OCR A Level Computer Science

Your notes

4.1 Data Types

Contents

Your notes

Primitive Data Types

What are Data Types?

- A data type is a classification of data into groups according to the **kind of data they represent**
- Computers use different data types to represent different types of data in a program
- The basic data types include:
  - **Integer**: used to represent **whole numbers**, either positive or negative
    - Examples: 10, -5, 0
  - **Real**: used to represent **numbers with a fractional part**, either positive or negative
    - Examples: 3.14, -2.5, 0.0
  - **Char**: used to represent a **single character** such as a letter, digit or symbol
    - Examples: 'a', 'B', '5', '$'
  - **String**: used to represent a **sequence of characters**
    - Examples: "Hello World", "1234", "@#$%"
  - **Boolean**: used to represent **true or false** values
    - Examples: True, False
- It is important to choose the correct data type for a given situation to ensure accuracy and efficiency in the program

Casting

- Casting is when you convert one data type to another data type
- When a user enters data into a program, this will more than likely be in a **string format**
- It's essential to convert some of this **string** data to a **numerical format** where possible
- For example, you may want to perform **calculations** on age-related data to determine if someone is eligible to vote
- Some programming languages can't execute numerical comparisons on text data, making this transformation crucial

- For example if you had "12" stored as a string and you wanted to know if this value was below 20
- Therefore, the string value of "12" will need to be cast as an integer to allow the comparison to take place

Your notes

Python example

```
int_value = int("123") // converts the string "123" to 123

float_value = float("3.14") // converts the string "3.14" to 3.14
```

Java example

```
int intValue = Integer.parseInt("123"); // converts the string "123" to 123

double doubleValue = Double.parseDouble("3.14"); // converts the string "3.14" to 3.14
```

Casting between data types

Conversion	Example	Output
From **Integer** to **Real**	`int_value = 5` `real_value = float(int_value)`	5.0
From **Real** to **Integer**	`real_value = 5.7` `int_value = int(real_value)`	5
From **String** to **Integer**	`int_str = "10"` `int_value = int(int_str)`	10
From **Integer** to **String**	`value = 5` `str_value = str(value)`	"5"
From **Boolean** to **String**	`bool_val = True` `str_val = str(bool_val)`	"True"
From **String** to **Boolean**	`bool_str = "True"` `bool_val = bool(bool_str)`	True


Your notes

Worked Example

You are tasked with designing a program for a bookstore.

The program should be able to store each piece of information given in table.

Book Detail	Recommended Data Type
The book's title	
The number of pages the book has	
The price of the book	
Whether the book is currently in stock or not	
The average rating given by customers which ranges from 1 to 5	

Complete the table by identifying a suitable data type for each piece of information in the table.

5 marks

Answer:

Answer that gets full marks

Book Detail	Recommended Data Type
The book's title	String
The number of pages the book has	Integer
The price of the book	Float
Whether the book is currently in stock or not	Boolean
The average rating given by customers ranging from 1 to 5	Float

Positive Binary Numbers

Your notes

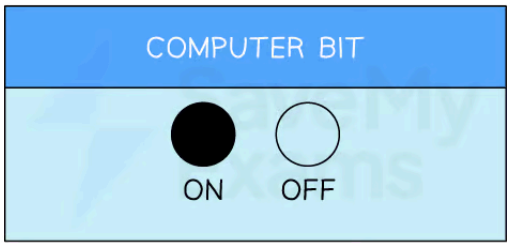
Binary

What is Binary?

- binary refers to a system of representing information using only two digits: **0** and **1**.

Bits

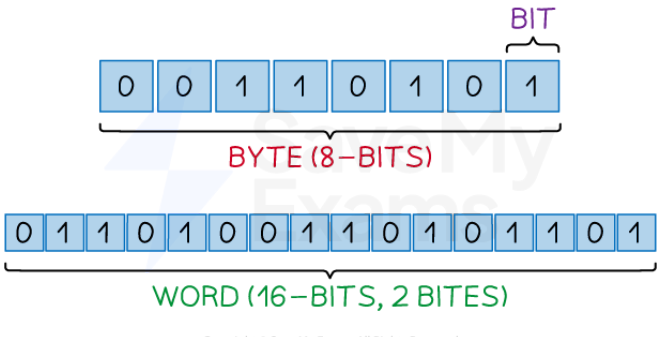
- A **bit** is the smallest unit of **digital information**, representing either an "off" (0) or an "on" (1) state.



The status of a computer bit being on or off.

Bytes

- **Bits** are grouped into larger structures to form **bytes** (8 bits), **words**, and **long words**
- These **groupings** allow us to represent more **complex information**, like numbers, text, and instructions



Groups bits to store more complex information

Your notes

What do the 0s and 1s represent?

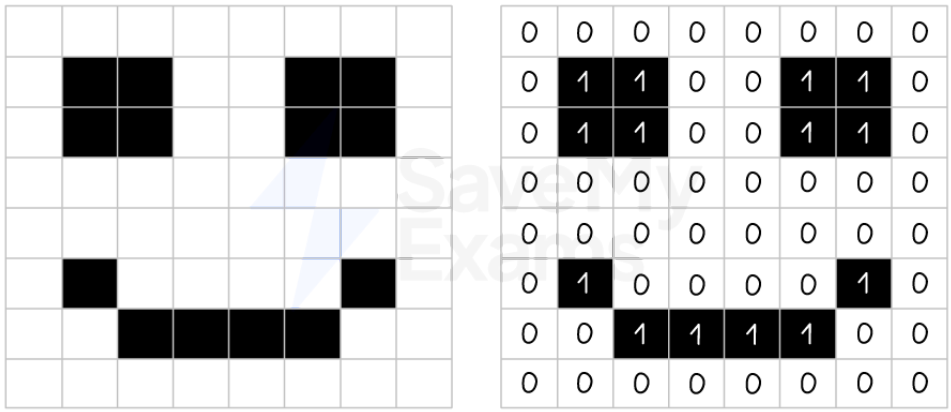
- In binary, each level is based on powers of 2
- In the **8-bit binary number** below, each of the 8 columns represents values of 2^n, e.g.
- $2^0 = 1$
- $2^1 = 2$
- $2^2 = 4$
- $2^3 = 8$
- $2^4 = 16$
- $2^5 = 32$
- $2^6 = 64$
- $2^7 = 128$

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

Binary powers of two

Encoding and representation

- Various encoding schemes, like ASCII for text or JPEG for images, map these binary values to human-readable forms
- For example, the binary value 01001000 represents the letter 'H' in ASCII
- In the example below, an image is shaded black or white depending on the binary value for each pixel
- Each row in the image requires 1 byte of storage



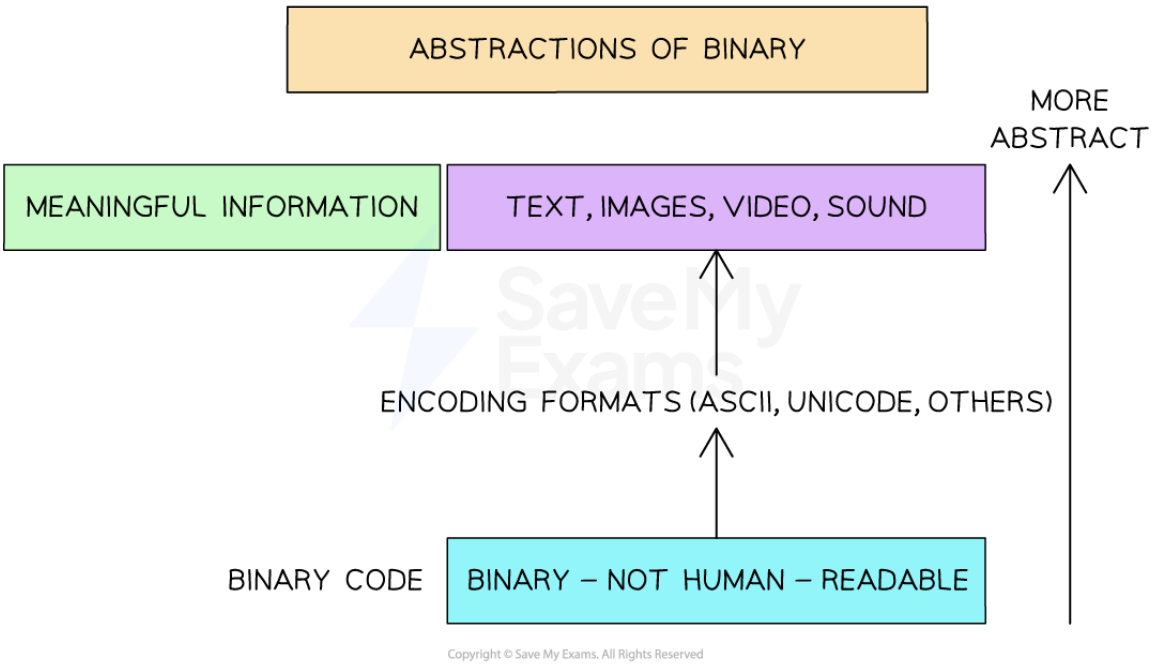
Pixel-shading in a bitmap image

Abstraction layers

- Computers handle large volumes of basic numeric data
- To create meaningful representations of data, layers of abstraction exist so that basic data can be interpreted upwards into other forms, e.g. images, sound, video
- The same principle applies to programming languages that compile down into binary code
- At the bottom, you have binary, and each layer above it allows for more meaningful information to be represented

Your notes

Your notes

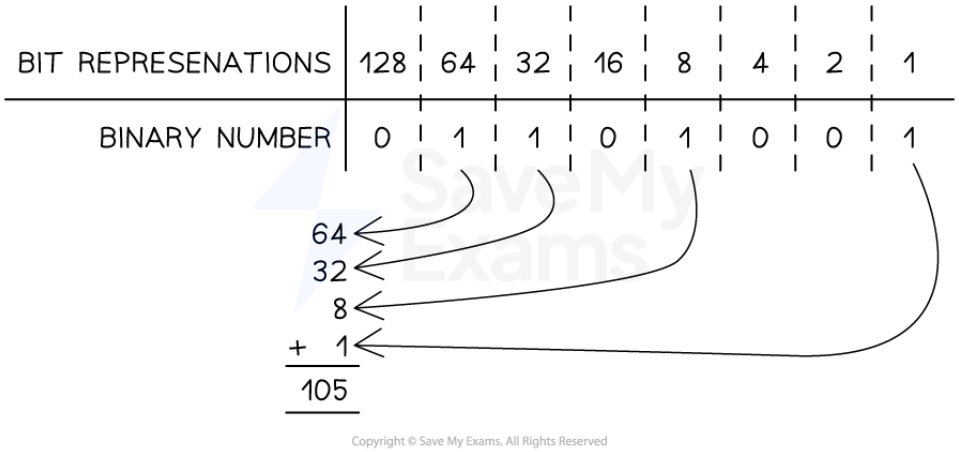


Abstractions of binary

Converting Between Binary & Denary

- Within computer science, two common number systems are:
  - **Denary numbers** – This is also known as base-10. These are used by humans and consist of 10 digits ranging from 0 to 9
  - **Binary numbers** - Computer systems store data using 1s and 0s. This is known as binary or base-2. Computer systems store data in binary format because computers are made up of circuits and switches that are either on (1) or off (0)
- Binary numbers can be converted into denary and vice-versa
- For example the 8-bit binary number **01101001** can be converted into denary using the following method:

Your notes



Binary to decimal conversion

Therefore the 8-bit binary number **01101001** is 105 as a denary value.

Converting Between Denary & Binary

- To convert the denary number **101** to binary, we firstly write out binary number system

128	64	32	16	8	4	2	1

- Then we start at the left and look for the highest number that is less than or equal to **101** and if so, place a 1 in that column. Otherwise, place a 0 in the column
- **128** is bigger than **101** and therefore we place a 0 in that column
- **64** is smaller than **101** so we place a 1 in that column. **101 – 64 = 37**. This now means we have **37** left to find
- **32** is smaller than **37** so we place a 1 in that column. **37 – 32 = 5**. This now means we have **5** left to find
- **16** is bigger than **5** and therefore we place a 0 in that column
- **8** is bigger than **5** and therefore we place a 0 in that column
- **4** is smaller than **5** so we place a **1** in that column. **5 – 1 = 1**. This now means we have **1** left to find
- **2** is bigger than **1** and therefore we place a 0 in that column
- **1** is equal to the number we have left so we place a 1 in that column

- 64 + 32 + 4 + 1 = **101**. Therefore the denary number **101** in binary is **01100101**

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


Your notes

Negative Binary Numbers

Your notes

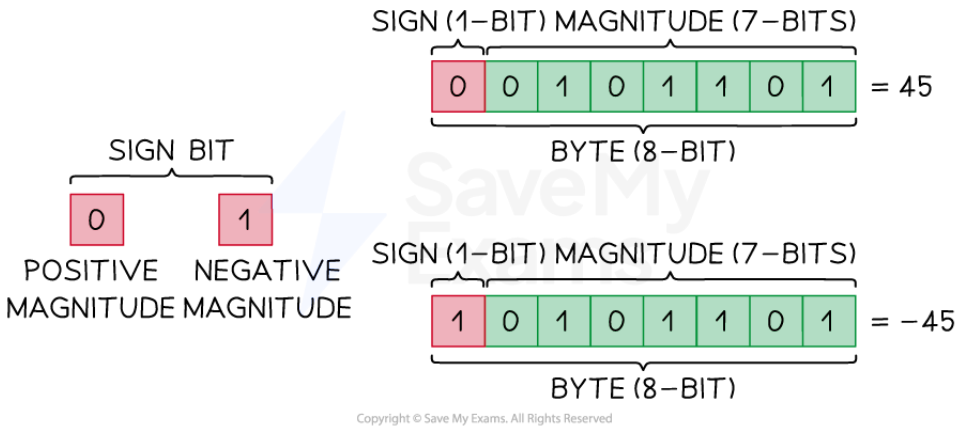
Signed Binary Numbers

What are Signed Binary Numbers?

- A binary number can be signed or unsigned:
  - **Unsigned** - used to represent positive binary numbers
  - **Signed** - used to represent both positive and negative binary numbers
- We can use signed binary numbers to represent negative numbers using methods such as:
  - Sign & magnitude
  - Two's complement
- Both of these methods use the **Most Significant Bit (MSB)** to represent whether the number is negative or positive:
  - If the **MSB is 0**, the number is **positive**
  - If the **MSB is 1**, the number is **negative**

Sign & Magnitude

- Sign & magnitude binary numbers contain a:
  - **Sign** - This is when the MSB is used to represent whether the number is negative (1) or positive (0)
  - **Magnitude** - This is used to describe the rest of the bits after the MSB

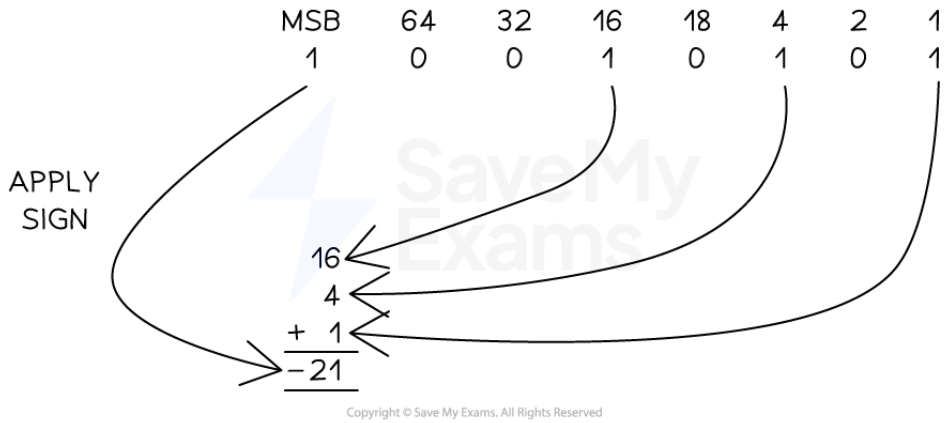


Representing negative binary numbers using sign & magnitude

Your notes

Binary to denary example

- To convert a sign & magnitude binary number to denary, you need to:
  - Convert the number as normal from binary to denary (as described in Positive Binary number: Binary to Denary)
  - Apply the MSB at the end of the calculation
    - If the **MSB** is 1, the number is **negative**
    - If the **MSB** is 0, the number is **positive**



Converting sign & magnitude binary numbers to denary

Denary to binary example

- To convert a denary number to a sign & magnitude binary number, you need to:
  - Identify whether the number is positive or negative
  - Convert the number to binary as normal (as described in Positive Binary Numbers: Denary to Binary)
  - If the number is negative, change the MSB to **1**

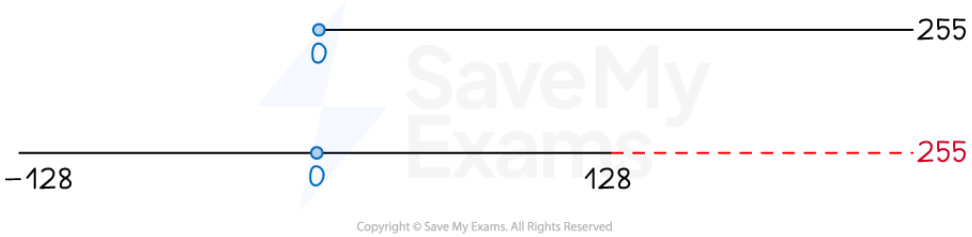
MSB	64	32	16	8	4	2	1
1	1	0	0	0	1	1	1

- 64 + 4 + 2 + 1 = **71**
- Apply a sign of 1 to make **-71**
- Therefore the denary number **-71** in binary is **11000111**

Your notes

A consequence of using a sign bit

- The MSB purpose changes from representing a value to representing positive or negative
- Losing the MSB halves the maximum size of the number that can be stored
- However, as a benefit it makes it possible to represent negative numbers



Sign & magnitude number system

Worked Example

Convert the 8-bit sign and magnitude binary number 10001011 to denary.

How to answer this question:

- **Identify the sign bit**: The MSB is 1, so the number is negative
- **Isolate the magnitude**: We are left with **0001011** by removing the sign bit
- **Convert to denary**: The binary number **0001011** converts to 11 in denary
- **Apply the sign**: The MSB was 1, so the number is **-11** in denary

Answer:

Answer that gets full marks

10001011 converts to **-11** in denary

Two's Complement

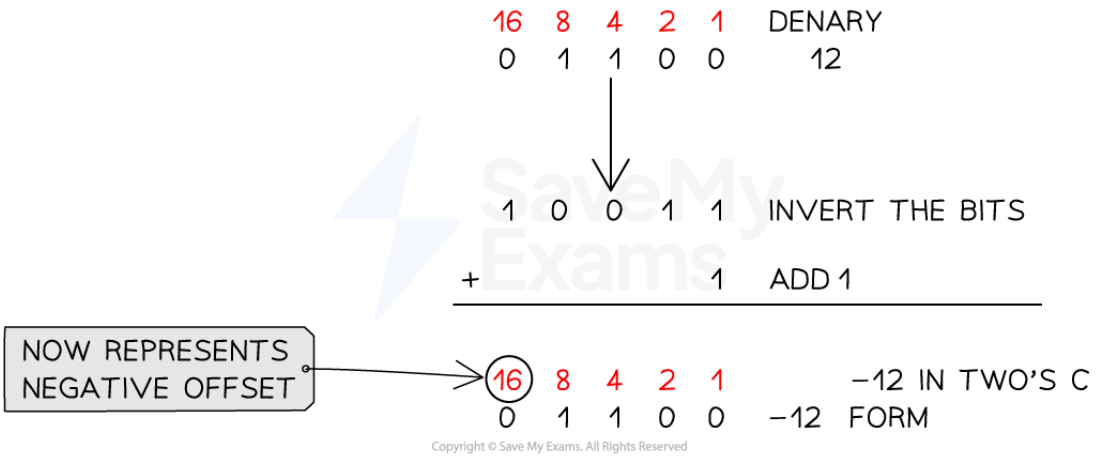
- Two's complement is a different method for representing negative binary numbers

- Calculations on two's complement numbers are less computationally intensive

Your notes

Method

1. Start with the **absolute value** of the number, in this case 12
2. Invert the bits so that all of the 1's become 0's and all of the 0's become 1's
3. Add 1



Representing two's complement binary numbers

- The purpose of the **MSB** has changed; it now represents **the negative starting point** of the number, and the rest of the **bits** are used to count **upwards** from that number. For example using the binary number from the image above:
  - Begin counting at **-16**
  - Add 4 to make **-12**
- Two's complement has a similar consequence to sign and magnitude as the maximum value of an 8-bit value is halved

Denary to binary example

- To convert the denary number -24 to a two's complement binary number, you need to:
  - Convert the number to binary

-128	64	32	16	8	4	2	1

0	0	0	1	1	0	0	0


Your notes

- Invert the bits

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

- Add 1 to the number

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

Binary Addition & Subtraction

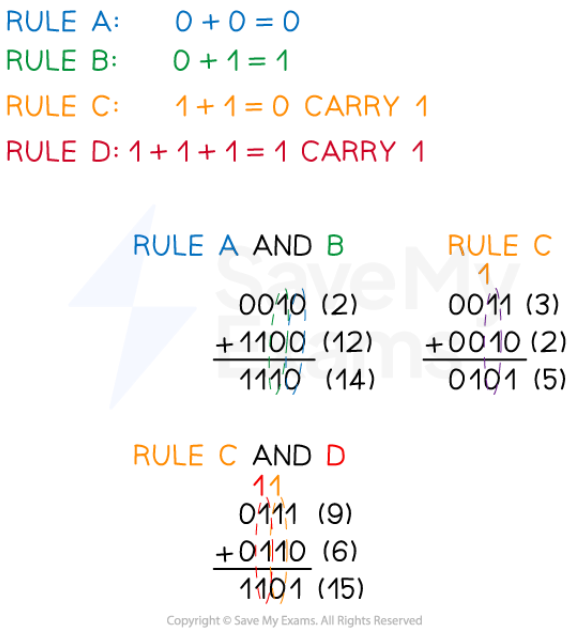
Your notes

Binary Addition

What is Binary Addition?

- Binary addition involves summing numbers in base-2, which uses only the digits 0 and 1
- Like denary addition, start from the rightmost digit and move towards the left
- Carrying over occurs when the sum of a column is greater than 1, passing the excess to the next left column

Example addition

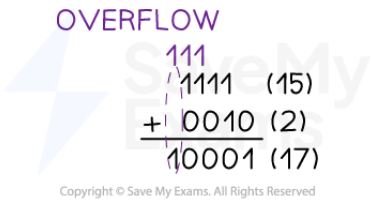


Binary addition example

Overflow Errors

- Overflow occurs when the sum of two binary numbers exceeds the given number of bits
- In signed number representations, the leftmost bit often serves as the sign bit; overflow can flip this, incorrectly changing the sign of the result

- Overflow generally leads to incorrect or unpredictable results as the extra bits are truncated or wrapped around



An overflow occurring after a binary addition

Binary Subtraction

- As well as adding binary numbers, we can also subtract binary numbers
- One method of doing this is to use two's complement

Example 1

Subtract 1001 (9) from 0011 (3)

1. Given numbers

Number 1	1	0	0	1
Number 2	0	0	1	1

2. Two's complement

- Convert the number to subtract (0011) to its **two's complement**
- Invert bits to get **1100**
- Add 1 to get **1101**

Number 1	1	0	0	1
Number 2 (Converted)	1	1	0	1

3. Addition Operation

- Now add **1001** and **1101**

Your notes

- **1001 + 1101 = (1) 0110 – including an overflow**

Carry	1			1	
Number 1		1	0	0	1
Number 2		1	1	0	1
Addition	1	0	1	1	0


Your notes

4. Remove Overflow

- The result is **10110** with overflow
- Remove the **overflow bit** to get **0110**
- In **two's complement** arithmetic, the **overflow bit** does not contribute to the actual value of the operation but is more of a **by-product** of the method.

Hexadecimal Numbers


Your notes

Representing Hexadecimal Numbers

What is Hexadecimal?

- It serves as a more human-friendly representation of binary-coded values
- Another numbering system is **hexadecimal** (base-16)
- It consists of 10 numbers (0–9) and 6 letters (A-F)

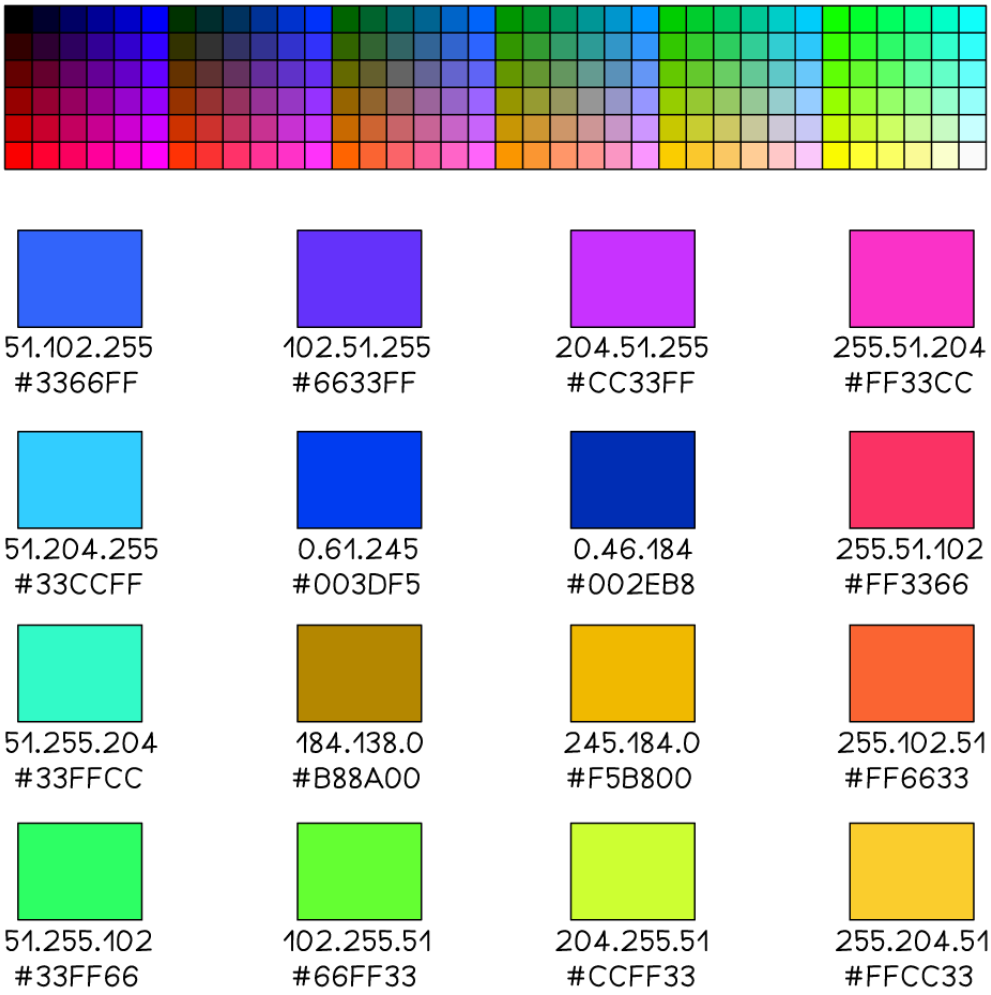
Why use hexadecimal instead of binary?

- It is more concise as four binary bits (e.g. 1010) can be represented with one hexadecimal character (e.g. A)
- It is easier for humans to read and write
- it is less prone to error as it is more likely to be communicated correctly

Potential uses of hexadecimal

- It is commonly used for debugging, configuring hardware devices, and in cryptographic algorithms
- It also also commonly used to define colours. As there are millions of colours in the visual spectrum that require very large binary numbers to represent them, they can be replaced with shorter hexadecimal values

Your notes



Sample hexcolours

Hexadecimal Lookup Table

- The hexadecimal lookup table serves as a quick reference for converting numbers between denary, binary and hexadecimal values
- The hexadecimal scale is identical to the denary scale until the tens column is introduced
- When the denary scale reaches 10, this is when the hexadecimal scale switches to letters, starting with A

Denary	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F


Your notes

Using subscript in number representation

- The subscript in number representation denotes the base of a number

- It helps in differentiating between number systems
- Common uses:
  - 10_2 indicates the number is in binary (base 2), and its value is '10' in binary
  - 10_{10} indicates the number is in denary (base 10), and its value is '10' in denary
  - 10_{16} indicates the number is in hexadecimal (base 16), and its value is '10' in hex, equivalent to '16' in denary
- It provides clarity, especially in contexts where multiple numbering systems are discussed


Your notes

Denary to Hexadecimal

Convert the denary number 241 to hexadecimal.

Step 1: Convert the number to binary

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

Step 2: Split the binary number into nibbles

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

Step 3: Convert each nibble into its hexadecimal value

8	4	2	1		8	4	2	1
1	1	1	1		0	0	0	1
15 (F in hexadecimal)					1 (1 in hexadecimal)			

Step 4: Final result

The denary number **241** is **F1** in hexadecimal.

Hexadecimal to Denary

Convert the hexadecimal value 1A to denary.

Your notes

Step 1: Convert each hexadecimal character into binary

8	4	2	1		8	4	2	1
0	0	0	1		1	0	1	0
1 (1 in hexadecimal)					10 (A in hexadecimal)			

Step 2: Join the binary values

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

Step 3: Final result

16 + 8 + 2 = 26

The hexadecimal value **1A** is **26** in denary.

Binary to Hexadecimal

Convert 1101 0101 to hexadecimal.

Step 1: Take the binary values

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

Step 2: Convert each nibble into its hexadecimal value

8	4	2	1		8	4	2	1
1	1	0	1		0	1	0	1
13 (D in hexadecimal)					5 (5 in hexadecimal)			

Step 3: Final result

The binary number **11010101** is **D5** in hexadecimal.

Hexadecimal to Binary

Convert **B2** from Hexadecimal to Binary.


Your notes

Step 1: Split the Hexadecimal into two separate digits and convert each to a nibble

B = 11					2			
8	4	2	1		8	4	2	1
1	0	1	1		0	0	1	0
11 in 4-bit binary is 1011					2 in 4-bit binary is 0010			

Step 2: Join the two nibbles to make a byte (8 bits)

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

Step 3: Final result

The **hexadecimal** number **B2** in **Binary** is **10110010**

Floating Point Binary Numbers

Your notes

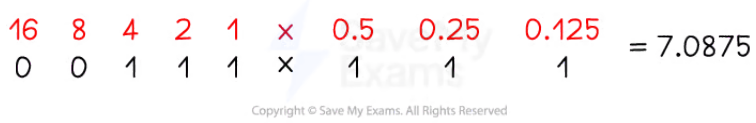
Floating Point Binary

What is Floating Point Binary?

- Floating point binary addresses the limitations of fixed-point binary in representing a wide array of real numbers
- It allows for both fractional and whole-number components
- It accommodates extremely large and small numbers by adjusting the floating point
- It optimises storage and computational resources for most applications

Representation of floating point

The **appearance** of floating-point binary is mostly **the same** except for the presence the decimal **point**



An example positive floating point number

- In the example above, an 8–bit number can represent a whole number and fractional elements
- The point is always placed between the whole and fractional values
- The consequence of floating point binary is a significantly reduced maximum value
- The benefit of floating point binary is increased precision

Representation of negative floating point

- Negative numbers can also be represented in floating point form using two's Complement
- The MSB is used to represent the negative offset of the number, and the bits that follow it are used to count upwards
- The fractional values are then added to the whole number

Your notes

Converting Denary to Floating Point

Denary to floating point binary

Example: Convert 6.75 to floating point binary

Step 1: Represent the number in fixed point binary.

-8	4	2	1	.	0.5	0.25
0	1	1	0	.	1	1

Step 2: Move the decimal point.

0	.	1	1	0	1	1

Step 3: Calculate the exponent

The decimal point has moved **three** places to the left and therefore has an exponent value of **three**.

-4	2	1
0	1	1

Step 4: Calculate the final answer:

Mantissa: 011011

Exponent: 011

Converting Floating Point to Denary

Binary floating point to denary

Example: Convert this floating point number to denary:

- **Mantissa –** 01100
- **Exponent –** 011


Your notes

Step 1: Write out the binary number.

0	.	1	1	0	0

Step 2: Work out the exponent value.

The exponent value is 3.

-4	**2**	**1**
0	1	1

Step 3: Move the decimal point three places to the right.

-8	**4**	**2**	**1**	**.**	**0.5**
0	1	1	0	.	0

Step 4: Calculate the final answer: 6

Normalising Floating Point Binary

A floating point number is said to be normalised when it starts with **01** or **10**.

Why normalise?

- Ensures a consistent format for floating point representation
- Makes arithmetic and comparisons more straightforward

Steps to Normalise a Floating Point Number

1. Shift the **decimal point left or right** until it starts with a 01 or 10
2. Adjust the **exponent value** accordingly as you move the decimal point
    - Moving the point to the left increases the exponent and vice versa.

Example

Before normalisation:

- **Mantissa** = 0.0011
- **Exponent** = 0010 (2)

After normalisation:

- **Mantissa** = 0.1100 – Decimal point has moved 2 places to right so it starts with 01
- **Exponent** = 0000 (0) - Exponent has been reduced by 2


Your notes

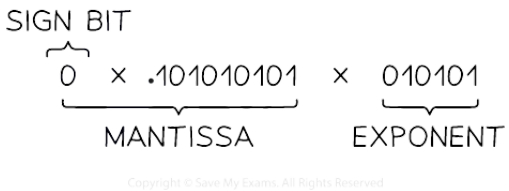
Floating Point Addition & Subtraction

Your notes

Floating Point Arithmetic

How do you represent Floating Point Numbers?

- Floating point numbers are represented with a **sign**, **mantissa**, and **exponent**
- Arithmetic operations must take into account these three components



Sections of a floating point number

Steps for adding or subtracting floating point numbers

1. **Ensure the numbers have the same exponent before performing arithmetic**
   - This might involve shifting the decimal point of one number and adjusting its exponent until both numbers have matching exponents.
   - Example:
     - Number A: 1.101×2^3
     - Number B: 1.010×2^2
     - Number A has an exponent of 2^3 and B has an exponent of 2^2, we need to adjust B to have the same exponent as A
     - This is achieved by moving the point one space to the left in Number B and increasing the exponent by 1
     - Resulting in: 0.101×2^3
2. **Perform the binary addition or subtraction on the mantissa**
   - $1.101 + 0.101 = 10.010$

3. **Ensure the result is in a normalised form**
   - The sum 10.010 exceeds the normal range for mantissa (1.0 to 1.111... in binary)
   - To normalise it, we shift the mantissa one position to the right and increment the exponent by 1
   - New Mantissa: 1.0010
   - New Exponent: Increment the exponent from 2^3 to 2^4
   - The final result would be 1.0010×2^4.
4. **Determine Sign**
   - For **addition**: If both numbers are positive or negative, the result takes the common sign
   - If they have different signs, the result's sign depends on the larger absolute value
   - For **subtraction**: The sign is determined by the sign of the number you're subtracting from and the result of the subtraction


Your notes

Example addition

1. $1.1001_2 \times 2^3 + 1.0110_2 \times 2^2$
2. **Align exponents:** $1.1001_2 \times 2^3 + 0.1011_2 \times 2^3$
3. **Add mantissa:** 10.0100_2
4. Normalise (if required) and determine the sign.

Example subtraction

1. $1.1001_2 \times 2^3 - 1.0110_2 \times 2^2$
2. **Align exponents:** $1.1001_2 \times 2^3 - 0.1011_2 \times 2^3$
3. **Subtract mantissas:** 0.1110_2
4. Normalise (if required) and determine the sign.

Bitwise Manipulation & Masks


Your notes

What are Logical Shifts?

- Logical shifts are the process of moving the bits in a binary number to the left or right by a specified number of places
- Bitwise manipulation uses logical operators like AND, OR, XOR, and NOT to manipulate binary numbers

Logical Shifts

- Logical binary shifts are operations performed on binary numbers where all the bits in the number are moved left or right by a specified number of positions
- These shifts are commonly used in computer programming and digital systems
- There are two types of logical binary shifts: Left and Right

Example left shift

The following number is shifted by **two** places to the left.

Step	128	64	32	16	8	4	2	1
Original number in binary	0	0	0	0	1	1	1	0
Left shift by 2 places	0	0	1	1	1	0	0	0

Original number: 00001110 = 14

Left shift (2) result: 00111000 = 56

Each left shift has doubled the number:

- **Original value** = 14
- **Left shift 1** - Doubled the number to 28
- **Left shift 2** - Doubled the number to 56

Example right shift

The following number is shifted by **three** places to the right.

Step	128	64	32	16	8	4	2	1
Original number in binary	1	1	0	0	1	0	0	0
Right shift by 3 places	0	0	0	1	1	0	0	1


Your notes

Original number: 11001000 = 200

Right shift (3) result: 00011001 = 25

Each right shift has halved the number:

- **Original value** = 200
- **Right shift 1** - Halved the number to 100
- **Right shift 2** - Halved the number to 50
- **Right shift 3** - Halved the number to 25

Bitwise Manipulation

Bitwise AND operation

If **both** bits are 1 in the binary number and the mask, the result will be 1. Otherwise, the result will be 0.

Description	128	64	32	16	8	4	2	1
Binary	1	0	1	1	1	0	0	1
Mask	0	0	1	1	0	0	0	0
Result	0	0	1	1	0	0	0	0

Bitwise OR operation

If **either** bit is 1 in the binary number or the mask, the result will be 1. Otherwise, the result will be 0.

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

Mask	0	1	1	1	0	0	0	0
Result	1	1	1	1	1	0	1	0


Your notes

Bitwise XOR operation

If **only 1 of the bits is** 1 in the binary number or the mask, the result will be 1. Otherwise, the result will be 0.

Description	128	64	32	16	8	4	2	1
Binary	1	0	1	0	1	0	1	0
Mask	0	0	1	1	0	0	0	0
Result	1	0	0	1	1	0	1	0

Character Sets


Your notes

Character Sets

How are characters represented?

- Computers only understand binary and therefore we need to represent characters using binary codes
- For example, the letter 'A' might be represented as 01000001 in binary

Character sets

- A character set is a list of all of the characters and their associated binary code
- Character sets standardise the binary codes for each character
- Without a character set, one system might interpret 01000001 differently from another
- Two common character sets are:
  - American Standard Code for Information Interchange (ASCII)
  - UNICODE

ASCII

- ASCII uses **7-bits** to encode each character, providing for **128** distinct characters
- For example, **'A'** is represented as 65 in decimal, which is 1000001 in binary
- ASCII was created to provide a common standard for encoding characters, which was necessary for compatibility among various types of hardware and software
- An extended version of ASCII exists which encodes each character using **8-bits** creating **256** characters

ASCII table

- The ASCII table shows the relationship between characters that humans recognise and the denary values that represent them in the system
- The denary values can then be converted to binary, representing the original character as binary

Your notes

Char.	ASCII	Char.	ASCII	Char.	ASCII
@	64	U	85	j	106
A	65	V	86	k	107
B	66	W	87	l	108
C	67	X	88	m	109
D	68	Y	89	n	110
E	69	Z	90	o	111
F	70	[	91	p	112
G	71	\	92	q	113
H	72	]	93	r	114
I	73	^	94	s	115
J	74	_	95	t	116
K	75	`	96	u	117
L	76	a	97	v	118
M	77	b	98	w	119
N	78	c	99	x	120
O	79	d	100	y	121
P	80	e	101	z	122
Q	81	f	102	{	123
R	82	g	103	|	124
S	83	h	104	}	125
T	84	i	105	~	126

ASCII Table

Limitations of ASCII

1. It has a limited number of characters

ASCII is limited to 128 characters, which include English alphabets, numerals, and some special and control characters.

A, B, C, ..., Z
a, b, c, ..., z
0, 1, ..., 9
!, @, #, ...

2. It is not suitable for multilingual text

ASCII cannot represent characters from languages other than English, limiting its applicability globally.

No representation for: 'α', 'ö', 'ñ',

3. There is no provision for modern symbols

ASCII does not include modern symbols or emoji's common in today's digital communication.


Your notes

Unicode

- UNICODE was created to be a solution to the limitations of ASCII
- UNICODE uses a much larger bit range, up to **32-bits** (depending on the encoding method), allowing for a wide variety of characters from different languages and scripts
  - Example: The Greek character Lambda 'λ' is represented as U+03BB
  - U+03BB breaks down to:
    - U+, meaning this is a Unicode character
    - 03BB, meaning character 03BB in the UNICODE set

Impact on storage

- ASCII is more storage-efficient, with characters requiring only **7-bits**
- UNICODE characters can require up to **32-bits**, thus potentially using more storage space

Comparison

	ASCII	UNICODE
Encoding system	7-Bits	16-bits or 32-bits
Number of characters	128 characters	65,536 characters (16-bit)
Uses	Used to represent characters in the English language.	Used to represent characters across the world.
Benefits	It uses a lot less storage space than UNICODE.	It can represent more characters than ASCII. It can support all common characters across the world. It can represent special characters such as emoji's.

Drawbacks	It can only represent 128 characters. It cannot store special characters such as emoji's.	It uses a lot more storage space than ASCII.

Your notes