CAMBRIDGE

Brighter Thinking

A/AS Level Computer Science for WJEC/Eduqas

Student Book

Mark Thomas with Alistair Surrall and Adam Hamflett

Contents

Introduction iv

Component 1:

1 Data structures 1

2 Logical operations 15

3 Algorithms and programs 28

4 Principles of programming 91

5 System analysis 110

6 System design 143

7 Software engineering 150

8 Program construction 158

9 Economic, moral, legal, ethical and cultural issues relating to computer science 175

Component 2:

10 Hardware and communication 199

11 Data transmission 230

12 Data representation and data types 238

13 Organisation and structure of data 259

14 Databases and distributed systems 272

15 The operating system 291

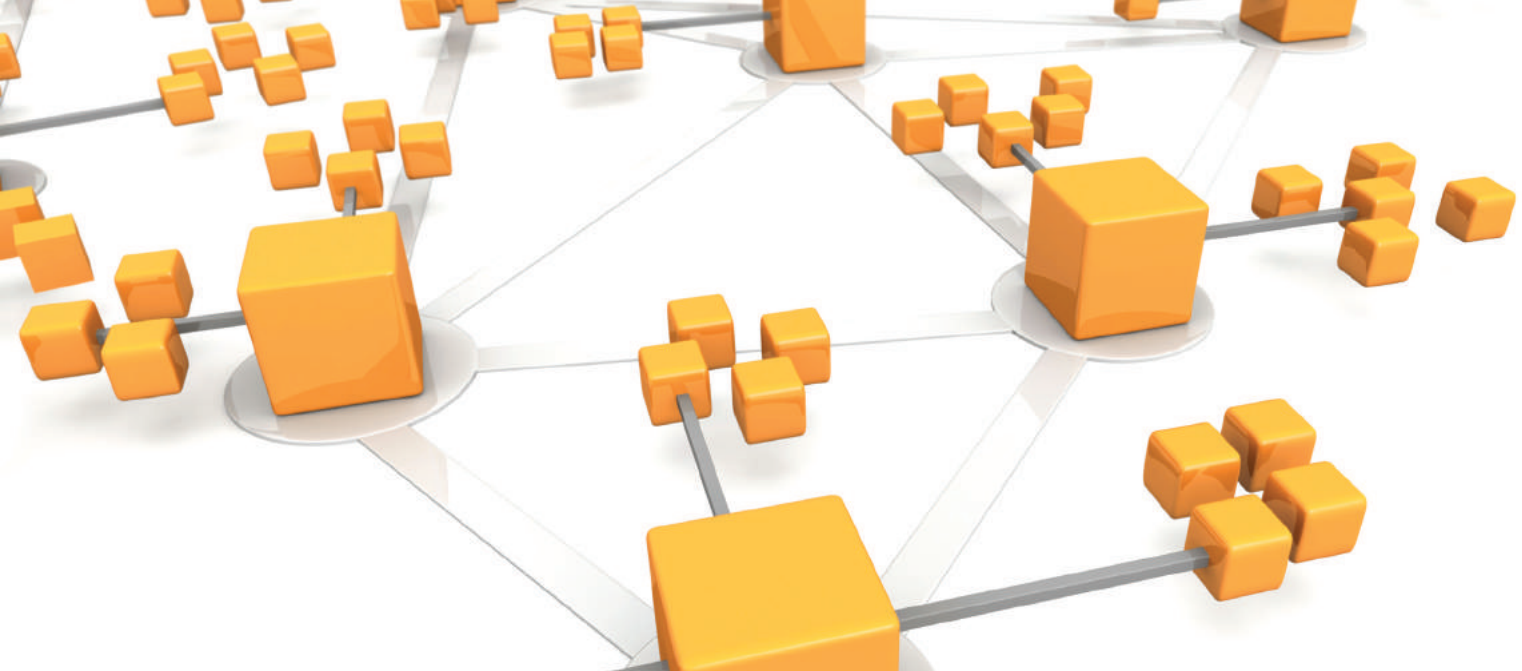
16 The need for different types of software systems and their attributes 316

17 Data security and integrity processes 332

Glossary 344

Index 347

Acknowledgements 360



Chapter 1
Data structures

Learning objectives

- A • Describe, interpret and manipulate data structures including one-dimensional arrays, two-dimensional arrays, three-dimensional arrays, stacks, queues, trees, linked lists, hash tables and records.
- A • Represent the operation of stacks and queues using pointers and arrays.
- • Represent the operation of linked lists and trees using pointers and arrays.
- • Select, identify and justify appropriate data structures for given situations.

Introduction

A data structure is a collection of related data items held in a computer's memory. There are hundreds of different data structures available to programmers but at this level you need only understand a few of the most common ones:

- one-dimensional arrays
- two-dimensional arrays
- A • three-dimensional arrays
- records
- stacks
- queues
- binary trees
- linked lists
- hash tables.

Data structures usually fall into one of two categories: static and dynamic. In static data structures, the size of the data structure is set at the beginning of the program and cannot be changed. This is very limiting (and inefficient) because it means that you have to know

in advance how large your data structure needs to be. However, static data structures are easy to program and you always know how much space they will take up. In contrast, dynamic data structures do not have a limited size and can expand or shrink as needed. They also tend to be more complex to implement.

It is also very common for more complex data structures to be made of simpler ones. For example, it is normal practice to implement binary trees by using a linked list, or to create a stack by using an array. It is important to remember, however, that data structures can be implemented in different ways. The idea behind a data structure (the abstract data type) is the important aspect to learn. How the data structure is implemented is up to the person coding the implementation. There are many ways of implementing the same data structure.

You will need to perform various operations on your data, such as inserting, sorting or searching data. The efficiency of such operations varies depending on the data structure in use. Generally speaking, you want to use the most efficient data structure possible. This is particularly important when you have large data sets. Your choice of data structure can have a big impact on the run time of your program. You can find out about the efficiency ratings (using Big O notation) of the data structures you need in the 'Further reading' section at the end of this chapter.

So why do we use data structures?

Data structures are a convenient way of organising data relating to a real problem.

One-dimensional arrays

Arrays are the simplest of all data structures. An array is a collection of variables of the same data type grouped under a single identifier. Each variable in the array is called an element and is accessed using its position in the array as an index.

Tip

NB: These examples use Python, which doesn't support arrays but uses lists instead. The examples here demonstrate the same properties as arrays.

This code shows how to declare an array (holding five names) using Python:

```
Names = ['Sam', 'Lucy', 'James', 'Jack', 'Jane']
```

Figure 1.1 shows how this is held in the computer's memory.

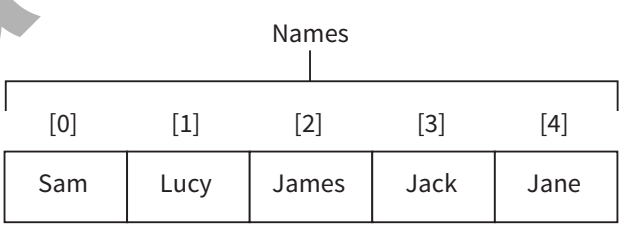


Figure 1.1: A one-dimensional array.

Changing or accessing the data in an array is simple. You simply refer to the element number (index) of the piece of data you want. For example, `print(Names[1])` would print the word 'Lucy'. `Names[3]='Sarah'` would change the value of element 3 from Jack to Sarah.

It is important to recognise that the element numbers (indices) of arrays usually start at 0.

Arrays are normally static data structures, so once they have been declared they can't grow beyond their initial size. As a result, the array in our example can never contain more than five items. If you were to write `Names[6]='David'` you would get an error message.

Two-dimensional and three-dimensional arrays

You can also create arrays with more than a single dimension. In theory, you can have arrays with a large number of dimensions, but this becomes very hard to visualise and at A-Level you only need to go up to three (at AS-Level you only need to go up to two).

This code shows how to declare a two-dimensional (2D) array using Python:

```
Names = [['Sam', 'Lucy', 'James', 'Jack', 'Jane'], ['Peter',
'Sarah', 'Adam', 'Karen', 'Verity'], ['Emily', 'Edward',
'Dominic', 'Justin', 'Jake']]
```

The implementation of multidimensional arrays differs from language to language. In Python, you simple create an array and then place another array inside each element of it.

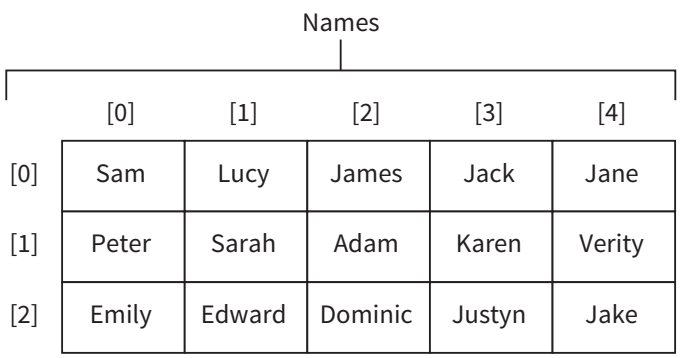


Figure 1.2: How a 2D array is held in the computer's memory.

Changing or accessing the data in a 2D array is the same as in a 1D array. You simply refer to the element number (index) of the piece of data you want. For example, referring to Figure 1.2, `print(Names[2][1])` would print the word 'Edward'. `Names[0][3]='Alistair'` would change the value of element (0, 3) from Jack to Alistair.

Notice that the element index operates in the same way as the coordinates on a map or graph.

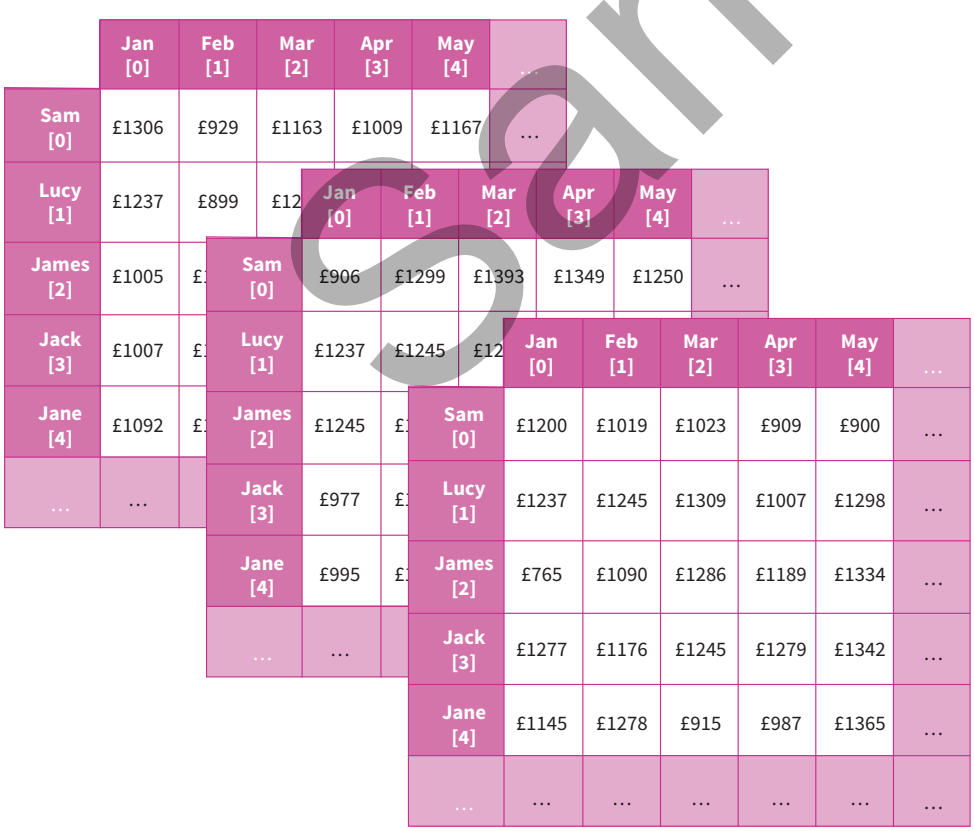


Figure 1.3: The sales figures for different salespeople in the first five months of the year over a number of years. These data can be stored in a 3D array.

The disadvantage of using 3D arrays is that they are more complex to program.

3D arrays can be thought of as an array of 2D arrays (see Figure 1.3). They can be defined in a similar manner as 2D. Consider a set of computing classes, as shown in the tables below.

Name	Tutor group
Sally	7TU
Bert	7AB

Name	Tutor group
Martin	7AB
Susan	7PB

A single row in each table can be represented by a 1D array, for example `['Sally', '7TU']`. The entire table can be represented by a 2D array. As we have two classes worth of data we can move to the third dimen.sion by expressing the final array as being the set of classes.

```
computingClasses = [[['Sally', '7TU'], ['Bert', '7AB']],
[['Martin', '7AB'], ['Susan', '7PB']]]
```

The first class will have an index of 0 while the second class will have an index of 1. To access Susan's tutor group we could use the notation `computingClasses[1][1][1]`.

To display the contents of both classes, a nested loop will be needed. The code example below uses three loops, one for each dimension.

```
for classes in computingClasses:
  for student in classes:
    print ('name tutor group')
    for data in student:
      print (data)
    print('')
```

Searching an array

Arrays are normally immutable, so you can't increase their size once they've been created. However, you can search through to find any empty spaces and put your new data in there. This has the added bonus that you don't waste memory reserving space for empty array elements.

This Python function takes a piece of data and searches through the array until an empty spot is found. If an empty element is found, the data is inserted and the function returns true. If there are no empty elements left, the function returns false.

```
def add_data (data):
  for x in range (0,len(array)):
    if array[x] == "":
      array[x] = data
      return True

  return False
```

The disadvantage of using 3D arrays is that they are more complex to program.

3D arrays can be thought of as an array of 2D arrays (see Figure 1.3). They can be defined in a similar manner as 2D. Consider a set of computing classes, as shown in the tables below.

Name	Tutor group
Sally	7TU
Bert	7AB

Name	Tutor group
Martin	7AB
Susan	7PB

A single row in each table can be represented by a 1D array, for example `['Sally', '7TU']`. The entire table can be represented by a 2D array. As we have two classes worth of data we can move to the third dimen.sion by expressing the final array as being the set of classes.

```
computingClasses = [[['Sally', '7TU'], ['Bert', '7AB']],
[['Martin', '7AB'], ['Susan', '7PB']]]
```

The first class will have an index of 0 while the second class will have an index of 1. To access Susan's tutor group we could use the notation `computingClasses[1][1][1]`.

To display the contents of both classes, a nested loop will be needed. The code example below uses three loops, one for each dimension.

```
for classes in computingClasses:
  for student in classes:
    print ('name tutor group')
    for data in student:
      print (data)
    print('')
```

Searching an array

Arrays are normally immutable, so you can't increase their size once they've been created. However, you can search through to find any empty spaces and put your new data in there. This has the added bonus that you don't waste memory reserving space for empty array elements.

This Python function takes a piece of data and searches through the array until an empty spot is found. If an empty element is found, the data is inserted and the function returns true. If there are no empty elements left, the function returns false.

```
def add_data (data):
  for x in range (0,len(array)):
    if array[x] == "":
      array[x] = data
      return True

  return False
```

As you might expect, it is also impossible to remove elements from an array once it has been created. You can, however, set elements to 'empty' or 'NULL' to signify that they are available to store data in:

Names[1] = ""

A Computing in context: Big O notation

An algorithm's efficiency is usually described using Big O notation. Big O notation gives an idea of how well an algorithm scales. For example, how long it will take to search an array of 10 items compared to an array of 10 000 items. Big O notation generally gives the worst case scenario, that is, the maximum possible number of steps required to complete the algorithm.

If an algorithm always takes the same time to complete, regardless of the size of the data set (n), it is said to be **O(1)**. For example, reading the first element of an array will be **O(1)** because it makes no difference if the array is 10 items long or 100 000 items long. The algorithm will take the same number of steps to complete.

If the length of time taken to complete an algorithm is directly proportional to the size of the data set (n), it is said to be **O(n)**. For example, carrying out a linear search on an array is **O(n)** because the number of checks carried out is directly proportional to the number of items in the array.

If the time to complete an algorithm is proportional to the square of the amount of data (n), it is said to be **O(n^2)**. This is very common in algorithms that have nested loops, such as the bubble sort algorithm.

All common algorithms and data structures will have a generally accepted Big O notation efficiency rating. It is up to the programmer to make sure that they choose the most efficient data structure for the job.

See Chapter 3 for more details on Big O notation.

Activity 1.1

Make a list of some common searching and sorting algorithms. Use Big O notation to find out which is the most efficient.

Records

A record is a set of data items all related to a single entity. Unlike an array, a record may contain data items of more than one data type. The following data structure may be used to store the details of an organisation's members:

Field Name	Data Type
Member ID	Integer
First name	String
Surname	String
Gender	Character
DOB	Date

A Stacks

Stacks are Last-In, First-Out (LIFO) data structures. This means that the most recent piece of data to be placed onto the stack is the first piece to be taken off it (Figure 1.4).

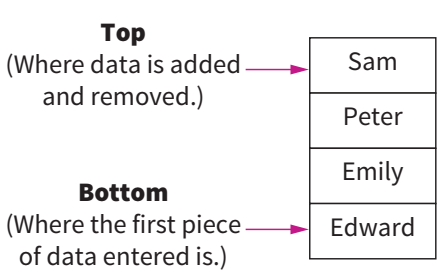


Figure 1.4: Stack.

Think of a stack like a pile of pancakes. The pancake on the top is the last one to be added to the pile and will be the first one to be taken off.

Adding data to the stack is called pushing and taking data off the stack is called popping.

Stacks are easily implemented using an array or a linked list to hold the data, and a variable to point to the top of the stack.

The code below shows how Python could be used to implement a simple stack using an array. Calling the pop function will remove a piece of data from the stack; calling the push function will add some data to the top of the stack.

```
myStack = ['Edward', 'Emily', 'Peter', 'Sam', 'Empty', 'Empty']

def pop(pointerTop): #LIFO, so removes the value from the top
of the stack and returns it
  value = myStack[pointerTop]
  myStack[pointerTop] = 'Empty'
  pointerTop = pointerTop - 1
  return value, pointerTop

def push (data, pointerTop): #add a new piece of data to the
top of the stack
  pointerTop = pointerTop + 1
  myStack[pointerTop] = data
  return pointerTop
  print (myStack)

pointerTop = 3 # points to the last valid element- the top of
the stack
  value, pointerTop = pop(pointerTop)
  print("Removed " + value)
  print (myStack)
  pointerTop = push ('Alex', pointerTop)
  print (myStack)
```

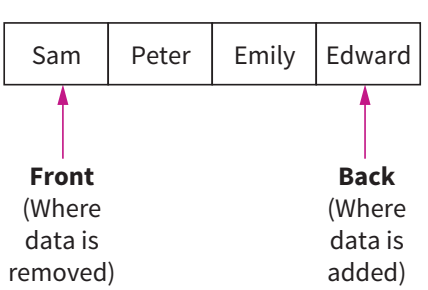


Figure 1.5: A queue data structure.

Queues

Queues are First-In, First-Out (FIFO) data structures. This means that the most recent piece of data to be placed in the queue is the last piece taken out.

Think of queues like the ones you find at supermarket checkouts. The first person in a queue is the first person to leave it (Figure 1.5).

Queues are easily implemented using the simple Python code shown here. It contains an array and two variables; one variable points to the first item in the queue, the other points to the next free space:

```
myQueue = ['Sam', 'Peter', 'Emily', 'Edward', 'Empty', 'Empty']
def pop(pointerFront): # FIFO, so removes the value from the
front of the queue and returns it
```

```
    value = myQueue[pointerFront]
    myQueue[pointerFront] = 'Empty'
    pointerFront = pointerFront + 1
    return value, pointerFront

def push(data, pointerBack): #add a new piece of data to the
end of the queue
    myQueue[pointerBack] = data
    pointerBack = pointerBack + 1
    return pointerBack

pointerFront = 0 # points to the front of the queue
pointerBack = 4 # points to the back of the queue
    print (myQueue)
    value, pointerFront = pop(pointerFront)
    print (myQueue)
    pointerBack = push ('Alex', pointerBack)
    print (myQueue)
```

Binary trees and binary search trees

Binary tree data structures are composed of nodes (sometimes called leaves) and links between them (sometimes called branches). Each node has up to two others coming off it, which creates a structure like the one shown in Figure 1.6.

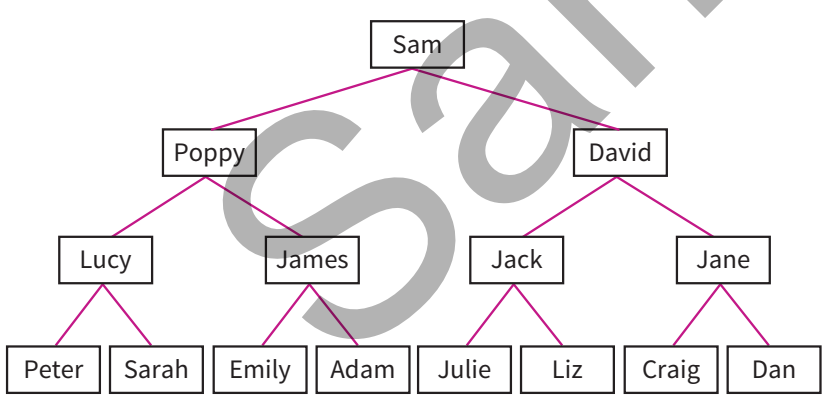


Figure 1.6: A binary tree.

Binary trees are used to implement another data structure called a binary search tree. These are identical to binary trees but have the additional constraint that the data on the left branch of a node must be less than the data in the node. Likewise, the data on the right branch of the node must be greater than the data in the node. All the following binary trees are also binary search trees.

Unlike stacks and queues, there are a range of different ways to traverse binary tree data structures. Each method will return a different set of results, so it is important that you use the correct one.

We'll use the binary search tree shown in Figure 1.7 to demonstrate the different methods of traversal.

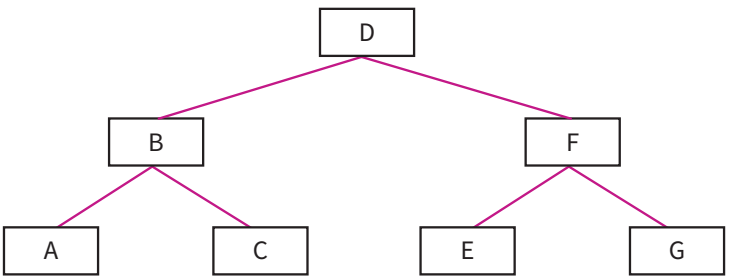


Figure 1.7: A binary tree.

- *Pre-order traversal (depth first search)*: Start at the root node, traverse the left sub-tree, then traverse the right sub-tree. This would give: *D B A C F E G*.
- *In-order traversal*: Traverse the left sub-tree, then visit the root node, and finally traverse the right sub-tree. This would give: *A B C D E F G*.
- *Post-order traversal*: Traverse the left sub-tree, then traverse the right sub-tree, and finally visit the root node. This would give: *A C B E G F D*.

Implementing a binary tree

Just like stacks and queues, it is possible to implement a binary tree using an array or a linked list. It is more useful to use a linked list as this allows the binary tree to be dynamic and grow to any size.

Using a linked list, the nodes of a tree are made up of a piece of data as well as pointers to the addresses of the nodes on the left and right. It is also sensible to include a pointer to a node's parent (the node above it) to aid in navigation. The C++ code below shows how this can be achieved:

```
class Node
{
public:
Node(string dataInput, Node* left=NULL, Node* right=NULL,
Node* parent=NULL);

 void setData(string dataValue);
 void setLeft(Node* leftNode);
 void setRight(Node* rightNode);
 void setParent(Node* parentNode);

private:
 string data;
 Node* left;
 Node* right;
 Node* parent;
};
const string& getData() const;
const Node* getLeft() const;
const Node* getRight() const;
const Node* getParent() const;
```

However, binary trees can also be created using an array. The first element would contain the root, at index 0. From that point on you would find the index which contains the left and right nodes using the formula:

Left index = $2n + 1$

Right index = $2n + 2$

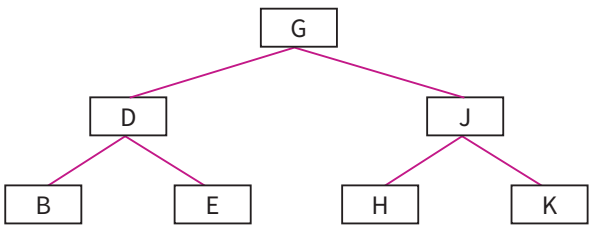


Figure 1.8: A binary tree.

Figure 1.8 would be encoded in the array below. To find the index for H, we start at 0. In order to go right we do the calculation $2n + 2$ or $2 \times 0 + 2$ which is 2. From index 2 we go left using the calculation $2n + 1$ or $2 \times 2 + 1$ which is 5. 'H' is at position 5.

```
myTree = ['G', 'D', 'J', 'B', 'E', 'H', 'K']
```

Adding data to a binary search tree

You can add data to a binary tree by following these simple steps, starting at the root of the tree:

1. Compare the data you want to add to the current node.
2. If the new data is less than the node, follow the left pointer, otherwise follow the right pointer.
3. Repeat this until you reach the end of the tree.
4. Update the pointers of the last node to point to the new one.

The Python code below gives you an idea of how this can be done.

```
class node:
  def __init__(self, data, parent):
    self.left = None
    self.data = data
    self.parent = parent
    self.right = None

def addToTree(self, currentNode, data, parent):
  if currentNode == None:
    newNode = node(data, parent)
  else:
    if data <= currentNode.data:
      addToTree(currentNode.left, data, currentNode)
    else:
      addToTree(currentNode.right, data, currentNode)
```

Removing data from a binary tree

Removing data from a binary search tree is more complicated. The method required differs depending on the number of branches coming off the node to be removed.

Case 1: a leaf with no branches

If the node to be removed is a leaf with no branches coming off it, removal is simple. You can delete the node and set the original pointer which is referencing the node to be null. In this example (Figure 1.9), node A is simply removed along with the pointer from node B.

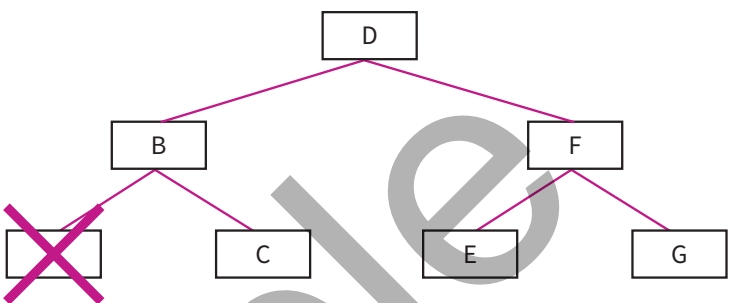


Figure 1.9: A binary tree.

Case 2: a leaf with a single branch

If the node has a single branch, you must update the pointer of the parent to point to the address of the child of the node to be deleted. Then you can delete the node without losing any information. In this example (Figure 1.10), the left pointer of node D is updated to point to node C before node B is deleted.

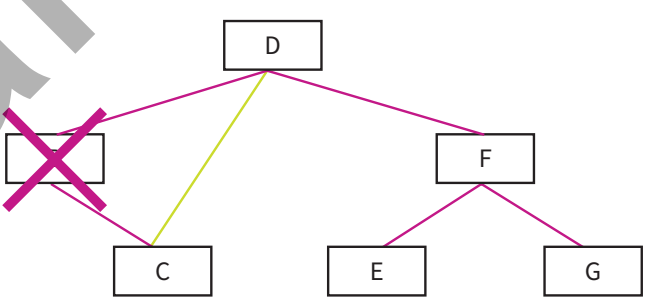


Figure 1.10: An element is removed from a binary tree and the necessary branches are updated.

Case 3: a leaf with two branches

If the node to be deleted has two branches, things are slightly more complicated. First you must use in-order traversal to find the smallest node beneath the one to be deleted, which is known as the successor node. The successor node has its left, right and parent pointers updated so that they are the same as the node to be deleted. Only then can the target node be removed from the tree. Although this may seem complicated, it is vital if the structure of the binary search tree (with small values on the left and large values on the right) is to be preserved.

Linked lists

Linked lists are probably the most common type of dynamic data structure. Being dynamic means that, unlike arrays, they can grow to whatever size is required and their size isn't set when they are declared at the beginning of the program.

Linked lists are made up of nodes, with each node linked to the next node in the list. They are really flexible, powerful data structures and can be used to implement almost every other type of data structure you need to know about.

Nodes in a linked list consist of two things: data and a pointer to the next piece of data in the list (Figure 1.11).

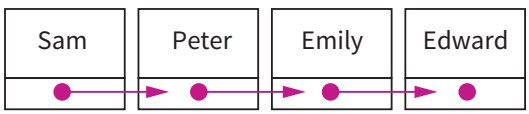


Figure 1.11: A linked list.

The best way to implement a linked list is using an object-oriented language such as C++; this allows the list to be completely dynamic:

```
class Node
{
public:
  Node(const Node* next=NULL);

  void setData(const string& dataValue);
  void setNext(constNode* nextNode);
  string getNext();
private:
  string data;
  Node* next;
};
```

Each node is made up of a piece of data and a pointer to the memory address of the next node in the list. This information is stored in the private section of the Node class. A node is created by calling the public Node constructor and passing it the data value of the node and a pointer to the next node.

Adding data to a linked list

Adding new data to the end of a linked list is easy. The first step is to create a new node (Figure 1.12):

```
Node* newNode = new Node();
newNode->SetData('Dan');
```

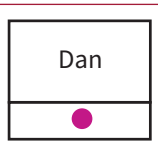


Figure 1.12: Node to be added.

The second step is to update the last item in the current list so that it points to the new item (Figure 1.13):

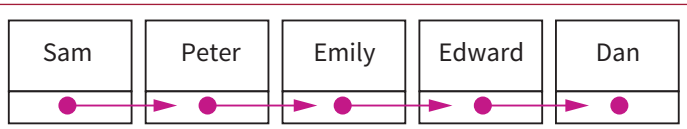


Figure 1.13: Linked list with new node added.

Of course, there is no reason why you have to put your new item at the end of the list. Imagine you want to create a linked list that is sorted in a particular order. You can easily insert a new node in the correct place by working out where you want it, adjusting the node that is currently there to point to the new node and then telling your new node to point to the next node in the list (Figures 1.14 and 1.15).

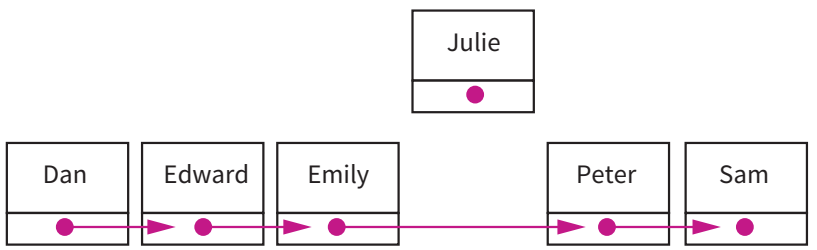


Figure 1.14: Linked list with node to be inserted.

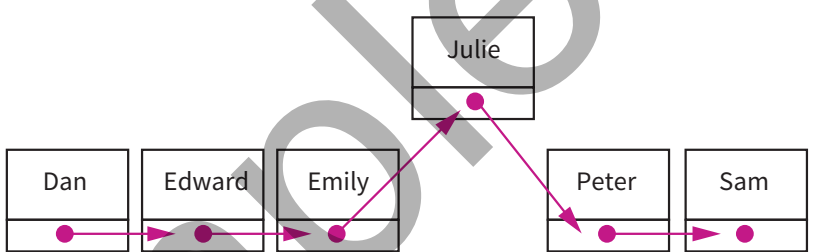


Figure 1.15: Inserting data into a linked list.

Removing data from a list

It is also very simple to remove an item from a list. We simply need to adjust the item before it to point to the item after it. Doing this removes the node from the list and preserves the order of items.

Hash tables

A hash table has two components, a table where the actual data is stored and a mapping function (called a hash function or hash algorithm). The hash function uses the data that is going to be entered in the table to generate the location in the table where the data should be stored.

In this example, we want to use a hash table to hold some names. The initial table shown below is the easy part; it looks like any other table:

Index	Name
0	
1	
2	
3	
4	

Next, we need to decide on our hash algorithm; let's say that we will take the length of the name entered and divide it by the size of the table (using the modulo (MOD) operation); the remainder gives the name's index number.

Let's say we want to enter the name 'Sarah' in the table. The length of the string is 5 and the length of the table is also 5:

5 MOD 5 = 0

So we'll store the name in index 0 as shown in the table below:

Index	Data
0	Sarah
1	
2	
3	
4	

To insert a new, name such as Adam, we carry out the same operation (4 MOD 5) and get 4, so the name Adam goes in index 4:

Index	Data
0	Sarah
1	
2	
3	
4	Adam

It's not hard to see that with our simple hash algorithm we'll soon have problems. If I want to add Jane to the table, her name is the same length as Adam's so the algorithm will try to overwrite Adam to insert Jane. Not ideal! There are many solutions to this problem, the most useful of which is called separate chaining.

Separate chaining uses the original table to store a dynamic data structure (like a linked list). When a new item creates the same hash result as an existing piece of data, a new node on the linked list in that location is created to hold the new data. While good at avoiding collision, this technique will slow down finding the data again, as once the hash location has been computed the linked list will need to be searched as well. It is important to pick a hash function which spreads data evenly over the table to minimise collisions and the potential slow down caused as a result.

Chapter summary

- Data structures are organised stores of data in a computer's memory. Each data structure has its own strengths and weaknesses that make it suitable for a given situation.
- Arrays hold data of a single type in rows and columns; they have one or more dimensions.
- The different sections in an array are called 'elements' and are accessed using an index (which usually starts from 0).
- A record is a data structure that allows items of data of different types to be stored together in a single file.
- Data is removed from linked lists by deleting the node and pointing the item behind it to the item in front of it.
- Stacks are Last-In, First-Out data structures that hold data in the order in which it was entered.

- Data is removed or added at the top of the stack.
- Queues are First-In, First-Out data structures in which the last piece of data entered is the last piece of data retrieved.
- Tree data structures consist of nodes (leaves) and branches. Nodes contain data and the branches link nodes together.
- Linked lists represent each piece of data as a node which holds the data and a link to the next piece of data in the list.
- Hash tables use mapping functions to calculate where in the table a new piece of data should be stored.

End-of-chapter questions

1 What notation is used to describe the efficiency of a data structure or algorithm? [1] A

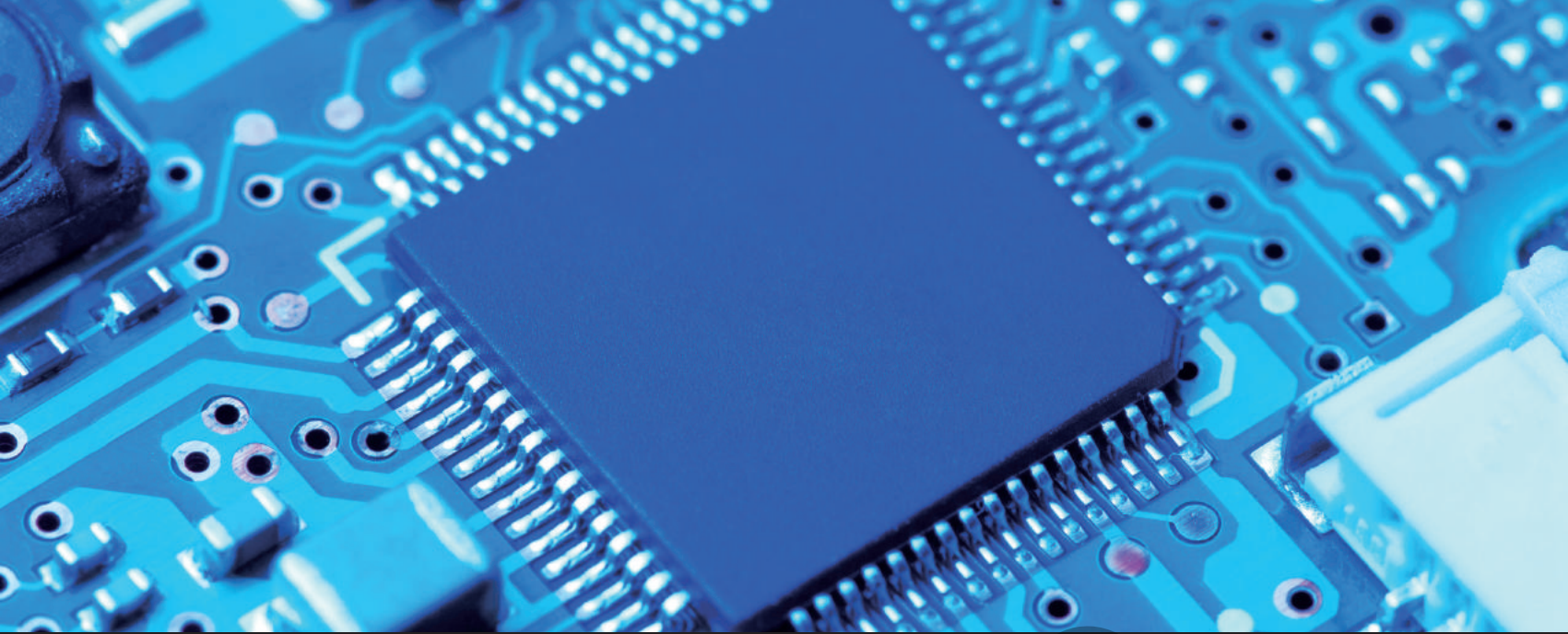
2 What is an array? [2]

3 What do the nodes in a linked list contain? [2] A

4 Give the name of a First-In, First-Out data structure (FIFO). [1]

Further reading

Efficiency of data structures	bigocheatsheet.com
Big O notation	Search for an article on Big O notation on www.khanacademy.org
Linked lists	Search for an article on Linked List Basics on Stanford's Computer Science library
Binary trees	Search for an article on Binary Trees on Carnegie Mellon's Computer Science website
Stacks and queues	Search for a lesson on Data Structures on Virginia Tech's Computer Science department website.



Chapter 10
Hardware and communication

Learning objectives

- Identify and describe the hardware and communication elements of contemporary computer systems and how they are connected.
- Identify and describe the main components of computer architecture, including von Neumann architecture.
- A • Identify and describe contemporary architectures.
- Describe different types of memory and caching.
- Describe and explain parallel processing.
- A • Describe and explain the limiting factors to parallelisation.
- Calculate the runtime of given tasks as a result of parallelisation and evaluate the effect of parallelisation.
- Describe the fetch-decode-execute cycle, including how data can be read from RAM into registers.
- A • Write simple programs in assembly language and demonstrate how these programs could be executed.
- Describe the use of contemporary methods and their associated devices for input and output.
- A • Explain the use of these methods and devices in contemporary computer systems and their suitability in different situations.
- Describe and differentiate between voice input for command and control systems, vocabulary dictation systems for general input, and voice print recognition for security. Discuss the suitability of each system in different situations.

- Compare the functional characteristics of contemporary secondary storage devices. A
- Explain fragmentation and its consequences and describe the need for defragmentation.
- Describe networks and how they communicate.
- Explain the importance of networking standards.
- Describe the importance and the use of a range of contemporary protocols including HTTP, FTP, SMTP, TCP/IP, IMAP, DHCP, UDP and wireless communication protocols. A

  Explain the role of handshaking.
- Identify and describe applications where connecting a portable device to a network is required.
- Describe the hardware required to make a wireless connection and explain how this might be achieved using contemporary wireless technologies.

Introduction

At the centre of all modern computer systems is a device referred to as the central processing unit (CPU), microprocessor or simply the processor. The processor is the brain of the computer; it carries out all the mathematical and logical operations necessary to execute the instructions given to it by the user. It is one of the most expensive parts of a computer system and upgrading a computer's processor remains one of the best ways of increasing a computer's performance.

Today, processors can be found in anything from smartphones and tablets to washing machines and microwaves.

As you might expect, processor designs are extremely complex and the way they are constructed (their architecture) changes rapidly. Innovations in specialist materials and improved design techniques are utilised to make them faster and more efficient.

To create a processor's architecture, specialists will first design the necessary circuitry. Silicon discs are then produced by melting sand, refining it and finely slicing the resulting crystal. Next, the circuit diagrams are transferred to the silicon discs, resulting in the creation of thousands of tiny transistors; these are joined together using copper to create small integrated circuits. Finally, these are packaged with the pins necessary to connect the processor to a computer's motherboard. This process takes place in a 'clean room', which is a very controlled and sterile environment. The smallest amount of dust could ruin the silicon.

This is a very basic overview of the manufacturing process used to create CPUs. It is well worth using the *further reading* section at the end of this chapter to find out more.

A processor is made up of a number of key parts, which work together to execute instructions. However, the processor requires software to perform any action and in this chapter you will learn about machine code, assembly code and their relationship with high-level programming. You will also explore how the processor transfers instructions from main memory to the processor.

Components of a processor

A processor is a small and complex device. Internally, the processor is made up of a number of key components, which include the Arithmetic Logic Unit, Control Unit and Registers (Figure 10.1). Each component does a different job. This type of processor is used in the von Neumann architecture, which will be discussed later in this chapter.

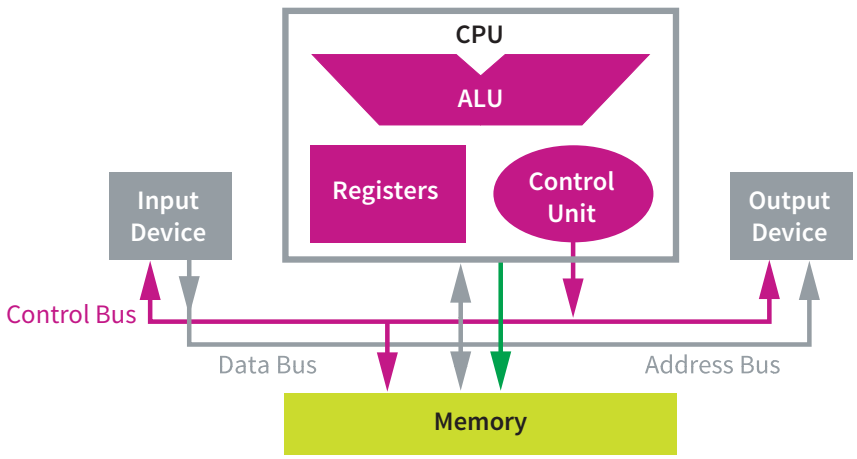


Figure 10.1: Overview of a computing system.

Arithmetic logic unit

The arithmetic logic unit (ALU) is responsible for calculations and logic operations. Calculations include floating point multiplication and integer division, while logic operations include comparison tests such as greater than or less than. The ALU also acts as a conduit for input and output to and from the processor.

Control unit

The control unit (CU) is a key part of the processor. It manages the execution of machine code by sending control signals to the rest of the computer. Control signals are sent via a control bus to connected devices, such as hard drives or the graphics card. Part of the CU's job is to synchronise instructions by using the processor's internal clock. This process will be based on the clock speed. An instruction will take one or more 'ticks' of the clock, known as clock cycles.

Registers

A register is a small block of memory used as temporary storage for instructions as they are being processed. The register runs at the same speed as the processor. A processor contains many registers; some are reserved for specific purposes while others are used for general purpose calculations. Machine code instructions can only work if they are loaded into registers.

General purpose registers are used as programs run, to enable calculations to be made, and can be used for any purpose the programmer (or compiler) chooses. Special purpose registers are crucial to how the processor works. Values are loaded in and out of registers during the execution of a process. Some of the key special purpose registers are:

- Program counter (PC)
- Memory address register (MAR) and memory data register (MDR)
- Current instruction register (CIR)
- Accumulator (ACC)

Program counter

The program counter (PC) stores the address of the next instruction to be executed.

Programs need to be loaded into memory by the operating system in order to function. As memory is referenced using memory addresses, the start of any given program has a specific memory address. It is not possible to know in advance where exactly in memory

a program will be loaded, as this is managed by the operating system. Each instruction of the program is assigned an address in memory. The PC initially contains the address of the first instruction of the program, not the actual instruction.

As the process is run, the instruction to which the PC is pointing is loaded into the memory address register (MAR) so that it can be fetched from memory into the processor, and the PC is incremented so that it points to the address of the next instruction. The value added to the PC is calculated from the size of the instructions for the given instruction set.

Sometimes the value of the PC changes as the result of an instruction being executed. This tends to be when a `JUMP` instruction is run; `JUMP` is used to provide the functionality of procedures, IF statements and iteration.

Memory address and memory data registers

Currently running programs are stored in memory, which means that they must be loaded into registers. This process is known as fetching and makes use of two special registers, the memory address register (MAR) and the memory data register (MDR). The MAR contains the address of the instruction or data to be fetched. The fetched instruction or data is stored in the MDR. Because fetching instructions from memory can be a slow process, the processor always tries to fetch one instruction ahead of the one it is currently working on. So, while one instruction is being executed, the next one is being fetched. This process of looking ahead is known as pipelining.

Current instruction register

Once an instruction has been fetched, it is copied into the current instruction register (CIR) for executing. As the instruction in the CIR is being decoded and executed, the next instruction is being fetched into the MDR.

Accumulator

Any instruction that performs a calculation makes use of the accumulator (ACC). Many instructions operate on, or update, the ACC. If a subtraction instruction is run, it performs the subtraction using the data part of the instruction and stores the result in the ACC. Calculations take a step-by-step approach, so the result of the last calculation is part of the next. In a certain computer system, an instruction set may use the ACC as part of the calculation. Only one value needs to be given in the data part of the instruction. This is shown in Table 10.1 below:

Instruction	ACC
Initial value	0
ADD 4	4
ADD 2	6
SUB 1	5

Table 10.1: Table showing how the value of the ACC changes when instructions are executed.

Memory and caching

For a program to work, it loads instructions and data from main memory (random access memory or RAM) by using the Fetch-Execute cycle. As memory runs a lot slower than the processor, the processor is likely to have to wait for main memory to fetch data, which results in wasted clock cycles. The overall efficiency of the computer system is reduced,

as a lot of time is wasted because of the speed mismatch between the memory and the processor (this is called the von Neumann bottleneck). Cache memory helps to solve this problem by acting as a 'middle man' between the processor and the memory.

Data and instructions that are used regularly are copied into the cache and when the processor needs this data, it can be retrieved much faster than if it were stored in RAM. The processor always tries to load data from the cache, but if the data it needs is not there, it is copied over from RAM. If the cache is full, the least used data in the cache is thrown out and replaced with the new data.

There are a number of algorithms that can be used to ensure that the cache contains the most regularly used data. This is a lot easier than you might expect, owing to the 80/20 rule. Programs tend to spend most of their time in loops of some description, which means that they will be working on the same instructions for a lot of their execution time. The 80/20 rule states that 80% of a program's execution time is spent in only 20% of the code. This proves to be fairly accurate in most cases.

When the processor requests data that is not in the cache, this is referred to as a cache miss. If a system has a high incidence of cache misses, the overall speed of the system will be substantially reduced. Whichever algorithm is employed to manage cache, the primary focus is to minimise cache misses.

Cache memory size is smaller than main memory owing to its high cost. It is categorised by different levels that describe its speed, size and how close it is to the CPU:

- **Level 1** cache is extremely fast but relatively small, and is usually embedded in the CPU.
- **Level 2** cache often has a higher capacity than Level 1, and as a result operates more slowly. It may be located on the CPU, dedicated to single or pairs of cores,
- or located on a separate chip, to avoid being slowed down by traffic on the main system bus.
- **Level 3** cache works to improve the performance of Level 1 and Level 2 cache and is typically shared by all cores of a processor.

Most modern computers share the same critical problem as a result of using the von Neumann architecture. Because of the speed mismatches between memory, secondary storage and the processor, most of the processor's time is spent idle. This is known as the von Neumann bottleneck.

Parallel processing

The classic von Neumann architecture uses only a single processor to execute instructions. In order to improve the computing power of processors, it was necessary to increase the physical complexity of the CPU. Traditionally this was done by finding new, ingenious ways of fitting more and more transistors on to the same size chip.

However, as computer scientists reached the physical limit of the number of transistors that could be placed on a silicon chip, it became necessary to find other means of increasing the processing power of computers. One of the most effective means of doing this came about through the creation of multi-core systems (computers with multiple processors).

One of the most common types of multi-core system is the parallel processor. They tend to be referred to as dual-core (two processors) or quad-core (four processors) computers.

Computing in context: von Neumann bottleneck

John von Neumann was a Hungarian-American mathematician, born in 1903. He is widely considered to be one of the fathers of modern-day computing. He also made many contributions to areas of mathematics such as game and operator theory.

In 1945, while consulting for an engineering group developing the EDVAC computer, von Neumann released a paper entitled 'First draft report on the EDVAC'. To give the report context, it is important to understand the EDVAC computer and generally how computing was achieved in the 1940s. The majority of computing systems in use today are based on the von Neumann architecture, named after him.

In parallel processing, two or more processors work together to perform a single task. The task is split into smaller sub-tasks (threads). These tasks are executed simultaneously by all available processors (any task can be processed by any processor). This hugely decreases the time taken to execute a program, but software has to be specially written to take advantage of these multi-core systems.

All the processors in a parallel processing system act in the same way as standard single-core (von Neumann) CPUs, loading instructions and data from memory and acting accordingly. However, the different processors in a multi-core system need to communicate continuously with each other in order to ensure that if one processor changes a key piece of data (for example, the players' scores in a game), the other processors are aware of the change and can incorporate it in their calculations. There is also a huge amount of additional complexity involved in implementing parallel processing, because when each separate core (processor) has completed its own task, the results from all the cores need to be combined to form the complete solution to the original problem.

This complexity meant that in the early days of parallel computing it was still sometimes faster to use a single processor, as the additional time taken to coordinate communication between processors and combine their results into a single solution was greater than the time saved by sharing the workload. However, as programmers have become more adept at writing software for parallel systems, this has become less of an issue.

Advantages	Disadvantages
More instructions can be processed in a shorter time because they are executed simultaneously.	It is difficult to write programs for multi-core systems.
Tasks can be shared to reduce the load on individual processors and avoid bottlenecks.	Results from different processors need to be combined at the end of processing, which can be complex and adds to the time taken to execute a program.
	Not all tasks can be split across multiple processors.
	Concurrency introduces new classes of software bugs.

Table 10.2: Table showing the advantages and disadvantages of a parallel computer.

Runtime calculations

You may be required in an exam to explain the limiting factors of parallelisation in parallel processing in terms of runtime. Programs written specifically for parallel processing may have a certain portion of code that cannot be parallelised.

In the following example, a program has a runtime of 5 hours when using a single core processor. If 80% (4 hours) of this program can be parallelised, then clearly a multi-core processor will reduce the runtime required. However, regardless of the number of cores used to execute this program, the minimum runtime cannot be less than the time taken to execute the non-parallelised 20% (1 hour). The remaining 20% will still be processed sequentially.

Amdahl's law is used to calculate the minimum runtime of executing a program on a certain number of threads:

$T(n) = T(1)\ (B + \frac{1}{n}(1 - B)$

Where:

- $T(n)$ = time taken on n threads
- n = number of threads
- B = fraction of the algorithm that is sequential

As in the single-core example above, Amdahl's law demonstrates that with one thread ($n = 1$), the runtime of executing the program is 5 hours:

$T(n) = T(1)\ (B + \frac{1}{n}(1 - B)$

$T(1) - 5 \text{ hours} \times (0.2 + \frac{1}{n}(1 - 0.2)) - 5 \text{ hours}$

However, using Amdahl's law to calculate the runtime in a quad-core processor with four threads ($n = 4$), you get:

$T(n) = T(1)\ (B + \frac{1}{n}(1 - B)$

$T(4) - 5 \text{ hours} \times (0.2 + \frac{1}{4}(1 - 0.2)) - 2 \text{ hours}$

Note that even with an infinite number of threads, the runtime of executing the program cannot be less than 1 hour.

Fetch-decode-execute cycle (FDE)

When instructions are to be executed by the processor, they must be loaded into the processor one after another, via a process known as the fetch-decode-execute cycle (FDE). In the previous section, you were introduced to a number of special purpose registers used during the FDE cycle and in this section you will see how they work together. It is important to remember that a single processor can only execute a single instruction at a time from the current instruction register. By using a large number of registers, the whole process can be made more efficient using a system called pipelining. Figure 10.2 shows a summary of the registers involved in the FDE cycle.

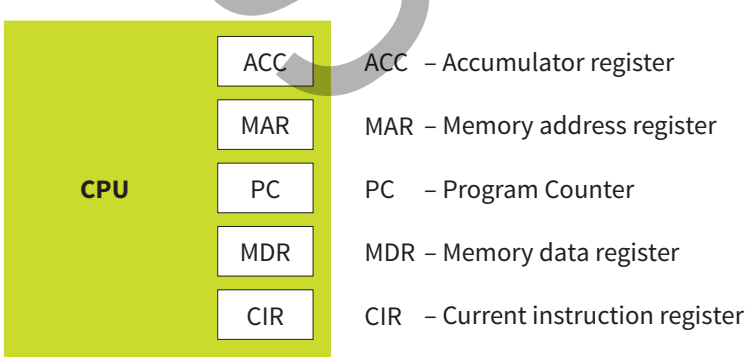


Figure 10.2: Specialist registers used during FDE cycle.

In Figure 10.3, the address stored in the PC is 0092. The first part of the cycle is to fetch the next instruction for processing. The PC is copied over to the MAR and then the PC is incremented. Most programs will run serially, unless a control instruction such as a jump is encountered, so by incrementing the PC it is highly likely that it will be pointing to the next instruction to be fetched.

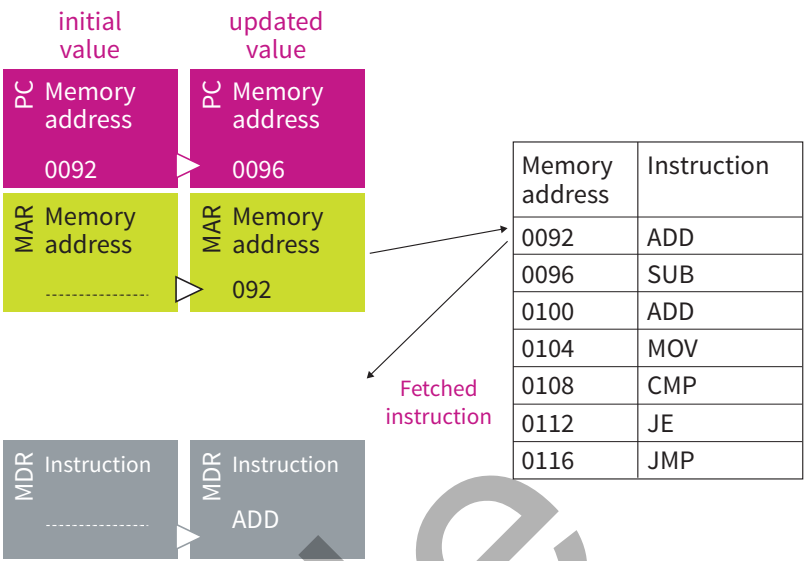


Figure 10.3: The content of registers change during FDE cycle.

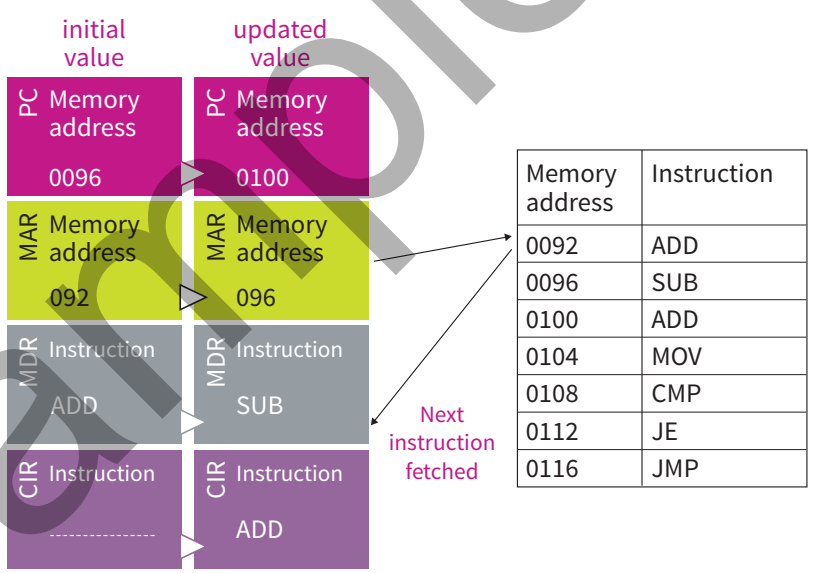


Figure 10.4: Registers during the second stage of the FDE cycle.

Figure 10.4 shows how the registers are altered in the first step of the cycle and sets up the fetch part of the cycle. In order to fetch, the address found in the MAR is looked up in memory and the contents of that memory address are loaded into the MDR. As memory is much slower than the processor, the fetch could take a number of clock cycles to complete and depends on whether the data is found in cache memory. An advantage of the FDE cycle is that while one instruction is being fetched, the previous one is being decoded and executed. This is why it is necessary to have separate registers for fetching and executing instructions.

In order for the processor to execute the instruction after the fetch, it must decode the **opcode** part of the instruction. The contents of MDR are copied over to the CIR. From here the instruction can be decoded and then executed, and as this is happening the next instruction can be fetched. Figure 10.5 shows what the registers look like during the second stage of the FDE cycle. As the first instruction, ADD, is being decoded, the second one is being fetched while the PC is pointing to the third. That way, the processor is always trying to be one step ahead of itself in order to make most efficient use of memory and clock cycles.

The cycle is designed to perform as many actions of the processor concurrently as possible, thus improving the overall speed of execution.

In nearly any program, at some point a control instruction that will force a jump from one instruction to another that is not in sequence will be encountered. The most common such instruction is a jump. A jump breaks the FDE cycle and forces it to start again from scratch, because the next instruction that has been fetched, the next one in sequence, is not the next instruction that should be executed. Consider the instruction JE 0092 (jump if equal to address 0092) and the CMP command has returned true for equality (Figure 10.5). Executing this instruction will cause a break in the linear flow of the program and will update the PC to be 0092.

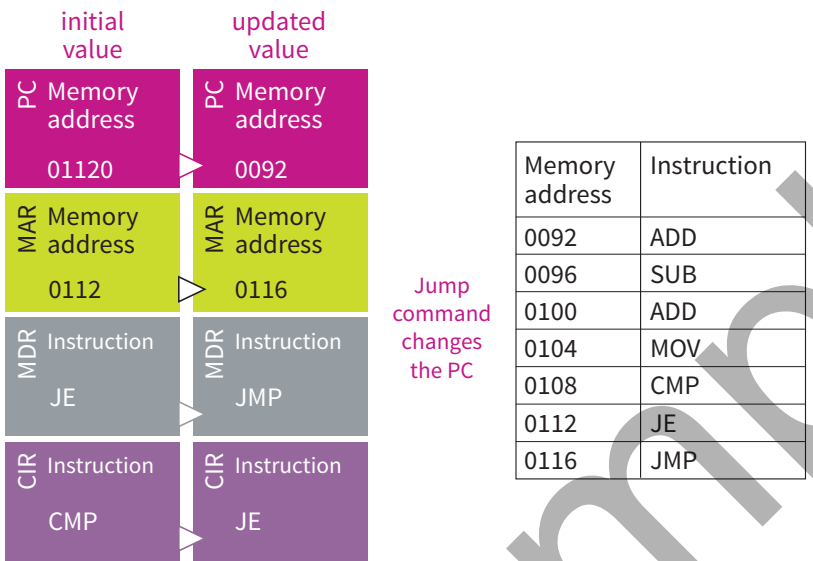


Figure 10.5: Jump command changes the PC.

Jumps slow down the processor by losing the benefits of pipelining. When programming in Assembly, or when a compiler optimises code, it is important to limit the number of jump commands.

Processor speed

Processor speed is measured by the number of clock cycles that the processor can perform in a second and is measured in hertz (Hz). A clock cycle is one 'tick' of the CPU clock. During a clock cycle the processor can fetch, decode and execute a simple instruction, such as load, store or jump. More complex instructions take more than one clock cycle.

It is easy to calculate the amount of time it takes to run one cycle for any speed of processor. The calculation for a 2 gigahertz (GHz) processor is shown in Figure 10.6:

$$Frequency = \frac{1}{Time}$$

$$Time = \frac{1}{Frequency}$$

$$Time = \frac{1}{2097152}$$

$$Time = 4.768 \times 10^{-7} s$$

Figure 10.6: Calculating the time taken to complete one clock cycle on a 2 Ghz CPU.

The number 2 000 000 is obtained by converting GHz to Hz.

Tip

You may be asked to calculate clock speed in the exam.

Processors are getting to the point where it is hard to increase speed. When the number of clock cycles per second is increased, the transistors within the processor have to switch faster, which generates more heat. Unless the heat is drawn away from the processor, it can easily overheat. A processor can reach temperatures in

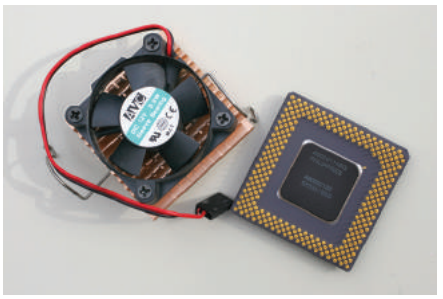


Figure 10.7: Heat sink and fan alongside a processor.

excess of 300 degrees Celsius, which is hotter than an oven, in a matter of seconds. If you open up a desktop PC, you will see a large heat sink and fan attached to the processor (Figure 10.7), which has the job of pulling all of this heat away from the processor. You should never run a system without a heat sink as this could cause permanent damage to both the processor and the motherboard. It would also pose a severe fire risk.

Processor manufacturers, such as AMD or Intel, try other methods to increase the performance of their CPUs. This can include increasing the number of cores (processors) in the CPU, making existing circuitry more efficient, or developing new instructions. In order to make a fair comparison of these processors, they should be subjected to controlled tests. An excellent place to see benchmarks of different processors is http://www.cambridge.org/links/kwse6059.

Assembly language programming

A

In some computer systems, there is a simple architecture designed to help you understand the concepts of machine code and instruction sets. An example of this is the Little Man Computer (LMC). The LMC has a very limited selection of instructions, shown in Table 10.3 along with their opcodes. An xx in the opcode refers to the data part of the instruction (if required – not every instruction needs data). For example, in order to add, you first need to know what you are adding. To keep things simple, the LMC has only two registers, ACC and PC. Data and instructions are stored in memory locations known as mailboxes; this is an exact simulation of a standard von Neumann-based computer. Conceptually, a mailbox represents one single byte of memory.

Opcode	Name	Data	Explanation
1xx	ADD	The number to add	It will add a value from a mailbox to the current value of the ACC
2xx	SUB	The number to subtract	It will subtract a value from a mailbox from the current value of the ACC
3xx	STA	Stores a value into a mailbox	Will take the contents of the ACC and store it in a mailbox
5xx	LDA	Loads a value from a mailbox	Will take a value from a mailbox and store it in the ACC
6xx	BRA	Line number to 'jump' to	Unconditional branch. Will jump to the given instruction number
7xx	BRZ	Line number to 'jump' to	Branch if zero. Will only jump if the ACC is zero
8xx	BRP	Line number to 'jump' to	Branch if positive. Will only jump if the ACC has a positive value. NB: Zero is classed as a positive number
901	INP	Input from the user	Will input a value and store it in the ACC
902	OUT	None	Will output the contents of the ACC
xxx	DAT	Address of a variable	Used at the end of the program to declare any variables used

Table 10.3: Table showing the instructions available in the LMC.

Mailboxes can be referred to directly, or indirectly, through the use of labels. A label is a text symbol that represents a mailbox, making coding in LMC easier. When a label is used to represent a mailbox, the LMC assembler assigns a suitable mailbox as the code is assembled.

The simple LMC program in Figure 10.8 will add two numbers input by the user. Each line of the code is converted into an LMC machine code instruction and stored in a single mailbox. Line 1 is converted to '901', which is stored in mailbox 0. Line 2 stores the value in the accumulator in mailbox 6, which is represented by the LMC code as the label FIRST. The fully assembled code is shown in Table 10.4.

```
0                INP
1                STA   FIRST
2                INP
3                ADD   FIRST
4                OUT
5                HLT
6   FIRST        DAT   0
```

Figure 10.8: Sample LMC program to add two numbers together.

Pos	0	1	2	3	4	5	6
Value	901	306	901	106	902	0	12

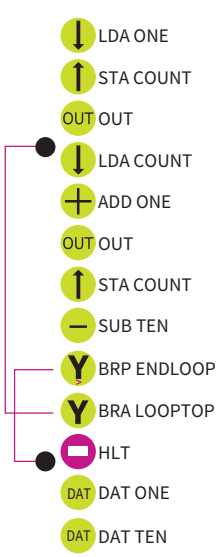
Table 10.4: Table showing an LMC program and its mailboxes.

A more complicated LMC program can be seen in Figure 10.9. This program will display the numbers from 1 to 10, but could easily be updated to perform actions on any sequence of numbers. Assembly code, regardless of **CPU** architecture, does not have control statements such as `IF` or `WHILE`. Instead, the flow of execution is changed by conditional jump commands. In LMC, jumps are referred to as branches. It is possible to represent any programming control statement through branches and jumps.

```
0
1                LDA   ONE
2                STA   COUNT
3                OUT
4   LOOPTOP      LDA   COUNT
5                ADD   ONE
6                OUT
7                STA   COUNT
8                SUB   TEN
9                BRP   ENDLOOP
10               BRA   LOOPTOP
11  ENDLOOP      HLT
12  ONE          DAT   1
13  TEN          DAT   10
14  COUNT        DAT   0
```

Figure 10.9: Sample LMC program to display the numbers from 1 to 10.

Branches allow decisions and loops to be coded in LMC. Figure 10.10 is a diagrammatic representation of the LMC code to add the numbers 1 to 10. The line **BRA LOOPTOP** does an unconditional jump back to the top of the loop. **BRP ENDLOOP** breaks out of the loop once the ten numbers have been added. BRP will only branch if the value in the accumulator is positive, which is why 10 is subtracted from the current count. Initially count will be 1, so 1 minus 10 will result in a negative number. One gets added to the count and we branch unconditionally. This continues until count becomes 10, as zero is considered to be a positive number. Rather than having a command that loops only if the accumulator is negative, you can use simple maths to remove the need for the command entirely.



Pos	0	1	2	3	4	5	6	7	8	9
Value	511	313	902	513	111	902	313	212	810	63
Pos	10	11	12	13	14	15	16	17	18	19
Value	0	1	10	10	0	0	0	0	0	0

Figure 10.10: Diagram showing the LMC code required to add the numbers 1–10.

Activity 10.1

Using an LMC simulator enter the following code.

```
0            LDA   ONE
1            STA   COUNT
2            OUT
3  LOOPTOP   LDA   COUNT
4            ADD   ONE
5            OUT
6            STA   COUNT
7            SUB   TEN
8            BRP   ENDLOOP
9            BRA   LOOPTOP
10 ENDLOOP   HLT
11 ONE       DAT   1
12 TEN       DAT   10
13 COUNT     DAT   0
```

Run this program to ascertain its purpose. Once you understand it, write the code out in a high level language or in pseudocode.

Consider the following high-level code below:

```
1  x = 0
2  total = 0
3  while x < 10
4        total = total + x
5        x = x + 1
6  end while
7  print total
```

Rewrite this code using the LMC simulator.

Input and output devices

Input devices allow interaction with a computing system. This could be through a standard interface, such as a mouse or keyboard, or through more advanced interfaces such as Microsoft's Kinect games controller. Once data has been captured by an input device, it must be processed by software before information can be given back to the user.

Optical character recognition (OCR)

Optical character recognition (OCR) converts printed media into editable text documents using a scanner. OCR empowers the user to change sections of text on a printed document. It also means that old books, for example, can be digitised, allowing them to be published in new formats for new markets such as e-readers.

OCR is a post-processing step that occurs after a document has been scanned. It performs pattern matching by scanning the image for shapes it can recognise as letters, numbers or symbols. This is done by comparing the binary data to an internal database of known character shapes. When a binary shape is found in a document, OCR transforms it (resizes

and rotates it) until it matches a known shape from the database. If no match is found, that shape is skipped.

It is very easy for a human to read text off a document, but very complicated for a computer to do it. This is why OCR will sometimes get it wrong, so the resulting document must be proofread.

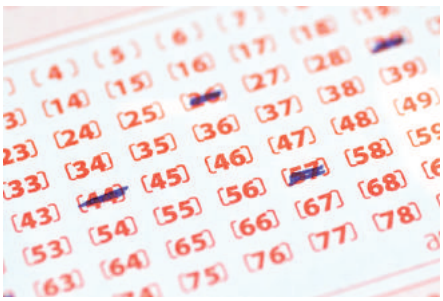


Figure 10.11: A typical input sheet for an OMR reader.

Optical mark recognition (OMR)

Optical mark recognition (OMR) is based around a predefined form, which has areas where someone can mark multiple choice responses (Figure 10.11). The form has specific parts for someone to fill out by putting a thick black mark on them. A special reader then reads in this form in order to detect the dark marks and note their position on the page. As the form is predefined, the scanner knows what option the user has selected. If two marks are made in the same section, the reader accepts only one of the answers, most likely the first one. Also, if the mark does not cover the whole of the circle, the reader may miss it. This type of input is used for multiple choice exams and registers, as they can be automatically marked and processed. It is not suitable for any form of written communication.

Magnetic ink character recognition (MICR)

Magnetic ink character recognition (MICR) uses a combination of ink containing iron oxide and specific fonts so that data written using this ink can be read by a specialist MICR reader. Normal ink does not contain iron oxide, so only the MICR ink will be picked up by the reader, effectively ignoring everything else. This means that regardless of what is written over the magnetic ink, it can still be read by the device.

This is almost exclusively used for cheques because of the high cost of the reader (Figure 10.12). The cheque number and account number are written on the bottom of each cheque using magnetic ink. This can be very quickly read in by the reader, so cheques can be processed quickly by the banks. One of the key benefits of MICR is that it does not have to rely on OCR, which can be problematic.

Figure 10.12: A standard bank cheque. Notice the number in electronic ink along the bottom.

Touch screens

Touch screens are very popular in most modern computing devices, especially since the release of Windows 8, the first major desktop operating system focused on providing a touch screen interface. Many laptops, desktops and the majority of mobile phones have a touch screen interface. Car entertainment, satellite navigation devices and printers often use touch screens. There are two types of touch screen, Resistive and Capacitive. Both versions send the X and Y coordinates of the touch to the operating system when a touch is registered. It works using a grid system, with the screen being split up as shown in Figure 10.13. The Y-axis increases in value going down rather than up, which is different from what you may be used to in mathematics.

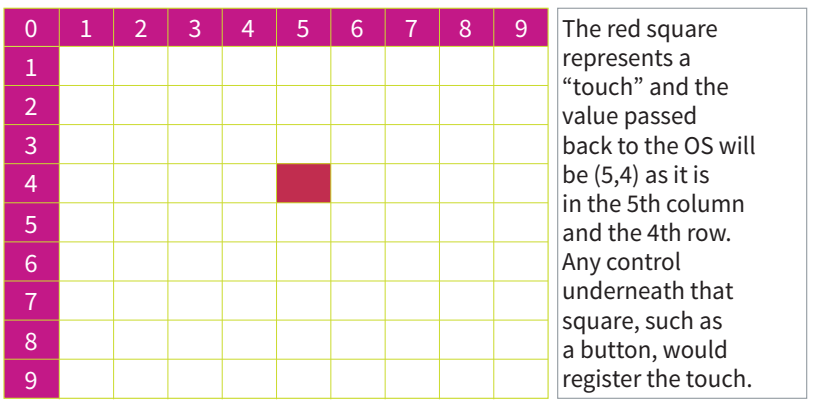


Figure 10.13: A touch screen grid.

Resistive touch screens are much cheaper and are made up of two thin transparent sheets. When these sheets touch each other, a voltage is recorded at that position. A certain amount of pressure is needed to make the sheets touch, which is why this type of touch screen is known as resistive. These screens do not provide as sharp an image as capacitive screens, nor do they allow for multiple touches.

Capacitive touch screens allow for a much sharper image and for multiple touch points to be recorded simultaneously, making them superior to, and more expensive than, resistive touch screens. Capacitive touch screens make use of the fact that the human body can conduct electricity: when you touch a capacitive screen you change the electric field of the area you touched. This change in field registers as a click and sends the X and Y coordinates back to the operating system. One downside, apart from the expense, is that you cannot make use of a capacitive screen when wearing gloves or using items such as a pen, as they do not conduct electricity.

Voice input

Voice input can be used to issue spoken commands to a computer system via a microphone, and the computer will try to interpret the commands and carry them out. An example of this may be a user asking a computer to increase the volume on a media player. Voice input is different to vocabulary dictation as there is a set amount of commands that the computer is able to carry out.

Vocabulary dictation

Vocabulary dictation can be used to input data into a computer system, where a user will speak into a microphone and the computer will try to change the spoken words into typed text. Vocabulary dictation is a popular interface as it is natural for people to communicate in this way. Specialist vocabularies are available with additional words for particular types of users, such as medical and financial users.

Table 10.5 shows some of the common advantages and disadvantages of voice input and vocabulary dictation:

Voiceprint recognition

Voiceprint recognition is the process of capturing a person's voiceprint then digitising and storing this data on a computer system. Voiceprint recognition is often used in security systems, where a person may attempt to gain entry to a high security room. When entry is attempted, the voiceprint of that person is again captured. The two data items are compared, with entry being allowed if there is a match.

Advantages	Disadvantages
• Speech input can be much faster than keyboard input • No need to learn to type • Less danger of RSI • Reduces typing mistakes such as spelling/hitting wrong key • Keyboard takes up room on the desk/ space on a small touch screen • Users with a disability that prevents typing can use speech input • Hands-free advantages – can multi-task • Users find talking more natural than typing.	• Background noise interferes with speech recognition • The software may not cope with all accents of the same language • A user with a speech impediment, sore throat or cold may not be understood • Users with a disability that prevents speech would need to find a different method for input • Difficult to keep data input private as people can hear what you are saying • Words that sound the same (heterographs, such as 'too' and 'two') may be incorrectly interpreted.

Table 10.5: Advantages and disadvantages of vocabulary dictation.

Secondary storage

Computing systems that need to save information will have some form of storage device. Magnetic hard drives (HDD) are very common in desktop and laptop PCs, but increasingly solid state drives (SSD) are being used. Tablet computers, smartphones and some modern laptops or desktops make use of SSDs as their secondary storage device.

Storage devices in a computing system commonly store:

- the operating system and utilities
- user applications
- user documents and files.

Magnetic storage devices

A hard drive is a high-capacity storage medium, common in most modern PCs. Hard drives are used to store the operating system, software and user data. As hard drives have a fast transfer rate and a fairly fast access time, they provide a good compromise between storage capacity, performance and cost. Their speed does not come close to the speed of memory, the CPU or even SSDs.

Hard drives are a magnetic medium and store data on a hard drive platter (Figure 10.15). Data is read and saved using an arm that has a special read/write head at the end. As the disc spins, the arm travels across the disc. Each sector of the platter can store data and the movement of both the disc and the read/write head means that every sector on the hard drive can be reached. In order to read or write data, a small magnetic flux is introduced on the disc. The oxide on the hard drive platter remembers this flux. Data is then encoded using standard binary techniques, with the flux able to store either a 1 or a 0.

The faster the platter spins, the faster data can be read from the disc. This speed is measured in revolutions per minute, or RPM. A common speed for hard drives is 7200 RPM, but it can vary. Every minute that a 7200 RPM hard drive is in use, the platter will have spun 7200 times. Hard drives can run at 15 000 RPM, but these are expensive and

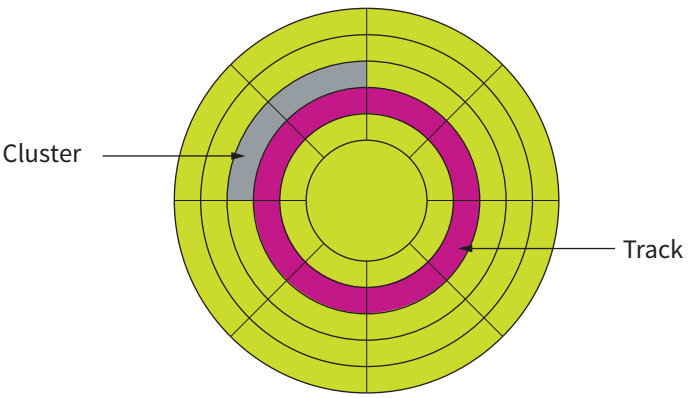


Figure 10.15: Hard drive platter.

tend to be used in servers rather than in home PCs. Laptops sometimes use slower drives to conserve power.

As hard drives provide a pivotal role in a PC, it is essential that they are well maintained. Hard drives do not last for ever and will develop errors over time. These errors are managed by the hard drive and the operating system; however, if there are too many errors, performance can be compromised. Also, these errors can lead to data corruption.

Flash storage devices

SSDs are becoming more popular in modern computing systems and use flash memory to store data rather than magnetic discs. They use a special type of memory that can retain its state once power has been disconnected. It is known as EEPROM or **e**lectronically **e**rasable **p**rogrammable **r**ead-**o**nly **m**emory. SSDs tend to have better performance than hard drives, but also tend to be more expensive and have lower capacity. Performance is increased due to the lack of physical motion. There is no spinning disc or moving read head and thus the time it takes to find data on an SSD is reduced.

Memory sticks and memory cards, such as SD (secure digital), use solid state technologies. They all have the same advantages and disadvantages as SSD drives.

Optical drives

Optical drives work by using lasers to store data by burning microscopic indentations into a disc such as a CD. This pattern of indentations is created in a spiral pattern, starting from the middle. Indentations and their absence create pits and lands (Figure 10.16), which are used to store binary data. A laser is aimed at the disc and reflected back, which can cause interference with the original laser. This change in interference is how the optical device can tell the difference between a pit and a land. DVD-ROM uses the same techniques to store data, but the data is stored on two layers. Two lasers of differing wavelength are used to read data from the two layers. Pits and lands are stored closer together, meaning that the laser's wavelength must be shorter.

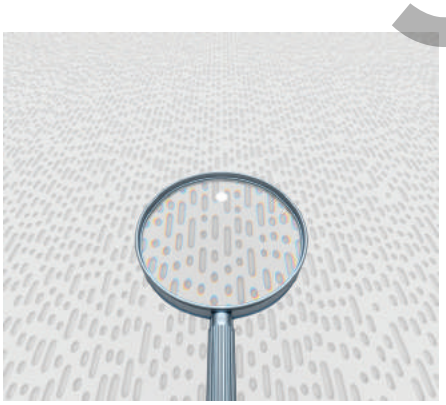


Figure 10.16: Pits and lands on the surface of a CD.

Optical media tend to be used to store multimedia information or programs, for example DVD movies and music CDs. Software such as Microsoft Office is often distributed on this type of media. The capacity, cost and size of these media make them suitable for storing high-definition data.

Blu-ray follows the same principles but uses a laser with a much smaller wavelength. The laser has a distinctive blue colour, which is why it is referred to as Blu-ray. With a smaller wavelength the pits and lands can be much closer together, allowing more data to be stored.

CD-R and DVD-R can store multimedia or user data onto CDs or DVDs. When a CD or CD-R is created, it cannot be modified. As such, it is useful for creating a permanent backup over time. As CDs are physically small, they will not take up too much storage space. Rewritable CDs and DVDs exist; they allow the user to write to them many times but are expensive and slow when compared to memory sticks and memory cards.

Contemporary storage devices

Storage systems for enterprises

Most of the storage devices considered so far are mainly intended for use by small systems or individual end users. Storage requirements for enterprises, however, are much more demanding. When you have over a thousand people trying to access the same set of files at the same time, some serious hardware is needed. A simple hard drive will not be sufficient.

In order to provide scalability and performance, multiple drives that can be brought together to act as a single drive will be needed. Building a custom network-attached storage device (NAS) can be very expensive. It is better to bring many disparate devices together and treat them as a single drive. This is known as virtual storage. Before we look at how virtual storage works, it is important to understand a few of the key technologies used for large-scale storage.

Redundant array of independent discs – RAID

Redundant array of independent discs (RAID) is one of the most common technologies that run behind the scenes. It can be used with both SSD and HDD and allows multiple drives to be brought together as a single drive. NAS devices will commonly support multiple drives and be in some form of RAID.

A RAID array can be in one of the following modes:

- *RAID 0 Striped*: this provides improved performance and additional storage. However, it does not provide any fault tolerance, so any errors on the discs could destroy the RAID.
- *RAID 1 Mirrored*: each disc provides the same information, which provides some fault tolerance. As data is repeated, read speed is increased (because it can be read from any disc), but write speed is decreased (because all discs must be updated).
- *RAID 3–6 Striped parity*: this requires at least three discs in the array. As well as fault tolerance it provides parity checks and error correction. The parity information is stored on a single drive, so the other drives can continue working when one of the drives fails. The lost data can be calculated using the parity data stored on the parity drive. This configuration involves sacrificing storage space to increase redundancy.

Common uses of storage devices

CD-ROMs and DVD-ROMs

CD-ROMs and DVD-ROMs are used to install software and store multimedia. Rewritable versions are expensive and slow and rarely used for backup.

Flash memory drive and portable hard drives

Flash memory drives are used to transport files from one place to another. Portable hard drives (whether magnetic or SSD) are external devices that function in exactly the same way as internal drives. They usually have a plastic or metal enclosure and normally connect to the computer through a USB cable.

Fragmentation and defragmentation

Files stored on computer systems can, over time, become fragmented. This means that they are split and stored on different parts of the disc. If a file is fragmented, it takes longer for the disc heads to move between parts of the file, which slows the process of loading it.

Defragmentation is the process where files are physically re-arranged on disc so that they are no longer fragmented and the parts of each file are stored together. This improves the speed of accessing data from disc.

Unlike magnetic storage devices, SSD uses direct access to data so there would be little point in defragmenting an SSD as there would be no improvement in read times as there's no physical read head to move. As a result, defragmentation may perform "trim" commands which may slightly improve the speed of future write operations. As SSD is currently made out of NAND based flash memory, which has a limited lifespan, the defragmentation process may shorten its lifespan.

Networks

Most computing systems are connected to some form of network, most commonly the internet. Computers may be connected through wired or wireless links. At its most basic, a computer network is a collection of interconnected computers that have the ability to communicate with one another.

The importance of networking standards

What underpins all communication is the idea of protocols. No matter what data is being sent or how the various devices in the network are connected, a set of protocols is required to ensure that both the sender and the receiver understand one another.

Types of network

Not every service or task a user wishes to perform will be done on a single computer. Networked computers can share files, offer services (such as email) and communicate with each other. Even when two PCs share a printer, a network is involved. The biggest network in the world is the internet.

When services are distributed over a network, you have a **distributed** system. A distributed system, in essence, is where computers work together by sharing services with one another in order to provide a complete system. In a distributed system, it is important to remember that both processing and data are distributed.

Networks differ in size depending on where they are located. At home, you will most likely have a small network. This is referred to as a local area network (LAN). Schools have a bigger network than at home, with many more computers connected together. Even though the size of a school network is large when compared to a home network, it is still referred to as a LAN. Local refers to the computers being near each other geographically rather than the number of computers connected to the network.

The internet is a worldwide communications infrastructure, which means that you can access websites, or more specifically web servers, in any country in the world. As the geographical distance between computers on the internet is very great, the internet constitutes a wide area network (WAN).

LANs are inherently more secure than WANs. In a LAN setting, all networked devices are likely to be under the control of a small group of network technicians, whereas WANs

involve many different people. As data travelling over WANs is likely to travel over many different networks owned by many different people, there is a much higher risk of data being intercepted. This form of threat, known as a man-in-the-middle attack, means that unencrypted data could be stolen or altered. This is why you should be careful about what information you send over the internet and why you should also make sure that you only use sites that encrypt your personal data before sending.

Structures for networked services

Two common structures for networks are client–server and peer-to-peer.

Client–server structure

Most distributed systems work on the client–server model (Figure 10.17). Services, such as file storage, are made available on servers to which clients connect in order to access the services. Web servers, which host websites over the internet, are among the most common. Clients use web browsers to access the webpages held on the server.

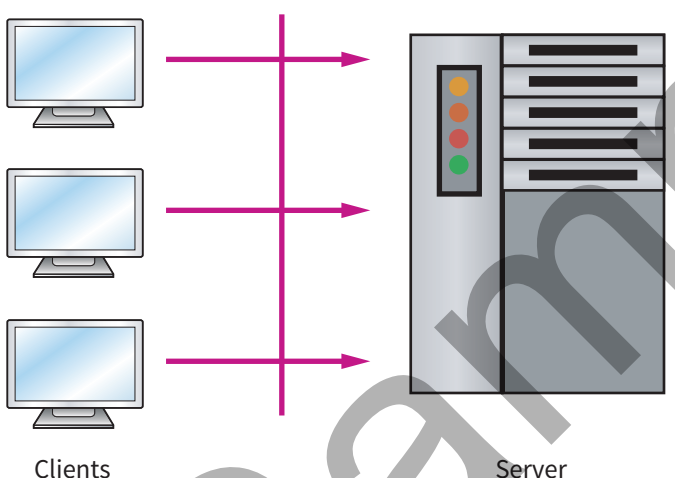


Figure 10.17: Client-server model.

Servers are also known as centralised computing, as services and processing power are held at a central point on a network rather than using the distributed model, where resources are spread across a set of peers.

Clients and servers make use of a request–response model. Clients request a service or resource from the server. The server responds either with a security challenge or with the requested resource. If the server fails for any reason, the service cannot be provided by that server.

Peer-to-peer networks

In a peer-to-peer network, each computer, or peer, has the same status as the others. No computer acts as a server, meaning that peers must work together to fulfil the required task. One of the most common uses of peer-to-peer networking is file sharing. A file required by a number of peers is downloaded in parts. Initially each peer downloads parts of the file, known as leeching, until they have enough of the file to start sharing. Other peers on the network then start downloading the parts of the file from the peers that have already downloaded them, until each peer has full access to every part of the file. As soon as a peer has the whole file, it becomes a seed and continues to share the file until that peer is taken out of the network. Peer-to-peer

networking allows very large files to be downloaded efficiently without the need for a powerful server.

Peer-to-peer technology can also be used to share the processing load for complicated calculations. This form of peer-to-peer networking is known as distributed processing. Difficult problems that require extensive processing time are split up into smaller chunks and these chunks are shared with peers. Each peer then solves part of the problem and returns the result to the peer that is responsible for consolidating and coordinating the calculations. Examples of distributed projects can be found at http://www.cambridge.org/links/kwse6061.

Protocols

When talking to someone who does not speak English as their first language, you may have difficulties being understood. At some point you will have to make a decision to aid communication. Maybe they speak a little English so you decide to use simple English, or maybe you speak a little of their language. It might be possible to mime what you want to say or even to try to draw or use symbols. If you're lucky, you might have a translator handy who could act as a go-between. However you decide to communicate, you will have needed to make a conscious choice of the communication style and decide on a set of rules to govern future communication. Computers, when they transmit data, can do so in many different ways. As such, it is crucial that each computer not only follows a set of rules, but ensures that both sides use the same ones. In computing, a set of rules that governs communication is known as a **protocol**.

In the early days of computers, there were many different ways of doing exactly the same job. This problem became very apparent when computers started to become networked. At that time, there were two main ways of communicating over a network, IPX (internetwork packet exchange) and TCP/IP (Transmission control protocol/nternet protocol). It is not possible for a computer using IPX to communicate with another computer using TCP/IP without making use of special hardware to translate, known as a bridge. IP addresses take the form of four sets of numbers ranging from 0 to 255, while IPX uses a 32-bit hexadecimal address. This is why the choice of protocols is so important: if each computer knows exactly how data should be sent over the network, when it arrives the computer will know exactly how to process it.

Tip

A classic exam question is to define a protocol and then explain why they are used in stacks.

Protocol stacks

There are many different tasks that need to be carried out when sending a packet over a network. How will the packet get there? How will the packet be formatted? Does the packet need to arrive in a specific order? What happens when an error is introduced during transmission? Owing to all the work that needs to be done, it is not practical to place all of it into a single protocol, so a number of protocols are chained together and known as a protocol stack.

Protocol stacks have many advantages:

- Separation of logic so problems in a single protocol can be dealt with in isolation.
- Protocols at different parts of the stack can be swapped out.
- More flexibility when choosing what properties you want your network to have.

The TCP/IP protocol stack is a prime example.

The TCP/IP protocol stack

The TCP/IP protocol stack, which is the protocol suite used in most modern network transmission, is named after the two of the protocols it contains, although it also contains other protocols. Figure 10.18 shows the four key layers of the TCP/IP stack:

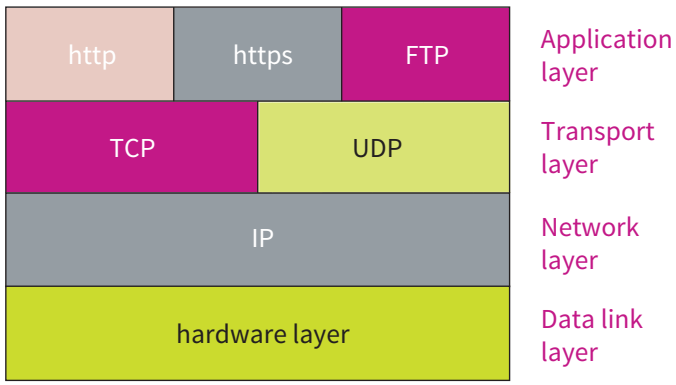


Figure 10.18: TCP/IP protocol stack.

At each layer, apart from the bottom layer, there is a choice of which protocol to use. Each choice provides a different service, which will provide different properties for the packet being sent.

The top layer has a large number of options and is sometimes referred to as the presentation or application layer. Protocols used in this layer are very specific to applications. For example, HTTP (hypertext transfer protocol) is used to transport the HTML of webpages, while FTP (file transfer protocol) is used to send simple files over a network. The transfer protocol used at layer three depends on the application that is running. Having this choice of layer three protocols means that it is very easy to add new protocols without having to change any of the existing structure.

This is possible because there is a common interface between layer three and layer two. So HTTP and FTP create a packet differently, but they set it up in a way that any of the layer two protocols will understand. Each protocol offers different services, but always formats packets so that they can be understood by the other layers. The service access point (SAP) is where protocols will communicate with each other (Figure 10.19) and is generic, so that any protocol can be swapped out without affecting the other protocols in the stack. That way, the application chooses the service it requires, without having to tailor the data for a specific protocol. It is important to note that protocols at both ends of the network must match, otherwise the packet will not be understood.

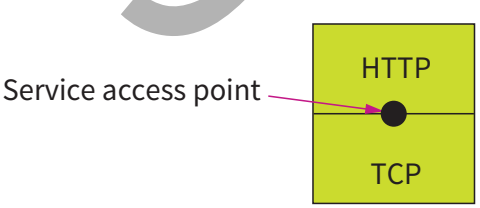


Figure 10.19: SAP.

Figure 10.20 shows how elements of the TCP/IP protocol stack communicate. The principle applies equally to other protocol stacks. Each protocol in the stack communicates along the logical communication lines. Each protocol considers that it communicates directly with its counterpart on the destination machine, and it does so by adding data to a packet's header for its counterpart to read. However, what actually occurs is that information is passed down each protocol, with data being added to the packet's header as it goes, before being placed on the network hardware, which transports it. At the destination, the packet is passed back

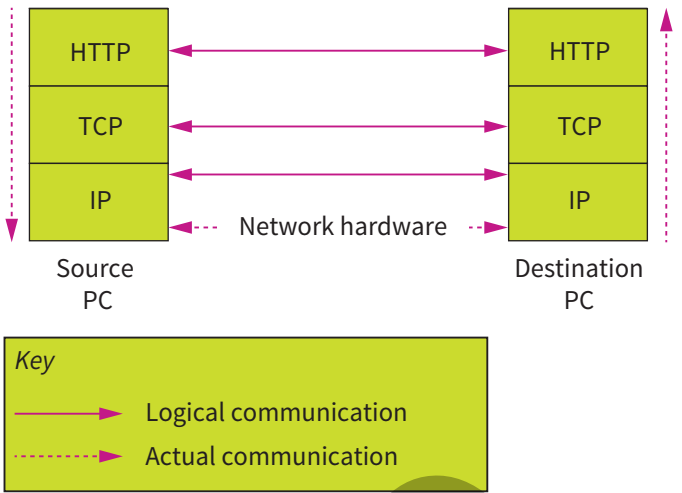


Figure 10.20: Packets travel through different layers of the stack.

up the protocol stack, with each protocol removing a part of the header. The actual path that a packet will travel is denoted by the dashed line in Figure 10.20.

HTTP

Hypertext transfer protocol (HTTP) allows resources, such as images or text files, to be transmitted over the network. Each item is identified using a, a uniform resource locator (URL), and is transmitted over port 80. HTML pages, which are stored as text files, are downloaded using HTTP along with any resources identified within. The web browser then interprets the HTML code and displays the webpage. HTTP is a stateless protocol, meaning that each HTTP request is independent. No data is recorded between requests. This makes interactive websites and web apps more difficult to program, which is why other technologies, such as sessions and AJAX, are becoming common on websites, as they overcome this shortcoming of HTTP.

When an HTTP request is sent, it is structured in the following way:

- Initial line
- Header line 1
- Header line 2 …
- Optional message.

The initial line is different if this is a request to, or a response from, the server. When a request for a resource is sent to the web server, it might look like the line below:

```
GET /path/to/file/index.html HTTP/1.0
```

GET instructs the server to send a file, which is pointed to by the path, which is the URL with the host name taken off. The last part refers to which version of the HTTP protocol should be used. The server may respond with an error code, should a problem arise when it tries to send the resource back. Some of the more common error codes include:

200 – OK!

404 – File not found

302 – Resource has moved

500 – Server error

Header information is sent, which includes information about the browser and other general purpose information that will be used by the server. A lot of this information is used for statistical purposes by the web server, such as recording the most common browser type. The message is the data to be sent back, which will be the contents of the file.

FTP

File transfer protocol (FTP) works on an interactive model and is used when copying (uploading or downloading) a file from one location to another via the internet. FTP remembers information about the state of the connection; that is, each subsequent request makes use of information sent in previous requests. FTP uses port 21.

A common FTP conversation may look like this:

>> OPEN ftp.example.com

<< Which account do you want to access

>> USER bert

<< That account requires a password

>> PASS mypassword

<< welcome

>> CD mydirectory

<< DIR changed

>> LIST

<< Contents of directory myfile.txt

>> RETRIEVE myfile.txt

<< File sent

SMTP

The simple mail transfer protocol (SMTP) is an internet standard for electronic mail (email) transmission. SMTP by default uses TCP port 25 and the protocol for mail submission is the same, but uses port 587. Mail servers use SMTP to send and receive mail messages. Client mail applications, on the other hand, normally use SMTP only for *sending* messages to the server for relaying. For receving messages, IMAP or POP are used.

IMAP

The internet message access protocol (IMAP) is a protocol used for transferring emails between computer systems via the internet. The IMAP protocol is generally used for email retrieval and storage as an alternative to POP. The IMAP protocol, unlike POP, allows multiple clients to connect to the same mailbox simultaneously.

DHCP

The dynamic host configuration protocol (DHCP) is a standardised network protocol used on internet protocol (IP) networks for assigning dynamic IP addresses to devices on a network. With DHCP, computers request IP addresses and networking parameters automatically from a DHCP server, reducing the need for a network administrator or a user to configure these settings manually.

TCP

The transmission control protocol (TCP) provides reliable and error-checked streams of packets over a network and is the core transport protocol for HTTP. Each set of data sent using TCP, known as an octet, contains a **checksum** generated using cyclic redundancy checks (CRC). A sequence number is also included, so that the stream of packets can be reorganised in the correct order at the destination. Port numbers, both source and destination, are included, so that the packet can be correctly routed to the corresponding application.

In order for a packet to be accepted, an acknowledgement field is included in TCP. This is simply the highest sequence number of the last accepted packet. This enables the source to resend data should packets get lost or damaged.

UDP

The user datagram protocol (UDP) provides an unreliable communication channel with a minimal level of features. There is no error checking, ordering of packets or resending of lost packets. UDP is commonly seen as a fire-and-forget protocol. At face value, this may not seem very useful, but UDP has a number of advantages. As it does not use any checksums or ordering, less data has to be sent with each packet, which makes better use of the bandwidth. Also, as there is less processing involved, data will be passed back to the application much faster. Applications that require data fast, but are not concerned about losing data along the way (such as video conferencing applications), are better served by UDP.

IP Address

In order for computers to talk to one another, each one must have a unique address, known as an IP address (internet protocol address). An IPv4 address is made up of a set of four numbers of up to three digits, ranging from 0 to 255, such as 192.168.0.1. When data is transmitted over a network, the IP address of both the sender and the receiver must be included. The destination IP address is needed to ensure that each packet can be routed to the correct destination. The source IP address must be included so that information can be sent back. Most networking communication takes the form of a request and a response, so without the source IP the response could not be sent back to the correct destination.

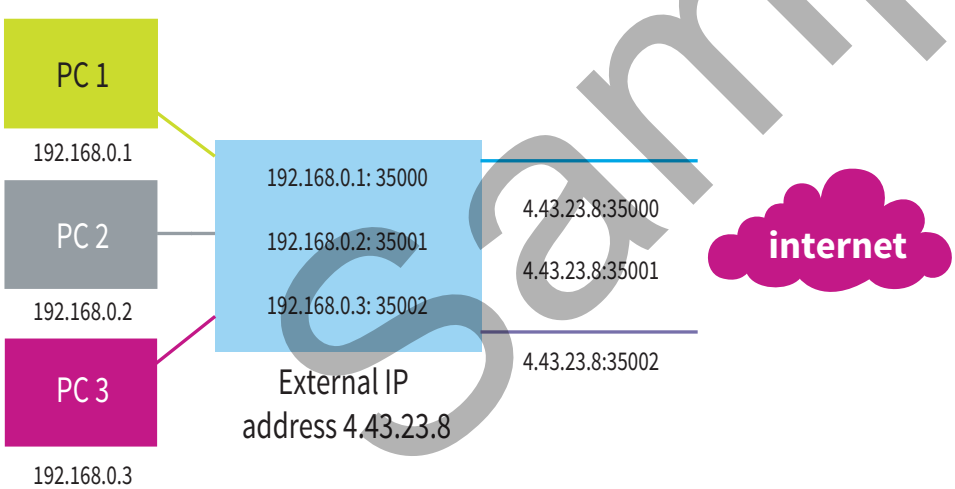
Computing in context: Running out of IP addresses

Each device connected to the internet must have a unique IP address. This uniqueness applies across the world, not just in a single country, which raises a massive problem: there are a limited number of IP addresses available. Version 4 of the internet protocol (IP), which uses four bytes to represent a single address, has a maximum of 4 294 967 296 or 2^{32} unique addresses. This value may seem huge, but considering that the world population is about 7.15 billion (7 150 000 000), not everyone on the planet can be connected at the same time using this version. Matters are made much worse by the fact that an individual may be using multiple devices connected to the internet at the same time, and every computer at work or at school must also have an IP address; it is easy to see how the number of addresses can run out.

This problem has been known for some time and steps have been taken to enable more devices to be connected:

- *Dynamic IP addresses*: Most devices connected to the internet do not have a fixed IP address, known as a static address, but rather a dynamic one. When a device connects to the internet, an IP address is assigned to it by the internet service provider (ISP). ISPs have a set of addresses assigned to them, which they use for their clients. A request is sent to a DHCP (dynamic host configuration protocol) server, which assigns an IP address. By issuing IP addresses on demand, they can be recycled.
- *Network address translation (NAT)*: Private LAN networks can share a smaller number of external IP addresses to connect to the internet by using network address translation, also known as IP masquerading. Each private LAN has an internal IP address range, allowing it to use as many IP addresses as needed. When connecting to the internet, these addresses must be translated to the external IP address. A router holds a table of these IP address translations so that responses can be returned to the sender. This effectively allows an entire LAN, such as a school might have, to share a single IP address. Modern home routers also use NAT to allow a single IP address to be shared by all connected devices.

NAT works by changing the TCP/UDP port numbers of the source IP address. In Figure 10.21, three PCs in a private LAN each have their own IP addresses. When they connect to the internet, these addresses are translated using NAT into the external IP, with each PC having a unique port number. When the result comes back, NAT uses the unique port numbers to translate the address back to the original private IP.



- *IP version 6*: IP addresses specified in this chapter are in accordance with version 4 of the IP protocol. There exists a version 6, which uses 16 bytes per address rather than four. This gives a maximum IP address range of $2128 = 3.4 \times 10^{38}$. This may seem like an obvious solution, but the problem is that devices that use version 4 do not work with version 6. A lot of the internet's infrastructure would have to be replaced, which is very difficult as no one person or organisation owns the internet or has influence over each router. Thus, version 6 rollout is very slow.

Activity 10.2

This task requires you to use Linux. This can be done by installing a Linux distribution of your choice, using a Raspberry Pi or using a virtual machine. The commands listed below will not work on a Windows PC.

1. Run the command ifconfig and write down the MAC address and the IP address of your current computer.
2. Run the command dig www.pwnict.co.uk +short. What does this command do?
3. Install traceroute (sudo apt-get install traceroute).
4. Run the command traceroute www.pwnict.co.uk. Explain the output of this command.
5. Run the command netstat -tp. Explain the result of this output.

Read over the tutorial for programming sockets in Python. A socket is a programming term for making and using a network connection.

Tip

A lot of the theory of handshaking is covered in this chapter. Make sure, where relevant, to make use it in your answers. The exam will normally focus on handshaking for communication over a cable rather than a network.

Handshaking

In order to set up a communication link, both devices need to agree on a set of protocols to use. This must occur before a single byte of data is sent and is known as handshaking. The handshaking signal is sent from one device to the other and is acknowledged by the second device. It is entirely possible that the second device disagrees with the choice of protocols, so either another set must be chosen or communication will fail.

Networking hardware

In order to connect computers together into a network, certain hardware is required. There is a massive range of hardware solutions for networking and the ones described in this chapter are the main ones.

Network interface card

In order for a PC to connect to a computer network, it needs a special piece of hardware called a network interface card or NIC. This hardware is responsible for placing packets onto network cables in the form of electronic signals, or bursts of light if the cable is optical. NICs produce electric signals based on the physical protocol being used on the network, one of the most common data transmission systems being Ethernet. NICs tend to be built into most modern motherboards, so extra hardware is not needed. Home networks make use of Ethernet and twisted pair cables, also known as CAT5.

Wireless interface cards, or WICs, are needed if the connection to the network is wireless. Most devices have WICs built onto their motherboards, with the exception of desktop PCs. Wireless networks make use of the 802.11.x protocol rather than Ethernet. Again, this is a physical protocol and uses electromagnetic waves to transmit data.

Hub

A hub allows more than one computer to be interconnected in a network by connecting them all into one central device or hub. When using a hub, each computer uses a single cable to connect to the hub, which ultimately creates a path between each computer. If a single cable breaks, it affects only the computer connected to the hub by that cable rather than the entire network. When a packet is sent to a hub, it will broadcast the packet to all other computers connected to the hub. A single hub, depending on the size, can have up to 48 PCs connected. If 48 computers were connected to a single hub, then, when a packet arrived, that packet would be repeated to all 48 connected computers. Internally, hubs are bus networks, with the bus being inside the hub.

Hubs work well for small networks with low network traffic, but the more PCs are connected, the more traffic will be generated, increasing the number of packets colliding or getting corrupted. Bandwidth is an issue with hubs, so it is not advisable to have too many devices connected to a single hub, nor is it sensible to chain them together. However, they are much cheaper than switches or routers, which means that they do have a part to play in most networks. A common approach is to use a hub to connect a small number of devices together, for example one room in a school, and then connect the hub to a switch so that some level of routing can be achieved.

Switch

Switches, unlike hubs, perform routing of packets from one port to another. A port on a networking device is the physical RJ45 connector for a networking cable, as shown in Figure 10.21. When the packet arrives, the switch determines the port on which the destination address exists; this process is known as routing. In order to ensure that packets are sent to the correct destination, a picture of the network must be built up in the form of a table.

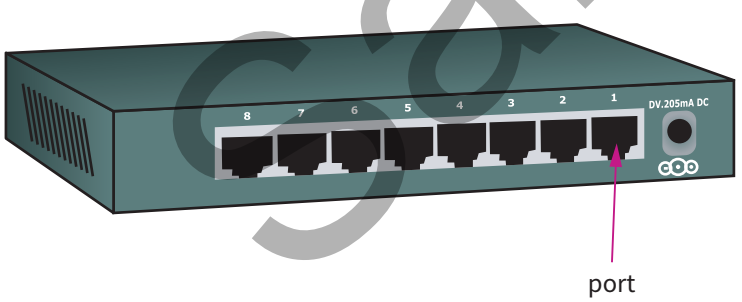


Figure 10.21: Network hub.

Switches perform routing at the MAC (media access control) level rather than at the IP level. MAC addresses are hard-coded into each device by the manufacturer, and take the format of 12 hexadecimal values. Every MAC address is unique, and while IP addresses can be changed, MAC addresses cannot. When a packet arrives at a switch, the switch looks at the packet's MAC address and forwards it on to the correct port. Initially the switch does not know which port a specific device is connected to, so it has a blank routing table. In this scenario the switch broadcasts a discovery packet to ascertain which port, if any, that networked device is on. When a device receives a discovery packet, it responds, allowing the switch to make a note of the port the reply came back on. A sample table (Table 10.6) is shown below:

Port number	MAC address
1	00:00:AE:3B:09:FF
2	00:00:4B:BC:00:E1
3	00:1A:D7:B6:CA:F2

Table 10.6: Table showing the MAC address of devices connected on various ports of a switch.

Router

Routers perform a very similar job to switches: they route packets to the correct port. Like a switch, they have an internal representation of which devices can be found on each port through the routing table. However, there are differences between the devices and they are used for very different purposes. Routers are used to connect different segments of a LAN together and allow access to the outside networks such as the internet.

Routers perform routing on IP addresses rather than MAC addresses. Sections of a network, known as sub-domains, can be assigned similar IP address ranges, which provide the router with more information to help it direct the packet. When connecting different parts of a LAN or WAN together, a router ultimately has to deal with more devices than a switch does. This is why routers tend to have a more powerful processor and are more expensive. Routers are necessary when connecting to any WAN, which is why the device you may use at home is a router.

Wireless interface card

A wireless interface card (WIC) connects to a wireless access point (WAP). A WAP is a device that allows wireless devices to connect to a wired network using Wi-Fi or similar standard. A WAP has a maximum number of devices that it can support and a physical broadcast range where connections are possible. It is possible to detect a wireless network by sitting outside a company's premises, which is why security on a wireless network is a top priority.

Wireless networks use encryption and require a special key in order to stop unauthorised people accessing the network. Home routers normally offer wireless connection and tend to have security set up by default. This is intended to protect data sent over the router, but the level of protection is only as strong as the encryption methods used. Some wireless security protocols, such as WEP, can be cracked very easily by a determined hacker. WPA2 is one of the more secure wireless protocols available at present.

Wireless networks minimise the amount of cable needed when setting up the network, but are more error prone. Distance from the router, interference from other devices and even bad weather can impact the speed and quality of connection. When a wireless signal is low, packets may contain errors and the data will need to be resent.

There are two main wireless technologies in use at the time of writing, 802.11 and Bluetooth. Both have their uses, but only 802.11 is used for networking. Bluetooth is used for connecting external devices, such as wireless handsets, to other devices.

Chapter summary

- A CPU is made up of the following components:
  - ALU – arithmetic logic unit
  - Registers
  - CU – control unit.
- A Machine code instructions are written in binary and are made up of an opcode and data.
- Little Man Computer a is simplified architecture used to help learners understand the basics of low-level programming.
- The fetch–decode–execute (FDE) cycle will fetch instructions from memory to be run on the CPU. During the cycle the following registers are used:
  - MAR – memory address register, which stores the address of the instruction to fetch
  - MDR – memory data register, which stores the data or instruction once fetched
  - PC – program counter, which stores the address of the next instruction to fetch
  - CIR – current instruction register, from where the instruction will be decoded and executed.
  - ACC – accumulator, which is used to store the results of a calcultion
- Cache is fast memory that sits between memory and the CPU. It is used to help remove the impact of the von Neumann bottleneck.
- Processors, to improve the throughput of instructions, perform parts of the FDE cycle concurrently, which is known as pipelining.
- A multi-core system is a common architecture where processors have more than one core, allowing multiple processes to be run concurrently.
- A Parallel systems are made up of multiple processors which can perform multiple operations concurrently. Code must be written in such a way as to take advantage of any parallelism.
- Random-access memory (RAM) stores the currently running programs, operating systems and user files.
- Read-only memory (ROM) stores the boot-up software required to initialise the hardware, load settings and initiate the operating system (OS).
- Input devices provide data for processing and come in the form of scanners, optical character recognition, optical mark recognition, barcodes, magnetic ink recognition, touch screens and sensors.
- Storage devices store data and programs when the computer is powered down. The main types of storage device are:
  - magnetic – back-up tape and hard drives
  - optical – CD, DVD and Blu-ray
  - solid state – solid state hard drives, memory sticks and secure digital (SD) cards.
- Redundant array of independent discs (RAID) arrays are used in servers and for more data-critical systems. They allow multiple drives to be connected together to form a single drive.
- RAID arrays can provide fault tolerance in the form of mirroring and parity bits.
- A Storage area networks (SAN) can connect numerous different types of storage device as a single device.
- Virtual storage will abstract the type of storage device in a SAN, allowing devices to be added and removed without having to administer the individual devices.

- Networks rely on the use of protocols and standards to ensure that communication between devices can occur even if those devices were produced by different companies.
- Client–server-based architecture is where a powerful computer (known as a server) provides functionality to the rest of the network (a service). This can include providing files, websites or databases.
- Peer-to-peer networks have no server; rather each computer has the same status as the other. These networks are commonly used for file sharing.
- Error detection ensures that packets arrive at their destination without corruption. Common error detection methods include:
  - parity bits: adding additional bits to make the number of ones odd or even, depending on the type of parity chosen
  - echo: packets are returned from the destination and then compared by the sender
  - checksum: calculations are performed on the packet generating a checksum that is sent with the packet; this calculation is repeated and the checksums compared.
- Protocols are a set of rules that govern communication.
- Handshaking is a process that devices undertake when a connection is set up. This is where the set of protocols to be used is agreed.
- Protocol stacks allow the separation of logic for each protocol, enabling different parts of the stack to be swapped.
- Transmission control protocol/internet protocol (TCP/IP) stack is the most common protocol stack used in networking.
- Hypertext transfer protocol (HTTP) is used for the transmission of webpages and webpage components. HTTP uses port 80.
- HTTPS (secured) is a secure version of HTTP and communicates over port 443. Certificates and the secure sockets layer (SSL) protocol are used to secure communication.
- File transfer protocol (FTP) uses port 21 and allows files to be transmitted.
- The transport layer of TCP/IP uses either TCP or UDP:
  - TCP provides a streamlike communication with error detection, ordering of packets and confirmation that packets have been received.
  - UDP (user datagram protocol) is a fire-and-forget approach with minimal protections.
- IP is a network layer protocol that provides routing. It uses the end-to-end principle where each node is considered to be unreliable and therefore each node is asked to perform routing.
- Common networking hardware includes:
  - network interface card: allows a computer to connect to the network
  - hub: connects multiple computers and packets are broadcast to all connected devices
  - switch: allows multiple devices to be connected and provides routing by media access control (MAC) address
  - router: connects different networks together and uses IP addresses for routing; commonly used to connect to the internet.

A

End-of-chapter questions

1 Explain the purpose of each of the following items of networking hardware.

   a Hub [2]

   b Switch [2]

   c Router [2]

2 Describe, using specific protocol examples, the TCP/IP protocol stack. [8]

3 Describe how the FDE cycle is used by the CPU and how pipelining can be used to make the process more efficient. [8]

Further reading

OSI model (reference model for networking)	Search for the OSI model on tech-faq.com
Warriors of the net (video introduction to networking)	Go to warriorsofthe.net
HTTP protocol (request fields definition)	Search for HTTP request fields on w3.org
Microsoft introduction to TCP/IP	Search for Introduction to TCP/IP on the Microsoft TechNet website
Socket programming introduction in Python	Search for Python Sockets on codingtree.com