Data Representation

Number Systems

Binary System

- Base **2** number system like $(1001)_2$
- It has two possible values only (0 and 1)
- 0 represents **OFF**, and 1 represents **ON**
- A point to be noted is that the most left bit is called the **MSB** (Most Significant Bit)

Denary System

- Base 10 number system
- Has values from 0 to 9

Hexadecimal (aka Hex)

- Base 16 number system
- Have values from 0 to 9 followed by A to F
- A represents 10, B represents 11 and so on until 15, which is F

Chart of Binary, Hex, Denary Equivalent

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

Number Conversions

Converting Binary to Denary

- Place the binary value in columns of 2 raised to the power of the number of values from the right starting from 0. e.g. For binary value 11101110, place it in a table like this:

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

  - As can be seen it starts from 1 and then goes to 128 from left to right
- Now values with 1 are to be added together, giving the final answer, as for the example, it is 128 + 64 + 32 + 8 + 4 + 2 = 238
- So, this in (11101110)$_2$ equals to (283)$_{10}$

Converting Denary to Binary

- Take the value and successively divide it by 2, creating a table like follows:

2	**142**		
2	71	Remainder:	0
2	35	Remainder:	1
2	17	Remainder:	1
2	8	Remainder:	1
2	4	Remainder:	0
2	2	Remainder:	0
2	1	Remainder:	0
	0	Remainder:	1

- Note that when the value itself is not divisible by 2, it is divided by the previous value of the current number and 1 is added to the remainder column for that specific number
- When you reach 0, the remainder has to be read from bottom to top giving us the binary value (as in this case, it is 1 0 0 0 1 1 1 0)

Converting Hexadecimal to Binary

- Separate each value from each other and convert them to denary
- Each separate denary value to be converted to binary
- All the binary values to be merged together

  e.g.

```
Hexadecimal: 2    1    F    D
Denary     : 2    1    15   13
Binary       : 0010 0001 1111 1101
Final Answer: 0010000111111101
```

Converting Binary to Hexadecimal

- Divide the binary value into groups of 4 starting from the right. If at the end, the last division is less than 4, add 0s until it reaches 4
- For each group, find the denary value as shown above, and then convert each denary value to its corresponding hexadecimal value (if less than 10, then itself, else, 10 is A, 11 is B, 12 is C, 13 is D, 14 is E and 15 is F).
- After conversion, just put all the hexadecimal values in order to get the final answer

```
Given Value : 1 0 0 0 0 1 1 1 1 1 1 1 0 1
When grouped: 10 0001 1111 1101
After 2 values added to left: 0010 0001 1111 1101
After Conversion to Denary: 2 1 15 13
Denary to Hexadecimal: 21FD
```

Converting Hexadecimal to Denary

- Convert the value to binary as shown above, and then convert the final answer to denary

Converting Denary to Hexadecimal

- Convert the value to binary, and then convert it to hexadecimal as explained above

Binary Calculations

- Binary values are not added the way denary values are added, as when adding 1 and 1, we cannot write two because it doesn't exist in binary.

Points to Note:

- 0 + 0 = 0
- 1 + 0 / 0 + 1 = 1
- 1 + 1 = 0 (1 carry)
- 1 + 1 + 1 = 1 (1 carry)

Overflow

- When adding two values, if the solution exceeds the limit of given values, e.g., the solution has 9 bits, but the question had 8 bits per value, the 9th bit (most left bit) is called overflow.
- This indicates that the memory doesn't have enough space to store the answer to the addition done in the previous part.

Steps to add Two Values (With Example)

- The values we will add are 1 1 0 1 1 1 0 and 1 1 0 1 1 1 1 0
  1. Convert both the bytes into 8 bits (add zero to the left-hand side to match them).

     e.g., 1 1 0 1 1 1 0 would become 0 1 1 0 1 1 1 0
  2. Add the values as follows with the points given above

Carry	**1**	**1**	**1**	**1**	**1**	**1**	**1**		
Byte 1		0	1	1	0	1	1	1	0
Byte 2		1	1	0	1	1	1	1	0
	OVERFLOW								
Solution	1	0	1	0	0	1	1	0	0

Note: We move from RHS(Right Hand Side) to LHS(Left Hand Side), and when adding values, we use the rules given above. If the bit crosses the limit (overflows), we put the value in brackets, denoting it is overflow.

iii. The solution would now be (1) 0 1 0 0 1 1 0 0

Logical Shifts

- The logical shift means moving a binary value to the left or the right
- When doing a logical shift, keep in mind that the bit being emptied is going to become 0

Explanation with Example

- **Shifting 10101010 - 1 place left:**
    1. The furthest bit in the direction to be logically shifted is removed (in this case, one at the LHS is removed) - ==(if it were two places, 2 bits would have been removed)==
    2. Every bit is moved in given places to the given direction (every bit is moved one place to the left in this case, and the leftover bit in the right is marked 0, so **10101010** would become **01010100**)

Two’s Complement (Binary Numbers)

- Two’s complement is a method used to represent negative values in binary. Here, the MSB (Most Significant Bit) is replaced from 128 to -128; thus, the range of values in a two’s complement byte is -128 to 127.

Converting Binary Values to Two’s Complement

- Firstly, write the binary value and locate the first one from the right; e.g., 1101100 would have the first one at the third position from the right.
- Now, switch every value to the left of the first one located above (not switching the one), e.g., the value in our example becomes 0010100, which is the two’s complement of itself.

Converting negative values to two complement

- Find the binary equivalent of the value ignoring the - sign
- Convert the binary value to two’s complement
- Make the MSB 1, if not already

Converting Two’s Complement Value to Denary:

- We do it the same way as a normal value is converted from binary to denary; we only replace 128 with -12,8 e.g., for 1011101,0 we do the:

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

-128 + 32 + 16 + 8 + 2 = -70

Use of the Hexadecimal System

Examples:

- Defining colours in Hypertext Markup Language (HTML)
- Media Access Control (MAC) addresses (a number that uniquely identifies a device on a network)
- Assembly languages and machine code
- Memory Dumps
- Debugging (method to find errors in a program)
- Display error codes (numbers refer to the memory location of the error)
- IP (Internet Protocol) addresses

Memory Dumps

- Hexadecimal is used when developing new software or when trying to trace errors.
- Memory dump is when the memory contents are output to a printer or monitor.

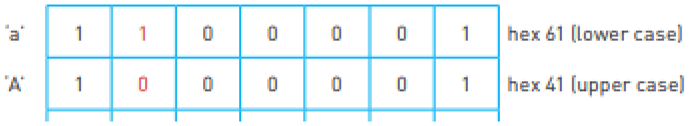
Assembly code and machine code (low-level languages)

- Computer memory is machine code/ assembly code
- Using hexadecimal makes writing code easier, faster, and less error-prone than binary.
- Using machine code (binary) takes a long time to key in values and is prone to errors.

Text, Sound and Images

ASCII

- The standard ASCII code character set consists of 7-bit code that represents the letters, numbers and characters found on a standard keyboard, together with 32 control codes
- Uppercase and lowercase characters have different ASCII values
- Every subsequent value in ASCII is the previous value + 1. e.g. “a” is 97 in ASCII, “b” will be 98 (which is 97 + 1)
- Important ASCII values (in denary) to remember are as follows:
- □ 0 is at 48
- □ A is at 65
- □ a is at 97
- ASCII uses one byte to store the value
- When the ASCII value of a character is converted to binary, it can be seen that the sixth-bit changes from 1 to 0 when going from lowercase to uppercase of a character, and the rest remains the same. e.g.



'a'	1	1	0	0	0	0	1	hex 61 (lower case)
'A'	1	0	0	0	0	0	1	hex 41 (upper case)

Unicode

- ASCII does not contain all of the international languages thus, Unicode is used to solve this problem
- The first 128 values are the same as ASCII.
- Unicode supports up to four bytes per character, storing multiple languages and more data.
- **To represent text in binary, a computer uses a character set, a collection of characters and the corresponding binary codes that represent them.**

Sound

- Sound is analogue, and for it to be converted to digital form, it is sampled
- The sound waves are sampled at regular time intervals where the amplitude is measured. However, it cannot be measured precisely, so approximate values are stored

How is Sound Recorded

- The amplitude of the sound wave is first determined at set time intervals
- The value is converted to digital form
- Each sample of the sound wave is then encoded as a series of binary digits
- A series of readings gives an approximate representation of the sound wave

Sampling Resolution:

- The number of bits per sample is known as the sampling resolution (aka bit depth)
- Increasing the sampling resolution increases the accuracy of the sampled sound as more detail is stored about the amplitude of the sound.
- Increasing the sampling resolution also increases the memory usage of the file as more bits are being used to store the data.

Sampling Rate

- The sampling rate is the number of sound samples taken per second, which is measured in Hertz (Hz)
- A higher sampling rate would allow more accurate sound as fewer estimations will be done between samples.

Images

Bitmap Images

- Bitmap images are made up of pixels
- A bitmap image is stored in a computer as a series of binary numbers

Colour Depth

- The number of bits representing each colour is called the colour depth.
- An 8-bit colour depth means that each pixel can be one of 256 colours (because 2 to the power of 8 = 256)
- A 1-bit colour depth means each pixel can store one colour (because 2 to the power of 1 is 2) - (This is done as the bit can either be 0 or 1, with 0 being white and 1 being black)
- Increasing colour depth increases the size of the file when storing an image.

Image Resolution

- Image resolution refers to the number of pixels that make up an image; for example, an image could contain 4096 × 3072 pixels.
- Photographs with a lower resolution have less detail than those with a higher resolution.
- When a bitmap image is ‘ blurry ‘ or ‘ fizzy ’ due to having a low amount of pixels in it or when zoomed, it is known as being **pixelated.**
- High-resolution images use high amounts of memory as compared to low-resolution ones.

Measurement of the Size of Computer Memories

- A binary digit is referred to as a **BIT**
- 8 bits is a **byte**
- 4 bits is a **nibble**
- Byte is used to measure memory size

IECB System (Most Common)

Name of memory size	No. of Bytes	Equivalent Denary Value
1 kibibyte (1KB)	2^{10}	1 024 bytes
1 mibibyte (1MB)	2^{20}	1 048 576 bytes
1 gibibyte (1GB)	2^{30}	1 073 741 824 bytes
1 tibibyte (1TB)	2^{40}	1 099 511 627 776 bytes
1 pibibyte (1PB)	2^{50}	1 125 899 906 842 624 bytes

Conventional System

Name of memory size	No. of Bytes	Equivalent Denary Value
1 kilobyte (1KB)	10^3	1 000 bytes
1 megabyte (1MB)	10^6	1 000 000 bytes
1 gigabyte (1GB)	10^9	1 000 000 000 bytes
1 terabyte (1TB)	10^{12}	1 000 000 000 000 bytes
1 petabyte (1PB)	10^{15}	1 000 000 000 000 000 bytes

Note:

A tip to remember is “kmgtp” and their power are starting with ^{3} to ^{15} with having each difference of 3 powers.

Calculation of File Size

- The file size of an image is calculated as:

  image resolution (in pixels) × colour depth (in bits)

- The size of a mono sound file is calculated as:

  sample rate (in Hz) × sample resolution (in bits) × length of sample (in seconds). (For a stereo sound file, you would then multiply the result by two.)

File Types

Musical Instrument Digital Format (MIDI)

- Storage of music files
- A communications protocol that allows electronic musical instruments to interact with each other
- Stored as a series of demands but no actual music notes
- Uses 8-bit serial transmission (asynchronous)
- Each MIDI command has a sequence of bytes:
  - The first byte is the status byte – which informs the MIDI device what function to perform
  - Encoded in the status byte is the MIDI channel (operates on 16 different channels)
- Examples of MIDI commands:
  - Note on/off: indicates that a key has been pressed
  - Key pressure: indicates how hard it has been pressed (loudness of music)
- Needs a lot of memory storage

MP3

- Uses technology known as Audio Compression to convert music and other sounds into an MP3 file format
- This compression reduces the normal file size by 90%
  - Done using file compression algorithms which use Perceptual Music Shaping
  - Removes sounds that the human ear cannot hear properly
  - Certain sounds are removed without affecting the quality too much
- CD files are converted using File Compression Software
- Use lossy format as the original file is lost following the compression algorithm

MP4

- This format allows the storage of multimedia files rather than just sound
- Music, videos, photos and animations can be stored
- Videos could be streamed without losing any real discernible quality

Joint Photographic Experts Group (JPEG)

- JPEG is a file format used to reduce photographic file sizes
- Reducing picture resolution is changing the number of pixels per centimetre
- When a photographic file undergoes compression, file size is reduced
- JPEG will reduce the raw bitmap image by a factor between 5 and 15

Lossless and Lossy File Compression

Lossless File Compression

- All the data bits from the original file are reconstructed when the file again is uncompressed.
- Important for files where the loss of data would be disastrous (spreadsheet)
- An algorithm is used to compress data
- No data is lost
- Repeated patterns/text are grouped together in indexes

Run-Length Encoding

- It reduces the size of a string of adjacent, identical data (e.g. repeated colours in an image)
- A repeating string is encoded into two values: the first value represents the number of identical data items (e.g. characters), and the second value

represents the code of the data item (such as ASCII code if it is a keyboard character), e.g. ‘aaaaabbbbccddddd’ becomes “05 97 04 98 02 99 05 100.”

- RLE is only effective where there is a long run of repeated units/bits
- One difficulty is that RLE compression isn't perfect for strings like "cdcdcdcdcd". We use a flag to solve this; e.g. 255 can be made the flag. Now 255 will be put before every repeating value, e.g. our previous example becomes 255 05 97 255 04 98 255 02 99 255 05 100 where 255 now indicates that the next character/set of characters is approaching

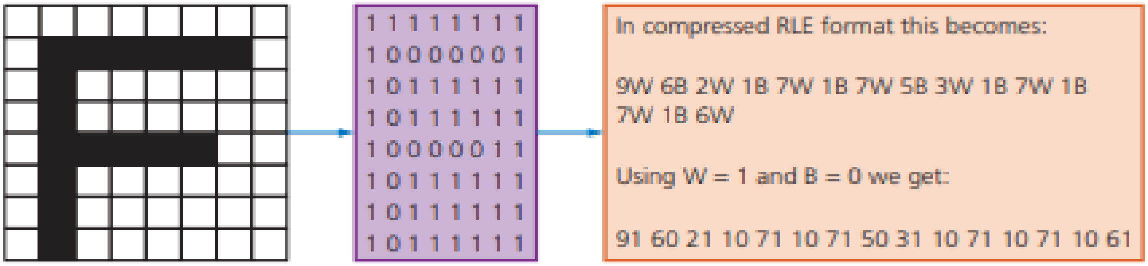
Lossy File Compression

- The file compression algorithm eliminates unnecessary data bits like MP3 and JPEG formats.
- It is impossible to get the original file back once it is compressed
- Reduces file quality
- In this, the image's resolution and colour depth are reduced.

Using RLE with images

Example 1: Black and white image

Figure 1.12 shows 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:



```
1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 1
1 0 1 1 1 1 1 1
1 0 1 1 1 1 1 1
1 0 0 0 0 0 1 1
1 0 1 1 1 1 1 1
1 0 1 1 1 1 1 1
1 0 1 1 1 1 1 1
```

In compressed RLE format this becomes:

9W 6B 2W 1B 7W 1B 7W 5B 3W 1B 7W 1B 7W 1B 6W

Using W = 1 and B = 0 we get:

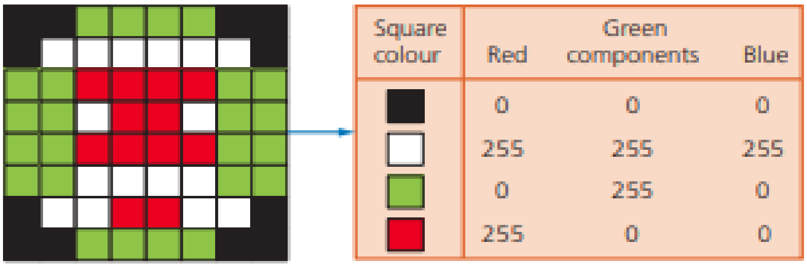
91 60 21 10 71 10 71 50 31 10 71 10 71 10 61

▲ **Figure 1.12** Using RLE with a black and white image

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.

Example 2: Coloured images

Figure 1.13 shows an object in four colours. Each colour is made up of red, green and blue (RGB) according to the code on the right.



Square		Green	
colour	Red	components	Blue
⬛	0	0	0
⬜	255	255	255
🟩	0	255	0
🟥	255	0	0

▲ **Figure 1.13** Using RLE with a coloured image

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.

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 × 8 × 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 (e.g. a file header).