4 Operating systems and computer architecture

In this chapter you will learn about:

- operating systems
- interrupts and buffers
- computer architecture and the von Neumann computer model
- the fetch–execute cycle.

4.1 Introduction

All modern computers have some form of operating system which users generally take for granted. The operating system makes it possible to communicate with the software and hardware that make up a typical computer system.

There are many ways of representing computer architecture, but one of the more common ones is known as the von Neumann model which will be fully described in this chapter.

4.2 Operating systems

The OPERATING SYSTEM (OS) is essentially software running in the background of a computer system. It manages many of the basic functions which are shown in Figure 4.1. Obviously not all operating systems carry out everything shown in the figure but it gives some idea of the importance and complexity of this software. Without it, most computers would be very user-unfriendly and the majority of users would find it almost impossible to work with computers on a day-to-day basis.

Figure 4.1

One of the most common examples of an operating system is known as Windows and is used on many personal computers. Other examples include: Linux, Android, UNIX and DOS. Windows is an example of a single-user multitasking

operating system – this means only one user can use the computer at a time but can have many applications open simultaneously. How operating systems actually work is beyond the scope of this textbook.

When a computer is first powered up, the initiating programs are loaded into memory from the ROM (read only memory) chip. These programs run a checking procedure to make sure the hardware, processor, internal memory and bios (basic input–output system) are all functioning correctly. If no errors are detected, then the operating system is loaded into memory.

It is worth mentioning here that simple devices with embedded microprocessors don't always have an operating system. Household items, such as cookers, microwave ovens and washing machines only carry out single tasks which don't vary. The input is usually a button pressed or a touchscreen option selected which activates a simple hardware function which doesn't need an operating system to control it.

Activity 4.1

Find out how appliances fitted with microprocessors can be controlled and activated by web-enabled devices such as smart phones.

4.3 Interrupts and buffers

An INTERRUPT is a signal sent from a device or from software to the processor. This will cause the processor to temporarily stop what it is doing and service the interrupt. Interrupts can occur when, for example:

- a disk drive is ready to receive more data
- an error has occurred, such as a paper jam in a printer
- the user has pressed a key to interrupt the current process – an example could be <CTRL><ALT><BREAK> keys pressed simultaneously
- a software error has occurred – an example of this would be if an .exe file couldn't be found to initiate the execution of a program.

Once the interrupt signal is received, the processor either carries on with what it was doing or stops to service the device/program that generated the interrupt.

Interrupts allow computers to carry out many tasks or to have several windows open at the same time. An example would be downloading a file from the internet at the same time as listening to some music from the computer library. Whenever an interrupt is serviced, the status of the current task being run is saved. This is done using an INTERRUPT HANDLER and once the interrupt has been fully serviced, the status of the interrupted task is reinstated and it continues from the point prior to the interrupt being sent.

BUFFERS are used in computers as a temporary memory area. These are essential in modern computers since hardware devices operate at much slower speeds than the processor. If it wasn't for buffers, processors would spend the majority of their time idle, waiting for the hardware device to complete its operation. Buffers are essentially filled from the processor or memory unit and whilst these are emptied to the hardware device, the processor carries on with other tasks. Buffers are used, for example, when streaming a video from the internet. This ensures that the video playback doesn't keep on stopping to wait for data from the internet.

Buffers and interrupts are often used together to allow standard computer functions to be carried out. These functions are often taken for granted by users

of modern computer systems. Figure 4.2 shows how buffers and interrupts are used when a document is sent to a printer.

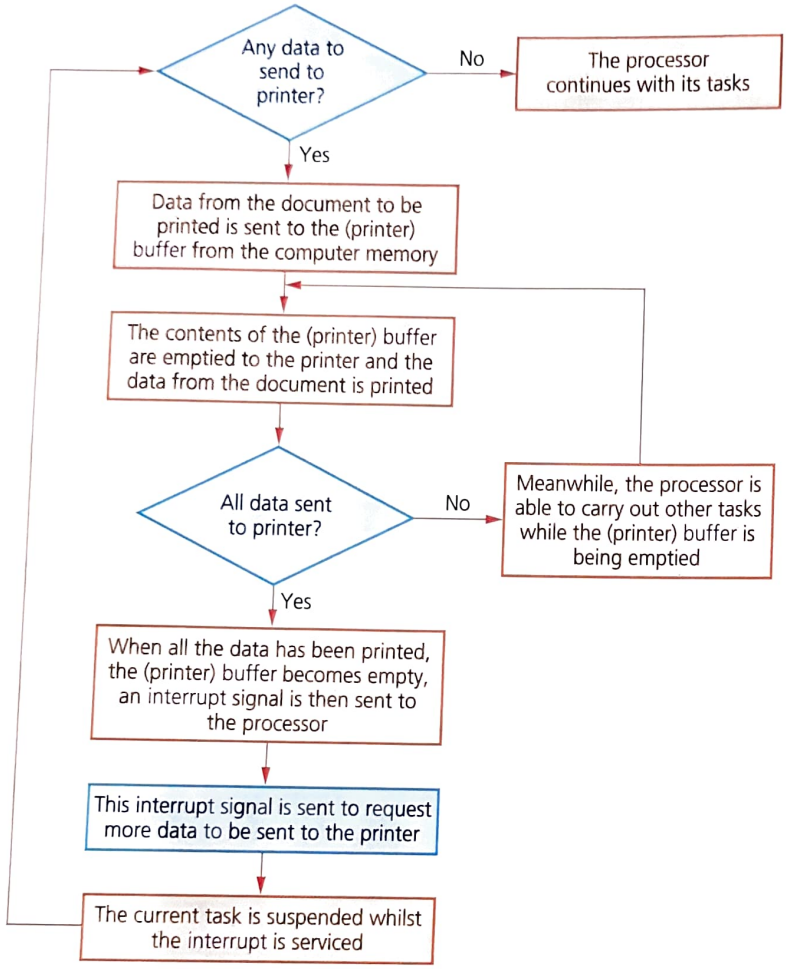


Figure 4.2

Activity 4.2

Find out how buffers and interrupts are used when sending data to memories such as DVDs and solid state (e.g. pen drive).

Activity 4.3

Find out how buffers are used when streaming a video or music from the internet to your computer.

Activity 4.4

Investigate the many ways that hardware and software can cause an interrupt to occur. Are ALL interrupts treated equally or do some have priority over others?

4.4 Computer architecture

Very early computers were fed data whilst the machines were actually running. They weren't able to store programs; consequently, they weren't able to run without human intervention. In about 1945, John von Neumann developed the idea of a stored program computer, often referred to as the VON NEUMANN ARCHITECTURE concept. His idea was to hold programs and data in a memory. Data would then move between the memory unit and the processor.

There are many diagrams which show von Neumann architecture. Figures 4.3 and 4.4 show two different ways of representing this computer model. The first diagram is a fairly simple model whereas the second diagram goes into more detail.

Figure 4.3 shows the idea of how a processor and memory unit are linked together by connections known as BUSES. This is a simple representation of von Neumann architecture. These connections are described in Table 4.1.

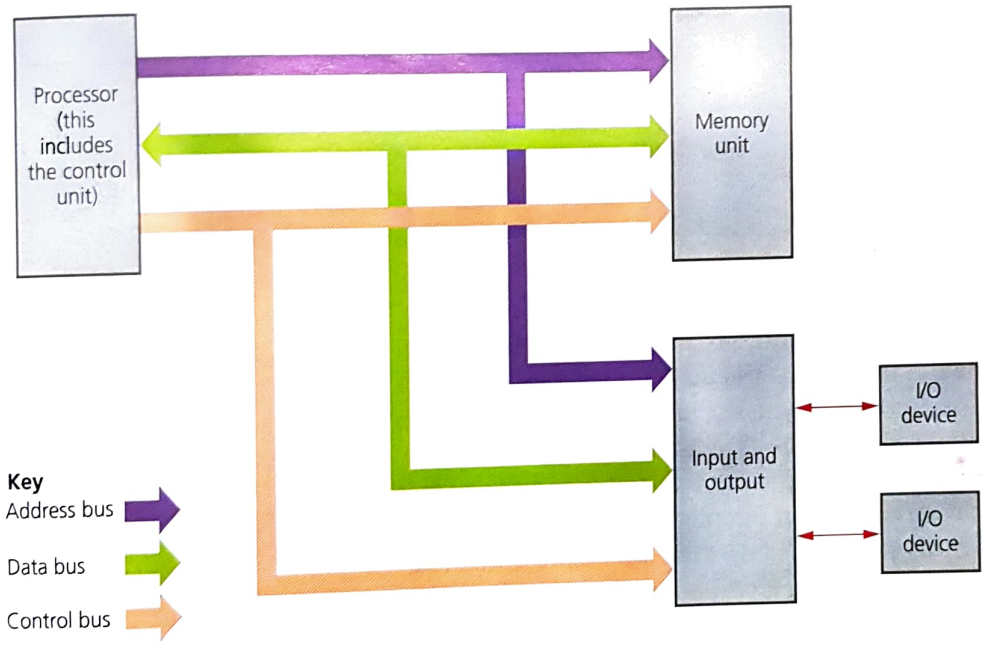


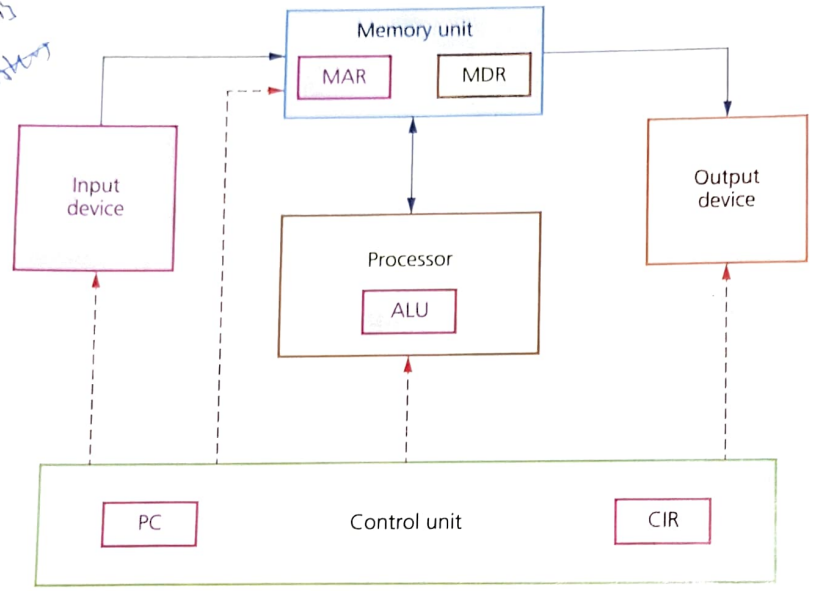
Figure 4.3

Table 4.1 describes the function of each of the three buses shown in Figure 4.3. Buses essentially move data around the computer and also send out control signals to make sure everything is properly synchronised.

Table 4.1

Type of bus	Description of bus	Data/signal direction
address bus	carries signals relating to addresses (see later) between the processor and the memory	**unidirectional** (signals travel in one direction only)
data bus	sends data between the processor, the memory unit and the input/output devices	**bi-directional** (data can travel in both directions)
control bus	carries signals relating to the control and coordination of all activities within the computer (examples include: the read and write functions)	this is regarded as being both **unidirectional** and **bi-directional** due to the internal connections within the computer architecture

Figure 4.4 shows a slightly more detailed diagram of the von Neumann architecture. It brings to our attention another new concept in this computer model – the idea of ADDRESSES and REGISTERS. Addresses indicate where the data is stored and registers are needed so that data can be manipulated within the computer.



Please note: registers are shown in the diagram as a form of representation and don't actually reflect computer architecture. This is a simplification to help understand the following chapter notes.

Figure 4.4

An address is the location of where data can be found in a computer memory. Each address in the memory is unique. The addresses aren't actually shown in Figure 4.4, but they are contained in the part of the diagram labelled memory unit. The actual function of the addresses is discussed in Section 4.4.1.

You will notice a number of items shown in the diagram known as registers: MAR, MDR, ALU, PC and CIR. These are a little more complex and their function will be fully described in the following pages. But essentially a register is simply a high-speed storage area within the computer. All data must be represented in a register before it can be processed. For example, if two numbers are to be added, both numbers must be stored in registers and the result of the addition must also be stored in a register.

Activity 4.5

Draw up a summary table that shows how buses, registers and addresses are all connected together. This can be done by doing a desk-top exercise:

- Use a sheet of A0 (flipchart size) paper and draw a large outline of the FIVE main components shown in Figure 4.4.
- Cut coloured arrows out of cardboard to show the buses and place on the diagram you've just drawn; the data and control buses will be easy to fit.
- However, you will need to do a little bit of thinking to decide how the address bus fits into your diagram.
- Cut the five registers out of cardboard and place them on the large A0 outline.
- Above the memory unit, place an example of some memory addresses and their contents.
- Cut out some 8-bit binary numbers representing your addresses and contents.
- Finally by moving data around you should be able to see how buses, addresses and registers all interconnect; you get further help in doing this desk-top exercise when you read Sections 4.4.1 to 4.4.4.
- Try to find out how simple operations like addition can be carried out using your desk-top exercise.

4.4.1 Memory unit

The computer memory unit is made up of a number of partitions. Each partition consists of an ADDRESS and its CONTENTS. The example shown here uses 8 bits for each address and 8 bits for the content. In a real computer memory, the address and its contents are actually much larger than this.

The address will uniquely identify every LOCATION in the memory and the contents will be the binary value stored in each location.

Table 4.2

Address	Contents
1111 0000	0111 0010
1111 0001	0101 1011
1111 0010	1101 1101
1111 0011	0111 1011
↓	↓
1111 1100	1110 1010
1111 1101	1001 0101
1111 1110	1000 0010
1111 1111	0101 0101

Figure 4.4 showed five examples of registers. Their function will be described later on, but for now, the abbreviations stand for:

- MAR memory address register
- MDR memory data register
- ALU arithmetic and logic unit
- PC program counter
- CIR current instruction register.

Let us now consider how the two registers (MAR and MDR) shown in the memory unit are used.

Consider the READ operation. We will use the memory section shown in Table 4.2. Suppose we want to read the contents of memory location 1111 0001; the two registers are used as follows:

- The address of location 1111 0001 to be read from is first written into the MAR (memory address register):

MAR:	1	1	1	1	0	0	0	1

Figure 4.5

- A 'read signal' is sent to the computer memory using the control bus.
- The contents of memory location 1111 0001 are then put into the MDR (memory data register):

(Look at Table 4.2 to confirm this.)

MDR:	0	1	0	1	1	0	1	1

Figure 4.6

Now let us consider the WRITE operation. Again we will use the memory section shown in Table 4.2. Suppose this time we want to show how the value 1001 0101 was written into memory location 1111 1101.

- The data to be stored is first written into the MDR (memory data register):

Figure 4.7

- This data has to be written into the memory location with the address 1111 1101; so this address is now written into the MAR:

Figure 4.8

- Finally, a 'write signal' is sent to the computer memory using the control bus and this value will then be written into the correct memory location. (Again confirm this by looking at Table 4.2.)

Activity 4.6

Use your model created in Activity 4.5 to show how the two operations, read and write, are carried out in the von Neumann architecture.

4.4.2 Processor

The PROCESSOR contains the ARITHMETIC AND LOGIC UNIT (ALU). The ALU allows arithmetic (e.g. add, subtract, etc.) and logic (e.g. AND, OR, NOT, etc.) operations to be carried out. How this is used is explained in Section 4.5.

4.4.3 Control unit

The CONTROL UNIT controls the operation of the memory, processor and input/output devices.

Essentially, the control unit reads an instruction from memory (the address of the location where the instruction can be found is stored in the Program Counter (PC)). This instruction is then interpreted. During that process, signals are generated along the control bus to tell the other components in the computer what to do.

How this all fits together with the other components of the von Neumann model is discussed in Section 4.5.

4.4.4 Input and output devices

The input and output devices are discussed in Chapter 5 and are the main method of entering data into and getting data out of computer systems. Input devices convert external data into a form the computer can understand and can then process (e.g. keyboards, touchscreens and microphones). Output devices show the results of computer processing in a humanly understandable form (e.g. printers, monitors and loudspeakers).

How all of these five registers are used by a typical computer system can be found in Section 4.5 – the fetch–execute cycle. Again other textbooks and websites will all have different ways of showing how this cycle operates. The important thing is to get a good general understanding of how the cycle functions.

4.5 The fetch–execute cycle

To carry out a set of instructions, the processor first of all FETCHES some data and instructions from memory and stores them in suitable registers. Both the address bus and the data bus are used in this process. Once this is done, each instruction needs to be decoded before finally being EXECUTED. This is all known as the FETCH–EXECUTE CYCLE and is the last part of this puzzle.

The CURRENT INSTRUCTION REGISTER (CIR) contains the *current* instruction being procesed. THE PROGRAM COUNTER (PC) contains the address of the *next* instruction to be executed.

Fetch

In the fetch–execute cycle, the next instruction is fetched from the memory address currently stored in the Program Counter (PC) and is then stored in the Current Instruction Register (CIR). The PC is then incremented (increased by 1) so that the next instruction can be processed.

This is then decoded so that each instruction can be interpreted in the next part of the cycle.

Execute

The processor passes the decoded instruction as a set of control signals to the appropriate components within the computer system. This allows each instruction to be carried out in its logical sequence.

Figure 4.9 shows how the fetch–execute cycle is carried out in the von Neumann computer model. As with most aspects of computer science, there will be slight variations on this diagram if other textbooks or websites are consulted. The main aim is to end up with a clear understanding of how this cycle works.

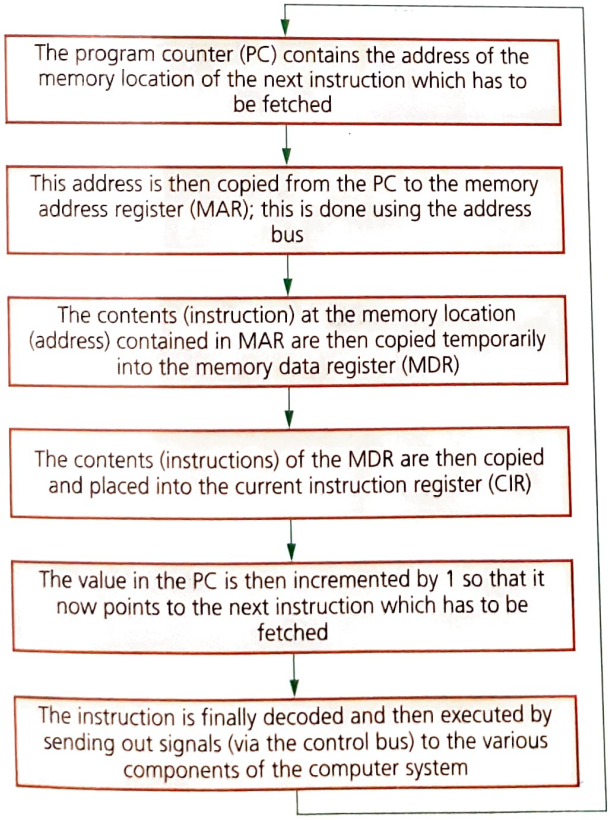


Figure 4.9

Figure 4.9 shows the actual stages that take place during the fetch–execute cycle, showing how each of the registers and buses are used in the process (the first five boxes are part of the fetch cycle and the last box is part of the execute cycle).

Activity 4.7

- It is worth trying to do a desk-top exercise to carry out a series of instructions using the fetch–execute cycle. Do some research and decide on four or five operations (add, subtract, etc.) and give the 8-bit binary codes. Then create a memory map as shown in Activity 4.5.
- Once this is done, use the five registers you created in Activity 4.5 and new 8-bit binary values to trace out exactly what happens as instructions are fetched and executed in your computer model.