A	65
B	66
C	67
D	68
E	69

Character	**Denary value**
a	97
b	98
c	99
d	100
e	101

Convert the following string into ASCII codes: 'Bed'

Answer: 66 101 100

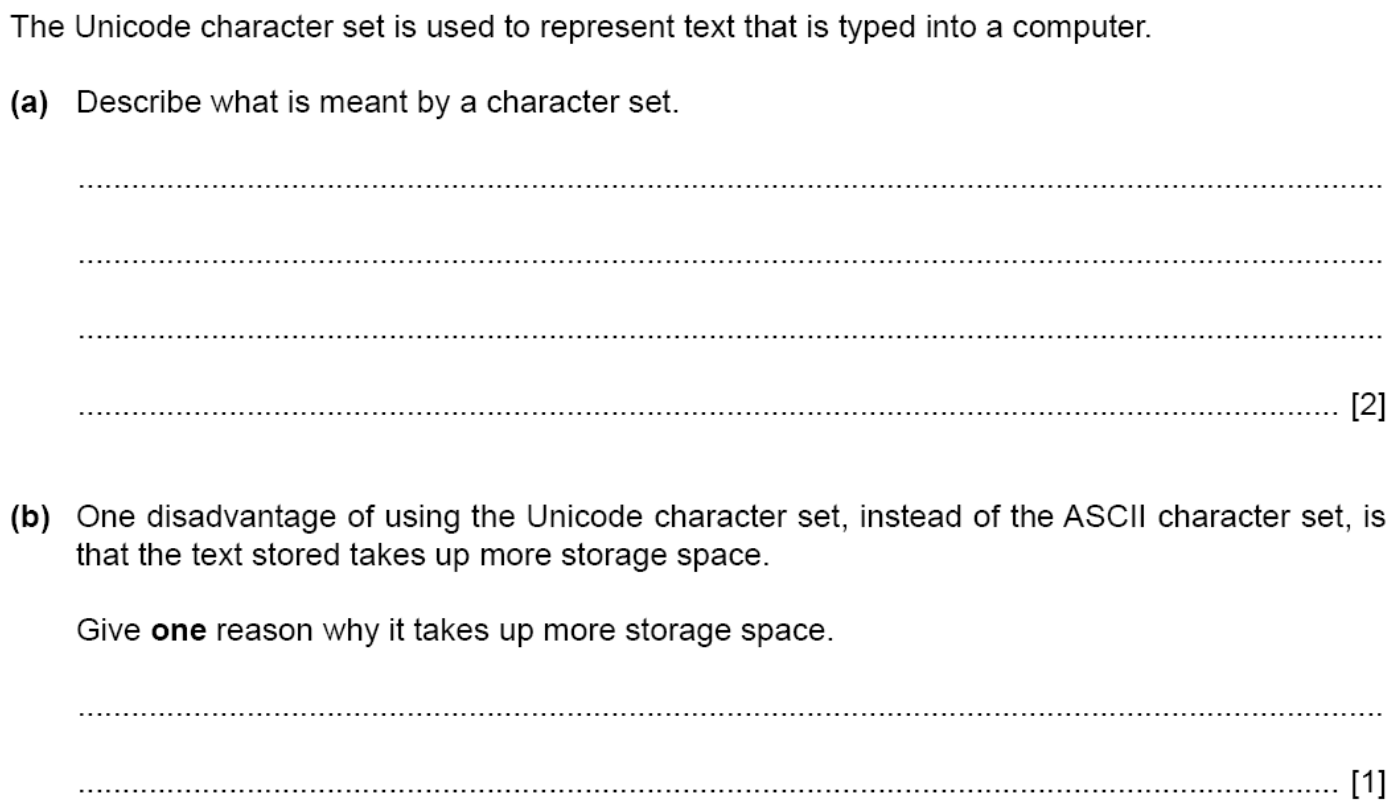
Working:

- The denary value for capital letter 'B' = 66.
- The denary value for small letter 'e' = 101
- The denary value for small letter 'd' = 100

Therefore, writing it together: 66 101 100 (Bed)

Exam Style Questions:

Question 1:



The Unicode character set is used to represent text that is typed into a computer.

(a) Describe what is meant by a character set.

....

....

....

.... [2]

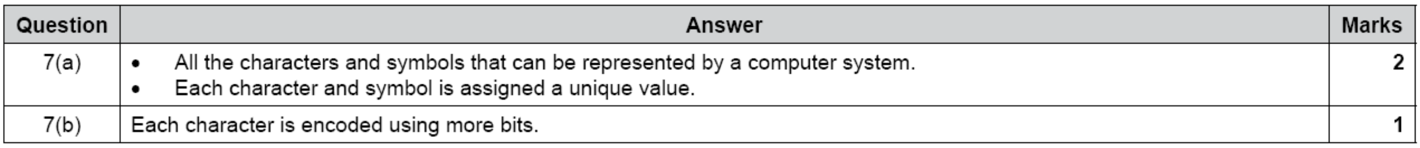
(b) One disadvantage of using the Unicode character set, instead of the ASCII character set, is that the text stored takes up more storage space.

Give **one** reason why it takes up more storage space.

....

.... [1]

Answer:



Question	Answer	Marks
7(a)	• All the characters and symbols that can be represented by a computer system. • Each character and symbol is assigned a unique value.	2
7(b)	Each character is encoded using more bits.	1

Question 2:

(a) A character set is used to represent characters in a computer.

(i) Describe what is meant by a **character set**.

....

....

....

....

....

.... [2]

(ii) Identify **two** character sets and state **one** difference between them.

Character set 1 ....

Character set 2 ....

Difference ....

....

[3]

Answer:

Question	Answer	Marks
1(a)(i)	**1 mark** per point to **max 2** • **All** of the characters/symbols that the computer can use/represent • Each character has a **unique** number/binary number/hexadecimal number	**2**
1(a)(ii)	**1 mark** for each character set to **max 2**, **1 mark** for difference • ASCII • Extended ASCII • UNICODE • ASCII has 7 bits whereas UNICODE has 16 bits • Extended ASCII has 8 bits whereas UNICODE has 16 bits • ASCII has 7 bits whereas extended ASCII has 8 bits • Unicode can represent more characters than ASCII/Extended// by example • Extended ASCII can represent more characters than ASCII	**3**

Question 3:

A computer uses the ASCII character set.

(a) State the number of characters that can be represented by the ASCII character set and the extended ASCII character set.

ASCII ....

Extended ASCII ....

[2]

(b) Explain how a word such as ‘HOUSE’ is represented by the ASCII character set.

....

....

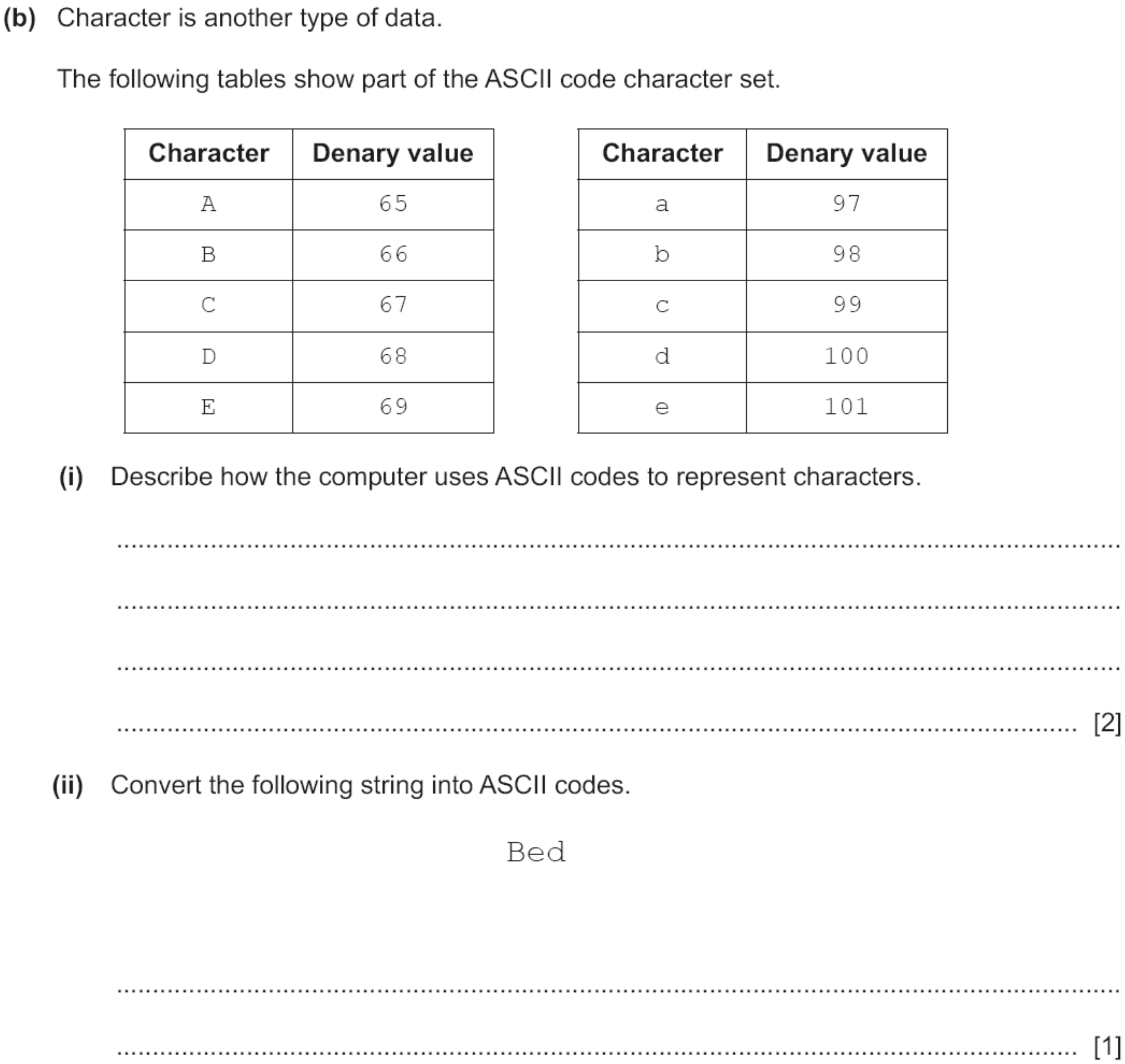
....

.... [2]

Answer:

Question	Answer	Marks
6(a)	1 mark for each correct answer ASCII = 128 // 2^7 Extended ASCII = 256 // 2^8	**2**
6(b)	**1 mark** per bullet point to **max 2** • Each character has its own **unique** code • Each character in the word is **replaced** by its code • The codes are stored **in the order in the word**	**2**

Question 4:



(b) Character is another type of data.

The following tables show part of the ASCII code character set.

Character	Denary value
A	65
B	66
C	67
D	68
E	69

Character	Denary value
a	97
b	98
c	99
d	100
e	101

(i) Describe how the computer uses ASCII codes to represent characters.

....

....

....

.... [2]

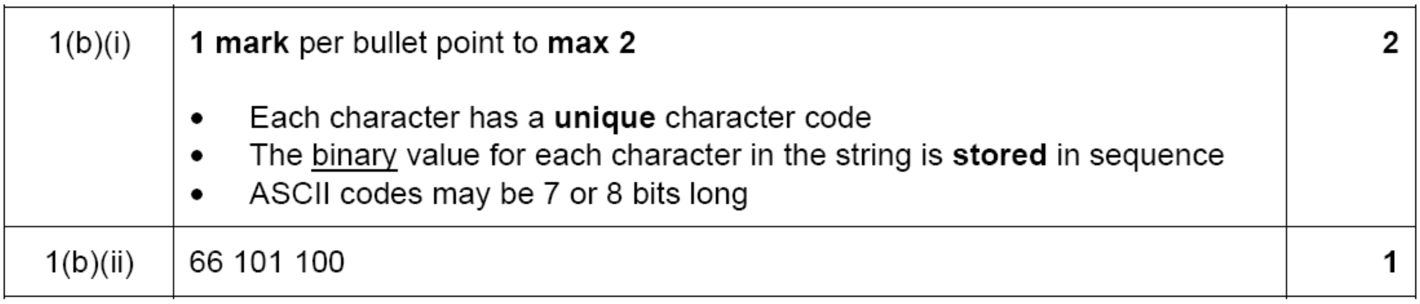
(ii) Convert the following string into ASCII codes.

`Bed`

....

.... [1]

Answer:



1(b)(i)	**1 mark** per bullet point to **max 2** • Each character has a **unique** character code • The binary value for each character in the string is **stored** in sequence • ASCII codes may be 7 or 8 bits long	**2**
1(b)(ii)	66 101 100	**1**

Question 5:

(c) The program used the ASCII coding system for character codes. An alternative coding system is Unicode.

(i) Give **two** disadvantages of using ASCII code.

1 ....

....

2 ....

....[2]

(ii) Describe how Unicode is designed to overcome the disadvantages of ASCII.

....

....

....

....[2]

Answer:

(c) **(i)** Any **two** from:

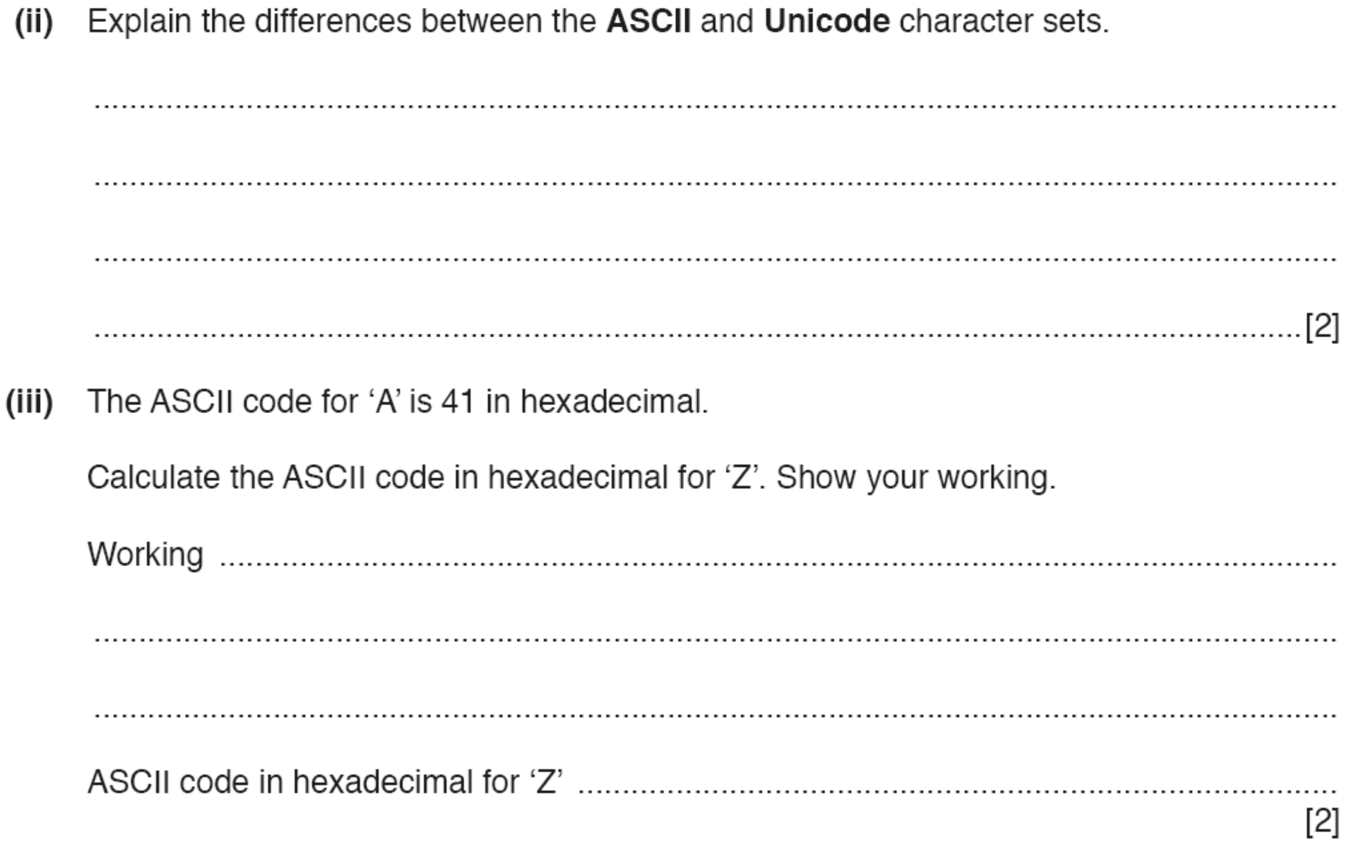
- Only 128 / 256 characters can be represented
- Uses values 0 to 127 (or 255 if extended form) / one byte
- Many characters used in other languages cannot be represented
- In extended ASCII the characters from 128 to 255 may be coded differently in different systems [2]

(ii) Any **two** from:

- Uses 16, 24 or 32 bits / two, three or four bytes
- Unicode is designed to be a superset of ASCII
- Designed so that most characters (in other languages) can be represented

[2]

Question 6:



(ii) Explain the differences between the **ASCII** and **Unicode** character sets.

....

....

....

....[2]

(iii) The ASCII code for ‘A’ is 41 in hexadecimal.

Calculate the ASCII code in hexadecimal for ‘Z’. Show your working.

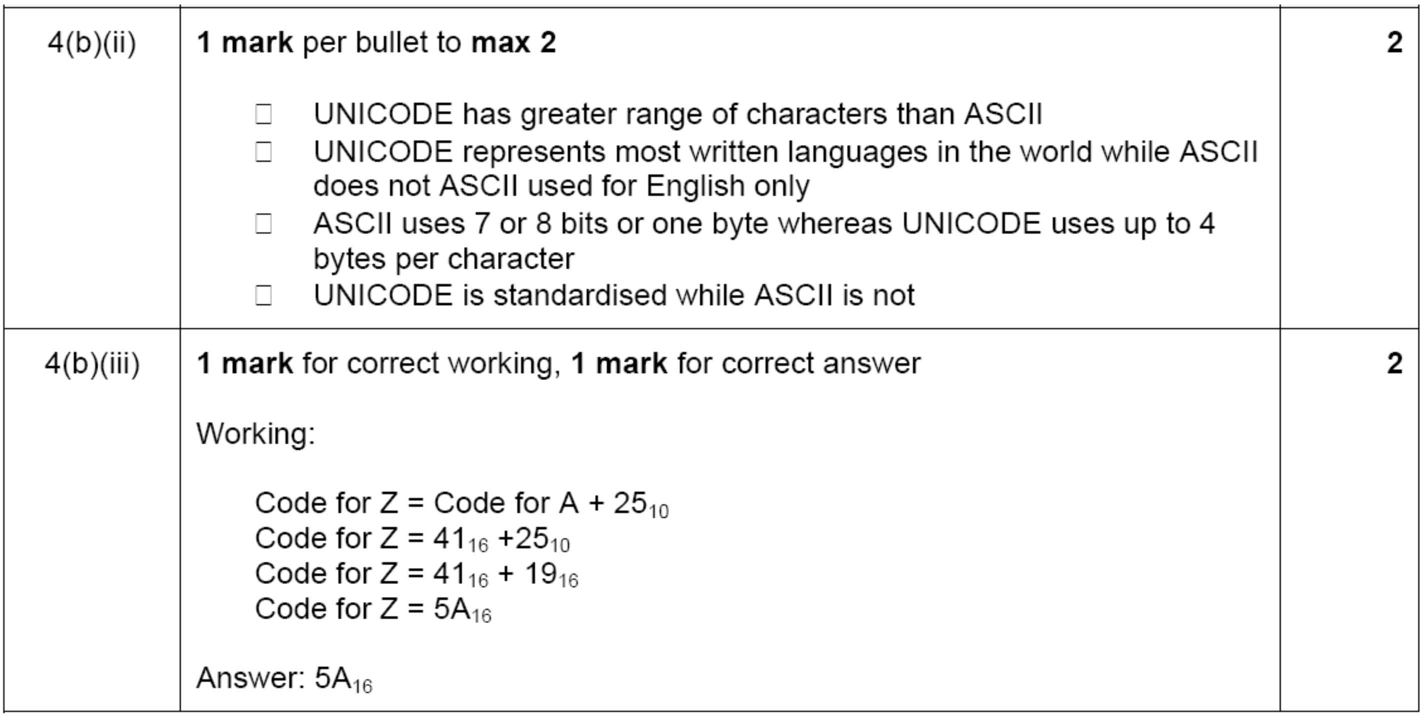
Working ....

....

....

ASCII code in hexadecimal for ‘Z’ .... [2]

Answer:



4(b)(ii)	**1 mark** per bullet to **max 2** □ UNICODE has greater range of characters than ASCII □ UNICODE represents most written languages in the world while ASCII does not ASCII used for English only □ ASCII uses 7 or 8 bits or one byte whereas UNICODE uses up to 4 bytes per character □ UNICODE is standardised while ASCII is not	**2**
4(b)(iii)	**1 mark** for correct working, **1 mark** for correct answer Working: Code for Z = Code for A + 25_{10} Code for Z = 41_{16} +25_{10} Code for Z = 41_{16} + 19_{16} Code for Z = $5A_{16}$ Answer: $5A_{16}$	**2**

1.2.2 Representation of Sound:

- The sounds waves are analogue in nature and computers cannot work with analogue data.
- The sound wave is sampled for sound to be converted to binary, which is processed by a computer.

The following table shows key terms related to sound along with their definitions:

Term	Definition
Sampling	It is the amplitude of analogue sound wave measured at regular/set time intervals.
Sampling Rate	It is the number of samples taken in a second.
Sampling Resolution	It is the number of bits used to store/represent each sample.

Sampling:

- It is the amplitude/height of analogue sound wave measured at set/regular time intervals.

How an analogue sound wave is sampled to convert it into digital format:

- The amplitude/height of analogue sound wave is measured at set/regular time intervals.
- The value of the measured sample is then stored/recorded as a binary number.

How the sound is encoded into a digital form:

- The amplitude/height of analogue sound wave is measured at set/regular time intervals (sound wave is sampled).
- The value of the measured sample is then stored as a unique binary number.
- The whole sound is encoded as a sequence of binary numbers which are stored in a sequence for each sample.

How sampling is used to record the sound clips:

- It measures the amplitude/height of analogue sound wave at set/regular time intervals.
- It is used to get an approximation of the sound wave and then encoding it as a sequence of binary numbers // converting it to a digital signal.
- Increasing the sampling rate will improve the accuracy of the recording.

Sampling Rate:

- It is the number of samples taken per second (1Hz means 'one sample per second').
- It is the number of times that the amplitude/height of analogue sound wave is measured per second.

Factors affecting sampling rate:

1. Increasing the sampling rate results in more accurate digital representation of sound.
2. Increasing the sampling rate will result in smaller quantization error.
3. Increasing the sampling rate will result in larger file size.

Effects of changing the sampling rate from 44.1 kHz to 22.05 kHz:

- A fewer samples will be taken per second.
- The file size will decrease.
- There will be larger gaps/spaces between samples which will result in greater quantization error.
- The sound accuracy will reduce (will not be as close to original sound).

Why sound is closer to the original when a higher sample rate is used:

- There are smaller time gaps between the samples
- It makes the digital sound wave more accurate
- There is a smaller quantization error.

Why the sound file size increases when a higher sample rate is used:

- There are more samples being taken per second.
- This results in more bits being stored altogether.

A voice is recorded twice. Each recording is the same length and has the same sampling resolution.

The first recording has a sampling rate of 44 100 Hz. The second recording has a sampling rate of 21 000 Hz.

How different sampling rates of 44 100 Hz and 21 000 Hz will affect the recording and the sound file:

- The sampling resolution of 44 100 Hz takes more samples per second, so the file size will be larger whereas the sampling resolution of 21 000 Hz takes fewer samples per second, so the file size will be smaller.
- At a resolution of 44 100 Hz, the sound recording is a more accurate representation of the voice whereas at the resolution of 21 000 Hz, the sound recording is a less accurate representation of the voice.

A sound file is being downloaded from a web server. A choice of a sampling rate of 44.1 kHz or 98 kHz is given before downloading the sound file.

Differences between the two sounds files stored on the server:

- 98 kHz has a larger file size because it is recording more samples per second meaning more binary values being stored per second.
- Therefore, it will take more time to download.
- 98 kHz sound will be closer to the original because the samples will be closer together due to smaller quantization error.

Sampling Resolution:

- It is the number of bits used to store each sample or bits available to encode/represent each sample.
- It is sometimes referred to as bit depth.
- It is usually 8-bit, 16-bit, 24-bit or 32-bit.

Factors affecting sampling resolution:

1. A larger sampling resolution means there is a greater range of values available to store each sample // more bits per sample.
2. A larger sampling resolution will improve the accuracy of the sound // more accurate representation of sound.
3. A larger sampling resolution will decrease the distortion of the sound.
4. Increased sampling resolution means a smaller quantization error.
5. A larger sampling resolution will result in a larger file size.

How different sampling resolutions affect the sound file and the sound it represents:

- The sampling resolution is the number of bits used to store each sample.
- Increasing the sampling resolution means a larger file size whereas decreasing the sampling resolution means a smaller file size.
- Increasing the sampling resolution gives a more accurate representation of analogue sound whereas decreasing the sampling resolution gives a less accurate representation of the analogue sound.
- Increasing the sampling resolution means a greater range of values can be stored whereas decreasing the sampling resolution gives a smaller range of values that can be stored.
- Increasing the sampling resolution reduces the quantization errors whereas decreasing the sampling resolution causes greater quantization errors.

Benefits of using a larger/higher sampling resolution:

1. It allows for larger dynamic ranges as dynamic range is approximately six times the bit depth.
2. It allows more accurate representation/crisper sound quality.
3. It decreases the distortion of sound.

Drawbacks of using a larger/higher sampling resolution:

1. A larger sampling resolution will result in a larger file size.
2. The larger file will take longer to transmit/download music files.
3. It requires greater processing power.

Sampling Rate & Sampling Resolution:

- The accuracy of the recording and the file size increases as the sample rate and sample resolution increase.

How the sampling rate and sampling resolution affect the file size of the soundtrack:

- Increasing the sampling rate means more samples per second hence more bits per second and a larger file size whereas decreasing the sampling rate means fewer samples per second hence fewer bits per second and a smaller file size.
- A higher sampling resolution means more bits per sample and a larger file size whereas a lower sampling resolution means fewer bits per sample and a smaller file size.

What is meant by a sampling rate of 88.2 kHz:

- The sound wave is sampled 88 200 times per second OR 88 200 samples/measurements are taken per second.

What is meant by a sampling resolution of 32 bits:

- Each sample is stored as a 32-bit binary number OR 32 bits are used to store each sample.

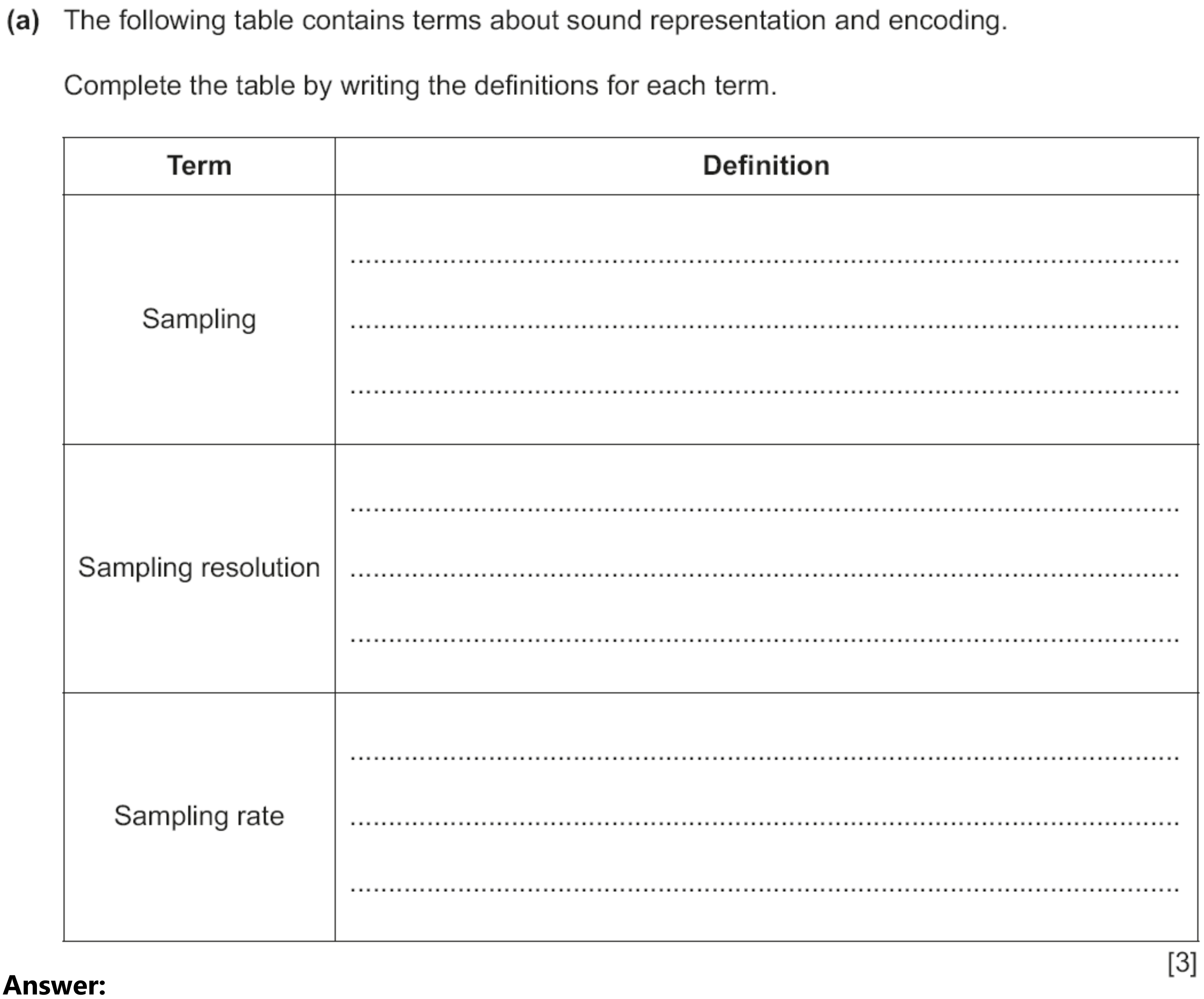
Compact Discs (CDs) have a 16-bit sampling resolution and a 44.1 kHz sample rate – that is 44 100 samples every second.

Why 16-bit sampling is used in an audio compact disc (CD):

- It is used to ensure that bit depth/sampling resolution is sufficient for good quality sound.
- Using a higher bit depth/sampling resolution would mean bigger files, hence less music content on each CD.
- It can represent a dynamic range of about 90 dB which is basically the maximum dynamic range of human hearing.
- It is done for the purpose of compromising between quality and reasonable file size.

Exam Style Questions:

Question 1:



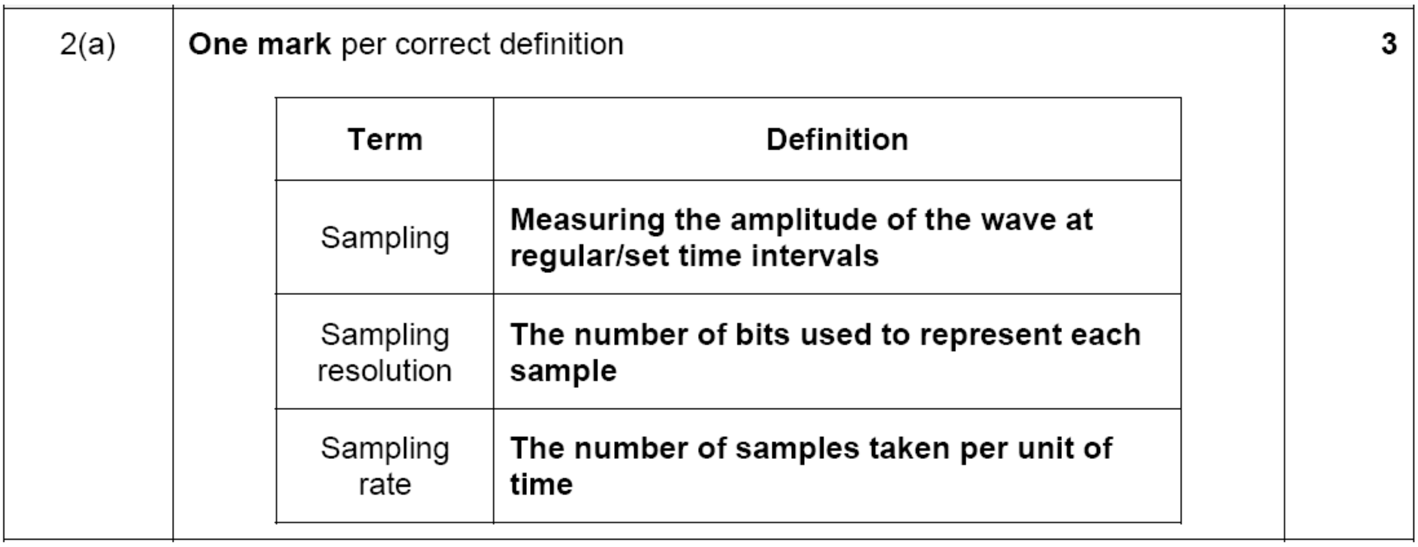
(a) The following table contains terms about sound representation and encoding.

Complete the table by writing the definitions for each term.

Term	Definition
Sampling	....
Sampling resolution	....
Sampling rate	....

[3]

Answer:



2(a)	**One mark** per correct definition	3

Term	Definition
Sampling	**Measuring the amplitude of the wave at regular/set time intervals**
Sampling resolution	**The number of bits used to represent each sample**
Sampling rate	**The number of samples taken per unit of time**

Question 2:

Daniel is creating a sound file for a school project.

(a) Daniel records the sound using a sampling rate of 44.1 kHz and a sampling resolution of 16 bits.

(i) State what is meant by a **sampling rate** of **44.1 kHz**.

....

.... [1]

(ii) State what is meant by a **sampling resolution** of **16 bits**.

....

.... [1]

Answer:

Question	Answer	Marks
2(a)(i)	44.1 kHz = **44 100** samples/measurements per second/unit of time	**1**
2(a)(ii)	**16** bits are used to store each sample	**1**

Question 3:

Bobby is recording a sound file for his school project.

(a) He repeats the recording of the sound several times, with a different sample rate each time.

(i) Describe the reasons why the sound is closer to the original when a higher sample rate is used.

....

....

....

.... [2]

(ii) Describe the reasons why the sound file size increases when a higher sample rate is used.

....

Answer:

Question	Answer	Marks
7(a)(i)	**1 mark** per bullet point • Smaller time gaps between the samples • Makes the **digital** sound **wave** more accurate • Smaller quantisation errors	**2**
7(a)(ii)	**1 mark** per bullet point • More samples/data are taken/recorded • … so more bits are stored altogether	**2**

Question 4:

(b) An analogue-to-digital converter uses sampling to encode the sound.

Explain how different sampling resolutions affect the sound file and the sound it represents.

....

....

....

....

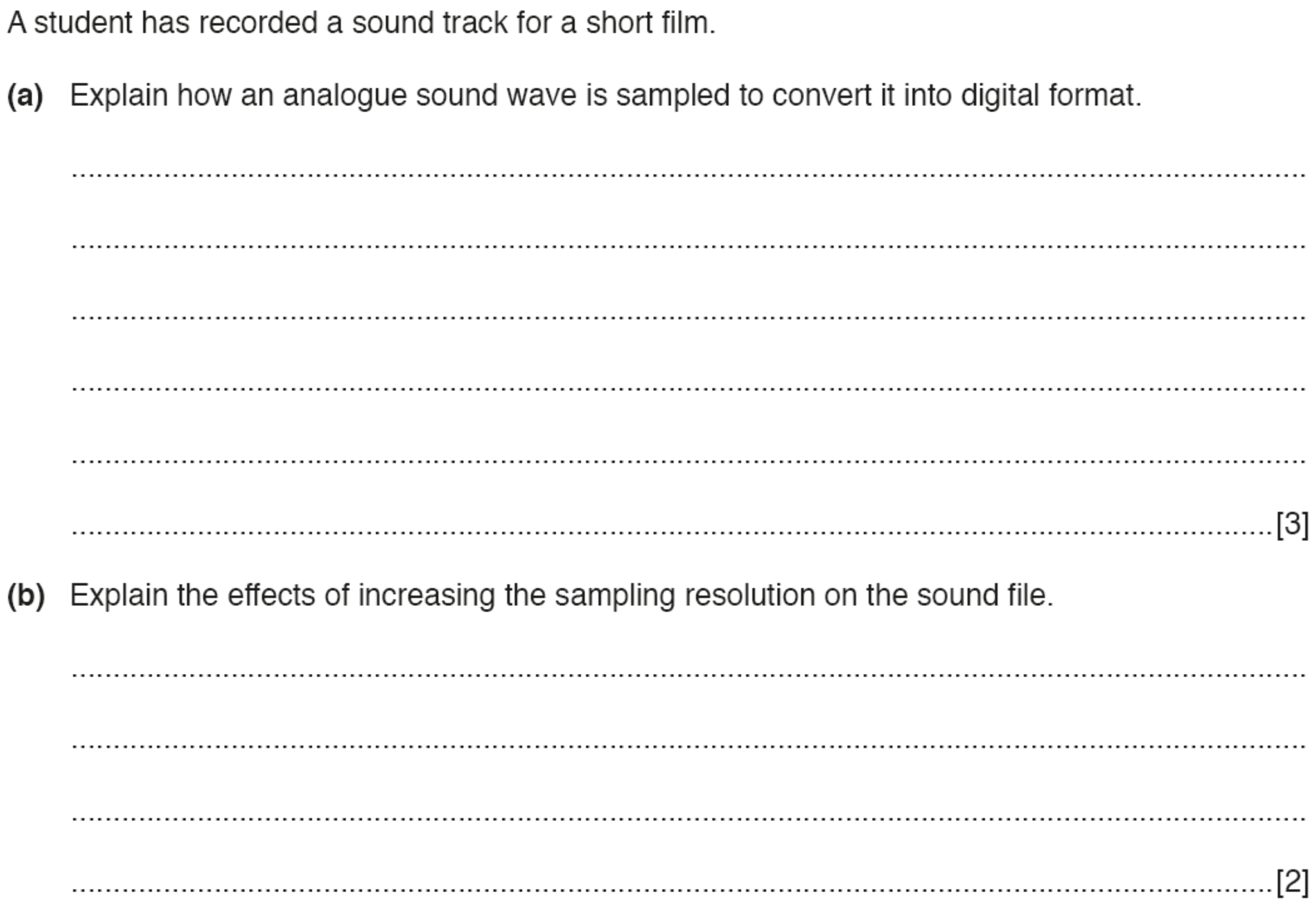
....

....[3]

Answer:

1(b)	**1 mark** per bullet point □ The sampling resolution number of bits used to store each <u>sample</u> □ Increasing the (sampling) resolution means a larger file size // Decreasing the (sampling) resolution means a smaller file size □ Increasing the (sampling) resolution gives a more accurate representation of the analogue sound // Decreasing the (sampling) resolution gives a less accurate representation of the analogue sound □ Increasing the (sampling) resolution means a greater range of values can be stored // Decreasing the (sampling) resolution gives a smaller range of values that can be stored □ Increasing the (sampling) resolution reduces the quantization errors // Decreasing the (sampling) resolution causes greater quantization errors	**3**

Question 5:



A student has recorded a sound track for a short film.

(a) Explain how an analogue sound wave is sampled to convert it into digital format.

....

....

....

....

....

....[3]

(b) Explain the effects of increasing the sampling resolution on the sound file.

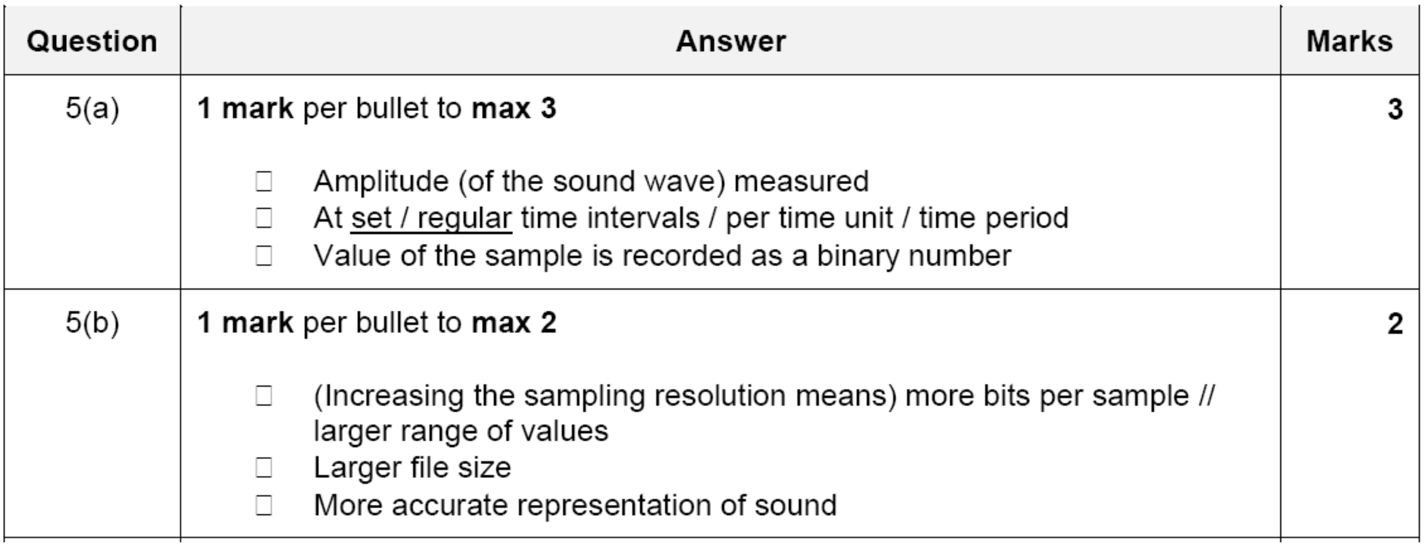
....

....

....

....[2]

Answer:



Question	Answer	Marks
5(a)	**1 mark** per bullet to **max 3** □ Amplitude (of the sound wave) measured □ At set / regular time intervals / per time unit / time period □ Value of the sample is recorded as a binary number	**3**
5(b)	**1 mark** per bullet to **max 2** □ (Increasing the sampling resolution means) more bits per sample // larger range of values □ Larger file size □ More accurate representation of sound	**2**

1.2.3 Representation of Images:

- An image is a series of pixels that are converted to binary, which is processed by a computer.

How a digital image file is stored by a computer:

- The image is made of pixels (picture elements).
- The image has a set number of pixels wide by pixels high.
- Each pixel stores one color.
- Each color has a unique binary value (color code).
- The color/binary value of each pixel is stored in sequence.
- File contains metadata to identify how the file should be displayed.
- The metadata can be the color depth/resolution.

The following table shows key terms related to images along with their definitions:

Term	Definition
Pixels	It is the smallest picture element which can be drawn.
Color Depth	It is the number of bits used to represent each color.
Image Resolution	It is the number of pixels in the image.

Bitmap Image:

- It is an image made up of rows and columns of pixels (picture elements).
- Each pixel has one color, and the color of each pixel is stored as a unique binary number.
- A bitmap image is stored in a computer as a series of binary numbers.
- A bitmap file has a file header.

How the images are encoded into a digital form:

- The images are stored as bitmaps.
- Each image is made up of pixels (picture elements).
- Each pixel has one color, and the color of each pixel has a unique binary number.
- The binary numbers are stored in a sequence for each image.

Image File Header:

- It stores data about the file contents/image/metadata (i.e., image size, number of colors etc.)

Examples of File Header Contents:

1. File size.
2. Dimensions of the image in pixels // image resolution.
3. Color depth (bits per pixel, 1, 4, 8, 16, 24 or 32).
4. Type of compression used, if any.

Color Depth:

- It is the number of bits used to represent each color.
- An 8-bit color depth means that each pixel can be one of 256 colors **(because $2^8 = 256$).**
- However, modern computers have a 24-bit color depth, which means over 16 million different colors can be represented.

Factors affecting color depth:

1. Increasing the color depth means there are more bits per pixel to represent each color.
2. It results in more accurate digital representation of image.
3. Increasing the color depth will result in better image quality.
4. Increasing the color depth will result in larger file size because more bits per pixel are used.

How changing the color depth of an image affects its file size:

- Increasing the color depth results in increased file size because more bits per pixel and hence more data is stored.
- Whereas decreasing the color depth results in smaller file size because fewer bits per pixel and hence less data is stored.

How different color depths affect the image file and the image it represents:

- The color depth is the number of bits used to represent each color.
- Increasing the color depth means a larger file size whereas decreasing the color depth means a smaller file size.
- Increasing the color depth gives a better quality of image whereas decreasing the color depth gives a poorer quality of the image.
- Increasing the color depth means a greater range of values can be stored whereas decreasing the color depth gives a smaller range of values that can be stored.

Number of bits required to store each pixel for a black and white image (2-color image):

- A black and white image only requires 1 bit per pixel (2^1 = 2)
- This means that each pixel can be one of two colors, corresponding to either 1 or 0 binary value.

Number of bits required to store each pixel for a 4-color image:

- A four-color image requires 2 bits per pixel (2^2 = 4).
- This means that each pixel can be one of four colors, corresponding to 00, 01, 10 or 11 binary values.

Number of bits required to store each pixel for an 8-color image:

- An eight-color image requires 3 bits per pixel (2^3 = 8).
- This means that each pixel can be one of eight colors, corresponding to 000, 001, 010, 001, 100, 101, 110 or 111 binary values.

Number of bits required to store each pixel for a 256-color image:

- 8 bits are needed, and each color is represented by one of 256 values.
- Values 0 to 255 / 0000 0000 to 1111 1111.
- 256 = 2^8

A picture has been drawn and is saved as monochrome bitmap image. How many pixels are stored in one byte:

- 8 pixels are stored in 1 byte because each pixel requires 1 bit.

A picture is drawn and is saved as a 16-color bitmap image. How many bits are used to encode the data for one pixel:

- 16 colors mean 2^4 = 16.
- Therefore, 4 bits are used to encode data for one pixel.

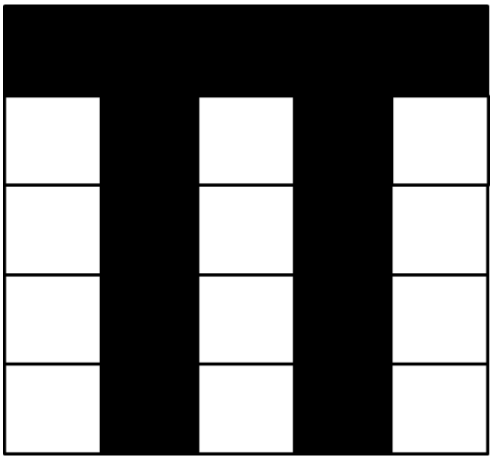
Image Resolution:

- It is the number of pixels in an image (per inch or per centimeter) OR it is the number of pixels wide by the number of pixels high in an image.
- For example, an image could contain 4096 x 3072 pixels (12 582 912 pixels in total).
- This value determines the amount of detail an image has.

Factors affecting image resolution:

1. A larger image resolution means there is a greater range of pixels to represent each image.
2. A larger image resolution will improve the quality of the image. // more accurate representation of image.
3. A larger image resolution will decrease the pixelation of the image.
4. A larger image resolution will result in a larger file size.

How a computer can store this black and white bitmap image:



- Each pixel in the image requires only one bit (as there are only two colors (2^1 = 2).
- The black color is represented by binary value 1 and white by 0 (or vice versa).
- The bits are stored for each pixel in sequence **e.g. 11111 01010 01010 01010 01010**.

Color Depth & Image Resolution:

- The file size and quality of the image increases as the resolution and color depth increase.

How the resolution and color depth affect the file size of the image:

- Increasing the resolution and color depth means more pixels per centimeter and more bits per color and a larger file size.
- Decreasing the resolution and color depth means fewer pixels per centimeter and fewer bits per color and a smaller file size.

Drawback of High-Resolution Images:

1. As the number of pixels used to represent each image is increased, the size of the file will also increase.
2. Larger image size means it takes longer to upload, download, or transfer images.

Exam Style Questions:

Question 1:

(a) Describe what is meant by a **bitmap image**.

....

....

....

....[2]

(b) The computer also has bitmap software.

(i) Define the term **image resolution**.

....

....[1]

Answer:

2(a)	**1 mark** per bullet, **max 2** □ Made up of pixels □ Each pixel has one colour □ Colour of each pixel stored as a binary number	**2**
3(b)(i)	**One** from: □ The <u>number of pixels</u> per <u>unit measurement</u> 1 □ The number of pixels in an image 1 □ The number of pixels wide by the number of pixels high 1 □ Number of pixels per row by the number of rows 1	**1**

Question 2:

(d) Dominic's tablet captures a video of Dominic to send to other people. The video is made of a sequence of images and a sound file.

(i) Describe how the images and sound are encoded into a digital form.

Images ....

....

....

....

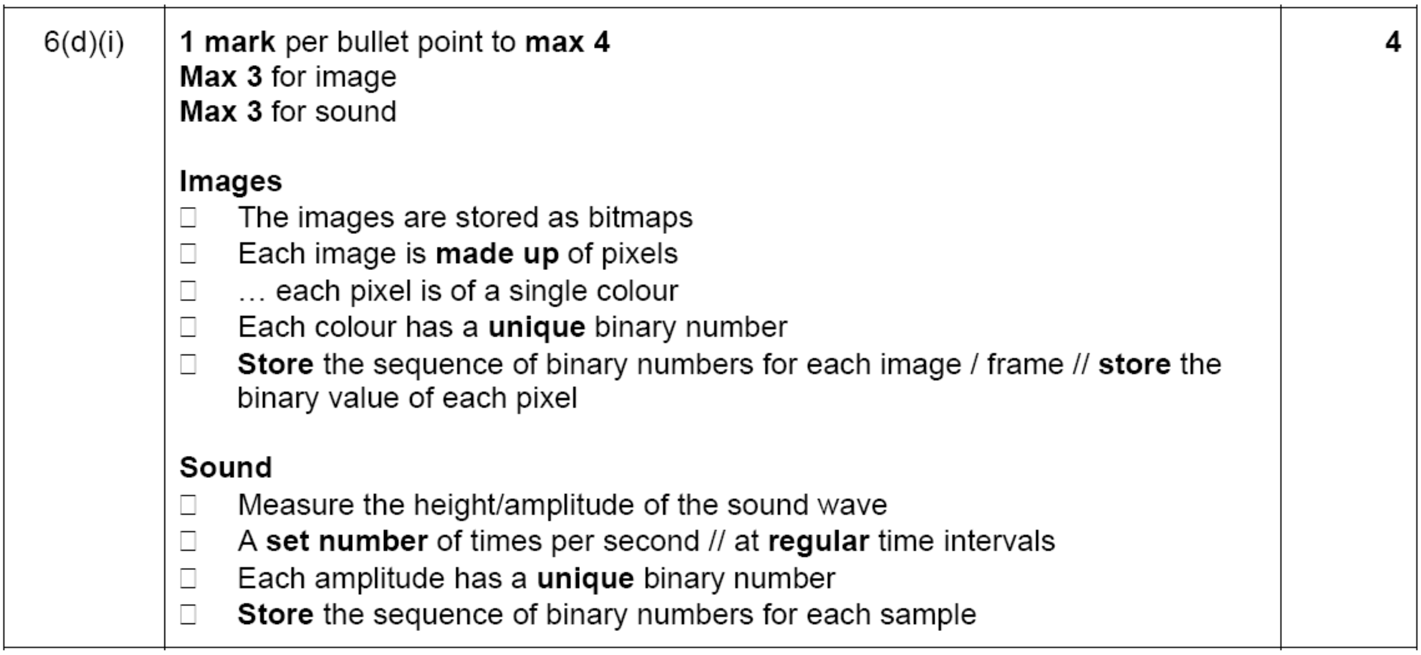
Sound ....

....

....

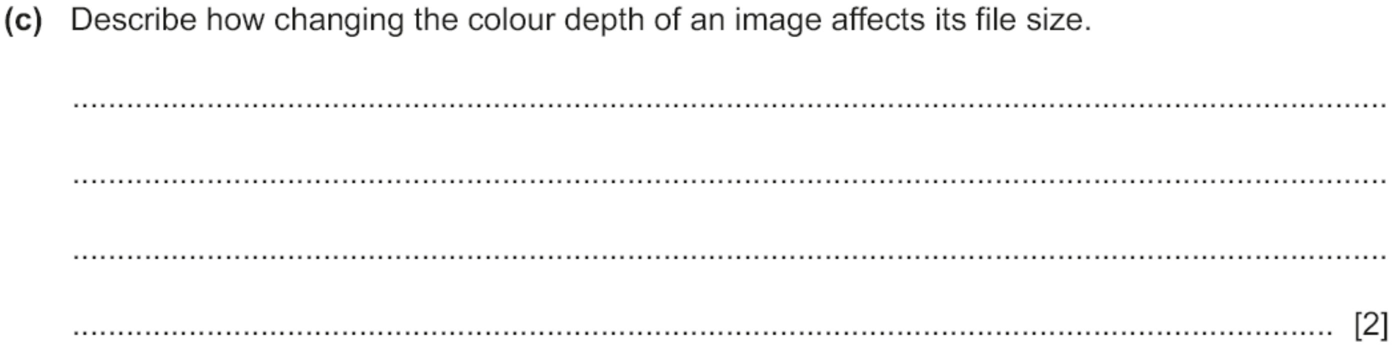
.... [4]

Answer:



6(d)(i)	**1 mark** per bullet point to **max 4** **Max 3** for image **Max 3** for sound **Images** □ The images are stored as bitmaps □ Each image is **made up** of pixels □ … each pixel is of a single colour □ Each colour has a **unique** binary number □ **Store** the sequence of binary numbers for each image / frame // **store** the binary value of each pixel **Sound** □ Measure the height/amplitude of the sound wave □ A **set number** of times per second // at **regular** time intervals □ Each amplitude has a **unique** binary number □ **Store** the sequence of binary numbers for each sample	**4**

Question 3:



(c) Describe how changing the colour depth of an image affects its file size.

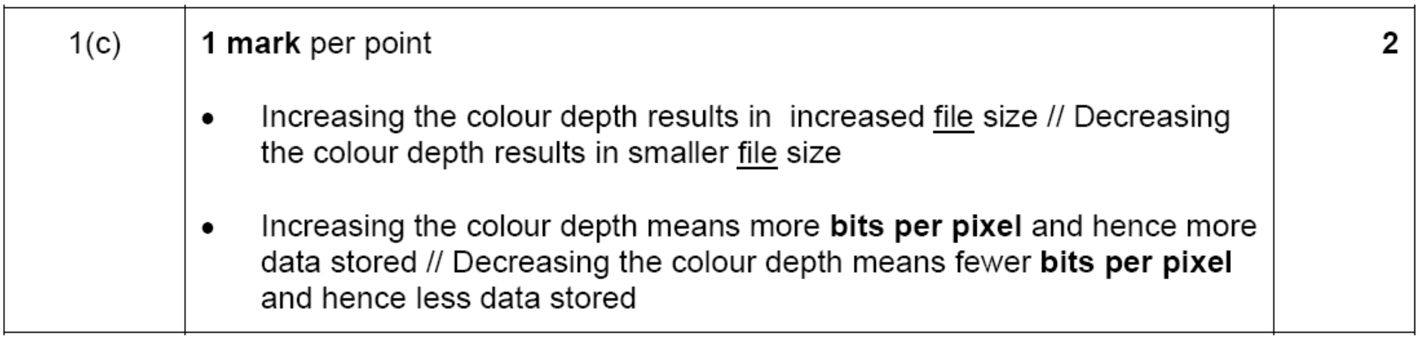
....

....

....

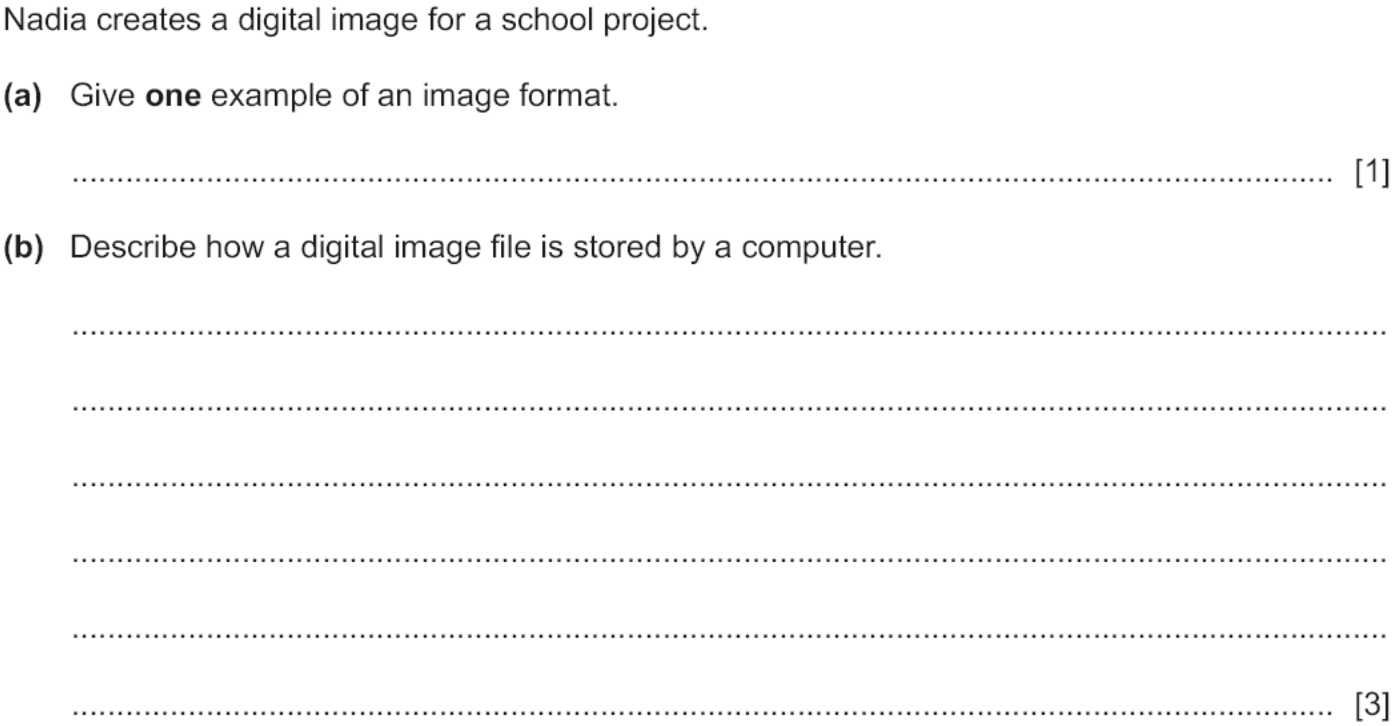
.... [2]

Answer:



1(c)	**1 mark** per point • Increasing the colour depth results in increased file size // Decreasing the colour depth results in smaller file size • Increasing the colour depth means more **bits per pixel** and hence more data stored // Decreasing the colour depth means fewer **bits per pixel** and hence less data stored	**2**

Question 4:



Nadia creates a digital image for a school project.

(a) Give **one** example of an image format.

.... [1]

(b) Describe how a digital image file is stored by a computer.

....

....

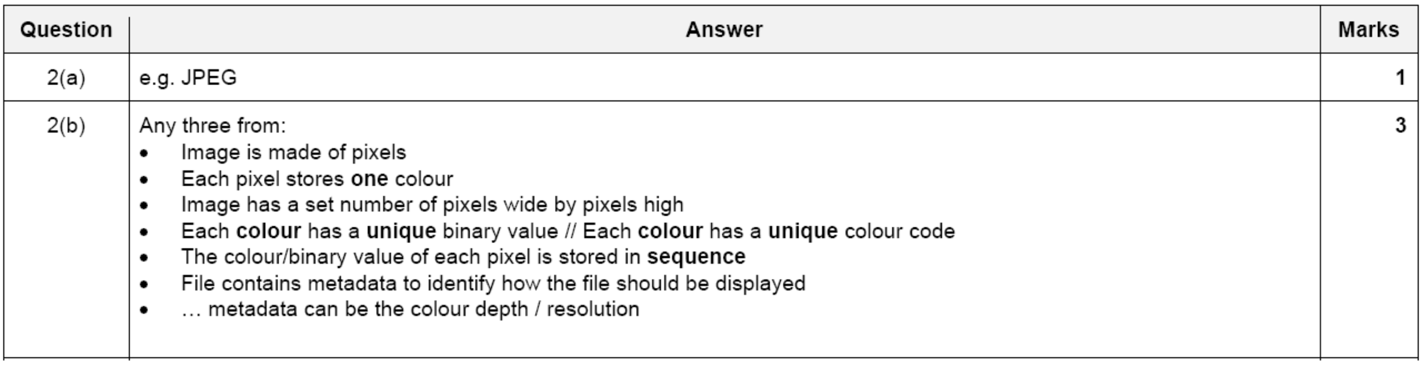
....

....

....

.... [3]

Answer:



Question	Answer	Marks
2(a)	e.g. JPEG	**1**
2(b)	Any three from: • Image is made of pixels • Each pixel stores **one** colour • Image has a set number of pixels wide by pixels high • Each **colour** has a **unique** binary value // Each **colour** has a **unique** colour code • The colour/binary value of each pixel is stored in **sequence** • File contains metadata to identify how the file should be displayed • … metadata can be the colour depth / resolution	**3**

1.3 | Data Storage & File Compression

1.3.1 Measurement of Data Storage:

- A binary digit is referred to as a bit which is either 1 or 0.
- 1 byte is 8 bits.
- The byte is the smallest unit of memory in a computer. Some computers use larger bytes, but they are always multiples of 8 (e.g. 16-bit and 32-bit systems).
- A 4-bit number is called a nibble which is half a byte.

A single letter or character would use one byte of memory (8 bits); two characters would use two bytes (16 bits).

The memory size system of numbering below is inaccurate as it is based on SI (base 10) system of units where 1 kilo is equal to 1000. It only refers to some storage devices but is technically inaccurate. However, it is included here for completeness:

Name of memory size	Equivalent denary value	Amount of previous denomination
1 nibble	4 bits	
1 byte	8 bits	
1 kilobyte (1 KB)	1 000 bytes	
1 megabyte (1 MB)	1 000 000 bytes	1024 KB
1 gigabyte (1 GB)	1 000 000 000 bytes	1024 MB
1 terabyte (1 TB)	1 000 000 000 000 bytes	1024 GB
1 petabyte (1 PB)	1 000 000 000 000 000 bytes	1024 TB
1 exabyte (1 EB)	1 000 000 000 000 000 000 bytes	1024 PB

This system is no longer a part of the new syllabus and so it will not be used in this course. However, you fill find some past paper questions related to this system as it was previously a part of the syllabus.

Since the memory size is actually measured in terms of powers of 2, a memory size system has been adopted by the IEC (International Electrotechnical Commission) that is based on the binary system given below:

Name of memory size	**Number of bytes**	**Equivalent denary value**	**Amount of previous denomination**
1 nibble	2^2	4 bits	
1 byte	2^3	8 bits	
1 kibibyte (1 KiB)	2^{10}	1 024 bytes	
1 mebibyte (1 MiB)	2^{20}	1 048 576 bytes	1024 KiB
1 gibibyte (1 GiB)	2^{30}	1 073 741 824 bytes	1024 MiB
1 tebibyte (1 TiB)	2^{40}	1 099 511 627 776 bytes	1024 GiB
1 pebibyte (1 PiB)	2^{50}	1 125 899 906 842 624 bytes	1024 TiB
1 exbibyte (1 EiB)	2^{60}	1 152 921 504 606 846 976 bytes	1024 PiB

You can either memorize it by increasing the powers of 2, or simply by remembering that 1 kibibyte is 1024 bytes, so accordingly 1 mebibyte would be 1024 x 1 KiB (1024), and so 1 gibibyte would be 1024 x 1 MiB (1024 x 1024) and so on.

This system is accurate, and all memories should be measured using the IEC system and therefore, only the IEC system is covered in the syllabus.

Example Questions:

Question 1: Give the number of bytes in a kibibyte.

- 1 024 bytes // 2^{10}

Question 2: Give the number of bytes in a mebibyte.

- 1 048 576 bytes // 1024 * 1024 // 2^{20}

Question 3: Give the number of bytes in a gibibyte.

- 1 073 741 824 bytes // 1024 * 1024 * 1024 // 2^{30}

Question 4: Give the number of bytes in a tebibyte.

- 1 099 511 627 776 bytes // 1024 * 1024 * 1024 * 1024 // 2^{40}

Question 5: State the difference between a kibibyte and a kilobyte.

- Kibibyte is 1024 bytes while kilobyte is 1000 bytes.
- Kibibyte has a denary prefix while kilobyte has the binary prefix.

Question 6: State the difference between a mebibyte and a megabyte.

- One mebibyte is 1024 kibibytes and one megabyte is 1000 kilobytes.

Question 7: State the difference between a gibibyte and a gigabyte.

- One gibibyte is 1024 mebibyte and one gigabyte is 1000 megabytes.

Question 8: State the difference between a tebibyte and a terabyte.

- One tebibyte is 1024 gibibytes and one terabyte is 1000 gigabytes.

1.3.2 Calculation of File Size:

In this section we will look at the calculation of the file size of an image file and a sound file, using the information given. The information given may include:

For image file:

1. image resolution
2. color depth

For sound file:

1. sound sample rate
2. resolution
3. length of track.

The size of an **image** file is calculated using the formula:

image resolution (in pixels) x color depth (in bits)

The size of a **mono** sound file is calculated using the formula:

sample rate (in Hz) x sample resolution (in bits) x length of sample (in seconds)

The size of a **stereo** sound file is calculated using the formula:

2 x [sample rate (in Hz) x sample resolution (in bits) x length of sample (in seconds)]

Calculations Based Examples:

1) A camera detector has an array of 2048 by 2048 pixels and uses a color depth of 16. Find the size of an image taken by this camera in MiB.

We will apply the **formula = image resolution (in pixels) x color depth (in bits)** to obtain the answer in bits. Then we will convert bits to byte and then bytes to mebibyte (MiB) using following information:

- 1 byte = 8 bits
- 1 KiB = 1024 bytes
- 1 MiB = 1024 KiB

Hence, 1 MiB can be taken as 1024 KiB where 1 KiB = 1024 bytes so (1 MiB = 1024 x 1024).

1. First calculate the image resolution by multiplying the number of pixels in vertical and horizontal directions to find the total number of pixels:

   2048 x 2048 = 4 194 304 pixels
2. Then multiply the number of pixels (image resolution) by color depth 16 given in bits:

   4 194 304 x 16 = 67 108 864 bits
3. Then divide number of bits by 8 to find the number of bytes in the file:

   67 108 864 / 8 = 8 388 608 bytes
4. Finally divide number of bytes by (1024 x 1024) to convert it to MiB (mebibytes):

   8 388 608 / (1024 x 1024) = 8 MiB

2) A photograph is 1024 x 1080 pixels and uses a color depth of 32 bits. How many photographs of this size would fit onto a memory stick of 64 GiB?

We will apply the **formula = image resolution (in pixels) x color depth (in bits)** to obtain the answer in bits. Then we will convert bits to byte and then divide the bytes by 64 gibibyte (GiB) to find out number of photographs that can fit onto a memory stick using following information:

- 1 byte = 8 bits
- 1 KiB = 1024 bytes
- 1 MiB = 1024 KiB
- 1 GiB = 1024 MiB

Hence, 1 GiB can be taken as 1024 MiB where 1 MiB = 1024 KiB and 1 KiB = 1024 bytes so 1 GiB = 1024 x 1024 x 1024.

Therefore 64 GiB = 64 x 1024 x 1024 x 1024 bytes

1. First calculate the image resolution by multiplying the number of pixels in vertical and horizontal directions to find the total number of pixels:

   1024 x 1080 = 1 105 920 pixels
2. Then multiply the number of pixels (image resolution) by color depth 32 given in bits:

   1 105 920 x 32 = 35 389 440 bits
3. Then divide number of bits by 8 to find the number of bytes in the file:

   35 389 440 / 8 = 4 423 680 bytes
4. Then calculate the number of bytes in memory stick of 64 GiB (gibibytes):

   64 GiB = 64 x 1024 x 1024 x 1024 bytes = 68 719 476 736 bytes
5. Then divide the memory stick size by the files size to find out how many photographs would fit onto a memory stick of 64 GiB:

   68 719 476 736 bytes / 4 423 680 bytes = 15 534 photographs

3) An audio CD has a sample rate of 44 100 and a sample resolution of 16 bits. The music being sampled uses two channels to allow for stereo recording. Calculate the file size for a 60-minute recording in MiB.

We will apply the **formula = 2 x [sample rate (in Hz) x sample resolution (in bits) x length of sample (in seconds)]** to obtain the answer in bits. Then we will convert bits to byte and then bytes to mebibyte (MiB) using following information:

- 1 byte = 8 bits
- 1 KiB = 1024 bytes
- 1 MiB = 1024 KiB

Hence, 1 MiB can be taken as 1024 KiB where 1 KiB = 1024 bytes so (1 MiB = 1024 x 1024).

1. First calculate the number of seconds in 60 minutes by multiplying 60 with 60 to find out the length of sample in seconds:

   60 x 60 = 3600 seconds
2. Then multiply the sample rate 44 100 in Hz, sample resolution 16 in bits, and length of sample 3600 in seconds:

   44 100 x 16 x 3600 = 2 540 160 000 bits
3. Then multiply the number of bits with 2 since the recording is stereo and uses two channels

   2 540 160 000 x 2 = 5 080 320 000 bits
4. Then divide number of bits by 8 to find the number of bytes in the file:

   5 080 320 000 / 8 = 635 040 000 bytes
5. Finally divide number of bytes by (1024 x 1024) to convert it to MiB (mebibytes):

   635 040 000 / (1024 x 1024) = 605 MiB

Calculations Based Questions:

Q1. Kamil wants to store a 16-bit color image file. The image size is 1000 pixels.

Calculate the size of the file. Give your answer in bytes. Show your working.

- image resolution (in pixels) = 1000
- color depth (in bits) = 16

image resolution (in pixels) x color depth (in bits)

- 1000 x 16 = 16 000 bits
- 16 000 / 8 = 2000 bytes

Answer = 2000 bytes

Q2. Marley needs to store ten 8-bit color images in a file for his project. Each image is 500 pixels wide and 300 pixels high.

Calculate the total tile size in mebibytes (MiB) for all Marley's images. Show all your working.

- image resolution (in pixels) = 500 x 300
- color depth (in bits) = 8
- total number of images = 10

image resolution (in pixels) x color depth (in bits) x total number of images

- 500 x 300 = 150 000 pixels
- 150 000 x 8 = 1 200 000 bits
- 1 200 000 x 10 images = 12 000 000 bits
- 12 000 000 / 8 = 1 500 000 bytes
- 1 500 000/(1024 x 1024) = 1.43 MiB

Answer = 1.43 MiB

Q3. Each image taken requires 1 MiB of storage. If the camera captures an image every 5 seconds over a 24-hour period, how much storage is required?

Give your answer in gibibytes and show all your working.

- 24 hours x 60 minutes x 60 seconds = 86 400 seconds
- 86 400 / 5 seconds = 17 280 images in 24 hours
- 1 image = 1 MiB therefore 17 280 images = 17 280 MiB
- 17 280 MiB / 1024 = 16.9 (16.875) GiB

Answer = 16.9 GiB

Q4. A security system records video footage. One minute of video requires 180 MiB of storage. The recording system can store several hours of video footage.

Calculate how much storage would be needed for 2 hours of video footage. Show your working and give the answer in gibibytes (GiB).

- 2 hours x 60 minutes = 120 minutes
- 1 minute of video = 180 MiB therefore 120 minutes of video = 120 x 180 MiB
- 120 MiB x 180 MiB = 21 600 MiB of video footage stored in 2 hours
- 21 600 MiB / 1024 = 21.1 GiB

Answer = 21.1 GiB

Q5. A computer has 2048 MiB of RAM.

How many GiB of RAM does the computer have?

- 2048 MiB / 1024 = 2 GiB

Answer = 2 GiB

Q6. The current status of the engine is sent to a computer in the aero plane. Each piece of data collected is 8 bytes in size. Data collection occurs every 30 seconds. Calculate the number of kibibytes that would be needed to store the data collected during a 10-hour flight. Show your working.

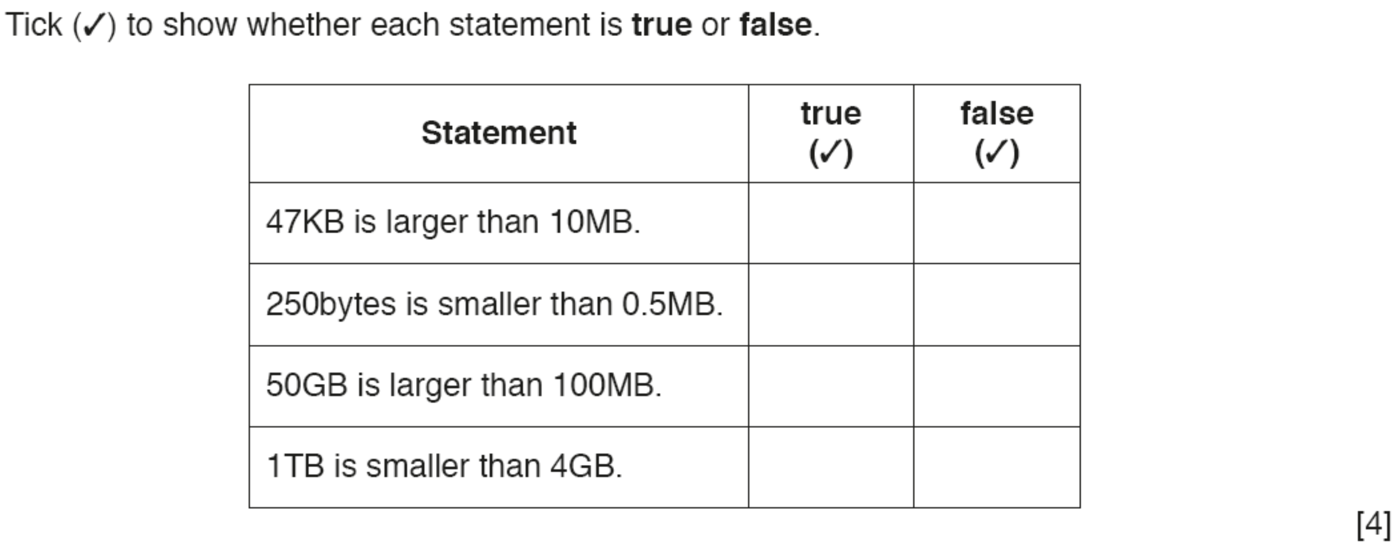
Step by step (i) data collection and (ii) working:

(i) One piece of data = 8 bytes
Data collected = every 30 seconds
Time of flight = 10 hour
Seconds in 1 hour = 3600
Answer needed in KiB

(ii) Data collected in 1 hour = 3600/30 = 120 times
Data collected in 10 hour = 120 x 10 = 1200 times
One piece of data = 8 bytes
1200 pieces of data = 8 x 1200 = 9600 bytes
In KiB = 9600/1024 = 9.4 KiB

Exam Style Questions:

Question 1:

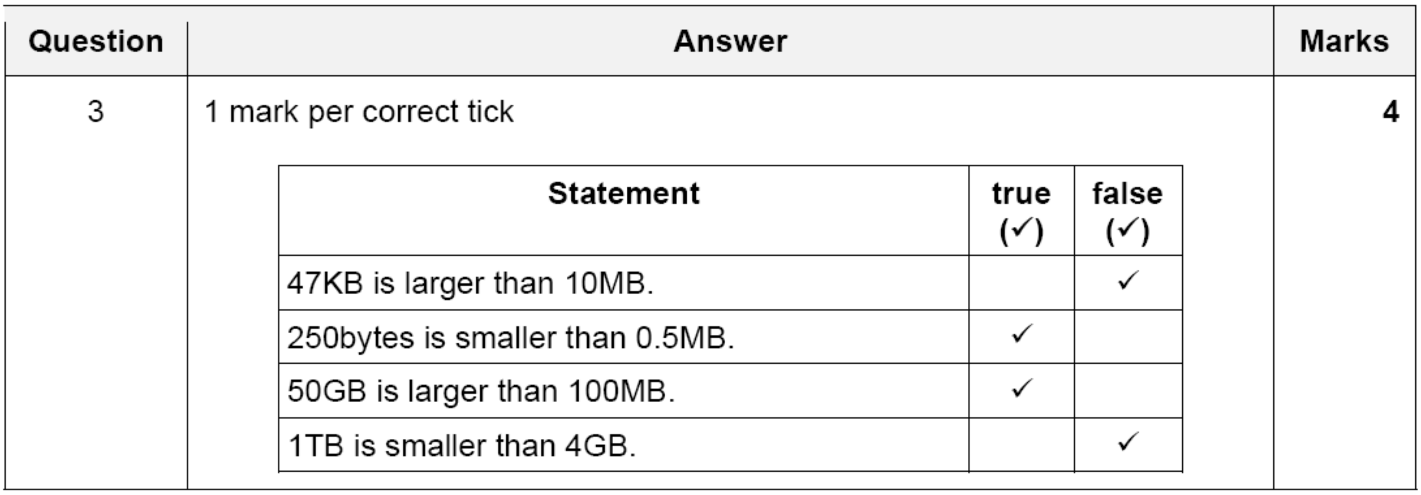


Tick (✓) to show whether each statement is **true** or **false**.

Statement	true (✓)	false (✓)
47KB is larger than 10MB.		
250bytes is smaller than 0.5MB.		
50GB is larger than 100MB.		
1TB is smaller than 4GB.		

[4]

Answer:



Question	Answer	Marks
3	1 mark per correct tick	**4**

Statement	true (✓)	false (✓)
47KB is larger than 10MB.		✓
250bytes is smaller than 0.5MB.	✓	
50GB is larger than 100MB.	✓	
1TB is smaller than 4GB.		✓

Question 2:

Different units of data can be used to represent the size of a file, as it changes in size.

Fill in the missing units of data, using the list given:

- byte
- gigabyte (GB)
- megabyte (MB)
- nibble

The units of data increase in size from smallest to largest.

Smallest bit

↓ ....

....

kilobyte (kB)

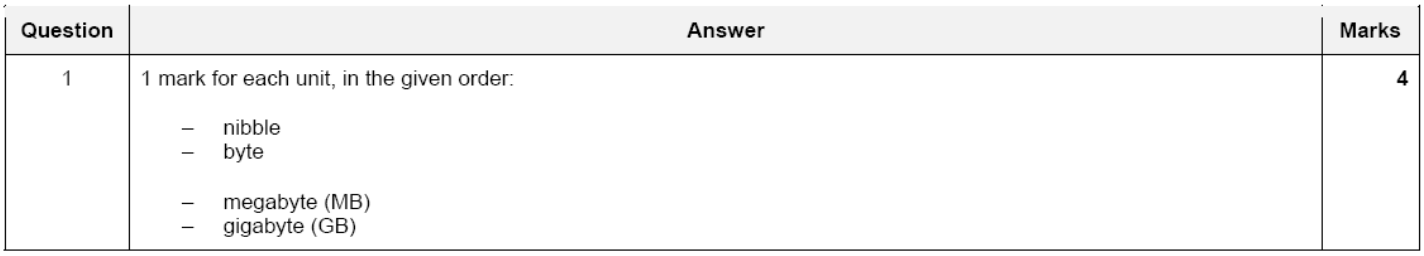
....

....

Largest terabyte (TB)

[4]

Answer:



Question	Answer	Marks
1	1 mark for each unit, in the given order: – nibble – byte – megabyte (MB) – gigabyte (GB)	4

Question 3:

An image is to be stored electronically.

The image is 256 pixels high by 200 pixels wide with a 16-bit colour depth.

Calculate the file size of the image. **You must show all of your working.**

File size .... kB [3]

Answer:

Question	Answer	Marks
12	256 □ 200 = 51 200 $\frac{51\,200 \square 16}{8} = 102\,400$ $\frac{102\,400}{1024} = 100$ Answer 100 kB One mark for correct answer and two marks for correct calculations.	3

Question 4:

Georgia is a wedding photographer. She wants to store 10 photographs on a USB flash memory drive for a customer. Each photograph is 100 pixels wide and 50 pixels high.

The photographs are 8-bit colour photographs.

(a) Calculate the total file size, in kilobytes (kB), of all the photographs. For this calculation, you may use the unit of measurement of 1024 or 1000.

Show all your working.

....

....

....

....

....

....

Answer .... kB

[3]

Answer:

4(a)	**Two** marks for any **two** correct workings and **one** mark for the correct answer. Working: – 100 × 50 = 5000 bits – 5000 × 8 = 40,000 bits – 40,000 / 8 = 5,000 bytes – 5,000 × 10 = 50,000 bytes – 50,000 / 1024 Answer: 48.83 kB // 49 kB **NOTE:** Alternative correct methods of working can be credited. Answer can be given to any number of dp.	**3**

Question 5:

A 32-second sound clip will be recorded. The sound will be sampled 16000 times a second.

Each sample will be stored using 8 bits.

Calculate the file size in kilobytes. **You must show all of your working.**

File Size .... kB

[3]

Answer:

9	Max 3 – 1 mark for correct answer and 2 marks for correct calculations. Any **two** from: 16000 □ 32 512000 / 1024 Or 16000 □ 8 128000 □ 32 4096000 / 8 512000 / 1024 **Correct answer:** 500 kB	**3**

Question 6:

(a) Elle has a file stored on her computer that is 20 MB in size. Jordan has a file that is 10 GB in size.

Tick (✓) to show which is the **larger** file.

File size	**Tick (✓)**
20 MB	
10 GB	

[1]

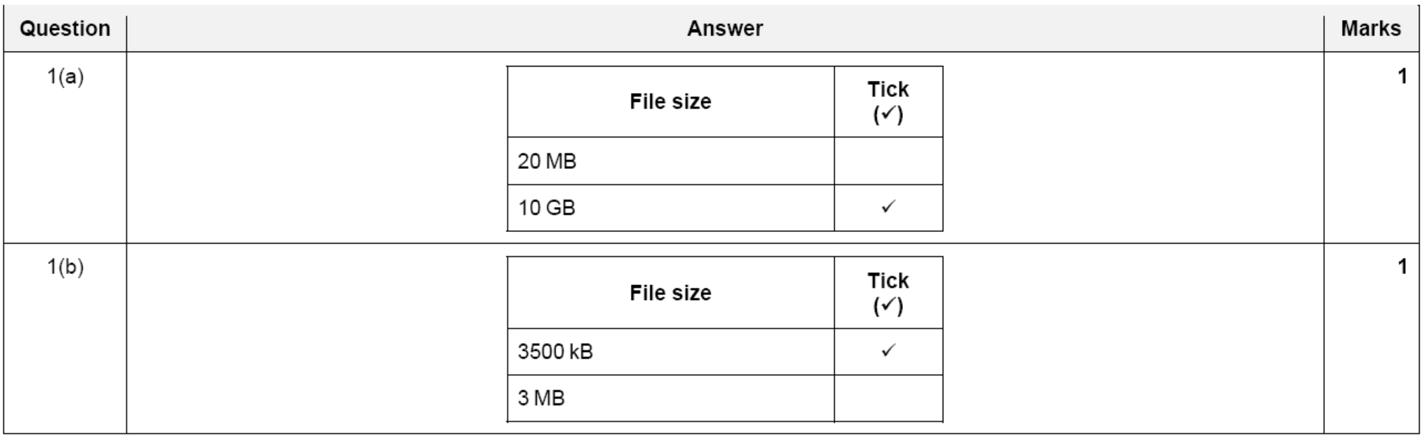
(b) Bob has a file stored on his computer that is 3500 kB in size. Gerty has a file that is 3 MB in size.

Tick (✓) to show which is the **larger** file.

File size	**Tick (✓)**
3500 kB	
3 MB	

[1]

Answer:



<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)</td><td><table><tr><th>File size</th><th>Tick
(✓)</th></tr><tr><td>20 MB</td><td></td></tr><tr><td>10 GB</td><td>✓</td></tr></table></td><td>1</td></tr>
<tr><td>1(b)</td><td><table><tr><th>File size</th><th>Tick
(✓)</th></tr><tr><td>3500 kB</td><td>✓</td></tr><tr><td>3 MB</td><td></td></tr></table></td><td>1</td></tr>
</table>

Question 7:

(a) Draw **one** line from each binary value to its equivalent (same) value on the right.

Binary value

8 bits

8000 bits

1000 kilobytes

1024 mebibytes

8192 bits

1 kibibyte

1 gigabyte

1 byte

1 kilobyte

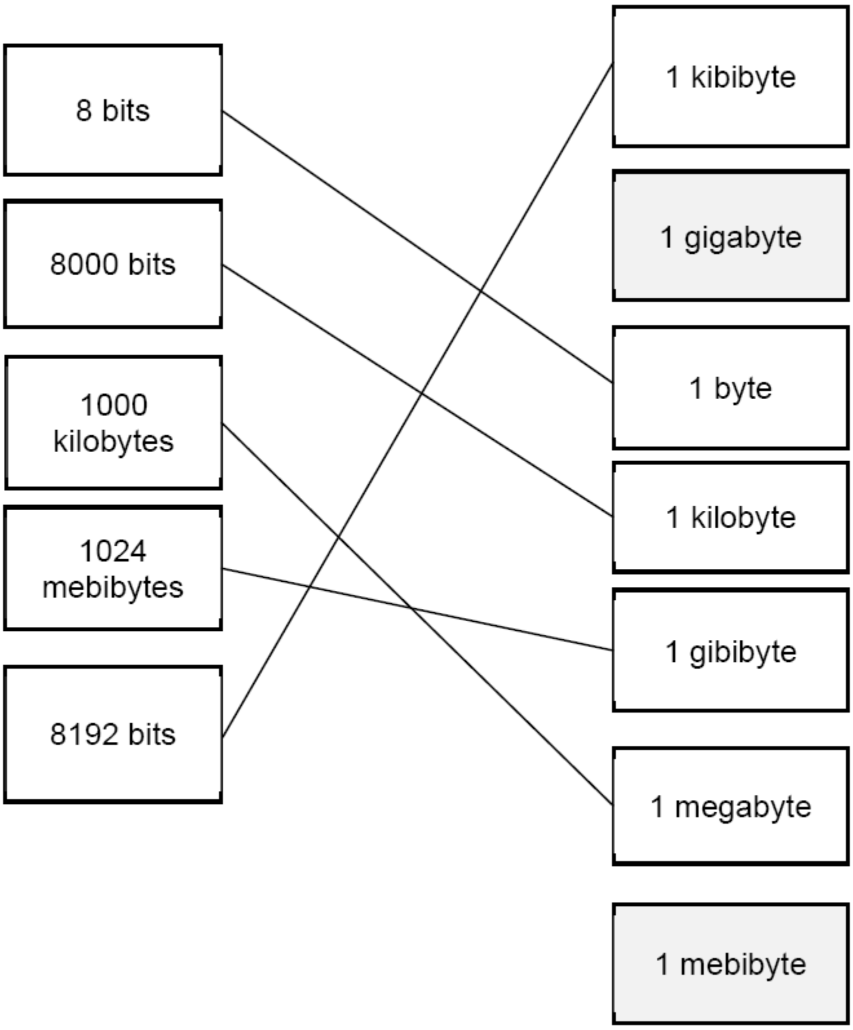
1 gibibyte

1 megabyte

1 mebibyte

[5]

Answer:

Question	Answer	Marks
1(a)	**1 mark** for each correct line  8 bits 8000 bits 1000 kilobytes 1024 mebibytes 8192 bits 1 kibibyte 1 gigabyte 1 byte 1 kilobyte 1 gibibyte 1 megabyte 1 mebibyte	**5**

Question 8:

(a) State **one** difference between a **kibibyte** and a **kilobyte**.

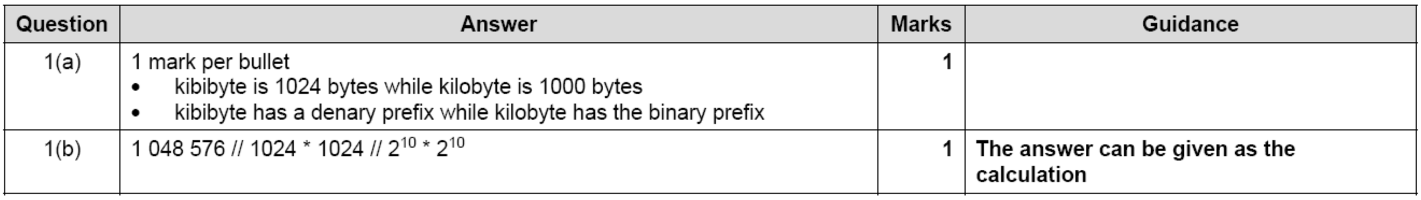
....

.... [1]

(b) Give the number of bytes in a **mebibyte**.

.... [1]

Answer:



Question	Answer	Marks	Guidance
1(a)	1 mark per bullet • kibibyte is 1024 bytes while kilobyte is 1000 bytes • kibibyte has a denary prefix while kilobyte has the binary prefix	1	
1(b)	1 048 576 // 1024 * 1024 // 2^{10} * 2^{10}	1	**The answer can be given as the calculation**

Question 9:

Anya scans an image into her computer for a school project.

(a) The scanned image is a bitmapped image.

(ii) The image is scanned with an image resolution of 1024 × 512 pixels, and a colour depth of 8 bits per pixel.

Calculate an estimate for the file size, giving your answer in mebibytes. Show your working.

Working ....

....

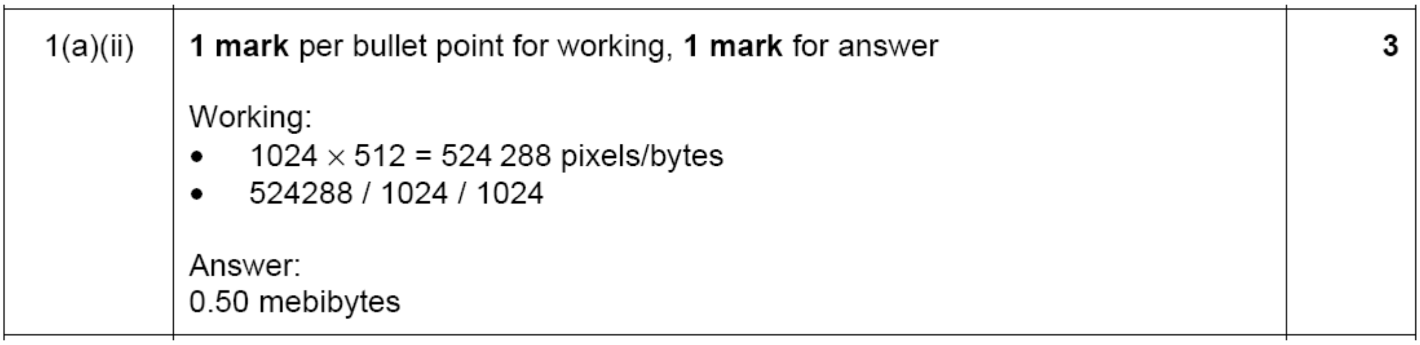
....

....

Answer .... mebibytes

[3]

Answer:



1(a)(ii)	**1 mark** per bullet point for working, **1 mark** for answer Working: • 1024 × 512 = 524 288 pixels/bytes • 524288 / 1024 / 1024 Answer: 0.50 mebibytes	3

Question 10:

A digital camera takes a bitmap image. The image is 2000 pixels wide by 1000 pixels high with a colour depth of 24-bits.

(a) Calculate an estimate of the file size for the image. Give your answer in megabytes. Show your working.

Working ....

....

....

....

....

Answer .... MB

[3]

(b) A second image is taken, this time in black and white. It has the same number of pixels, but the file size is smaller.

Explain why the file size is smaller.

....

....

....

.... [2]

Answer:

Question	Answer	Marks
4(a)	**1 mark** per bullet point • 2000 * 1000 * 24 = 48 000 000 bits • 48 000 000 / 8 / 1024 / 1024 • = 6 MB or 5.7 MB	**3**
4(b)	**1 mark** per bullet point to **max 2** • Only 1 bit needed to store the colour of each pixel ... • ... so number of pixels * bit depth is 2000 * 1000 * 1 (rather than 2000 * 1000 * 24) • ... so the calculation (in part 4(a)) results in smaller figure for file size	**2**

Question 11:

Xander creates a presentation that includes images, video and sound.

(a) The images are bitmap images. A bitmap image can be made up of any number of colours. Each colour is represented by a unique binary number.

Draw **one** line from **each** box on the left, to the correct box on the right to identify the minimum number of bits needed to store each maximum number of colours.

Maximum number of colours

68

256

127

2

249

Minimum number of bits

1

2

3

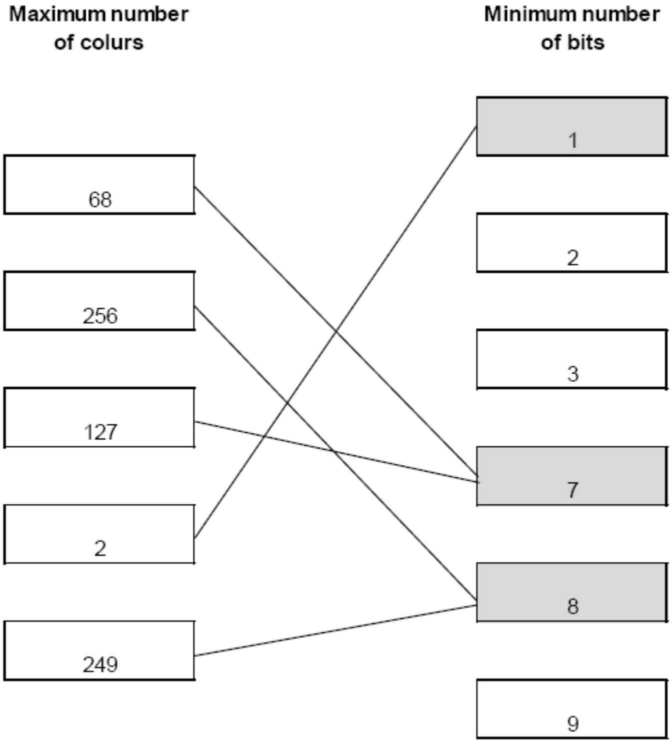
7

8

9

[3]

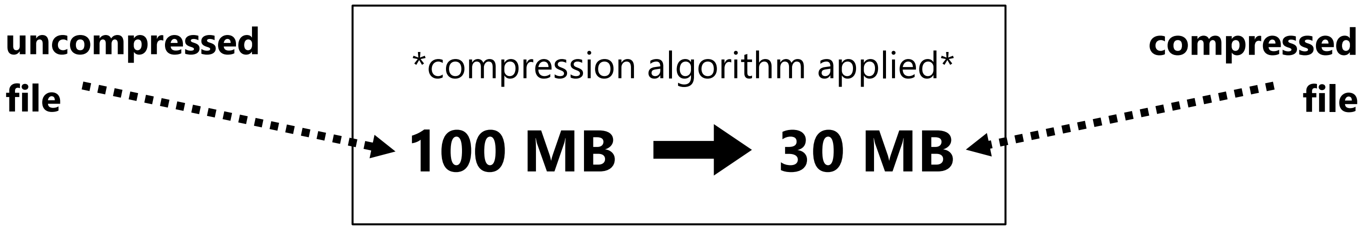
Answer:

Question	Answer	Marks
5(a)	**1 mark** for each correctly linked box on the right  Maximum number of colurs Minimum number of bits 68 256 127 2 249 1 2 3 7 8 9	**3**

1.3.3 Data Compression:

Purpose & Need of Compression:

- The size of sound and image files can be very large and so compression exists to reduce the size of the file.
- It means reducing the file size by either temporarily (lossless compression) or permanently (lossy compression) removing data.



Why do files need to be compressed?

1. The data files are very large so it would take a long time to send the uncompressed file.
2. A higher bandwidth would be needed to transmit the uncompressed file.
3. A greater amount of storage space would be required for storing uncompressed files.

Benefits of Compression:

1. Compressing the file reduces its size and hence less storage space is required.
2. Since the file is smaller, it is downloaded, uploaded, and transmitted quicker/in shorter time.
3. It requires less bandwidth if transmitted.
4. It will require less time to transmit.

1.3.4 Lossy & Lossless Compression:

File compression can either be lossy or lossless:

- Lossy compression reduces the file size by permanently removing data, e.g. reducing resolution or color depth, reducing sample rate or sampling resolution.
- Lossless compression reduces the file size without permanent loss of data, e.g. run length encoding (RLE)

Lossy File Compression:

- Lossy compression is the removal of unnecessary bits of data.
- It results in loss of detail compared to original file.
- It does not allow original file to be re-created exactly.
- It can compress and reduce data to about 10%.
- The decompressed file is not the same as the original.
- Examples of compressed formats include jpeg, mp3, mp4 etc.

Use of lossy compression:

1. It is used to compress music files, video files & photos.
2. It is used where the data being compressed does not need to be exactly the same as original file.
3. It is used for files where removing certain bits doesn’t detract from the quality.

Working:

- In an image, lossy compression may reduce the image resolution and/or the color depth (bit depth).
- In a sound file, lossy compression may reduce the sampling rate and/or the sampling resolution.

The common lossy file compression algorithms are:

1. MPEG-3 (MP3) and MPEG-4 (MP4)
2. JPEG

i) In case of sound/music/audio files:

- A compression algorithm called perceptual music shaping is used.
- It involves removing unnecessary sounds such as removing background noise and sounds humans can't hear very well (e.g. if two sounds played at the same time, softer sound is removed).
- It reduces the sample rate and sample resolution.
- As a result, it reduces size and permanently removes certain parts of the sound without affecting the quality too much.
- After compression, due to smaller size, the music can be streamed faster, sent quicker, uploaded faster, and downloaded faster as well.

Lossy compression techniques that can be used to reduce the size of the sound file:

- Reduce sampling rate so fewer samples taken per second means less data is being stored.
- Reduce sample resolution so fewer bits are used to represent each sample, so less data is stored.

MP3 file format reduces file size by about 90%. How the music quality is apparently retained:

- MP3 is a lossy compressed format.
- It uses psycho-acoustic modeling and perceptual music shaping (compression algorithm).
- It eliminates certain parts of the music without significantly degrading the listeners experience.
- It removes the sound that the human ear cannot hear.
- It only keeps sounds human ear can hear better than others.
- It discards softer sound if two sounds are played together.

MPEG-3 (MP3):

- It is the digital recording of sound produced by a recording software or microphone.
- It uses lossy file compression algorithm called perceptual music shaping which involves removing sounds human ear can’t hear very well (e.g. if two sounds are played at the same time, softer sound is removed).
- As a result, it removes certain parts of the music without affecting the quality too much.
- It is widely used when distributing sound files.

ii) In case of video files:

- It uses a compression algorithm that permanently removes all the unnecessary data from the file such as details that the human eye cannot see or the sound that the human ear cannot hear (perceptual music shaping).
- It reduces the color depth, so each pixel requires fewer bits.
- It reduces the image resolution by decreasing the number of pixels per centimeter.
- It removes the repeating frames or reduces frame rate.
- It reduces the sample rate and sample resolution for the sound.
- As a result, the size and quality of the video is reduced but it can still be viewed.
- After compression, due to smaller size, the videos can be streamed faster, sent quicker, uploaded faster, and downloaded faster as well.

MPEG-4 (MP4):

- This format allows the storage of multimedia files rather than just sound (e.g. music, videos, photos and animation etc.).
- MP4 format allows videos to be streamed over the internet without losing any real discernable quality.
- It's a highly versatile format that can also store audio, subtitles, and still images.

iii) In case of photos:

- It uses a compression algorithm that permanently removes all the unnecessary data from the file such as details that the human eye cannot see.
- It reduces the color depth, so each pixel requires fewer bits.
- It reduces the image resolution by decreasing the number of pixels per centimeter.
- It reduces color palette/number of colors, so fewer bits are needed to store each color.
- As a result, the size of the image is reduced without real loss of quality.
- After compression, due to smaller size, the photos can be sent quicker, uploaded faster and downloaded faster as well.

Lossy compression techniques that can be used to reduce the size of the image file:

- Reduce the color depth so number of bits per color are reduced and so each pixel has fewer bits.
- Reduce image resolution so fewer pixels per unit measurement which means fewer binary numbers are stored.
- Reduce color palette/number of colors so fewer colors mean fewer bits needed to store each color.

JPEG (Joint Photographic Experts Group):

- It uses lossy file compression to reduce the size of photographs.
- It reduces the quality of an image by decreasing number of pixels per cm (picture resolution).
- It reduces a raw bitmap image by a factor of between 5 and 15 depending on the quality of the original image.
- It relies on certain properties of human eye as the eye detects limited colors or differences in brightness and so reduces the size without real loss of quality.

Advantages of Lossy Compression:

1. It reduces the file size more than lossless hence saves more memory.
2. Due to smaller size, file can be downloaded, uploaded or sent quicker.
3. It will require less time and less bandwidth if transmitted.

Disadvantages of Lossy Compression:

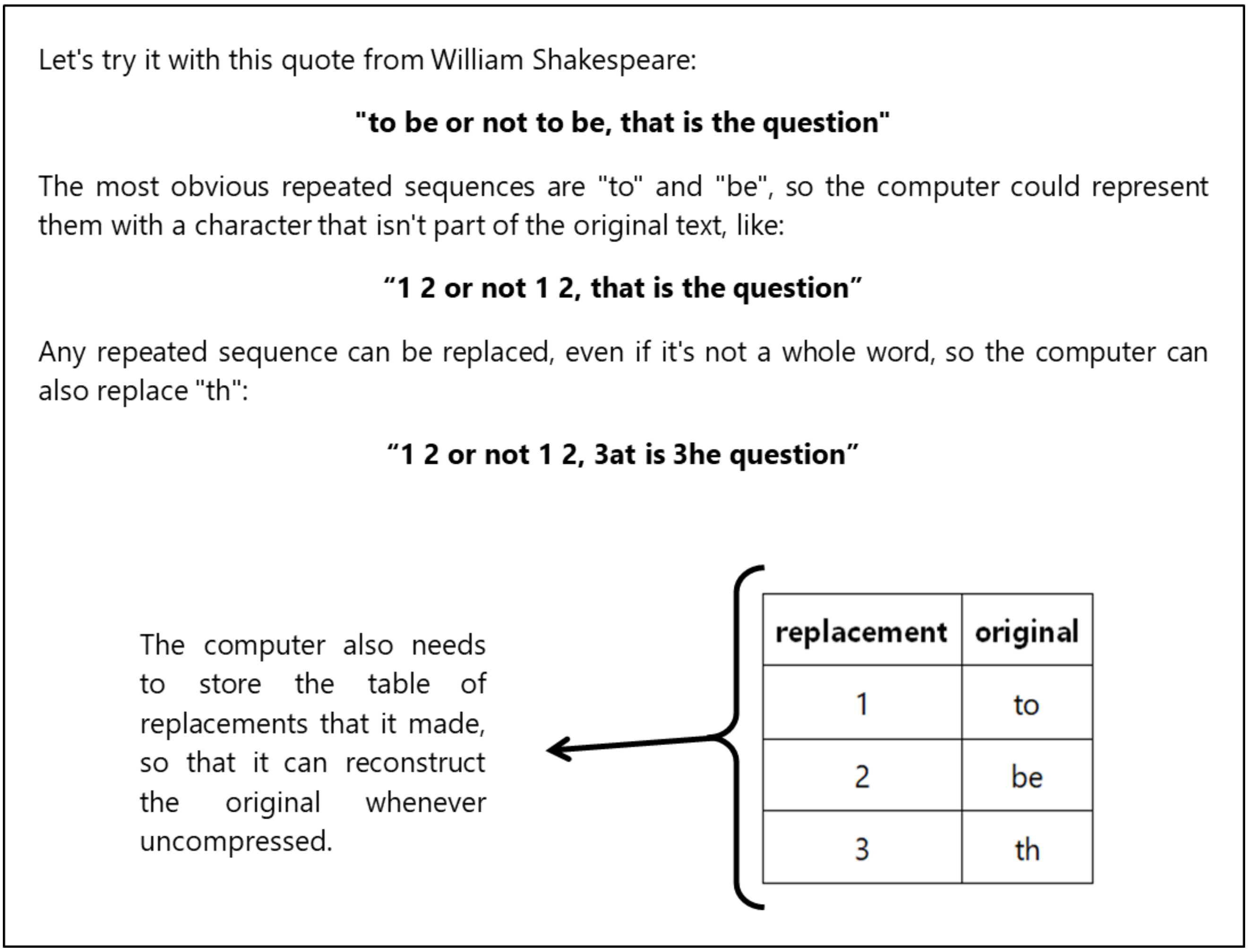
1. The quality of the image, audio or video will be reduced.
2. The original file cannot be restored.

Lossless File Compression:

- Lossless compression is designed to lose none of the original detail.
- It allows original file to be re-created exactly when uncompressed.
- It is based on some form of replacement, for example RLE, FLAC etc.
- By example: 000-1111-222222-333 = 3–0, 4–1, 6–2, 3–3 etc.
- It can compress and reduce data to about 50% maximum.

Example:

- Computers can compress text by **finding repeated sequences** and **replacing them** with **shorter representations** such as any numerical value or a character.



Let's try it with this quote from William Shakespeare:

"to be or not to be, that is the question"

The most obvious repeated sequences are "to" and "be", so the computer could represent them with a character that isn't part of the original text, like:

“1 2 or not 1 2, that is the question”

Any repeated sequence can be replaced, even if it's not a whole word, so the computer can also replace "th":

“1 2 or not 1 2, 3at is 3he question”

The computer also needs to store the table of replacements that it made, so that it can reconstruct the original whenever uncompressed.

replacement	original
1	to
2	be
3	th

Use of lossless compression:

1. It is used in compressing text files.
2. It is used where the loss of any data would be disastrous such as spreadsheets file.
3. It is used where the data being compressed needs to be exactly the same as original file such as program codes for computer.

Run-Length Encoding:

- It is a lossless method of data compression.
- It reduces the physical size of a string of adjacent, identical characters, pixels, or bytes etc.
- The repeating string (a run) is encoded into two values.
- The first value represents the number of identical data items (e.g. characters) in the run (run count).
- The second value is the code of the data item (e.g. ASCII code/color code of pixel etc.) in the run (run value).
- The run value and run count combination may be preceded by a control character.
- For example: 000-1111-222222-333 = 3–0, 4–1, 6–2, 3–3 etc.

i) In case of sound/music/audio files:

- It will reduce the amplitude to only the range used so limited amplitudes mean fewer bits per sample.
- A compression algorithm Run-Length Encoding (RLE) is used, and no data is permanently removed.
- The binary value of repeated patterns of notes or same sounds are identified, grouped, and replaced with a numerical value (indexed).
- The value is then stored in an index/dictionary along with the number of times the patterns/notes are repeated so the data can be restored.
- It records the changes instead of the actual sounds.
- The original file/data is restored when uncompressed.

ii) In case of video files:

- A compression algorithm Run-Length Encoding (RLE) is used, and no data is permanently removed.
- The repeated frames/pixels in the video are identified, grouped, and replaced with a numerical value (indexed).
- The value is then stored in an index/dictionary along with the number of times frames/pixels are repeated so the data can be restored.
- It just records the changes between the frames/pixels of the actual video.
- The original file/data is restored when uncompressed.

iii) In case of text files:

- A compression algorithm Run-Length Encoding (RLE) is used, and no data is permanently removed.
- The groups of repeated characters are identified and replaced with a numerical value.
- The value is then stored in an index/dictionary along with the number of times the character occurs so the data can be restored.
- For example, instead of storing aaaa, it will simply store a4
- The original file/data is retrieved when uncompressed.

Why run-length encoding will sometimes increase the size of a text file:

- The repeated sequences of characters rarely occur in text files as most characters are used only once in any sequence.
- The character code and the fact that it is stored once will both be stored, which will use as much if not more space.

The following text is stored as a text file:

She sells sea shells on the seashore. The shells that she sells are sea shells I am sure.

How lossless compression would compress this file.

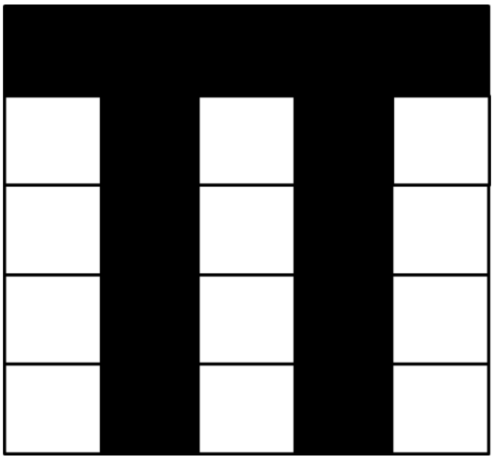
- The data is compressed using a compression algorithm (e.g. RLE).
- The repeating words or sections of words are identified and replaced with a numerical value.
- The value is then stored in an index/table along with word/sections of words so the data can be restored.
- The original file/data is retrieved when uncompressed.
- For example, the word “shell” repeats thrice so it can be replaced with a numerical value and stored in an index. (e.g. shell = 7).

iv) In case of photos:

- A compression algorithm Run-Length Encoding (RLE) is used, and no data is permanently removed.
- The sequences of the same color pixels in the picture are identified, grouped, and replaced with a numerical value (indexed).
- The value is then stored in an index/dictionary along with the number of identical pixels that occurs so the data can be restored.
- It just records the changes between the frames/pixels.
- The original file/data is restored when uncompressed.

Example Questions:

Question 1: How a computer can store this black and white bitmap image.



- Each pixel in the image requires only one bit (as there are only two colors).
- The black color is represented by binary value 1 and white by 0 (or vice versa).
- The bits are stored for each pixel in sequence e.g. 11111 01010 01010 01010 01010.

How run-length encoding (RLE) will compress the image.

- RLE will store the color and the number of times it occurs.
- In this case, it will store black and the number of times it occurs as well as white and its occurrence e.g. B5, W1, B1, W1, B1, and so on.

Why is it important to estimate the file size of an image?

- It is important for estimating how many images can be stored.
- It is important for deciding if it can be sent as an email attachment.

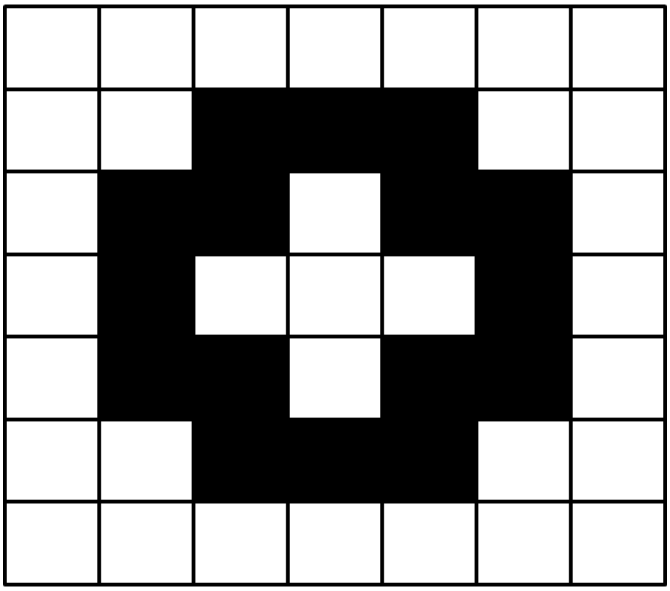
A digital camera takes a bitmap image. The image is 2000 pixels wide by 1000 pixels high with a color depth of 24-bits.

A second image is taken, this time in black and white. It has the same number of pixels, but the file size is smaller.

Why the file size is smaller:

- For black and white, only 1 bit is needed to store the color of each pixel so the number of pixels * bit depth is 2000 * 1000 * 1 (rather than 2000 * 1000 * 24).
- This calculation results in a smaller figure for file size.

Question 2: How run-length encoding (RLE) is used to store the image with the following color codes:



Color	Code
Black	1A
White	3B

- 3B9 1A3 3B3 1A2 3B1 1A2 EB2
- 1A1 3B3 1A1 3B2 1A2 3B1 1A2 3B3 1A3 3B9

Question 3: How run-length encoding (RLE) would compress this image.

R	R	P	P	P	G
B	R	R	P	G	G
B	W	B	B	O	O
B	W	W	P	P	O
B	B	R	P	G	O
B	R	R	P	G	O

- RLE will look for runs of consecutive pixel of the same color.
- It stores the color value once and the number of times it occurs e.g. R2 P3 G1 B1 R2 P1 G2 and so on.

Question 4: The following diagrams show:

- **the denary color code that represents each color**
- **the first three rows of a bitmap image**

Colour symbol	Colour code (denary)
B	153
W	255

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	B	B	B	B	B	B	B	B	B	B	W	W	W	B	B	B
1	B	B	B	B	B	B	B	B	B	W	W	W	W	W	W	B
2	B	B	B	B	B	B	B	W	W	W	W	W	W	W	W	W
...																
95																

How RLE will compress the first three rows of this image:

- **Row 1:** 153 10 255 3 153 3
- **Row 2:** 153 9 255 6 153 1
- **Row 3:** 153 7 255 9

Question 5: A section of bitmap image is shown. Each color is represented by a letter, such as B is blue.

B	B	B	B	B	B	B	B	B	B
Y	Y	P	Y	Y	Y	P	Y	Y	Y
R	R	M	R	P	K	T	T	R	R
B	O	P	Y	Y	Y	P	G	P	P
R	O	R	P	P	P	R	R	R	R

The first row of pixels in the image is shown:

B	B	B	B	B	B	B	B	B	B

How this row of pixels can be compressed using lossless compression:

- It can be compressed using run-length encoding (RLE).
- It stores the color value Blue and the number of times it occurs.
- For example: B10.

Advantages of Lossless Compression:

1. It reduces the file size hence saves memory.
2. The original file can be restored.
3. Due to smaller size, file can be downloaded, uploaded or sent quicker.
4. It will require less time and bandwidth if transmitted.

Disadvantages of Lossless Compression:

1. It reduces the file size lesser than lossy compression.

Exam Style Questions:

Question 1: Nigel wants to send a large text file electronically to Mashuda. Describe how the size of the text file can be reduced. (4)

- The file can be compressed using lossless compression.
- A compression algorithm is used (e.g. RLE).
- The repeating words or sections of words are identified and replaced with a numerical value.
- The value is then stored in an index/table along with word/sections of words so the data can be restored.
- The original file/data is retrieved when uncompressed.
- Another way of compressing the data is by converting it into either ZIP or PDF file formats.

Question 2: Two types of compression are lossy and lossless. Choose the most suitable type of compression for the following and explain your choice. (6)

(i) Downloading the code for a computer program: (3)

- Lossless compression should be used to ensure that no data is lost.
- It is important the code must be exactly the same as the original file.
- If it does not match the original file, it will not work.

ii) Streaming a video file: (3)

- Lossy compression should be used so redundant data bits are removed.
- It would make the file smaller than lossless compression and the file will also stream faster.
- The quality of the video will be reduced but it can still be viewed.

Question 3: The file Michele is sending contains the source code for a large computer program. Identify which type of compression would be most suitable for Michele to use. Explain your choice. (4)

- Lossless compression should be used so that the file can be restored to the exact same/original state it was in before compression.
- It is a computer program so no data should be lost.
- If any data is lost, then it will not run correctly so lossy compression should not be used at all.
- Lossless compression will give the repeating words or sections of words a numerical value.
- The value is then recorded in an index along with the words/sections of words so the data can be restored when uncompressed.

Question 4: Nancy wants to email the photos to Nadia.

Many of the photos are very large files, so Nancy needs to reduce their file size as much as possible. Identity which type of compression would be most suitable for Nancy to use.

Explain your choice (4)

- Lossy compression would be more suitable as it would reduce the file size more than lossless.
- It will remove all the unnecessary data from the file such as details that the human eye cannot see (e.g. humans can only detect limited colors or differences in brightness).
- The image quality will be reduced by decreasing number of pixels per centimeter (picture resolution).
- As a result, the size of the image is reduced without any real loss of quality.
- Moreover, there is no requirement for the files to be exactly the same as original file so lossless should not be used.
- After compression, due to smaller size, the photos can be sent quicker, uploaded faster and downloaded faster as well.

Question 5: David needs to send a large section of the programming code as an email attachment. He uses lossless compression to reduce the file size.

Explain how the file size is reduced. (3)

- The data is compressed using a compression algorithm (e.g. RLE).
- The repeating words or sections of words are identified and replaced with a numerical value.
- The value is then stored in an index/table along with word/sections of words so the data can be restored.
- The original file/data is retrieved when uncompressed.

Question 6: Audrey wants to send a sound tile to Nico using email.

The file is too large to attach to an email so Audrey decides to compress the file.

She uses lossy compression to reduce the size of the sound film.

(a) Describe how lossy compression reduces the size of the sound file. (4)

- A compression algorithm called perceptual music shaping is used.
- It involves removing unnecessary sounds such as removing background noise and sounds humans can't hear very well (e.g. if two sounds played at the same time, softer sound is removed).
- It reduces the sample size along with sample rate and the sound is clipped.
- As a result, it reduces size and permanently removes certain parts of the music without affecting the quality too much.

(b) Nico asks Audrey why she used lossy compression rather than lossless.

(i) State one advantage Audrey could give of using lossy rather than lossless to compress the sound file. (1)

1. The file size will be smaller than lossless.
2. The file will require less storage space.
3. It will require less time to transmit.
4. It will use less bandwidth to be transmitted.

(ii) State one disadvantage Nico could give of using lossy rather than lossless to compress the sound file. (1)

1. The quality of the sound will be reduced.
2. The original file cannot be restored.

Question 7: Videos on the library website show customers which books the library will soon have in stock.

The library wants the file size of a video to be as small as possible.

Identity and describe a method the library could use to reduce the file size of a video as much as possible. (4)

- Compression should be used and the best choice of compression would be lossy.
- It uses a compression algorithm that removes all the unnecessary data from the file such as details that the human eye cannot see or the sound that the human ear cannot hear.
- It reduces the color palette so each pixel requires fewer bits and decreases the quality by decreasing the number of pixels per centimeter (video resolution).
- It only stores what changes between frames.
- As a result, the size and quality of the video is reduced but it can still be viewed.

Question 8:

A music company has a website that allows users to stream music. The music is stored in sound files.

(a) The sound files are compressed using lossless compression.

(i) Describe how the sound files are compressed using lossless compression.

....

....

....

....

....

....

....

....

....

....

.... [4]

(ii) State **one** reason why the music company would compress the sound files using lossless, rather than lossy, compression.

....

.... [1]

(iii) Give **one** benefit, to the user, of the music company compressing the sound files.

....

.... [1]

(iv) Give **one** drawback of the music company using lossless, rather than lossy, compression for the sound files.

....

.... [2]

Answer:

Question	Answer	Marks
7(a)(i)	**Four** from: – (Compression) **algorithm** is used – No data will be removed // original file can be restored – Example of type of algorithm that would be used e.g. RLE – Repeated patterns in the music are identified – … and indexed NOTE: If another lossless method is described, marks can be awarded.	**4**
7(a)(ii)	Any **one** from: – To provide the highest quality of music file (that compression will allow) – The user is able to listen to the original sound file – No loss of quality for the sound file provided	**1**
7(a)(iii)	Any **one** from: – Allow for quicker streaming speed – Would not require as much bandwidth (to stream) – Does not need as much RAM – Smoother listening experience // less lag – Will not use as much of data allowance	**1**
7(a)(iv)	**Two** from: – Streaming speed may be slower – … and may affect listening experience // buffering may occur – User may need more bandwidth to stream – … that could be more expensive – It would be a larger file size – … so may take longer to upload – … so will take up more storage space … – … on webserver	**2**

Question 9:

(b) Sammi creates videos for the finance company website that give customers advice about their finances.

He uses lossy compression to reduce the file size of the videos for the website.

(i) Give **three** ways that lossy compression can reduce the file size of the videos.

1 ....

....

2 ....

....

3 ....

....
[3]

(ii) Give **one** drawback of using lossy compression to reduce the file size of the videos.

....

.... [1]

(c) Sammi could have used lossless compression to compress the videos for the website.

(i) Give **one** reason why he would use lossless compression, rather than lossy compression, for the videos.

....

.... [1]

(ii) Give **two** disadvantages of Sammi using lossless compression, rather than lossy compression, for the videos.

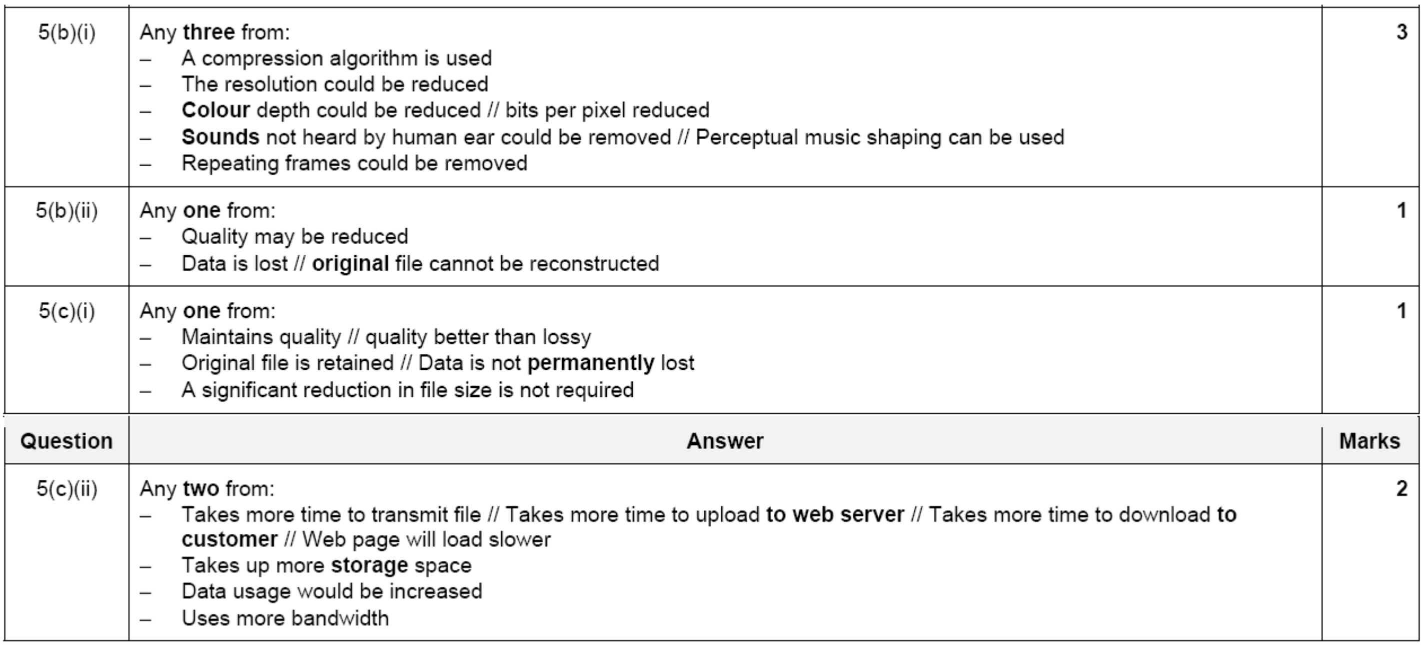
Disadvantage 1 ....

....

Disadvantage 2 ....

....
[2]

Answer:



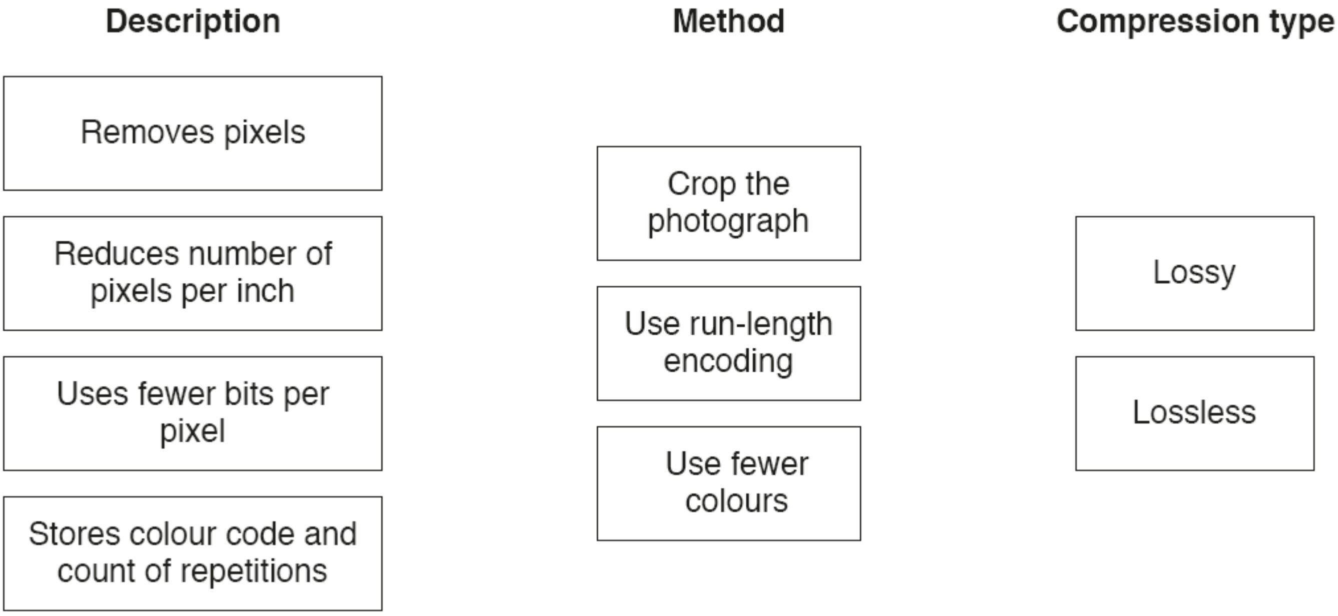
5(b)(i)	Any **three** from: – A compression algorithm is used – The resolution could be reduced – **Colour** depth could be reduced // bits per pixel reduced – **Sounds** not heard by human ear could be removed // Perceptual music shaping can be used – Repeating frames could be removed	3
5(b)(ii)	Any **one** from: – Quality may be reduced – Data is lost // **original** file cannot be reconstructed	1
5(c)(i)	Any **one** from: – Maintains quality // quality better than lossy – Original file is retained // Data is not **permanently** lost – A significant reduction in file size is not required	1
Question	**Answer**	**Marks**
5(c)(ii)	Any **two** from: – Takes more time to transmit file // Takes more time to upload **to web server** // Takes more time to download **to customer** // Web page will load slower – Takes up more **storage** space – Data usage would be increased – Uses more bandwidth	2

Question 10:

(ii) The file size of the photograph needs to be reduced before it is placed on the website.

Draw lines to link each method of reducing the file size of the image to:

- its description and
- its compression type, where appropriate.



Description

Removes pixels

Reduces number of pixels per inch

Uses fewer bits per pixel

Stores colour code and count of repetitions

Method

Crop the photograph

Use run-length encoding

Use fewer colours

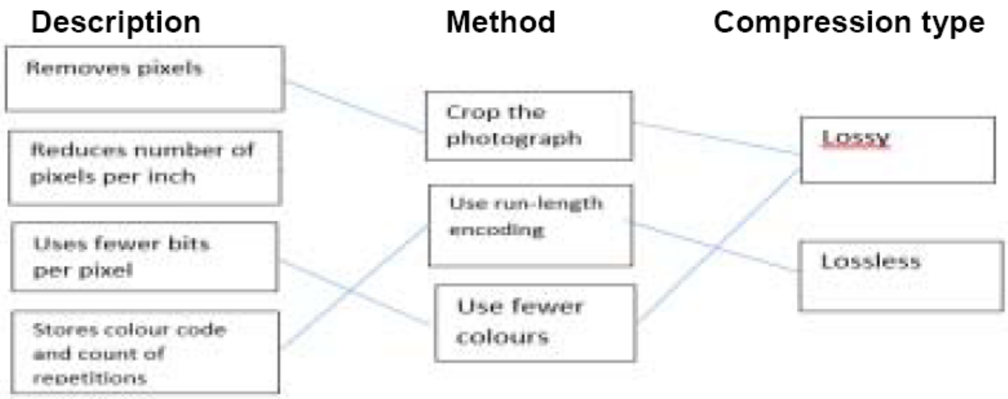
Compression type

Lossy

Lossless

[5]

Answer:

1(b)(ii)	**1 mark** per method correctly linked to its description **max 3** **1 mark** for each compression type correctly linked to its method(s). **max 2** 	**5**

Question 11:

(ii) The photograph needs to be sent by email but the file size is too big. It needs to be compressed.

The table lists several methods of making an image file size smaller.

Tick (✓) **one** box on each row to indicate whether each method is lossy or lossless.

Compression method	Lossy	Lossless
Cropping the image		
Reducing the resolution of the image		
Using run-length encoding (RLE)		
Reducing the colour depth of the image		

[4]

Answer:

1(b)(ii)	**1 mark** for correct tick in each row.	**4**

Compression method	Lossy	Lossless
Cropping the image	✓	
Reducing the resolution of the image	✓	
Using run-length encoding (RLE)		✓
Reducing the colour depth of the image	✓	

Question 12:

(e) Data can be compressed using either lossy or lossless compression.

Tick (✓) **one** box in each scenario to identify whether lossy or lossless compression should be used. Justify your choice.

(i) A program written in a high-level language.

Lossy	**Lossless**

Justification ....

....

....

.... [2]

(ii) A photograph that needs to be emailed to a friend.

Lossy	**Lossless**

Justification ....

....

....

.... [2]

(iii) You need to upload a video that you have created to a website.

Lossy	**Lossless**

Justification ....

....

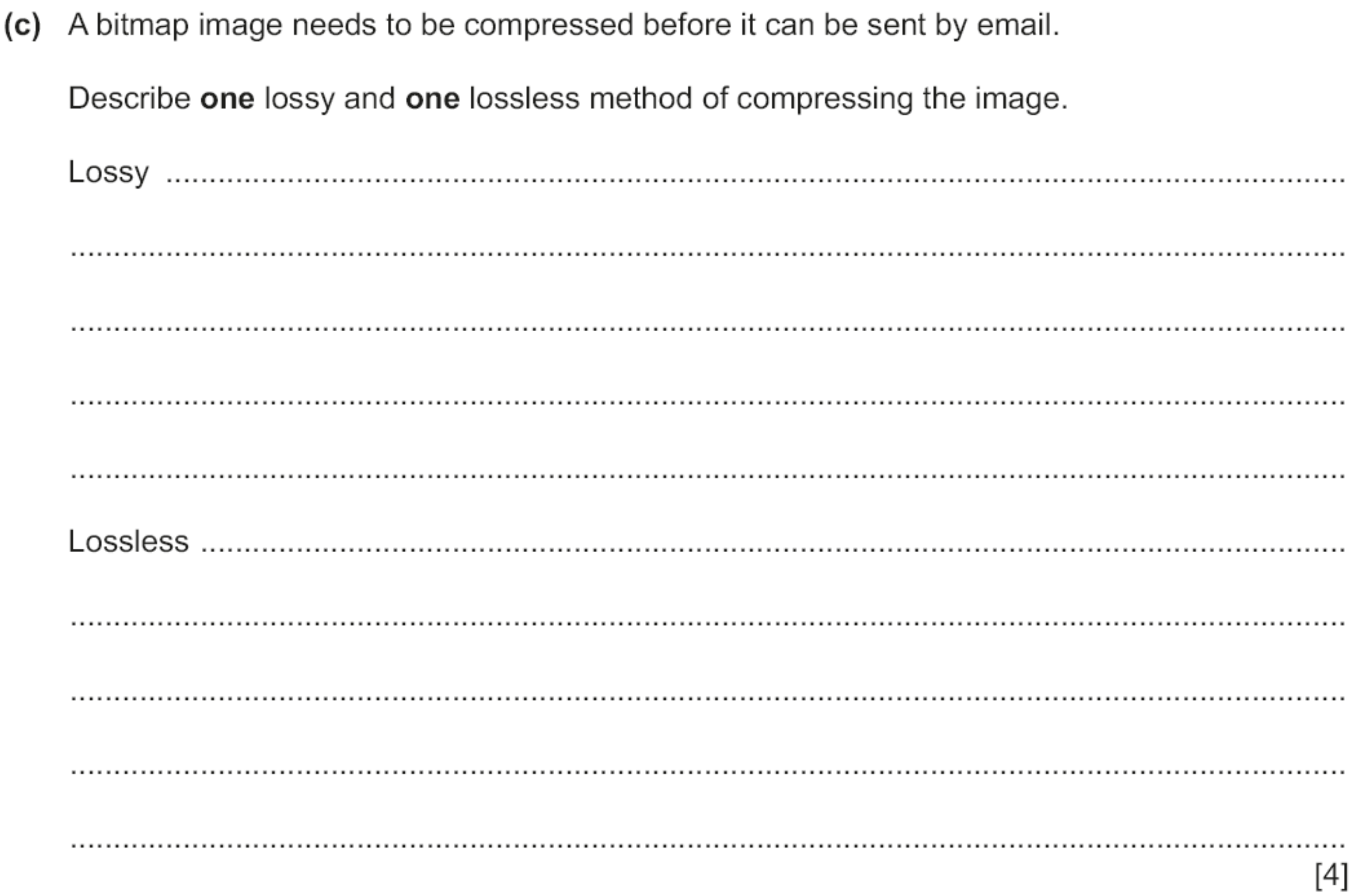
....

.... [2]

Answer:

5(e)(i)	**1 mark** per bullet to **max 2** Lossless: □ All the data is needed // the original file is fully restored □ If any data is lost, the program will not run □ Probably does not require significant reduction in file size // a program written in a high-level language is just text, so does not need much reduction in size	**2**
5(e)(ii)	**1 mark** per bullet to **max 2** Lossy: □ All the data is not required □ The number of colours / resolution can be reduced without the user noticing □ Email requires a significantly smaller file size // takes less time to transmit Lossless: □ A high quality image may be needed □ All of the data is needed // cannot afford to lose any data // the original file is fully restored	**2**
5(e)(iii)	**1 mark** per bullet to **max 2** Lossy: □ Some loss of quality will not be noticed // high quality video not needed on the website □ A more significant reduction may be needed □ Takes a shorter time to upload / download // requires less bandwidth Lossless: □ A high quality video may be needed □ Might only be a short video clip □ All of the data is needed // cannot afford to lose any data // the original file is fully restored	**2**

Question 13:



(c) A bitmap image needs to be compressed before it can be sent by email.

Describe **one** lossy and **one** lossless method of compressing the image.

Lossy ....

....

....

....

....

Lossless ....

....

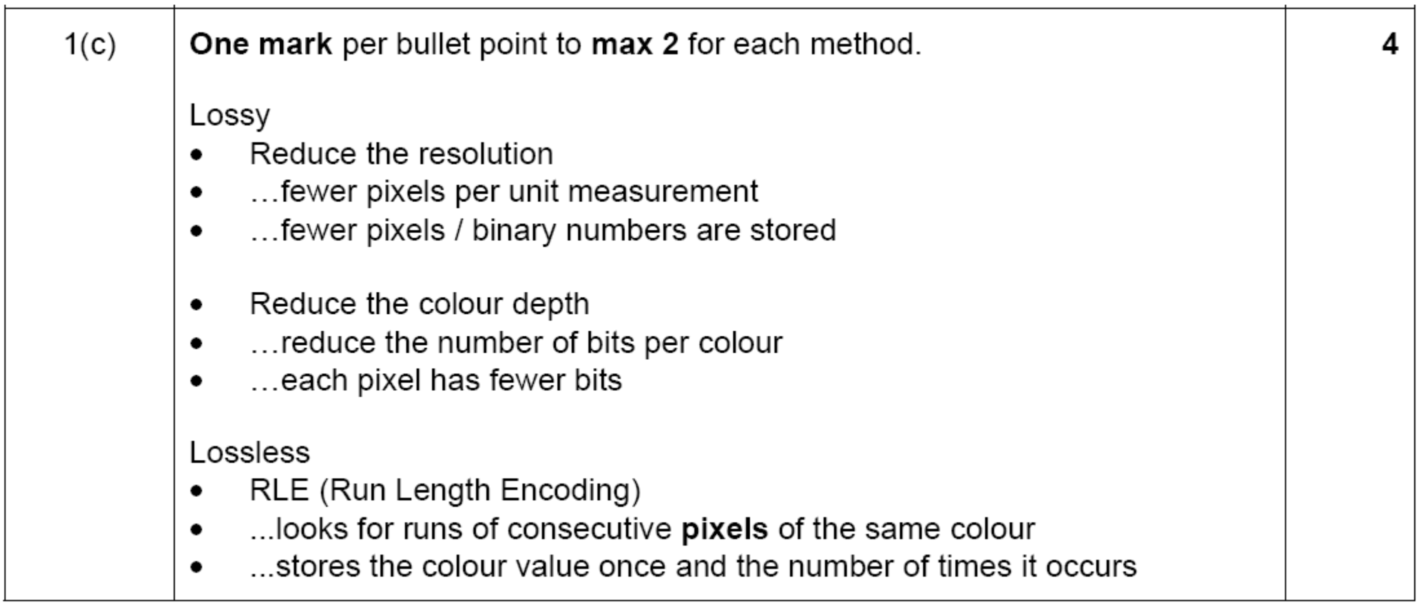
....

....

....

[4]

Answer:



1(c)	**One mark** per bullet point to **max 2** for each method. Lossy • Reduce the resolution • …fewer pixels per unit measurement • …fewer pixels / binary numbers are stored • Reduce the colour depth • …reduce the number of bits per colour • …each pixel has fewer bits Lossless • RLE (Run Length Encoding) • ...looks for runs of consecutive **pixels** of the same colour • ...stores the colour value once and the number of times it occurs	**4**

Question 14:

(iii) Describe how lossless compression can be used to reduce the file size of a text file.

....

....

....

....

....

.... [2]

(iv) Explain why lossy compression should **not** be used on a text file.

....

....

....

....

....

.... [2]

Answer:

1(a)(iii)	**1 mark** per point to **max 2** • Can use run-length encoding • Identifies groups of **repeated** characters ... • ... replaces them with a one copy of the character and the number of times it occurs	**2**
1(a)(iv)	**1 mark** per point to **max 2** • None of the original data can be lost / deleted • The (text) file would be corrupted // the (text) file cannot be opened	**2**