TEMASEK JUNIOR COLLEGE

Temasek Junior College

2023 JC2 H2 Computing

Data Representation 2 - Number Representation 2

Objectives

- Understand that values can be represented in different number bases: denary, binary and hexadecimal
- Represent data in binary and hexadecimal forms. (3.1.1)
- Write programs to perform the conversion of positive integers between different number bases: denary, binary and hexadecimal forms; and display results (3.1.2)

1 Numbers and Quantities

There are several different types of numbers within the denary system. Examples of these are provided in table below.

Type of Number	Examples	Remarks
Integer	3 or 47	A whole number used for counting
Signed integer	–3 or 47	Includes the arithmetic sign of the number Either positive or negative Positive integers have an implied '+' sign
Fraction	2/3 or 52/17	Rarely used in computer science
A number with a whole number part and a fractional part (float)	–37.85 or 2.83	Positive values have an implied '+' sign
A number expressed in exponential notation	-3.6×10^{8} or 4.2×10^{-9}	Values can be positive or negative and the exponent can be positive or negative

We will focus on how large values are represented. If we have a quantity that includes units of measurement, it can be written in three different ways. For example, a distance could be written in any one of these three ways:

- 23 567 m
- 23.567×10^{3} m
- 23.567 km

The second example has used an exponential notation to define the magnitude of the value. The third example has added a prefix to the unit to define its magnitude. We read this as 23.567 kilometres.

The 'kilo' is an example of a **decimal prefix**. A **decimal prefix** is used to define the magnitude of a value

There are four decimal prefixes commonly used for large numbers. These are shown below.

Decimal Prefix Name	Symbol Used	Factor Applied to the Value
kilo	k	10^3
mega	M	10^6
giga	G	10^9
tera	T	10^{12}

Unfortunately, for a long time, the computing world used these prefix names but with a slightly different definition The value for 2^{10} is 1024. As this is close to 1000, computer scientists decided that they could use the kilo prefix to represent 1024. For example, if a computer system had the following values quoted for the processor speed and the size of the memory and of the hard disk:

Processor Speed	1.6 GHz
Size of RAM	8 GB
Size of Hard Disk	400 GB

The prefix G would represent 10^9 for the processor speed but would almost certainly represent $1024 \times 1024 \times 1024$ for the other two values.

This unsatisfactory situation has now been resolved by the definition of a new set of names which can be used to define a **binary prefix**. A **binary prefix** is also used to define the magnitude of value,

A selection of these is shown in the table below.

Binary Prefix Name	Symbol Used	Factor Applied to the Value
kibi	Ki	2^{10}
mebi	Mi	2^{20}
gibi	Gi	2^{30}
tebi	Ti	2^{40}

When a number or a quantity is presented for a person to read it is best presented with either one denary digit or two denary digits before the decimal point. If a calculation has been carried out, the initial result found may not match this requirement. A conversion of the presented value will be needed by choosing a sensible magnitude factor. For example, consider the following two answers calculated for the size of a file:

(a) 34 560 bytes

Here, a conversion to kibibytes would be sensible using the calculation:

$$34560 \text{ bytes} = \frac{34\,560}{1024} \text{ KiB} = 33.75 \text{ KiB}$$

(b) 3 456 000 bytes

Here, a conversion to mebibytes would be sensible using the calculation:

$$3\ 456\ 000 \text{ bytes} = \frac{3\ 456\ 000}{1024^2} \text{ MiB} = 3.296 \text{ MiB}$$

If a calculation is to be performed with values quoted with different magnitude factors, there must first be conversions to ensure all values have the same magnitude factor. For example, if you needed to know how many files of size 2.4 MiB could be stored on a 4 GiB memory stick, there should be a conversion of the GiB value to the corresponding MiB value.

The calculation would be:

$$\frac{(4 \times 1024) \text{ MiB}}{2.4 \text{ MiB}} = 1706$$

2 Binary Arithmetic

Before considering the addition of binary numbers, it is useful to recall how we add two denary numbers. Two rules apply. The first rule is that the process is carried out starting with addition of the two least significant digits and then working right to left. The second rule is that if an addition produces a value greater than 9, there is a carry of 1. For example, in the addition of 48 to 54, the first step is adding 8 to 4 to get 2 with a carry of 1. Then 5 is added to 4 plus the carried 1 to give 0 with carry 1. The rules produce 102 for the sum which is the correct answer.

For binary addition, starting at the least significant position still applies. The rules for the addition of binary digits are:

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 1 = 0 with a carry of 1
- 1 +1 + 0 = 0 with a carry of 1
- 1 + 1 + 1 = 1 with a carry of 1

The last two rules are used when a caried 1 is included in the addition of two digits.

As an example, the addition of the binary equivalent of denary 14 to the binary equivalent of denary 11 can be examined.

$$\begin{array}{rcccc} & 1 & 0 & 1 & 1 \\ + & 1 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 0 & 1 \end{array}$$

The steps followed from right to left are:

- 1 + 0 = 1 with no carry
- 1 + 1 = 0 with carry 1
- 0 + 1 + carried 1 = 0 with carry 1
- 1 + 1 + carried 1 = 1 with carry 1

The rules have correctly produced the 5-bit answer which is the binary equivalent of 25. In a paper exercise like this, these rules for addition will always produce the correct answer.

Again for subtraction, we can first consider how this is done for denary numbers. As for addition the process starts with the least significant digits and proceeds right to left. The special feature of subtraction is the "borrowing" of a 1 from the next position when a subtracting digit is larger than the digit it is being subtracted from.

For example, in subtracting 48 from 64 the first step is to note that 8 is larger than 4. Therefore 1 has to be borrowed as 10. The 10 added to 4 gives 14 and 8 subtracted from this gives 6. When we proceed to the next digit subtraction, we first have to reduce the 6 to 5 because of the borrow So we have subtraction of 4 from 5 leaving 1. The answer for the subtraction is 16.

For binary subtraction, starting at the least significant position still applies. The rules for the subtraction of binary digits are:

- 0 – 0 = 0
- 0 – 1 = 1 after a borrow
- 1 – 0 = 1
- 1 – 1 = 0

As an example, the subtraction of the binary equivalent of denary 11 from the binary equivalent of denary 14 can be examined.

$$\begin{array}{rcccc} & 1 & 1 & 1 & 0 \\ - & 1 & 0 & 1 & 1 \\ \hline & 0 & 0 & 1 & 1 \end{array}$$

The steps followed from right to left are:

- 1 is larger than 0 so 1 is borrowed giving subtraction of 1 from 10 leaving 1
- Because of the borrow, the 1 is reduced to 0 so that 1 is to be subtracted from 0. This requires a further borrow giving subtraction of 1 from 10 leaving 1
- Because of the borrow the 1 is reduced to 0, leaving subtraction of 0 from 0
- 1 – 1 gives 0

The answer is the binary value for denary 3.

2's complement
↳ Repeat -ve in binary.

TEMASEK JUNIOR COLLEGE

Temasek Junior College

2023 JC2 H2 Computing

Data Representation 3 - Character Encoding

Syllabus Objectives

Aim: Understand the use of ASCII code and Unicode to represent characters

3.2.1 Give examples of where or how Unicode is used.
3.2.2 Use ASCII code in programs

1 Internal Coding of Text

- To store text in a computer, we need a coding scheme that provides a unique binary code for each distinct individual component item of the text.
- Such a coding scheme is referred to as a **character coding scheme**.
- There have been many different examples of character coding schemes throughout the history of computing.

1.1 American Standard Code for Information Interchange (ASCII) Code

- The ASCII coding scheme is the character coding scheme which has been used for the longest time.
- The **7-bit** version of the code (often referred to as US ASCII) was standardised many years ago by the American National Standards Institute (ANSI).
- The codes are always presented in a table. An edited version of a typical table is shown below.

Binary Code	Hexadecimal equivalent	Decimal	Character	Description
0000 0000	00	0	NUL	Null character
0000 0001	01	1	SOH	Start of heading
0010 0000	20	32		Space
0010 0001	23	35	#	Number
0011 0000	30	48	0	Zero
0011 0001	31	49	1	One
0100 0001	41	65	A	Uppercase A
0100 0010	42	66	B	Uppercase B
0110 0001	61	97	a	Lowercase a
0110 0010	62	98	b	Lowercase b

Figure 1: Some examples of ASCII codes stored in 1 byte with the remaining, most significant bit set to 0

- A full table would show the 2^7 **= 128** different codes available for a 7-bit code.

- It is important to know following key facts about the ASCII coding scheme:
  - ✓ A limited number of the codes represent non-printing or control characters, these were introduced to assist in data transmission or for data handling at a computer terminal.
  - ✓ The majority of the codes are for characters that would be found in an English text and which are available on a standard keyboard.
  - ✓ These include uppercase and lowercase letters, punctuation marks, denary digits and arithmetic symbols.
  - ✓ The codes for numbers and for letters are in sequence so that, for example, if 1 is added to the code for seven, the code for eight is produced.
  - ✓ The codes for the uppercase letters differ from the codes for the corresponding lowercase letters only in the value of bit 5, which allows a simple conversion from uppercase to lowercase or the reverse. (Don't forget that the least significant bit is bit 0)

- Note that this coding for numbers is exclusively for use in the context of stored, displayed or printed text.
- All of the other coding schemes for numbers are for internal use in a computer system and would not be used in text.
- Although a standard version of ASCII has been created, different versions of 7-bit ASCII are tailored to different software or different countries.
- Mostly, the coding for the printable characters has remained unchanged.
- A notable exception was the use in some countries of the code 00100001 to represent a currency ($) symbol rather than #.
- However, because most of the control characters became of limited use, there were versions of ASCII that used these codes to produce small graphic icons. For example, the code 00000001 would show ☺.

1.2 Extended ASCII

- Extended ASCII is a code that uses all 8 bits in a byte.
- The most used standardized version is often referred to as ISO Latin-1.
- The name Latin-1 reflects the fact that many of the new character definitions are for accented or otherwise modified alphabetic characters found in European languages, for example Ç or ü.
- As with the 7-bit code, there are many variations of the standard code.

1.3 Unicode

- Although ASCII codes are widely used, they do not cover all the characters needed for some uses. For this reason, new coding schemes have been developed and continue to be developed further.

- The discussion here describes one of the Unicode schemes, It should be noted that Unicode codes have been developed in tandem with the Universal Character Set (UCS) scheme, standardised as ISO/IEC 10646.

- The aim of Unicode is to be able to represent any possible text in code form. In particular, this includes all languages in the world.

- The most popular version of Unicode which is discussed here is named UTF-8. The inclusion of 8 in the name indicates that this version of the standard includes codes defined by one byte in addition to codes using two, three or four bytes.

- The table below shows the structure of codes.

One-byte	0???????			
Two-bytes	110?????	10??????		
Three-bytes	1110????	10??????	10??????	
Four-bytes	11110???	10??????	10??????	10??????

Figure 2: Byte formats for Unicode UTF-8

- The one-byte code reproduces 7-bit ASCII. Because the byte has the most significant bit set to 0, there can be no confusion with any byte which is part of a multiple byte code.

- Note that for the two-byte, three-byte and four-byte representations all continuing bytes have the two most significant bits set to 10. Whenever a byte has the most significant bits set to 11, there will be at least one continuation byte following.

- The number of codes available is determined by the number of bits that are not predefined by the format. For example, there are eleven bits free to identify codes in the 2-byte format. This allows $2^{11} = 2048$ different codes.

- Unicode has its own special terminology and symbolism. A character code is referred to as a 'code point'. In any documentation, a code point is identified by U+ followed by a 4-digit hexadecimal number.

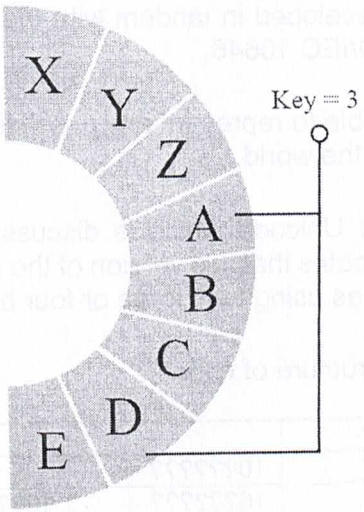
  - ✓ The code points U+0000 to U+00FF define characters which are a duplicate of those in the standard Latin-1 scheme.

  - ✓ The binary codes corresponding to U+0000 to U+007F use one byte only and range from 00000000 through 01111111.

  - ✓ Then the first binary codes for U+0080 to U+00FF require two bytes and range from 11000000 for the first byte followed by 10000000 for the second byte through to 11000001 followed by 10111111.

1.4 Caesar Cipher – An interesting application of character encoding

A Caesar cipher is a simple shift cipher where each letter is shifted a certain number of places in the alphabet according to a key. With 3 as the key, the letter A shifts three places and becomes D, the letter B becomes E, and so on.



If you want to write code to create a Caesar cipher, it is easiest to:

1. Convert the character to its corresponding ASCII code
2. Add the value of the key to the ASCII code number
3. Convert the code back into a character

A partial pseudocode is provided below:

```
letter = "A"
key = 3
character_code = ASC(letter)
shifted_code = character_code + key
encrypted_letter = CHR(shifted_code)
PRINT(encrypted_letter)
```

While this piece of code illustrates how to convert between ASCII code and characters, the code isn't enough to create a Caesar cipher: the program also has to take into account letters 'wrapping around' from the end of the alphabet to the start i.e. X → A, Y → B and Z → C.

Are you able to write the full code for the Caesar Cipher?

Once you are able to do so, encode your own name.