Data Representation

Shayan Naqvi

September 7, 2022

Contents

1 Number systems

1.0.1 Binary

A base 2 number system consisting only of 1s and 0s.

1.0.2 Denary

A base 10 number system consisting of ten separate digits (0-9).

1.0.3 Hexadecimal

A base 16 number system consisting of digits 0-9 and letters A-F.

1. Uses of hexadecimal
   (a) Error codes/memory dumps
      - Error codes are often shown as hexadecimal values. These numbers refer to the memory location of the error. The numbers need to be interpreted correctly in order to fix the error.
   (b) MAC addresses
      - MAC stands for Media Access Control. It is a unique hexadecimal number which identifies a device on a network. The MAC address refers to the network interface card (NIC) of the device. This address is rarely changed so that a particular device may be identified regardless of where it is.
      - The format of a MAC address is usually as follows: $NN : NN : NN : DD : DD : DD$ or $NN - NN - NN - DD - DD - DD$.
        – The first half of the address (NN:NN:NN) identifies the manufacturer of the device.
        – The second half of the address (DD:DD:DD) is the serial number of the device. This part of the address is unique to each device.
   (c) IP addresses
      - Each device connected to a network is given an address known as an IP (Internet Protocol) address. An IP (version 6) address is a 128-bit number broken down into 16-bit chunks, represented by a hexadecimal number. E.g.,

- a8fb:7a88:fff0:0fff:3d21:2085:66fb:f0fa

(d) Colour codes (HEX)

- The HEX colour system is a method of representing specific colours through hexadecimal values. Their structure is as follows:
  - #FF 00 00 represents red.
  - #00 FF 00 represents green.
  - #00 00 FF represents blue.
- These values are then mixed together to form a specific colour. For e.g.,
  - #FF 00 FF represents fuchsia.
  - #FF 80 00 represents orange.
  - #B1 89 04 represents tan.
  - #FF FF FF represents white.
  - #00 00 00 represents black.
  - #73 73 73 represents grey.
- There are 256 variants for red, green and blue, making a total of about 16.7 million colours.

1.1 Conversions

1.1.1 Binary –> Denary

The binary digits are placed under the corresponding denary headings. Any heading with a 1 below it is added together to form a complete denary number. E.g.,

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

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

1.1.2 Denary –> Binary

Find the appropriate denary headings that add up to the denary number to be converted. Place a 1 below each of the appropriate headings and group them up to form a converted binary digit. E.g., 142 –> Binary

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

$128 + 8 + 4 + 2 + 1 = 142\ (10001110)_2$

1.1.3 Binary <-> Hexadecimal

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

1. Split the binary digit into 4 parts. If the binary digit can't be evenly divided, add zeroes as needed. E.g.,
   - 101101011 –> 1011 0101 1 –> 1011 0101 1000
2. Refer to the table above and make the appropriate conversions. E.g.,
   - 1011 0101 1000 –> B 5 8 –> $(B58)_{16}$

1.1.4 Hexadecimal –> Denary

The headings of hexadecimal digits need to be used. They are multiples of 16: 1, 16, 256, 4096, etc. Multiply each hexadecimal digit with the corresponding heading. E.g.,

- 45A –> Denary
- (4 * 256) + (5 * 16) + (A(10) * 1) = 1024 + 80 + 10 = $(1114)_{10}$

1.2 Binary addition

Note the following when adding two binary digits:

Addition	Carry	Sum
$0 + 0$	0	0
$0 + 1$	0	1
$1 + 0$	0	1
$1 + 1$	1	0

Note the following when adding three binary digits:

Addition	Carry	Sum
$0 + 0 + 0$	0	0
$0 + 0 + 1$	0	1
$0 + 1 + 0$	0	1
$0 + 1 + 1$	1	0
$1 + 0 + 0$	0	1
$1 + 0 + 1$	1	0
$1 + 1 + 0$	1	0
$1 + 1 + 1$	1	1

Shorthand:

- $0 + 0 = 0$
- $0 + 1$ (in any way) $= 1$ as sum, 0 carried
- $1 + 1$ (in any way) $= 0$ as sum, 1 carried
- $1 + 1 + 1 = 1$ as sum and carried

1.2.1 Examples

1. 00100111 + 01001010

$$\begin{array}{cccccccccc} & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ + & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ = & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}$$

2. 01111110 + 00111110

$$\begin{array}{cccccccccc} & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ + & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ = & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{array}$$

1.2.2 Overflow errors

When the sum of two binary numbers exceeds 8 bits (i.e. 9 bits) the computer returns an overflow error. This means the number is too large to store in an 8-bit register. Therefore the solution would be to use a register with a larger capacity, i.e. a 16, 32, or 64-bit register.

1.3 Multiplication/division with binary digits

A logical shift on a binary digit refers to shifting binary digits either to the left or right.

- A shift to the right is equivalent to dividing the binary number by 2.
- A shift to the left is equivalent to multiplying the binary number by 2.

Any vacant positions are replaced with a zero.

1.3.1 Examples

1. The denary number 21 is 00010101 in binary. Putting this into an 8-bit register:

   $$0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1$$

   Shifting the bits once place to the left:

   $$0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0$$

   The value of this register is now 21 * 2 = 42. This can be checked by converting this new binary number into denary.

2. The denary number 21 is 00010101 in binary. Putting this into an 8-bit register:

   $$0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1$$

   Shifting the bits two places left:

   $$0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0$$

   The binary number 1010100 is 84 in denary, which is $21 * 2^2$. Now let us shift the original number 4 places left:

0 1 0 1 0 0 0 0

The left-most 1-bit has been lost. The result of 0101000 is 80, which is incorrect. This raises an error because the maximum number of left shifts possible in this register have been exceeded.

1.4 Two's complement

This is a method of binary <-> denary conversion that allows us to represent negative numbers.

1. Converting -17 into binary

   (a) Represent -17 in positive binary

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

   (b) Invert all binary digits (1's complement)

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

   (c) Add 1 to the inverted binary digits (2's complement)

	1	1	1	0	1	1	1	0
+	0	0	0	0	0	0	0	1
=	1	1	1	0	1	1	1	1

   (d) Verify by converting the new binary number into denary, with a negative 128 heading.

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

   -128 + 64 + 32 + 8 + 4 + 2 + 1 = -17

2. Converting -44 into binary

   (a) Represent -44 in positive binary

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

   (b) Invert all binary digits

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

(c) Add 1 to the inverted digits

	1	1	0	1	0	0	1	1
+	0	0	0	0	0	0	0	1
=	1	1	0	1	0	1	0	0

(d) Verify

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

-128 + 64 + 16 + 4 = -44

2 Text, sound and images

2.1 Character sets (ASCII + Unicode)

2.1.1 ASCII

ASCII stands for the **American Standard Code for Information Interchange**. It is a character set consisting of 7-bit codes that represent the letters, numbers and characters found on a standarf keyboard. The codes range from 0-127 (denary)/00-7F (Hexadecimal).

1. Extended ASCII Extended ASCII uses 8-bit (0-255 in denary/00-FF in hexadecimal) that allow for another 128 codes that represent non-English letters and some graphical characters.

2. Disadvantages of ASCII

   - Does not represent non-latin character scripts (i.e. Chinese or Arabic).
   - Limited set of characters.

2.1.2 Unicode

Unicode is a character set, similar in concept to ASCII, that aims to create a universally standard set of characters, inclusive of different scripts and languages (i.e. non-latin scripts).

2.2 Sound

By nature, sound is analogue data. Computers can process only digital data, therefore, in order for a computer to process sound, it needs to be sampled

(measuring the amplitude of the sound wave) via an ADC, an **Analogue to Digital Converter.**

2.2.1 Converting sound to digital data

1. The amplitude of the sound wave is measured at regular time intervals (2.2.3). Since the sound cannot be measured precisely, approximate values are stored. These values can be represented in binary with an appropriate number of bits.

2. Sampling yields an approximate digital representation of the sound wave.

3. Each sample of the sound wave is subsequently encoded as a series of binary digits.

2.2.2 Sampling resolution (bit depth)

The number of bits used to measure a sample affects how accurate the final representation of the sound is. If a sound is sampled using 10 bits, the accuracy of the representation would be lacking in comparison to a sound sampled with a higher bit depth, such as 127.

2.2.3 Sampling rate

The number of sound samples taken per second is known as the sampling rate. This rate is measured in Hertz (Hz). In this context, 1 Hz refers to 1 sample per second. 2 Hz would refer to 2 samples per second.

2.2.4 Quality

Higher sampling rates or larger sampling resolution yeilds a more faithful representation of the original sound. However, the higher the sampling rates and resolution, the higher the file size.

Benefits	Drawbacks
Larger dynamic range	Larger file size
Better sound quality	Takes longer to transmit/download audio files
Less sound distortion	Requires greater processing power

2.3 Bitmap images

A bitmap image is made up of pixels. Each pixel can be represented as a binary number, so an image is sotred as a series of binary numbers.

- A BW image requires 1 bit per pixel, meaning any pixel can be either a 1 or 0 to represent black or white.
- If each pixel is represented by 2 bits, then there are 4 possible colours (00, 01, 10, 11).
- If each pixel is represented by 4 bits, then there are 16 possible colours (0000, 0001, 0010, 0011, 0100, etc. . .).

2.3.1 Colour depth

The number of bits used to represent each pixel is called colour depth. 8 bits of colour depth mean that each pixel can be one of 256 colours (2^8). Most modern computers handle 24 bit colour depth, meaning over 16 million colours can be represented.

2.3.2 Image resolution

Image resolution refers to the number of pixels that make up a photograph. More pixels = a higher image resolution. A greater image resolution, however, comes with a larger file size (and all subsequent large file-size drawbacks).

3 Data storage + file compression