IGCSE Computer Science CIE

YOUR NOTES
↓

1. Data Representation

CONTENTS

YOUR NOTES
⬇

1.2 Text, Sound and Images

Character Sets

Character Sets

- Text is a collection of **characters** that can be represented in binary, which is the language that computers use to process information
- To represent text in binary, a computer uses a **character set**, which is a **collection of characters and the corresponding binary codes that represent them**
- One of the most commonly used character sets is the American Standard Code for Information Interchange (**ASCII**), which assigns a unique **7-bit binary code to each character**, including uppercase and lowercase letters, digits, punctuation marks, and control characters
- E.g. The ASCII code for the uppercase letter **'A' is 01000001**, while the code for the character **'?' is 00111111**
- ASCII has limitations in terms of the number of characters it can represent, and **it does not support characters from languages other than English**
- To address these limitations, **Unicode** was developed as a character encoding standard that allows for a **greater range of characters and symbols** than ASCII, including **different languages and emojis**
- Unicode uses a **variable-length encoding scheme** that assigns a unique code to each character, which can be represented in binary form using multiple bytes
- E.g. The Unicode code for the **heart symbol is U+2665**, which can be represented in binary form as **11100110 10011000 10100101**
- As Unicode requires **more bits per character than ASCII**, it can result in **larger file sizes** and slower processing times when working with text-based data

1.1 Number Systems

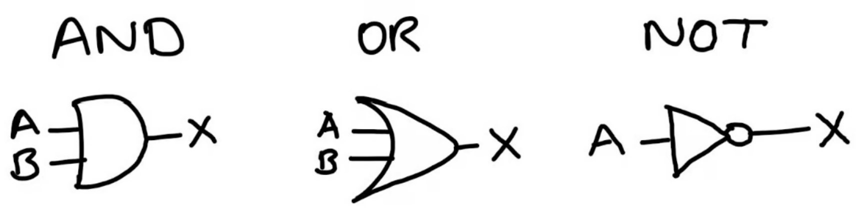
Computers & Binary

Why Computers Use Binary

- All data needs to be converted to binary to be processed by a computer
- Binary is a numbering system that uses only 0s (**off**) and 1s (**on**)
- A computer uses binary data for all of its operations
- The 0's and 1's are called **bits**. "Bits" is short for binary digits
- A bit is the smallest unit of data a computer can use (a single 0 or 1)
- Bits can be grouped to form larger units of data, such as bytes, kilobytes, etc.
- By using binary, computers can process and store data using electronic **switches** that can be either on or off
- Millions or billions of switches fit onto a microchip
- Any form of **data**, including text, images, and sound, needs to be **converted** to **binary** before it can be processed by a computer
- This conversion process involves assigning a binary code to each character, pixel, or sample in the data
- The resulting binary code can then be processed using logic gates and stored in registers

Logic Gates

- **Logic gates** are electronic devices that perform logical operations on binary data
- The most common types of logic gates are **AND, OR**, and **NOT** gates, which can be combined to perform more complex operations



- AND and OR gates both have 2 inputs (A and B) whereas a NOT gate only has 1 input (A)
- X is the output from the logic gate
- Logic gates are used to process **binary** data by applying **Boolean logic** to the input values and producing a binary output
- **Registers** are temporary storage areas in a computer's **CPU** (central processing unit) that hold binary data during processing
- Registers are used to **store data** that needs to be accessed quickly, such as variables in a program or data being manipulated by logic gates
- The size of a register determines the maximum amount of binary data that can be stored in it at one time

1.3 Data Storage and Compression

Data Storage

YOUR NOTES
⬇

Data Storage

- **Data storage** is measured in a variety of units, each representing a different size of storage capacity. The smallest unit of measurement is the **bit**, which represents a single binary digit **(either 0 or 1)**
- A **nibble** is a group of **4 bits**, while a **byte** is a group of **8 bits**
- **Kibibyte** (KiB), **mebibyte** (MiB), **gibibyte** (GiB), **tebibyte** (TiB), **pebibyte** (PiB), and **exbibyte** (EiB) are all larger units of measurement
- Specifically, 1 KiB is equal to 2^10 bytes, 1 MiB is equal to 2^20 bytes, 1 GiB is equal to 2^30 bytes, 1 TiB is equal to 2^40 bytes, 1 PiB is equal to 2^50 bytes, and 1 EiB is equal to 2^60 bytes

To calculate the file size of an image file:

- Determine the resolution of the image in pixels (width x height)
- Determine the colour depth in bits (e.g. 8 bits for 256 colours)
- Multiply the number of pixels by the colour depth to get the total number of bits
- Divide the total number of bits by 8 to get the file size in bytes
- If necessary, convert to larger units like kibibytes, mebibytes, etc

Worked Example

Calculating image file size walkthrough:

An image measures 100 by 80 pixels and has 128 colours (so this must use 7 bits)

100 x 80 x 7 = 56000 bits ÷ 8 = 7000 bytes ÷ 1024 = 6.84 kibibytes

To calculate the file size of a sound file:

- Determine the sample rate in Hz (e.g. 44,100 Hz)
- Determine the sample resolution in bits (e.g. 16 bits)
- Determine the length of the track in seconds
- Multiply the sample rate by the sample resolution to get the number of bits per second
- Multiply the number of bits per second by the length of the track to get the total number of bits
- Divide the total number of bits by 8 to get the file size in bytes
- If necessary, convert to larger units like kibibytes, mebibytes, etc

YOUR NOTES

Worked Example

Calculating sound file size walkthrough:

A sound clip uses 48KHz sample rate, 24 bit resolution and is 30 seconds long.

48000 x 24 = 1152000 bits per second x 30 = 34560000 bits for the whole clip

34560000 ÷ 8 = 4320000 bytes ÷ 1024 = 4218.75 kibibytes ÷ 1024 = 4.12 mebibytes

Exam Tip

Remember to always use the units specified in the question when giving the final answer.

YOUR NOTES
↓

Compression

- Lossless Compression:
  - A compression **algorithm** is used to **reduces the file size** without **permanently** removing any data
  - **Repeated patterns** in the file are **identified** and **indexed**
  - Techniques like **run-length encoding (RLE)** and Huffman encoding are used
  - RLE **replaces sequences of repeated characters with a code** that represents **the character and the number of times it is repeated**
  - Huffman encoding replaces frequently used characters with shorter codes and less frequently used characters with longer codes

- Lossy Compression:
  - Lossy compression reduces the file size by **permanently removing some data** from the file
  - This method is often used for **images** and **audio** files where minor details or data can be removed **without significantly impacting the quality**
  - Techniques like **downsampling**, **reducing resolution or colour depth**, and **reducing the sample rate or resolution** are used for lossy compression
  - The amount of data removed depends on the level of compression selected and can impact the quality of the final file

- Overall:
  - Compression is necessary to **reduce the size of large files** for **storage**, **transmission**, and faster processing
  - The choice between lossy and lossless compression methods depends on the **type of file and its intended use**
  - **Lossy** compression is generally used for **media files** where minor data loss is acceptable while **lossless** compression is used for **text, code, and archival purposes**

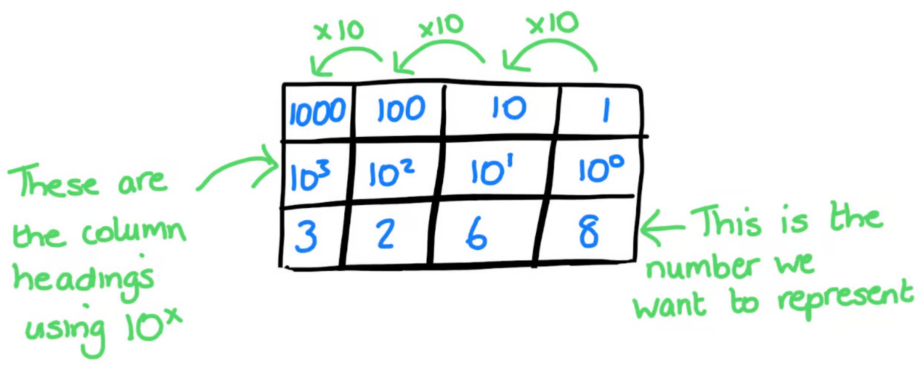
YOUR NOTES
⬇

1.1 Number Systems

Number Systems

The Denary, Binary & Hexadecimal Number Systems

- In Computer Science there are 3 **numbering systems** used to **represent data**:
  - Denary
  - Binary
  - Hexadecimal
- The **denary** number system, also known as the decimal system, is a **base-10** numbering system that uses 10 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9
- Denary numbers can be used to represent any quantity, including whole numbers, fractions, and decimals
- Each digit in a denary number represents a power of 10, with the rightmost digit representing 100, the next representing 101, and so on
- It is necessary to convert data from one number system to another eg. denary to binary or denary to hexadecimal
- It is much easier to do conversions with a table:
  - The number **3268** (three thousand two hundred and sixty-eight) can be represented in the following table:

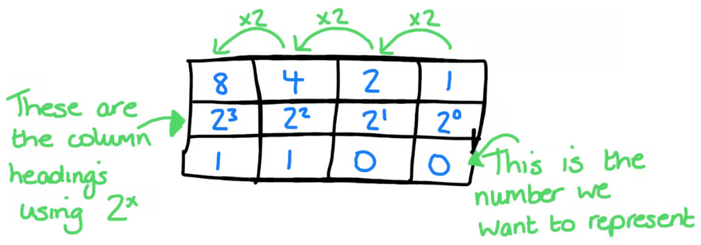

x10
x10
x10
1000
100
10
1
10^3
10^2
10^1
10^0
3
2
6
8
These are the column headings using 10^x
This is the number we want to represent

- (3 x 1000) + (2 x 100) + (6 x 10) + (8 x 1) = 3268

The Binary Number System

- The **binary** number system is a **base-2** numbering system that uses only two digits: 0 and 1
- Each digit in a binary number represents a power of 2, with the rightmost digit representing 20, the next representing 21, and so on.
- Eg: the number **12** represented in binary is **1100**:

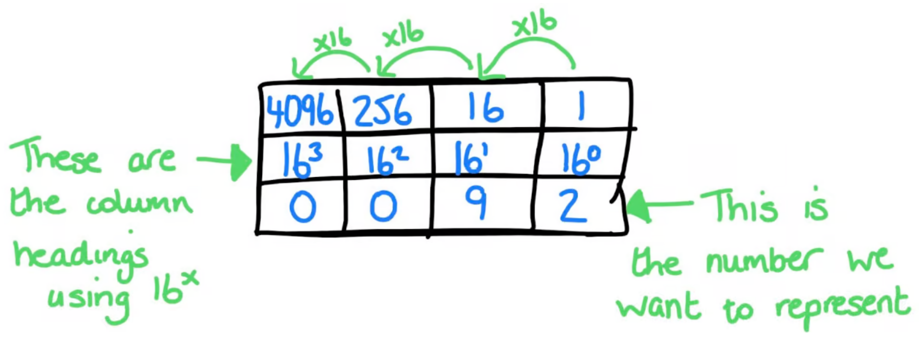
YOUR NOTES
⬇


x2
x2
x2
8
4
2
1
2³
2²
2¹
2⁰
1
1
0
0
These are the column headings using 2ˣ
This is the number we want to represent

- We know this as (1 x 8) + (1 x 4) + (0 x 2) + 0 x 1) =12

The Hexadecimal Number System:

- The **hexadecimal** number system is a **base-16** numbering system that uses 16 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F where 10 is represented by A, 11 by B and so on up to 15 represented by F
- Each digit in a hexadecimal number represents a power of 16, with the rightmost digit representing 160, the next representing 161, and so on
- E.g. the number **146** represented in hexadecimal is **92**


x16
x16
x16
4096
256
16
1
16³
16²
16¹
16⁰
0
0
9
2
These are the column headings using 16ˣ
This is the number we want to represent

- (0 x 4096) + (0 x 256) + (9 x 16) + (2 x 1) = 146

Exam Tip

You need to be able to make conversions in both directions, e.g. denary to binary or binary to denary. Make sure you write the table the correct way round (with 1 in the right hand column) & use the correct number of bits in your answer (the question should tell you but use multiples of 4 bits if you're not sure)

1.2 Text, Sound and Images

Representing Sound

YOUR NOTES
⬇

Representing Sound

- **Sound** is a type of **analog signal** that is captured and **converted into digital form** to be processed by a computer.
- To convert sound into digital form, a process called **sampling** is used. This involves **taking measurements of the sound wave** at **regular intervals** and **converting these measurements into binary data**
- The **quality** of the digital sound **depends on the sample rate**, which is the **number of samples taken per second**. A **higher sample rate** results in a **more accurate representation of the original sound wave**, but also **increases the file size** of the digital sound
- E.g. A typical **CD-quality** digital sound has a sample rate of **44.1 kHz**, which means that **44,100 samples are taken per second**
- The **sample resolution** is another factor that affects the quality of the digital sound. This refers to the **number of bits per sample**, which determines the **level of detail and accuracy of each sample**
- A **higher sample resolution** results in a **more accurate representation of the sound wave**, but also **increases the file size** of the digital sound
- E.g. A **CD-quality** digital sound typically has a **sample resolution of 16 bits**, which means that each sample is represented by a 16-bit binary number
- It's important to choose the appropriate sample rate and resolution based on the specific requirements of the digital sound application. E.g. A high-quality music recording may require a higher sample rate and resolution than a voice recording for a podcast
- MIDI
    - Musical Instrument Digital Interface (file)
    - Stores a set of instructions (for how the sound should be played)
    - It does not store the actual sounds
    - Data in the file has been recorded using digital instruments
    - Specifies the note to be played
    - Specifies when each note plays and stops playing
    - Specifies the duration of the note
    - Specifies the volume of the note
    - Specifies the tempo
    - Specifies the type of instrument
    - Individual notes can be edited
- MP3
    - MP3 is a format for digital audio
    - MP3 is an actual recording of the sound
    - MP3 is a (lossy) compression format
    - It is recorded using a microphone

YOUR NOTES
↓

Converting Between Binary & Denary

Converting Denary to Binary Walkthrough:

- Write down the powers of 2 in binary from **right to left**, starting with 2^0 (1), 2^1 (2), 2^2 (4), 2^3 (8), and so on, until you reach 128 (as answers must be given in 8 bits)

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

- Starting from the leftmost column, **write a 1** if the corresponding power of 2 is less than or equal to the number you're converting, 171 in this example, otherwise **write a 0**.

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

- **Check your working** by adding together all column headings with a 1 underneath (128+32+8+2+1=171)
- Read the binary digits from left to right to get the binary equivalent of 171.
  10101011

Converting Binary to Denary Walkthrough:

- Write down the powers of 2 in decimal from **right to left**, starting with 2^0 (1), 2^1 (2), 2^2 (4), 2^3 (8), and so on, until you reach 128 (as answers must be given in 8 bits)

128	64	32	16	8	4	2	1

- Starting from the leftmost column, write the binary digit in the column if it is a 1 write 1, and write 0 if it is a 0.

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

- **Add up the values** in each column **where the binary digit is 1**.
  64 + 32 + 4 + 2 = 102

1.2 Text, Sound and Images

Representing Images

Representing Images

- A **bitmap image** is made up of a series of **pixels**, which are small dots of colour that are arranged in a grid. Each pixel can be represented by a **binary code**, which is processed by a computer
- The **resolution** of an image refers to the **number of pixels in the image**. A **higher resolution image has more pixels** and is, therefore, **sharper** and **more detailed** but also **requires more storage space**
- The **colour depth** of an image refers to the **number of bits used to represent each colour**. A **higher colour depth** means that **more colours can be represented**, resulting in a **more realistic image** but also **requires more storage space**
- E.g. an **8-bit colour depth** allows for **256 different colours** to be represented (28=256), while a **24-bit colour depth** allows for **over 16 million different colours** to be represented (224=16,777,216)
- The **file size of an image increases** as the **resolution** and **colour depth increase**. This is because **more pixels** and **colours** require **more binary data** to represent them
- The **quality** of an image also **increases** as the **resolution** and **colour depth increase**. However, it's important to balance the desired quality with the practical limitations of storage space

Converting Between Hexadecimal & Binary

Converting Binary to Hexadecimal Walkthrough:

- Group the binary digits into **groups of 4**, starting from the **rightmost** digit. If there are not enough digits to make a group of 4, **add leading zeros** as needed.
  0110 1110
- Add **column headings** to work out the value of each nibble

8	4	2	1		8	4	2	1
0	1	1	0		1	1	1	0

4+2 = 6 8+4+2=14 = E in hex

- Replace each group of 4 bits with its corresponding **hexadecimal value**:
  0110 1110
  6 E
- Write down the resulting hexadecimal values to get the final answer:
  6E

Converting Hexadecimal to Binary Walkthrough:

- To convert a Hex number like A2 into binary write each hex digit in **4-bit binary**
- A=10

8	4	2	1		8	4	2	1
1	0	1	0		0	0	1	0

- The binary representation is the numbers altogether: 10100010

YOUR NOTES
⬇

Converting Between Hexadecimal & Denary

Converting Denary to Hexadecimal Walkthrough:

- **Divide** the decimal number (in this example 57) **by 16** and write down the answer including the remainder:
  57 ÷ 16 = 3 **remainder** 9
- If the remainder is above 9, replace this with the **corresponding letter**
- **Repeat** steps 1 and 2 until the number you're dividing is **zero**:
  3 ÷ 16 = 0 remainder 3
- Write the hexadecimal values from step 3 to step 1 in **reverse** order:
  39

Alternatively, you can turn your **denary** number into **binary**, and then turn the **binary** number into **hexadecimal**

Converting Hexadecimal to Denary Walkthrough:

- Write down the place value of each digit in the number, starting from the **right** and increasing by a **power of 16**:

16^1	16^0
16	1

- If the hex digit is a **letter**, convert it to its denary equivalent (Using the following table to help you):

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

- The hexadecimal value of the leftmost digit is **E**, which has a decimal value of **14**. The hexadecimal value of the rightmost digit is 5, which has a decimal value of 5.
- **Multiply** each decimal value by its corresponding place value, and **sum** the products:

16a	1

YOUR NOTES
⬇

E	5

(14 x 16) + (5 x 1) = 224 + 5 = 229

Therefore, the denary equivalent of E5 is 229.

Alternatively, you can turn your **hexadecimal** number into **binary**, and then turn the **binary** number into **denary**

Exam Tip

When doing conversions don't remove any 0s on the right hand side of your answer as this will cost you marks. E.g. B0 isn't the same as B, just like 30 isn't the same as 3.

YOUR NOTES
⬇

Hexadecimal

- **Hexadecimal** numbers are often used as a beneficial method of **data representation** in computer science
- It takes **fewer digits** to represent a given **value** in hexadecimal **than in binary**
- **One hexadecimal digit** can represent **four bits** of binary data
- It is beneficial to use hexadecimal over binary because:
    - The more bits there are in a binary number, the harder it is to read
    - Numbers with more bits are more **prone to errors** when being copied

Worked Example

- The following binary number has 24 bits: 101110110011101011101010. Write the number in hexadecimal and explain why it is better to write the number in hexadecimal rather than in binary.
- This number can be represented with only 6 digits in hexadecimal: B [1 mark] B [1 mark] 3 [1 mark] A [1 mark] D [1 mark] 2 [1 mark] .
- This number is:
    - Much shorter and a more efficient way of representing numbers [1 mark]
    - Easier to spot any errors in [1 mark]

Uses of Hexadecimal in Computers:

- Using hexadecimal allows computer scientists to represent **large amounts of binary** data in a more manageable and readable format
- There are many uses of hexadecimal in computers:
    - **Memory addresses** and other hardware related values
    - Representation of **colour** values
    - **Error messages**
    - **Memory dump**
    - **Debugging**
    - **IP addresses**
    - **ASCII / Unicode**
    - **Assembly language**
    - **URLs**
- Representation of **colour** values in **HTML/CSS** and in computer graphics: eg. RGB (red-green-blue) colour values can be represented in a compact and readable format. E.g. **#FF0000**
- **Error messages** & **debugging**: eg. 0x80070643 to give technicians/developers information about **what the error is in relation to**
- **Memory dump**: this is a record of what happened in the computer's memory at the time of an error. These are often difficult to read because they contain a lot of technical information, but computers can use hexadecimal values in memory dumps to identify **specific memory locations and values.**
    - Eg. A video game on a computer crashes and a memory dump is created. A technician analyses the memory dump and looks for hexadecimal values to identify the cause of

the crash. They might look for specific memory addresses and values that relate to the game being played or the graphics card in the computer.

- **IP addresses**:
    - Hexadecimal values can be used to represent each set of numbers in an IP address
    - Each set of numbers can be converted into a **two-digit hexadecimal value**, giving a total of eight digits in the IP address
    - Eg. the IP address "**192.168.0.1**" could be represented in hexadecimal as "**C0A80001**"
    - The hexadecimal values can make IP addresses easier to work with in some cases, such as when configuring network devices or writing scripts that interact with IP addresses, however, they are **not commonly used in everyday applications** or browsing the internet
- **ASCII / Unicode**: eg. The letter "**A**" is represented in **ASCII** as the decimal value **65**, which is equivalent to the **binary value 01000001**. This binary value can also be represented in hexadecimal as the value **41**
- **Assembly language**:
    - Hexadecimal values are used in assembly language to represent the binary code for the **instructions and data**
    - Each instruction or data value is represented by a specific sequence of hexadecimal digits, which can make it easier for programmers to read and understand the code
    - Eg. "**MOV AX, 5**" instruction is represented by the hexadecimal value "**B805**", which tells the computer to **move the value 5 into register AX**
- **URLs**:
    - To encode a **space** in a URL using hexadecimal, the space character is replaced by a percent sign ("**%**") followed by the **hexadecimal value** of the **space character**. In ASCII, the space character has a decimal value of **32**, which can be represented in hexadecimal as "**20**".
    - So, to encode the URL "**my website.com/page one**" using hexadecimal, it would look like this: **my%20website.com/page%20one**

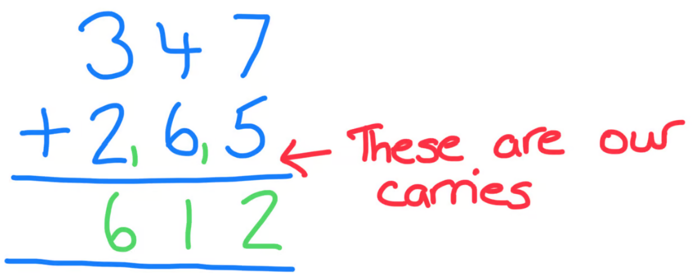
Exam Tip

When a question is asked which asks you to name a certain number of uses of hexadecimal, ensure you write the number asked for and no more. E.g. name 3 uses of hexadecimal - if you write more than 3, the last ones will be ignored by the examiner, even if they're correct

YOUR NOTES
⬇

Binary Addition

- Adding binary numbers follows a similar process to adding denary numbers



- The binary adding rules are:
  - 0+0=0
  - 0+1=1
  - 1+1=10 (The 1 is carried into the next column on the left)
  - 1+1+1=11 (The 1 is carried into the next column on the left)

Adding binary steps:

Step 1:

Start by writing the two binary numbers you want to add underneath each other, with the least significant bit (LSB) on the right.

Step 2:

Begin by adding the LSBs together. If the sum is less than or equal to 1, write it down in the sum column. If the sum is 2 or greater, write the remainder of the sum (i.e., the sum minus 2) in the sum column and carry over the quotient (i.e., 1) to the next column

Step 3:

Repeat this process for the next column to the left, adding the two bits and any carryover from the previous column. Again, if the sum is less than or equal to 1, write it down in the sum column; if the sum is 2 or greater, write the remainder of the sum in the sum column and carry over the quotient to the next column.

Step 4:

Continue this process for each subsequent column until you have added all the bits.

Step 5:

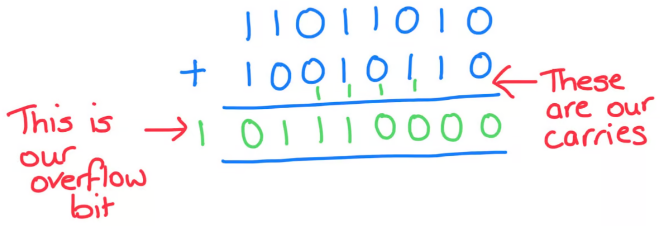
If the sum of the last two bits produces a carryover, add an additional bit to the left of the sum to represent the carryover.

Step 6:

Check the sum to make sure it fits within 8 bits. If it doesn't, you will need to use more bits to represent the sum.

YOUR NOTES
↓

Adding binary walkthrough:



In this example, we start by adding the two LSBs: 0 + 0 = 0, which we write down in the sum column. We then move to the next column to the left and add the two bits and the carryover from the previous column: 1 + 1 + 0 = 10. We write down the remainder of the sum (i.e., 0) in the sum column and carry over the quotient (i.e., 1) to the next column. We repeat this process for the next two columns, and end up with the sum 101110000.

Overflow

- An **overflow** error occurs when the result of a binary addition **exceeds the maximum value** that can be represented. In the case of 8-bits, the maximum value is 255
- Overflow occurs when the addition of two numbers **results in a carry bit that cannot be accommodated**
- To avoid overflow errors, it's important to **check the result** of binary addition to ensure that it doesn't exceed the maximum value that can be represented
- Overflow errors can **also occur** in other operations besides addition, such as **multiplication or division**

Exam Tip

You can convert your binary numbers to denary, then perform the calculation and then convert them back to check you've got the right answer. Label this as checking to make sure that the examiner knows this is a check and not part of your working out

YOUR NOTES
↓

Binary Shifts

- A **binary shift** is the term used for **multiplying** or **dividing** in binary
- A binary shift moves all the bits in a binary number a certain number of positions to the left or right
- When performing a logical binary shift to the left, the **bits shifted from the end of the register are lost**, and zeros are shifted in at the opposite end
- When performing a logical binary shift to the right, the bits shifted from the beginning of the register are lost, and **zeros are shifted in at the opposite end**
- When performing a logical binary shift, the positive binary integer is multiplied or divided according to the shift performed. **A shift to the left** is equivalent to **multiplication by a power of 2**, while a **shift to the right** is equivalent to **division by a power of 2**
- When performing a logical binary shift, the most significant bit(s) or least significant bit(s) are lost, depending on the direction of the shift

Binary shift walkthrough:

- E.g. shifting the binary number 11001100 to the left by two positions gives 00110000. The two bits shifted from the end are lost, and two zeros are shifted in at the opposite end
- E.g. shifting the binary number 11001100 to the right by two positions gives 00110011. The two bits shifted from the beginning are lost, and two zeros are shifted in at the opposite end
- E.g. shifting 11001100 to the left by two positions, is multiplying it by 2^2, or 4. The result is 00110000, which is equal to 4 times the original value
- Shifting the binary number 11001100 to the right by two positions, is dividing it by 2^2, or 4. The result is 00110011, which is equal to the original value divided by 4

Exam Tip

Make sure you've got the same number of bits in your answer as there were in the question. Check your answer by converting the binary number to denary, working out your answer and converting it back again. Make sure to label this checking so the examiner knows it isn't part of your working out

YOUR NOTES
⬇

Two's Complement

Two's Complement

- **Two's complement** is a method of representing **signed integers** in binary form, where the **leftmost bit** represents the **sign (0 for positive and 1 for negative)**
- To convert a positive number to a two's complement 8-bit integer, first represent the number in binary form with leading zeros until it is 8 bits long. **If the number is positive, the leftmost bit should be 0**
- To convert a negative number to a two's complement 8-bit integer, first **invert all the bits in the binary representation of the positive equivalent** of the number (i.e., flip all the 1's to 0's and all the 0's to 1's). Then, add 1 to the result to obtain the two's complement representation

Two's complement walkthrough:

- The binary number 00101011 represents the positive integer 43 in 8-bit binary form
- E.g. to represent the negative integer -43 in two's complement 8-bit form, start by representing the positive equivalent of 43 in binary form: 00101011. Then invert all the bits to get 11010100, and add 1 to get 11010101