

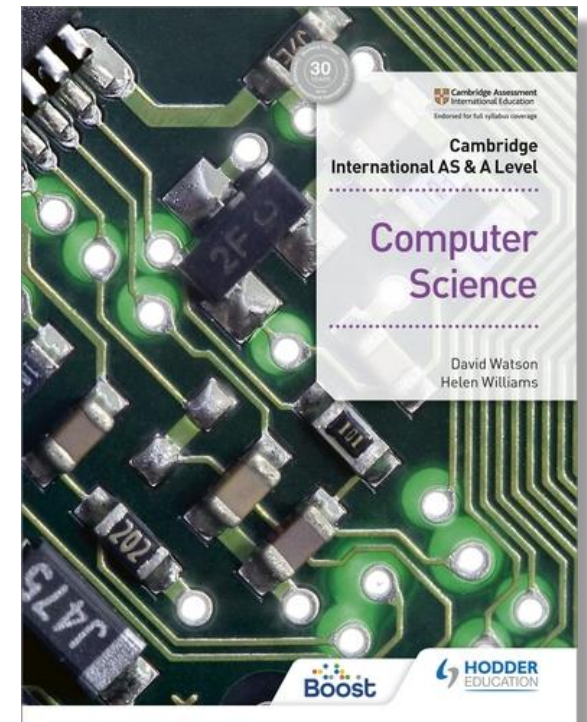
Chapter 1

Information representation and multimedia

1.1 Data Representation

1.2 Multimedia

1.3 File Compression



1 Information Representation and Multimedia

LEARNING OBJECTIVES:

1. Binary magnitudes, binary prefixes and decimal prefixes
2. Binary, denary and hexadecimal number systems
3. How to carry out binary addition and subtraction
4. The use of hexadecimal and binary coded decimal (BCD) number systems
5. The representation of character sets (such as ASCII and unicode)
6. How data for a bit-mapped image is encoded
7. How to estimate the file size for a bit-map image
8. Image resolution and colour depth
9. Encoding of vector graphics
10. The representation of sound in a computer
11. The effects of changing sampling rate and resolution on sound quality
12. The need for file compression methods (such as lossy and lossless formats)
13. How to compress common file formats (such as text files, bit-map images, vector graphics, sound files and video files).

1.1 Data Representation

KEY TERMS:

- **Binary** – base two number system based on the values 0 and 1 only.
- **Bit** – abbreviation for binary digit.
- **One's complement** – each binary digit in a number is reversed to allow both negative and positive numbers to be represented.
- **Two's complement** – each binary digit is reversed and 1 is added in right-most position to produce another method of representing positive and negative numbers.
- **Sign and magnitude** – binary number system where left-most bit is used to represent the sign (0 = + and 1 = –); the remaining bits represent the binary value.
- **Hexadecimal** – a number system based on the value 16 (uses the denary digits 0 to 9 and the letters A to F).
- **Memory dump** – contents of a computer memory output to screen or printer.
- **Binary-coded decimal (BCD)** – number system that uses 4 bits to represent each denary digit.
- **ASCII code** – coding system for all the characters on a keyboard and control codes.
- **Character set** – a list of characters that have been defined by computer hardware and software. It is necessary to have a method of coding, so that the computer can understand human characters.
- **Unicode** – coding system which represents all the languages of the world (first 128 characters are the same as ASCII code).

1.1.1 Number Systems

- **Decimal (base 10)** number system: uses digits **0** to **9** in weighted columns.

10000	1000	100	10	Units
3	1	4	2	1

- **Denary number** represented above: thirty-one thousand, four hundred and twenty-one
- Computer systems use **binary (base 2)** number system: only **0** and **1**
- **Binary system** is the basic building block in all computers
- Computers use **binary digits** known as **bits**
- **Binary**: **1** represents **ON**, **0** represents **OFF**

1.1.2 Binary Number Systems

- The binary system uses **1s** and **0s** only which gives these corresponding weightings:

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

- A typical binary number would be:

1	1	1	0	1	1	1	0
----------	----------	----------	----------	----------	----------	----------	----------

1.1.2 Binary Number Systems

Converting from binary to denary:

- Add the column value to the total for each 1 in a column
- Ignore 0 values when calculating the total

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

↑ ↑ ↑ ↑ ↑ ↑

$$128 + 64 + 32 + 8 + 4 + 2 = 238$$

1.1.2 Binary Number Systems

Converting from denary to binary:

- Reverse operation of converting from denary to binary.
- Two basic methods exist for this process.
- Consider the conversion of the denary number **107** into binary

Method 1:

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

↑ ↑ ↑ ↑ ↑

$$64 + 32 + 8 + 2 + 1 = 107$$

1.1.2 Binary Number Systems

Converting from denary to binary:

Method 2:

2	107	
2	53	remainder: 1
2	26	remainder: 1
2	13	remainder: 0
2	6	remainder: 1
2	3	remainder: 0
2	1	remainder: 1
2	0	remainder: 1
	0	remainder: 0



Write the remainder
from bottom to top to
get the **binary number**:

0 1 1 0 1 0 1 1

1.1.2 Binary Number Systems

Binary addition and subtraction:

- There are a number of methods to represent both **positive** and **negative** numbers.
- We will consider:
 - **One's** complement
 - **Two's** complement

1.1.2 Binary Number Systems

Binary addition and subtraction:

One's complement

- Each **digit** in the binary number is **inverted**.
- **0** becomes **1** and **1** becomes **0**.

$0 \rightarrow 1$

$1 \rightarrow 0$

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

90

1.1.2 Binary Number Systems

Binary addition and subtraction:

Two's complement

- Each **digit** in the binary number is **inverted** and...
- a **'1'** is **added** to the **right-most bit**.
- The **two's** complement uses these **weightings** for an 8-bit number representation

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

1.1.2 Binary Number Systems

Binary addition: Example 1

Add **37** and **58**

-128	64	32	16	8	4	2	1	
0	0	1	0	0	1	0	1	37
0	0	1	1	1	0	1	0	<u>58</u>
0	1	0	1	1	1	1	1	95

1.1.2 Binary Number Systems

Binary addition: Example 2

Add **82** and **69**

-128	64	32	16	8	4	2	1	
0	1	0	1	0	0	1	0	82
0	1	0	0	0	1	0	1	<u>69</u>
1	0	0	1	0	1	1	1	-105

- The binary number **10010111** (-105 in denary) is obtained.
- Adding two **positive numbers** should always result in a **positive answer**.
- However, the addition of two positive numbers in this case resulted in a **negative answer**.
- The **negative result** is due to the **addition** producing a **value outside** the **range** of the **8 bits** being used.
- The range of values that can be represented by the 8 bits is **-127 to +127**.
- The **calculation** produced a value of **151**, which is **larger than** 127 and causes **overflow**.¹³

1.1.2 Binary Number Systems

Binary subtraction:

- To carry out **subtraction** in binary, we **convert** the **number** being **subtracted** into its **negative** equivalent using **two's complement**.
- Then **add** the **two numbers**.

1. **Convert** the two numbers into **binary**:

-128	64	32	16	8	4	2	1	
0	1	0	1	1	1	1	1	95
0	1	0	0	0	1	0	0	68

1.1.2 Binary Number Systems

Binary subtraction: Example 1

Subtract 95 – 68 in binary

2. Find the **two's complement** of 68:

-128	64	32	16	8	4	2	1	
0	1	0	0	0	1	0	0	
1	0	1	1	1	0	1	0	
							1	
1	0	1	1	1	1	0	0	-68

1.1.2 Binary Number Systems

Binary subtraction: Example 1

Subtract 95 – 68 in binary

3. **Add** 95 and -68:

-128	64	32	16	8	4	2	1	
0	1	0	1	1	1	1	1	95
1	0	1	1	1	1	0	0	<u>-68</u>
0	0	0	1	1	0	1	1	27

1.1.2 Binary Number Systems

Binary subtraction: Example 2

Subtract 49 – 80 in binary

1. **Convert** the two numbers into **binary**:

-128	64	32	16	8	4	2	1	
0	0	1	1	0	0	0	1	49
0	1	0	1	0	0	0	0	80

1.1.2 Binary Number Systems

Binary subtraction: Example 2

Subtract 49 – 80 in binary

2. Find the **two's complement** of 80:

-128	64	32	16	8	4	2	1	
0	1	0	1	0	0	0	0	
1	0	1	0	1	1	1	1	
							1	
1	0	1	1	0	0	0	0	-80

1.1.2 Binary Number Systems

Binary subtraction: Example 2

Subtract 49 – 80 in binary

3. **Add** 49 and -80:

-128	64	32	16	8	4	2	1	
0	0	1	1	0	0	0	1	49
1	0	1	1	0	0	0	0	<u>-80</u>
1	1	1	0	0	0	0	1	-31

1.1.2 Binary Number Systems

Measurement of the size of computer memories:

- The **byte** is the **smallest unit** of **memory** in a **computer**.
- Some computers use **larger bytes**, such as **16-bit** systems and **32-bit** systems, but they are **always multiples** of 8.
- **1 byte** of memory **wouldn't allow** you to **store** very much information.
- So **memory size** is **measured** in **these multiples**.

Name of memory size	Equivalent denary value (bytes)
1 kilobyte (1 KB)	1 000
1 megabyte (1 MB)	1 000 000
1 gigabyte (1 GB)	1 000 000 000
1 terabyte (1 TB)	1 000 000 000 000
1 petabyte (1 PB)	1 000 000 000 000 000

Memory size and denary values

1.1.2 Binary Number Systems

Measurement of the size of computer memories:

- The system of numbering in previous slide only refers to some storage devices, but is **technically inaccurate**.
- Another system has been proposed by the **International Electrotechnical Commission (IEC)**; it is based on the **binary system**.
- This system is more **accurate**. Internal memories (such as RAM) should be measured using the **IEC system**.

Name of memory size	Number of bytes	Equivalent denary value (bytes)
1 kibibyte (1 KiB)	2^{10}	1 024
1 mebibyte (1 MiB)	2^{20}	1 048 576
1 gibibyte (1 GiB)	2^{30}	1 073 741 824
1 tebibyte (1 TiB)	2^{40}	1 099 511 627 776
1 pebibyte (1 PiB)	2^{50}	1 125 899 906 842 624

IEC Memory size and denary values

1.1.3 Hexadecimal Number System

Measurement of the size of computer memories:

- The **hexadecimal** system is very **closely related** to the **binary** system.
- **Hexadecimal** (hex) is a **base 16 system** with the weightings.

1048576	65536	4096	256	16	1
(16^5)	(16^4)	(16^3)	(16^2)	(16^1)	(16^0)

- Because it is a system based on **16 different digits**, the numbers **0** to **9** and the letters **A to F** are used to represent **hexadecimal digits**.

1.1.3 Hexadecimal Number System

Measurement of the size of computer memories:

Binary value	Hex Value	Denary Value
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9

Binary value	Hex Value	Denary Value
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

The link between binary, hexadecimal and denary.

1.1.3 Hexadecimal Number System

Converting from binary to hexadecimal & from hexadecimal to binary:

- Starting from the **right** and moving **left**, **split** the **binary** number into **groups** of **4 bits**.
- If the **last group** has **less** than **4 bits**, then simply fill in with **0s** from the **left**.
- Take each group of **4 bits** and **convert** it into the equivalent **hexadecimal digit**.

1.1.3 Hexadecimal Number System

Converting from binary to hexadecimal:

- Example 1:

Convert **101111100001** from binary to hexadecimal.

1011	1110	0001
B	E	1

- Example 2:

Convert **10000111111101** from binary to hexadecimal.

0010	0001	1111	1101
2	1	F	D

1.1.3 Hexadecimal Number System

Converting from hexadecimal to **binary** :

- Example 3:

Convert this hex **45A** to its binary equivalent.

4	5	A
0100	0101	1010
010001011010		

- Example 4:

Convert this hex **BF08** to its binary equivalent.

B	F	0	8
1011	1111	0000	1000
1011111100001000			

1.1.4 Binary-coded decimal (BCD) System

Converting from BCD to digit:

- The **binary-coded decimal (BCD) system** uses a **4-bit code** to represent **each denary digit**:

0 0 0 0 = 0

0 1 0 1 = 5

0 0 0 1 = 1

0 1 1 0 = 6

0 0 1 0 = 2

0 1 1 1 = 7

0 0 1 1 = 3

1 0 0 0 = 8

0 1 0 0 = 4

1 0 0 1 = 9

- Therefore:

Denary number **3 1 6 5** = **0 0 1 1 0 0 0 1 0 1 1 0 0 1 0 1** in BCD format.

1.1.4 Binary-coded decimal (BCD) System

- The **4-bit code** can be **stored** in the **computer** either as **half a byte** or **two 4-bit codes** stored together to form **one byte**.
- For example, using 3 1 6 5:

Method 1:
Four single bytes

0 0 0 0 0 0 1 1 = 3
0 0 0 0 0 0 0 1 = 1
0 0 0 0 0 1 1 0 = 6
0 0 0 0 0 1 0 1 = 5

Method 2:
Two bytes

0 0 1 1 0 0 0 1 = 3 1
0 1 1 0 0 1 0 1 = 6 5

1.1.5 ASCII codes and Unicodes

- To represent text digitally, each character needs to have its own **unique bit-pattern**.
- Bit-patterns are combinations of **1s** and **0s** used to represent data inside of a computer. The bit-pattern used for each character becomes a numeric **character code**.
- A character can be any of the following:
 - **Letters** (upper and lower case letters have separate codes)
 - **Punctuation** (e.g. ?/|\£\$)
 - **Numbers** (0–9)
 - **Non-printing commands** (e.g. Enter, Delete, F1)
- For computers to be able to communicate and exchange text between each other efficiently, they must have an agreed standard that defines which character code is used for which character.
- A standardised collection of characters and the bit-patterns used to represent them is called a **character set**.

1.1.5 ASCII codes and Unicodes

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

1.1.5 ASCII codes and Unicodes

- The **problem with ASCII** is its limited character representation (128 for standard 7-bit ASCII).
- ASCII can't represent **all languages, scripts, numbers, and symbols** worldwide.
- **Unicode** is the commonly used character set, with over a million possible characters.
- Unicode allows encoding with **three different standards** based on the minimum number of bits used.
- With Unicode, we can store **every character from every alphabet** and include special symbols like mathematical operators, emojis, etc.
- The first 128 codes in Unicode and ASCII **represent the same characters**.

1.1.5 ASCII codes and Unicodes

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F		
0000																																		Symbols
0020		!	"	#	\$	%	&	'	()	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?		Number
0040	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_		Alphabet
0060	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{		}	~			
0080	€		,	f	„	…	†	‡	^	%	Š	‹	Œ		Ž		’	’	“	”	•	–	—	ˆ	™	š	›	œ		ž	ÿ			
00A0		ı	ı	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯	°	±	²	³	´	µ	¶	·	,	˚	°	»	¼	½	¾	¿		
00C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß		Latin
00E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ		
0100	Ā	ā	Ă	ă	Ą	ą	Ć	ć	Ĉ	ĉ	Ċ	ċ	Č	č	Ď	ď	Đ	đ	È	è	Ě	ě	Ê	ê	Ë	ë	Ĝ	ĝ	Ğ	ğ				
0120	Ġ	ġ	Ģ	ģ	Ĥ	ĥ	Ħ	ħ	Ī	ī	Ĭ	ĭ	Į	į	İ	ı	U	ü	Ï	ï	Ĵ	ĵ	Ķ	ķ	κ	Ĺ	ĺ	Ł	ł	ℓ	ℓ	ℓ		
0140	ƒ	ℓ	ı	Ń	ń	Ņ	ņ	Ň	ň	ň	Ŋ	ŋ	Ō	ō	Ŏ	ö	Œ	œ	Ŕ	ŕ	Ŗ	ŗ	Ř	ř	Ś	ś	Ŝ	ŝ	Ş	ş				
0160	Š	š	Ţ	ţ	Ť	ť	Ŧ	ŧ	Ū	ū	Ŭ	ŭ	Ů	ů	Ű	ű	Ų	ų	Ŵ	ŵ	Ŷ	ŷ	Ÿ	Ž	ž	Ž	ž	Ž	ž					
0180	Ɓ	Ƃ	ƃ	Ƅ	ƅ	Ɔ	Ƈ	ƈ	Ɖ	Ɗ	Ƌ	ƌ	ƍ	Ǝ	Ə	Ɛ	Ɔ	Ɖ	Ɗ	Ƌ	ƌ	ƍ	Ǝ	Ə	Ɛ	Ɔ	Ɖ	Ɗ	Ƌ	ƌ	ƍ	Ǝ	Ə	
01A0	Ɛ	Ɔ	Ɖ	Ɗ	Ƌ	ƌ	ƍ	Ǝ	Ə	Ɛ	Ɔ	Ɖ	Ɗ	Ƌ	ƌ	ƍ	Ǝ	Ə	Ɛ	Ɔ	Ɖ	Ɗ	Ƌ	ƌ	ƍ	Ǝ	Ə	Ɛ	Ɔ	Ɖ	Ɗ	Ƌ	ƌ	ƍ
01C0	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı
01E0	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı
0200	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı
0220	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı
0240	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı

1.2 Multimedia

KEY TERMS:

- **Bit-map image** – system that uses pixels to make up an image.
- **Pixel** – smallest picture element that makes up an image.
- **Colour depth** – number of bits used to represent the colours in a pixel, e.g. 8 bit colour depth can represent $2^8 = 256$ colours.
- **Bit depth** – number of bits used to represent the smallest unit in, for example, a sound or image file – the larger the bit depth, the better the quality of the sound or colour image.
- **Image resolution** – number of pixels that make up an image, for example, an image could contain 4096×3192 pixels (12 738 656 pixels in total).
- **Screen resolution** – number of horizontal and vertical pixels that make up a screen display. If the screen resolution is smaller than the image resolution, the whole image cannot be shown on the screen, or the original image will become lower quality.
- **Resolution** – number of pixels per column and per row on a monitor or television screen.
- **Pixel density** – number of pixels per square centimetre.
- **Vector graphics** – images that use 2D points to describe lines and curves and their properties that are grouped to form geometric shapes.
- **Sampling resolution** – number of bits used to represent sound amplitude (also known as bit depth).
- **Sampling rate** – number of sound samples taken per second.
- **Frame rate** – number of video frames that make up a video per second.

1.2.1 Multimedia – Bitmapped Graphics

Bitmapped Graphics:

- A **bitmapped graphic** (also called a bitmap image) is made up of a grid of **pixels**.
- A **pixel** (short for ‘picture element’) is the smallest element in an image.
- If you **magnify** an image, you will see that it is made up of these **tiny elements**.

Pixels:

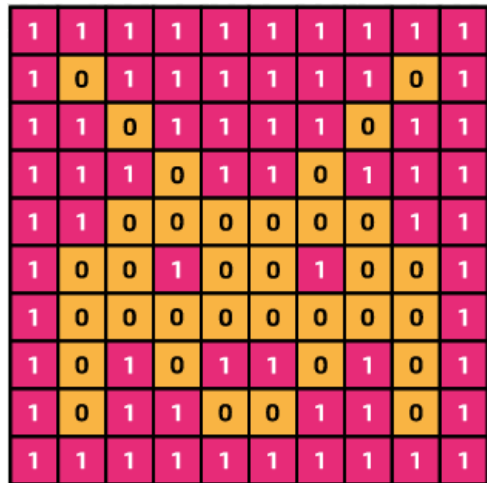
- A **pixel** represents the **smallest element** of a **bitmapped graphic**: a **single colour**.
- As all data in a computer is stored in **1s** and **0s**, each **colour** needs to have a **binary code** assigned to it.
- For example, the **black-and-grey image** can be represented by using just **one bit per pixel**, by assigning a **0** to each **black pixel** and a **1** to each **grey pixel**.

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

1.2.1 Multimedia – Bitmapped Graphics

Colour Dept:

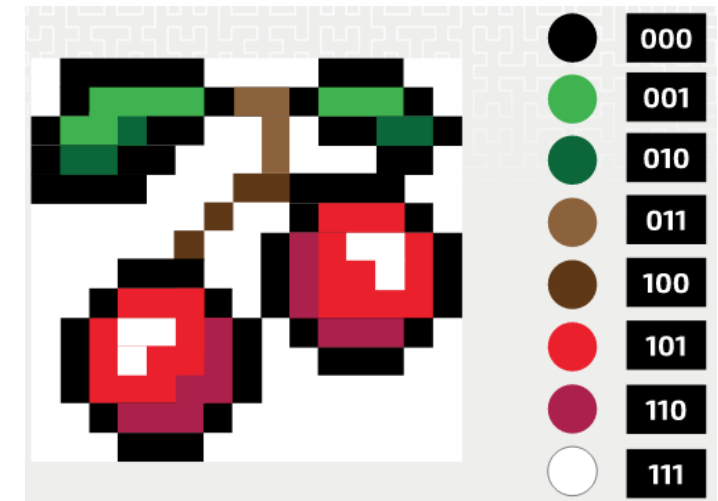
- Bitmap images with **two colours** use **one bit per pixel** (0 for orange, 1 for pink).
- For images with **more colours**, more bits per pixel are needed.
- With **2 bits** per pixel, you get **4 different bit patterns**, representing four colours.
- Using **3 bits** allows for **8 colours** ($2 \times 2 \times 2 = 8$).
- To calculate the number of colours, you can use **exponents: $2^3 = 8$ colours**.



Binary	Colour
0	Orange
1	Pink

Binary	Colour
00	Black
01	White
10	Blue
11	Red

Bits	Number of colours
2^1	2
2^2	4
2^3	8
2^4	16



1.2.1 Multimedia – Bitmapped Graphics

Image Resolution:

- **Image resolution** determines the **clarity of an image** on **screen** or **paper**.
- **Zooming** in bitmap images **stretches pixels** into **larger blocks**, causing **poor quality** at high magnifications.
- Image resolution is measured in **pixels per inch (ppi)**, for example, **300 ppi**.
- It's important to note that 'image resolution' can also refer to the **size** of a **bitmapped graphic in pixels**, calculated by **multiplying** the **width** by the **height** of the image.

1.2.1 Multimedia – Bitmapped Graphics

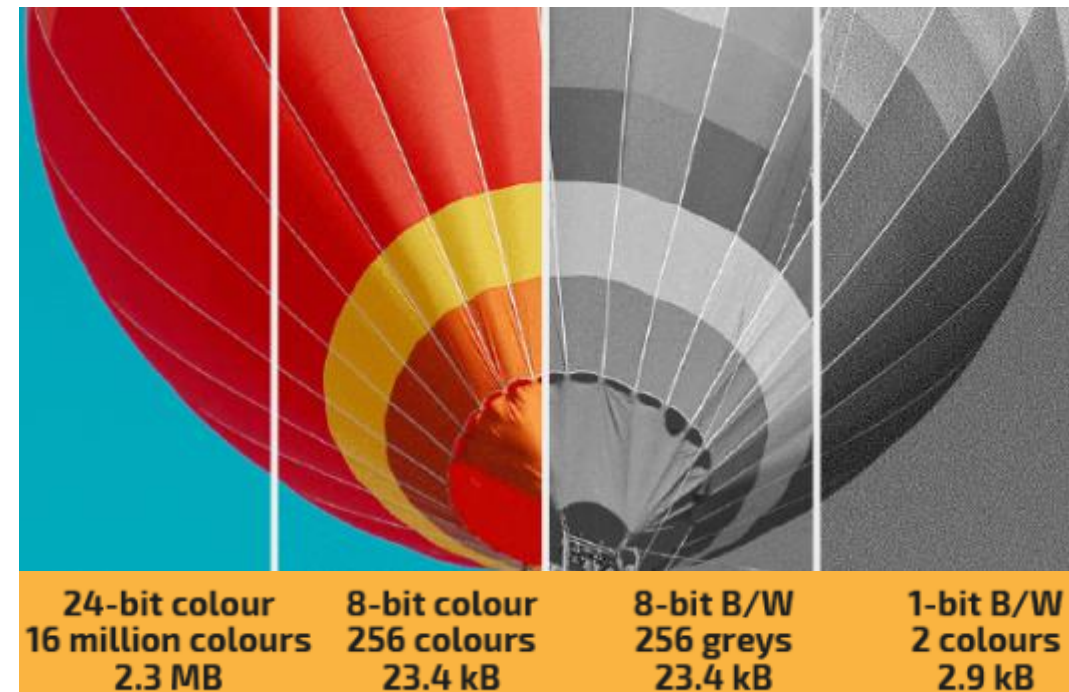
Calculating Bitmap File Size:

- **Colour depth** refers to the **number of bits** used **per pixel**, determining the **available colours** for an image.
- **Image resolution** is the **size** of a **bitmapped graphic**, calculated by multiplying **width** and **height** in **pixels**.
- To find the **image file size in bits**, multiply **resolution** by **colour depth**:
image file size (in bits) = width (in pixels) × height (in pixels) × colour depth.
- Increasing either the **colour depth** or **image resolution** will result in a **larger** image file size.
- For example, with an image of **8x8 pixels** and a **colour depth** of **4 bits**:
Total bits = $8 \times 8 \times 4 = 256$ bits.

1.2.1 Multimedia – Bitmapped Graphics

Effects of changing resolution and colour dept:

- The **8-bit colour depth** represents **256 colours**, while the **24-bit image** represents **16.7 million colours**, making the latter of higher quality due to human vision perceiving more than **256 colours**.
- **File size** of an image is influenced by its **colour depth**. As shown in the picture, increasing the colour depth results in an **increase in file size**.



1.2.1 Multimedia – Bitmapped Graphics

Colour dept and Image quality:

- **Colour depth** influences **image quality** by representing more colours for a closer resemblance to reality.
- Some images don't need a very high colour depth as they don't aim to capture real-life details.
- Even for real-life images, reducing the colour depth may not always be noticeable.
- Finding a suitable colour depth often involves **trial and error**.

Colour depth and file size

- **Reducing the colour depth** reduces the **file size**.
- More bits used for colours mean larger storage space needed.
- A smaller file size enables **faster downloads** on web pages and quicker transmission in emails.

1.2.1 Multimedia – Bitmapped Graphics

Metadata:

- The **actual file size** of a bitmapped graphic is larger than the calculated file size due to **metadata** storage.
- **Metadata** includes information about the image, such as:
 - Dimensions
 - file format
 - creation date and time
 - geographical location
 - camera settings

1.2.1 Multimedia – Bitmapped Graphics

File Header:

- **Metadata** must be in a **fixed position** in the image file for easy access by any software.
- It's usually placed in the **file header** at the beginning of the file.
- **Exif** (exchangeable image file format) is one way to organize image metadata, covering various tags like camera settings and time details.

Property	Value
Dimensions	2753 x 3060
Width	2753 pixels
Height	3060 pixels
Horizontal resolution	72 dpi
Vertical resolution	72 dpi
Bit depth	24
Compression	
Resolution unit	2
Colour representation	sRGB
Compressed bits/pixel	
Camera	
Camera maker	Google
Camera model	Pixel 4a (5G)
F-stop	f/1.7
Exposure time	1/2924 sec.
ISO speed	ISO-56
Exposure bias	0 step
Focal length	4 mm
Max aperture	1.58

1.2.1 Multimedia – Bitmapped Graphics

Screen or Display Resolution:

- **Image resolution** is different from **screen resolution**, which indicates the **number of pixels** a screen displays.
- **Screen resolution** is expressed as the **width × height** of the screen in **pixels** (e.g., 1920×1080).
- **High-definition screens** have a **large number of pixels** (e.g., 1280×720 and above) for enhanced clarity in **images** and **videos**.
- Viewing a **low-resolution image** (e.g., 300×240 pixels) on a higher-resolution screen (e.g., 1280×720 pixels) in full-screen mode will cause the image to look **pixelated** due to **rescaling**.

1.2.1 Multimedia – Bitmapped Graphics

File Formats:

- Common **file formats** of bitmapped graphics are:
 - **Bitmap** – file extension is .bmp
 - **PNG** (Portable Network Graphic) – file extension is .png
 - **JPEG** (Joint Photographic Experts Group) – file extension is .jpg or .jpeg
 - **GIF** (Graphics Interchange Format) – file extension is .gif

1.2.1 Multimedia – Bitmapped Graphics

How Bitmapped Graphics are created:

- **Bitmapped graphics** are typically created by taking **photos** with **digital cameras** or **scanners**, storing the image as a grid of **pixels**.
- Creating bitmaps manually with editing software can be tedious, especially for complex images.
- Simple bitmapped graphics, like game sprites, can be easily created using **online software**.

1.2.1 Multimedia – Bitmapped Graphics

Compression of Bitmapped Graphics:

- **Bitmapped graphic files** can be very **large**, depending on the number of **pixels** and **color depth**.
- For example, a **12-megapixel image** with **24-bit** color code would result in a **36 MB** file size.
- Bitmapped graphics can be **compressed** to take up **less storage space**, making them **faster** to **transmit** and **display**.


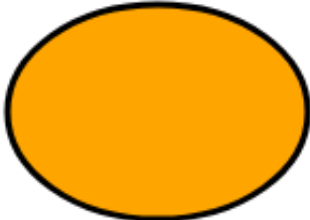

1.2.2 Multimedia – Vector Graphics

Vector Graphics:

- **Vector graphics** are images made up of lists of objects with their properties, like rectangles, lines, polygons, or circles.
- Each object has properties for dimensions, appearance, and position stored in a **drawing list**.
- **SVG** (Scalable Vector Graphics) is a common file format for vector graphics, widely supported by web browsers.

1.2.2 Multimedia – Vector Graphics

Examples of drawing lists of three simple vector graphics:

Object	Circle	Ellipse	Rectangle
Drawing list	<pre><circle cx="100" cy="100" r="50" fill="lightgreen" stroke="black" stroke- width="5" /></pre>	<pre><ellipse cx="80" cy="60" rx="70" ry="50" fill= "orange" stroke="black" stroke-width="3"/></pre>	<pre><rect x="10" y="10" width="140" height="100" fill="yellow" stroke="black" stroke-width="1" /></pre>
Explanation of properties	cx, cy: define the coordinates of the centre of the circle r: defines the radius of the circle	cx, cy: define the coordinates of the centre of the ellipse rx, ry: define the horizontal and vertical radii	x, y: define the position of the top left hand corner of the rectangle
Image			

1.2.2 Multimedia – Vector vs Bitmapped Graphics

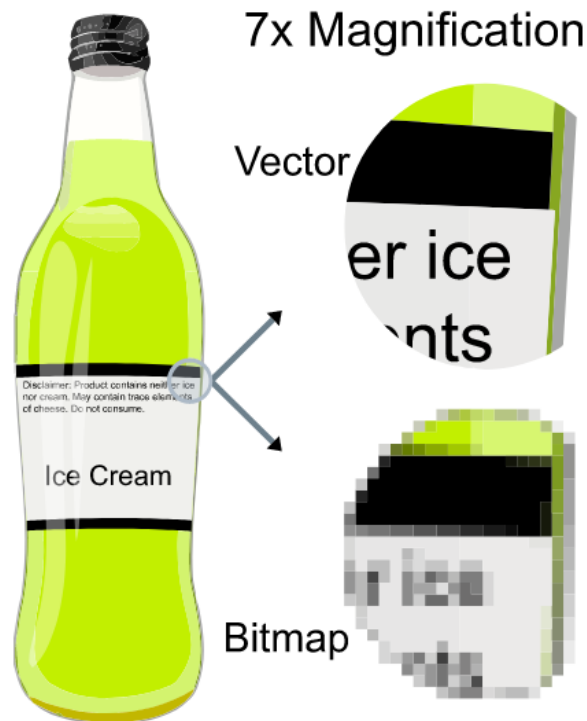
Advantages of Vector Graphics:

- **Vectors** can be resized without **distortion**.
- Vector graphics are **scalable**, adapting properties of objects for different sizes without pixelation.
- Bitmapped graphics may look **pixelated** when resized, requiring pixel stretching.
- Vector graphics scale without changing file size, while bitmapped graphics increase file size when resized.
- **Vector graphics** usually result in **smaller file sizes** than bitmapped graphics.
- Creating and editing vector graphics is simpler, and objects can be modified independently.
- Vector graphics maintain **good quality** regardless of the resolution, while bitmapped graphics' quality depends on their resolution and color depth.

1.2.2 Multimedia – Vector vs Bitmapped Graphics

Advantages of Bitmapped Graphics:

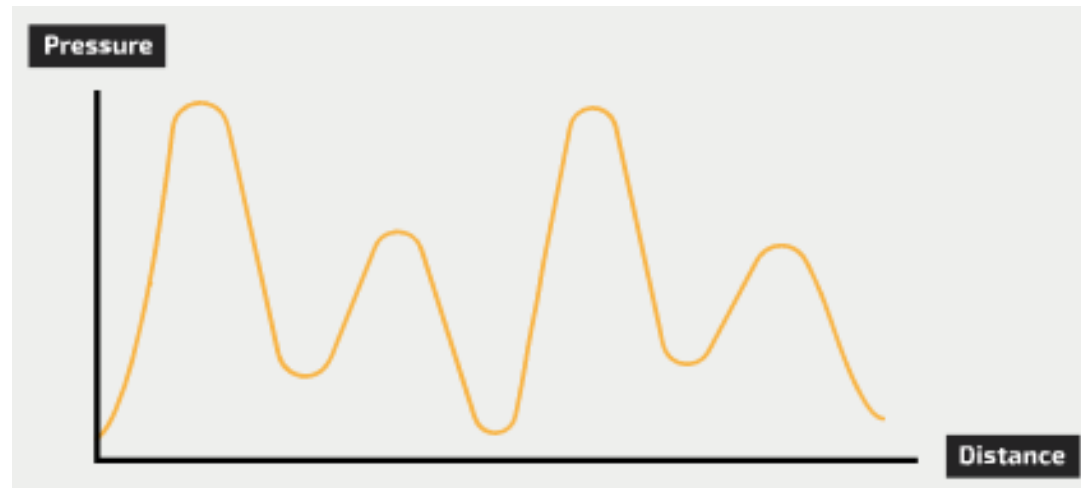
- **Vectors graphics** are created by combining a limited number of shapes, limiting the variety of images they can represent.
- They are commonly used for **illustrations, cartoons, logos, web designs**, etc.
- **Bitmapped graphics** can depict **complexity** and **detail**, making them suitable for storing photographs.



1.2.3 Multimedia – Sound

Sound and Sound Waves:

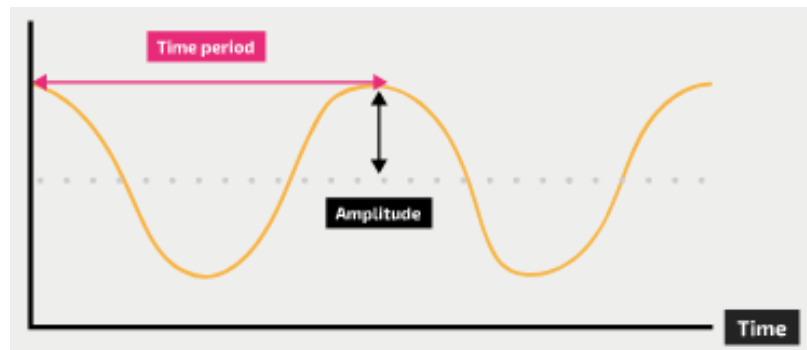
- **Sound is vibration** created by vibrating objects like guitar strings, voice boxes, or loudspeakers.
- **Sound waves** consist of vibrating particles that travel away from the source.
- We hear sounds when **vibrations in the air** make our eardrums vibrate and send signals to our brains.
- **Microphones** detect air vibrations and convert them into electrical signals.
- Sound waves are **analog**, allowing smooth changes in air pressure on a graph, rather than discrete values.



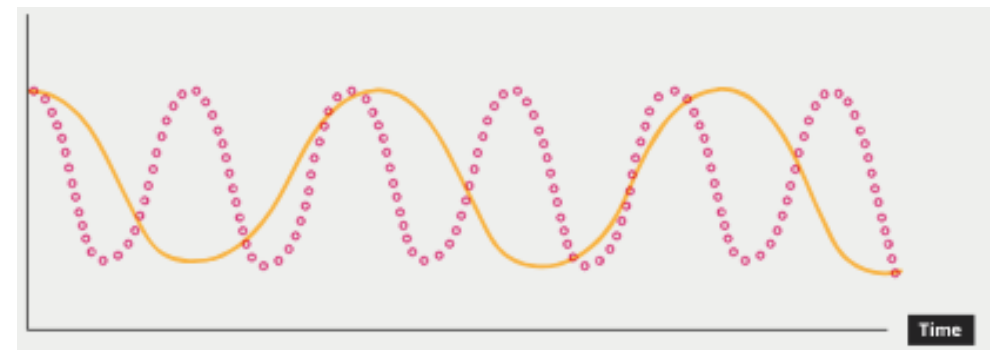
1.2.3 Multimedia – Sound

Amplitude and Frequency:

- **Amplitude and frequency** are two important features of sound waves affecting how we perceive sound.
- **Amplitude** represents the wave's height on the graph and corresponds to the sound's **volume** (higher amplitude = louder sound).
- **Frequency** is the time between two wave peaks, inversely related to the sound's **pitch** (higher frequency = higher pitch).
- The graph of a **shrill sound** like a whistle looks more bunched up, while a **deep sound** like a double bass has a more spread-out graph.



Amplitude and time period

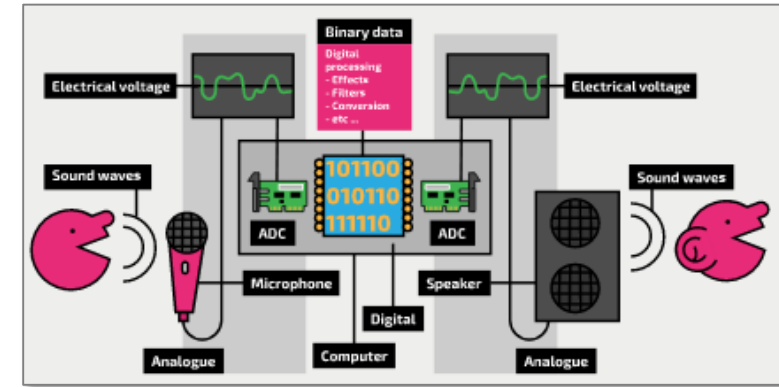


Two waves of different frequencies

1.2.3 Multimedia – Sound

Sound Processing:

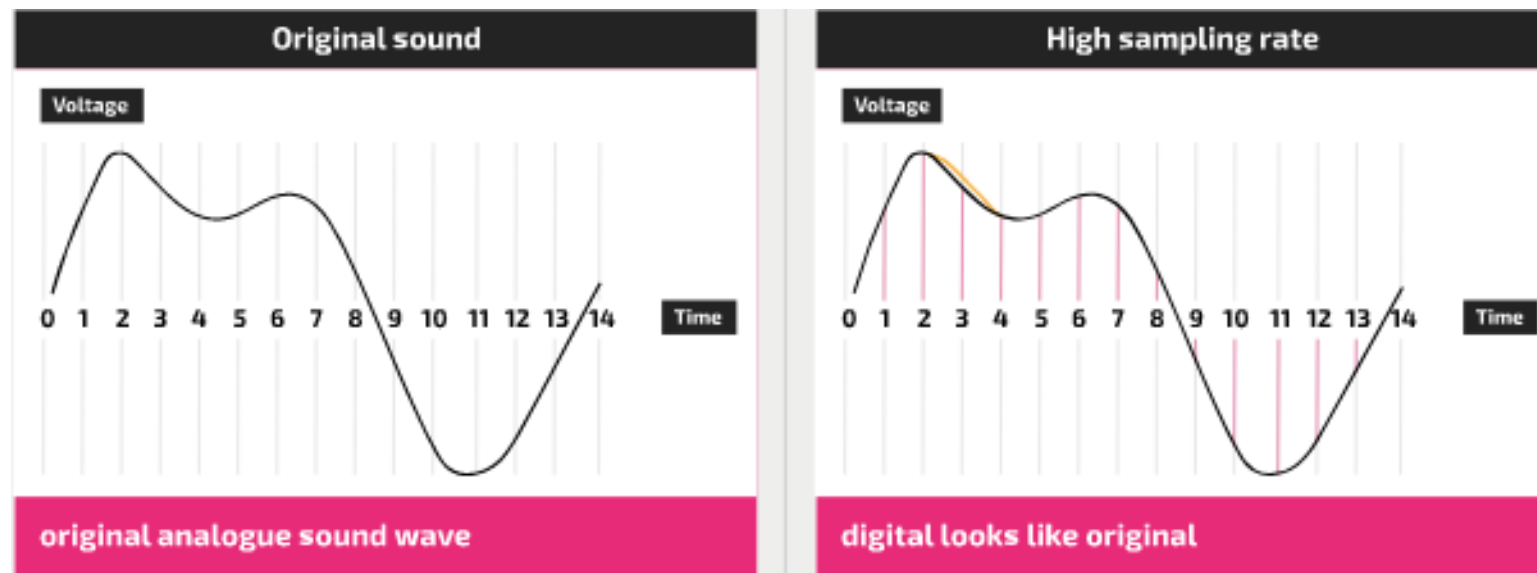
- **Digital sound files** recreate analogue sound waves using a computer.
- The process involves:
 - Using a **microphone** to convert sound waves into **analogue electrical signals**.
 - Processing signals with an **ADC** to convert them into **digital values** (sampling).
 - Storing the digital values in the computer's memory as a **binary pattern**.
- Digital audio files can be edited using sound processing programs.
- To play the digital audio file, it needs to be converted back to **analogue sound waves**.
- This is done using a **DAC** to convert digital values to **analogue electrical signals**.
- The electrical signals are then passed to a **speaker**, which vibrates the cone to create **analogue sound waves**.
- The produced analogue waves may differ significantly from the original sound waves. 52



1.2.3 Multimedia – Sound

Sampling Rate:

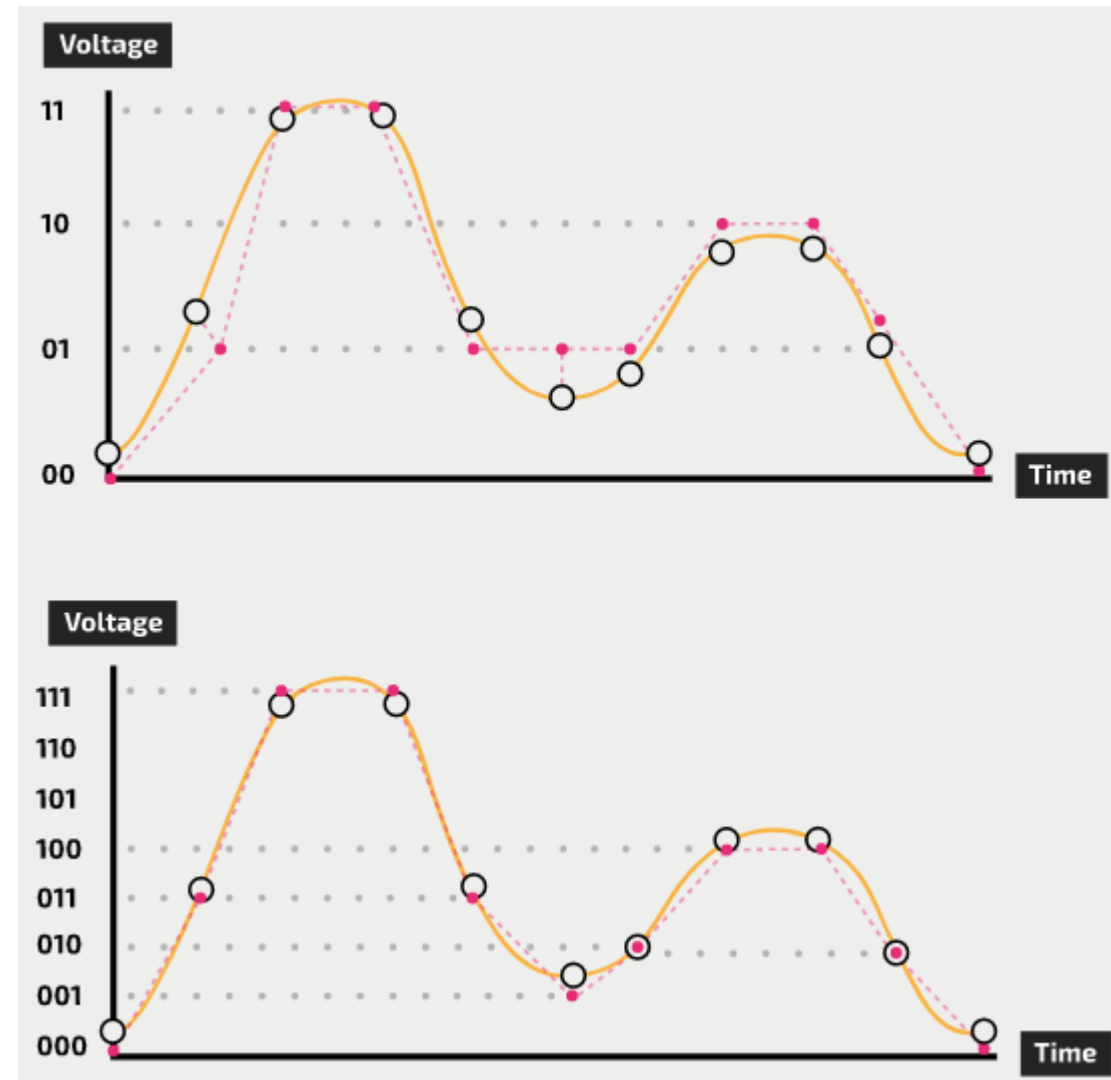
- **Sampling rate** determines how often the ADC samples the sound wave.
- Low frequency sampling results in **different sound** from the original.
- High frequency sampling produces **similar sound**, but **larger file size** due to more samples.
- Sampling rate is measured in **hertz** (samples per second).
- **Higher sampling rate** improves audio quality and increases file size.



1.2.3 Multimedia – Sound

Sample Resolution:

- **Sample resolution** determines the number of bits used to represent each sample.
- Low resolution limits the **accuracy** of representing variations in the analogue signal digitally.
- Higher resolution allows for a more **accurate representation** of each sample.
- High resolution increases **file size**, requiring more storage space.
- A CD has a sample resolution of **16 bits per sample** ($2^{16} = 65536$ binary patterns).



2-bit and 3-bit sample resolution for the same waveform

1.2.3 Multimedia – Sound

Storage requirements for audio files:

- **Storage requirements for an audio file** in bits can be found using the formula:
storage (in bits) = sampling rate × seconds × sample resolution.
- For stereo sound files, include an additional factor of **2** in the equation due to the use of two tracks.

1.2.3 Multimedia – Sound

Changing the sampling rate and resolution:

- When creating and storing digital audio files, you need to balance **accuracy and file size**.
- Increasing the **sampling rate or sample resolution** improves audio quality but increases file size.
- Decreasing the sampling rate or resolution reduces file size but degrades audio quality.
- **Example:** For a 3-minute file with CD quality (44.1kHz, 16 bits), the size is approximately 127MB.
- **Increasing sampling rate** to 48kHz results in a file size of 138.2MB, around 9.9% larger.
- **Decreasing sampling rate** to 32kHz results in a file size of 92.2MB, around 27.3% smaller.
- **Increasing sample resolution** to 24 bits results in a file size of 190.5MB, around 50.5% larger.
- **Decreasing sample resolution** to 12 bits results in a file size of 95.3MB, around 25.25% smaller.

1.3 File Compression

KEY TERMS:

- **Lossless file compression** – file compression method where the original file can be restored following decompression.
- **Lossy file compression** – file compression method where parts of the original file cannot be recovered during decompression, so some of the original detail is lost.
- **JPEG** – Joint Photographic Expert Group – a form of lossy file compression based on the inability of the eye to spot certain colour changes and hues.
- **MP3/MP4 files** – file compression method used for music and multimedia files.
- **Audio compression** – method used to reduce the size of a sound file using perceptual music shaping.
- **Perceptual music shaping** – method where sounds outside the normal range of hearing of humans, for example, are eliminated from the music file during compression.
- **Bit rate** – number of bits per second that can be transmitted over a network. It is a measure of the data transfer rate over a digital telecoms network.
- **Run length encoding (RLE)** – a lossless file compression technique used to reduce text and photo files in particular

1.3 Compression

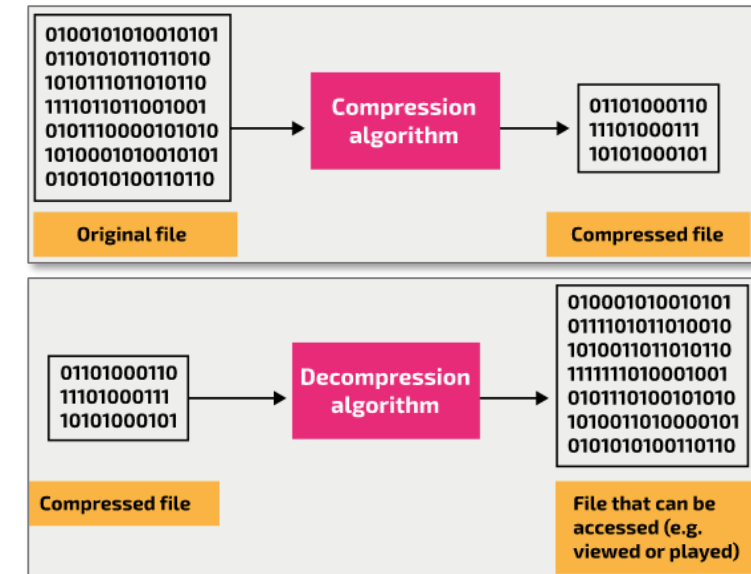
Compression:

- Compression reduces file sizes, making them smaller.
- Large files can be a problem when storage capacity is limited.
- Smaller file size allows for more files to be stored in the same space.
- Compressed files are quicker to transfer, upload, and download.

1.3.1 Compression Fundamentals

Compression Fundamentals:

- A **compression technique** is an **algorithm** used to **reduce file size**.
- **Digital files** are **long series of binary digits**.
- **Compression algorithm** reduces the **number of binary digits**, resulting in **fewer bits** in the **compressed file**.
- **Compressed files** must be **decompressed** to be accessed meaningfully.
- **Decompression** may or may not produce the **original file**, depending on whether the compression technique is **lossy or lossless**.
- **Compression** doesn't change the **fundamental properties** of the file (e.g., **resolution, duration**).
- **Compression** and **decompression** processes are usually **automatic**.
- Most users save files in **compressed formats** without actively choosing compression (e.g., **JPEG** for images on smartphones).
- Applications automatically apply **compression and decompression algorithms** based on the **file format** to display the data.



1.3.2 Compression Ratios

Compression Ratios:

- **Compression ratio** gives an indication of the effectiveness of a compression method in **reducing file size**.
- It's calculated by dividing the **size of the uncompressed file** by the **size of the compressed file**.
- A **higher compression ratio** means **better compression** or a **higher percentage of space saved** during compression.
- **Example**: If the original file size is **10MB** and the compressed file size is **2MB**, the compression ratio is **$10\text{MB} / 2\text{MB} = 5:1$** .

1.3.3 The need to compress images

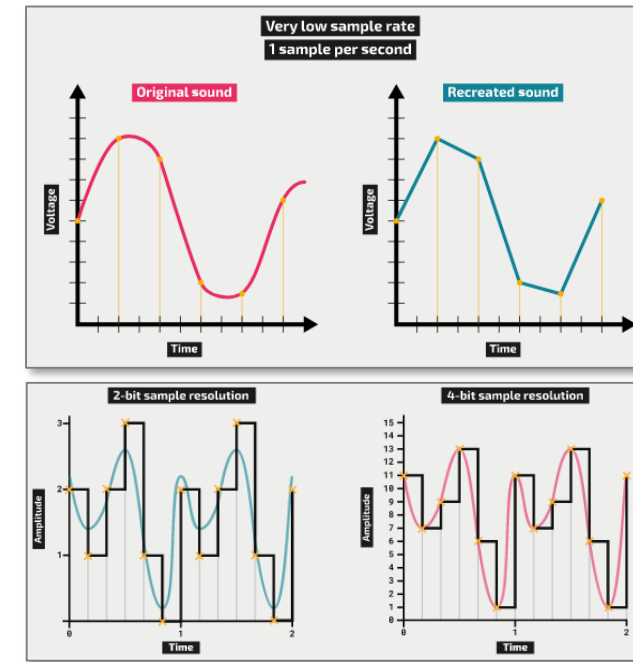
The need to compress images:

- Image files from mid-range cameras can be **very large** (24 Megapixel).
- Typical photographic images use **12 bits per color channel** (R, G, B), totaling **36 bits per pixel**.
- The size of an **uncompressed image** from a mid-range camera would be around **108MB**.
- Image size calculation: 24 million pixels x 36 bits/pixel = 864 million bits = 108 million bytes = 108MB.
- Often, storage space is limited, e.g., in digital cameras with memory cards.
- To reduce storage and transfer time, images are commonly stored in **JPEG format** (highly compressed), reducing the file size to around **10% of its uncompressed size**.

1.3.4 The need to compress sound files

The need to compression sound files:

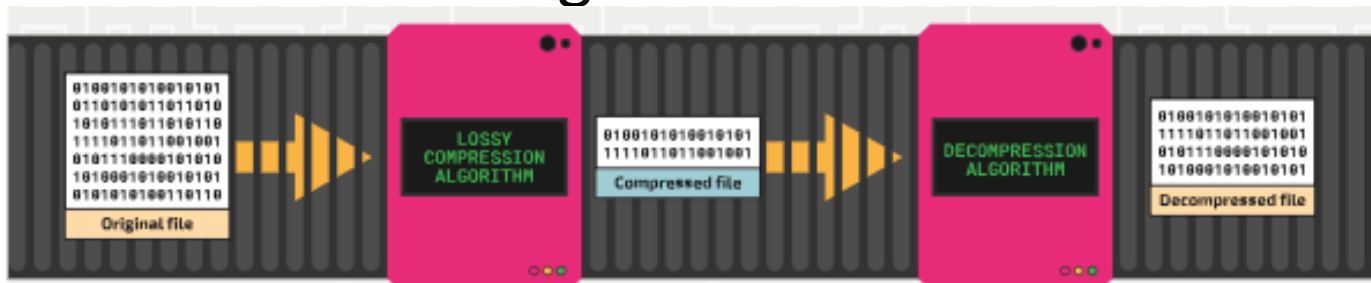
- Sound files can be **very large**. The popular **MP3** format is commonly used for streaming and downloads due to its **compression** and reduced size.
- Music enthusiasts and professionals work with **uncompressed WAV** format to maintain high sound quality.
- **Analogue sound** is converted into a **digital format** through **analogue-to-digital conversion (ADC)** using **sampling**.
- **Sampling rate** determines the accuracy of sound representation in digital audio.
- **Sample resolution** (e.g., 8-bit or 16-bit) affects the number of values a sample can take and the accuracy of sound representation.
- The size of a sampled sound file is calculated by: **Sampling rate × Length of sound (seconds) × Sample resolution**.
- **MP3 compression** typically reduces file size by **90%**. For example, a **15MB uncompressed file** becomes around **1.5MB** in compressed form.



1.3.5 Lossy Compression

Lossy Compression:

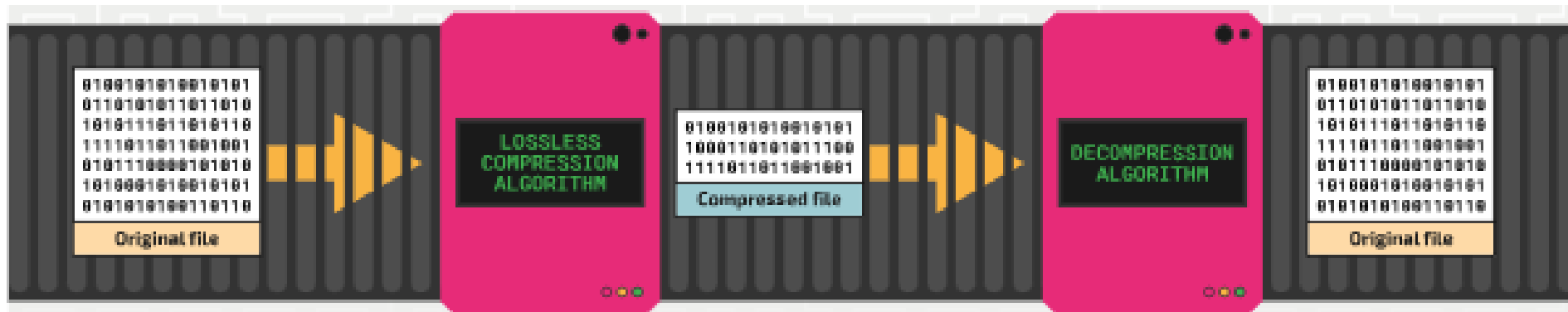
- **Lossy compression** permanently removes data from a file, resulting in an approximation of the original data after decompression.
- It is commonly used for **image and sound files** where slight data loss is not noticeable to the human eye or ear.
- Common lossy formats include: **MP3** (audio), **JPEG** (photographs), **MPEG-1**, and **MPEG-2** (video).
- **Files with words or numbers** require **lossless compression** as their contents need to be exactly the same after decompression.
- **Repeated rounds of lossy compression** can cause severe **data loss**, making the file output **unrecognizable** from the original.



1.3.6 Lossless Compression

Lossless Compression:

- **Lossless compression** retains all data during the compression process, making the decompressed contents identical to the original file.
- Suitable for files where every detail needs to be preserved, like **documents** with **text** and **numbers**.
- Users often don't actively choose compression; it's automatically applied when saving files to reduce size.
- **Common lossless formats: PNG** (high-quality images), **GIF** (images and short animations), **ZIP** (compressing multiple files).



1.3.7 File compression applications

MPEG-3 (MP3) and MPEG-4 (MP4):

- **MPEG-3 (MP3)** technology compresses music and sounds into an MP3 file format, reducing file size by about **90%**.
- MP3 files are used in **MP3 players, computers, and mobile phones** for downloading or streaming music from the internet.
- Compression uses **perceptual music shaping**, removing inaudible frequencies and softer sounds without significant quality loss.
- MP3 is a **lossy format**, meaning part of the original file is lost during compression, and it cannot be fully restored.
- MP3 quality depends on the **bit rate** used during file creation, typically between **80 to 320 kilobits per second**.
- **MPEG-4 (MP4)** files allow storage of **multimedia files**, including music, videos, photos, and animations, without losing discernible quality during streaming.

1.3.7 File compression applications

Photographic (bit-map) images:

- **Photographic files** are compressed using the JPEG format, resulting in reduced **file size** and **image quality** (lossy compression).
- JPEG compression creates a **new file** that cannot be reconstructed back to the original.
- JPEG reduces the raw bit-map image by a factor of **5 to 15**, depending on the original quality.
- **Vector graphics** like **.svg** files can also be compressed since they are defined in XML text files, enabling compression.

1.3.7 File compression applications

Run-length encoding (RLE):

- **Run-length encoding (RLE)** compresses various file formats using a **lossless/reversible method**.
- It reduces the size of **repeated, adjacent data** (e.g., colors in an image) by encoding them into **two values**: count and data code.
- RLE is most effective when there is a **long run of repeated units/bits**.

1.3.7 File compression applications

Using RLE on text data:

- Consider the **text string** 'aaaaabbbbccdddd'.
- Assuming **each character** requires **1 byte**, then this **string** needs **16 bytes**.
- If we assume **ASCII code** is being used, then the string can be **coded** as follows:

a	a	a	a	a	b	b	b	b	c	c	d	d	d	d	d
05 97					04 98				02 99		05 100				

- This means we have **five characters** with ASCII code **97**, **four** characters with ASCII code **98**, **two** characters with ASCII code **99**, and **five** characters with ASCII code **100**.
- Assuming **each number** in the **second row** requires **1 byte** of **memory**, the **RLE code** will need **8 bytes**.
- This is **half** the **original file size**.

1.3.7 File compression applications

Using RLE on text data:

- One **issue** occurs with a string such as 'cdcdcdcd', where compression is **not very effective**.
- To cope with this we use a **flag**.
- A **flag preceding data** indicates that **what follows** are the **number of repeating units** (for example, **255 05 97** where **255** is the **flag** and the other **two numbers** indicate that there are **five items** with ASCII code **97**).
- When a **flag is not used**, the **next byte(s) are taken** with their face **value** and a run of **1** (for example, **01 99** means **one character** with ASCII code **99** follows).



EXAMPLE

1.3.7 File compression applications

Using RLE on text data:

- Consider this example:

String	aaaaaaaa	bbbbbbbbbb	c	d	c	d	c	d	eeeeeeee
Code	08 97	10 98	01 99	01 100	01 99	01 100	01 99	01 100	08 101

- The **original string** contains **32 characters** and would occupy **32 bytes of storage**.
- The **coded** version contains **18 values** and would require **18 bytes of storage**.
- Introducing a **flag** (255 in this case) produces:

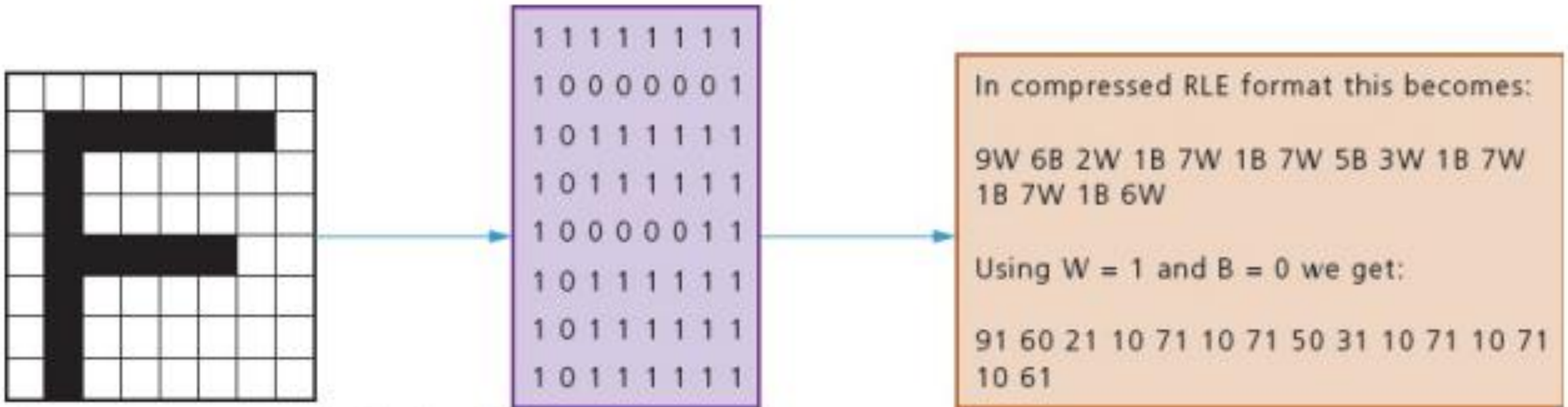
255 08 97 255 10 98 99 100 99 100 99 100 255 08 101

- This has **15 values** and would, therefore, require **15 bytes of storage**.
- This is a **reduction in file size** of about **53%**.

1.3.7 File compression applications

Using RLE with images (*Black and white images*):

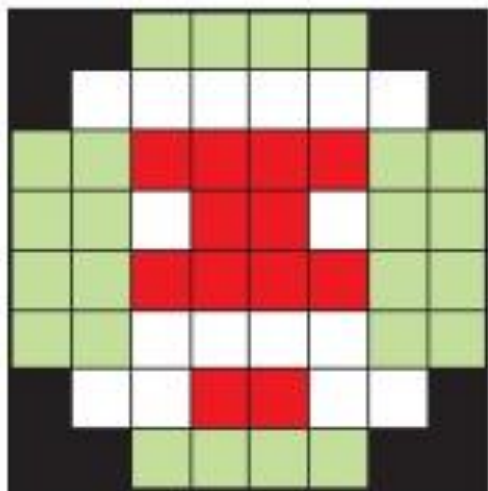
- The letter **F** in a grid where **each square** requires **1 byte** of storage.
- A **white square** has a value **1** and a **black square** a value of **0**.
- The **8 × 8 grid** would need **64 bytes**; the compressed **RLE** format has **30 values**, and therefore needs only **30 bytes** to **store the image**.




1.3.7 File compression applications

Using RLE with images (*Colour Images*):

- This is an **object** in **four colours**.
- **Each colour** is made up of **red, green and blue (RGB)** according to the code on the right.



Square colour	Red	Green Components	Blue
	0	0	0
	255	255	255
	0	255	0
	255	0	0

- This produces the following data:

2 0 0 0 4 0 255 0 3 0 0 0 6 255 255 255 1 0 0 0 2 0 255 0 4 255 0 0 4 0 255 0 1 255 255 255 2
255 0 0 1 255 255 255 4 0 255 0 4 255 0 0 4 0 255 0 4 255 255 255 2 0 255 0 1 0 0 0 2 255 255
255 2 255 0 0 2 255 255 255 3 0 0 0 4 0 255 0 2 0 0 0

1.3.7 File compression applications

Using RLE with images (*Colour Images*):

```
2 0 0 0 4 0 255 0 3 0 0 0 6 255 255 255 1 0 0 0 2 0 255 0 4 255 0 0 4 0 255 0 1 255 255 255 2
255 0 0 1 255 255 255 4 0 255 0 4 255 0 0 4 0 255 0 4 255 255 255 2 0 255 0 1 0 0 0 2 255 255
255 2 255 0 0 2 255 255 255 3 0 0 0 4 0 255 0 2 0 0 0
```

- The **original** image (**8×8 square**) would need **3 bytes per square** (to include all three RGB values).
- Therefore, the **uncompressed file** for this image is $8 \times 8 \times 3 =$ **192 bytes**.
- The **RLE** code has **92 values**, which means the **compressed file** will be **92 bytes** in size.
- This gives a **file reduction** of about **52%**.
- It should be noted that the file reductions in reality will not be as large as this due to other data which needs to be stored with the compressed file (such as a file header).

1.3.7 File compression applications

General Methods of Compressing Files :

- All the above **file compression techniques** are excellent for very **specific types** of file.
- However, it is also worth considering some **general methods** to **reduce** the **size** of a **file without** the **need** to use **lossy** or **lossless** file compression:

