Chapter 21 - File organisation

21.1 Introduction

Data is stored in files on storage devices, for example, on hard drives and magnetic tape. There is a choice of storage devices and there is also a choice in the way that files are organised on the storage device. Files can be organised so that a piece of data in the file can be accessed **directly**, without going through other pieces of data to get to it. This is a fast method of data access. Files could also be organised so that they are accessed **serially** or **sequentially**. This means that to find a file, you start at the beginning of the file and go through all the records in turn until you get to the one you want. This is a slow method of data access but has other benefits. When deciding what file organisation to use and what storage device to use, the key questions to ask are:

- How quickly do I need to get back a particular piece of data?
- What do I want to do with the data?

There are five ways to organise files of data that we will discuss:

1) Serial files
2) Sequential files
3) Index sequential files
4) Random files
5) Direct-access file.

21.2 Serial files

Data stored in a serial file is stored in **time order** only. When a new piece of data is added to the file, it is simply added on to the end of the file. If you had a serial file with 10000 items in it, they would be in the order that they were received on the storage device. This makes getting back any individual piece of data a rather slow process. The only way you can retrieve a particular piece of data is to start at the beginning of your file and compare each data item in turn with the one you want.

Data access with serial files is slow and is therefore not suitable for applications that require fast access to data. You wouldn't want to use this system of organisation, for example, for a driving licence database used to deal with customer enquiries. If a driver rang up DVLC (the driving licence centre) and wanted to query their details, it would take a very long time to search serially the whole database! Some applications, however, don't require fast access to data. Here are some examples.

21.2.1 Backups

Most networks backup the users' data onto magnetic tape during the night. The data will generally not be needed although occasionally a user will need to get back a copy of a file that they have accidentally deleted and very occasionally the whole network will need to be recovered after a system crash.

21.2.2 A shop

A shopkeeper might keep a record of each transaction made in the shop in time order (a serial file) over a 24-hour period. This file will hold details of each sale; what was sold and how many of each item. This file (also known as a 'transaction file') can then be used at the end of the 24-hour period to update the 'master file' of products. The master file is a record of the products and what is in stock. It doesn't change except when it is updated using the transaction file. Once updated, the stock control system can then be used to automatically re-order items.

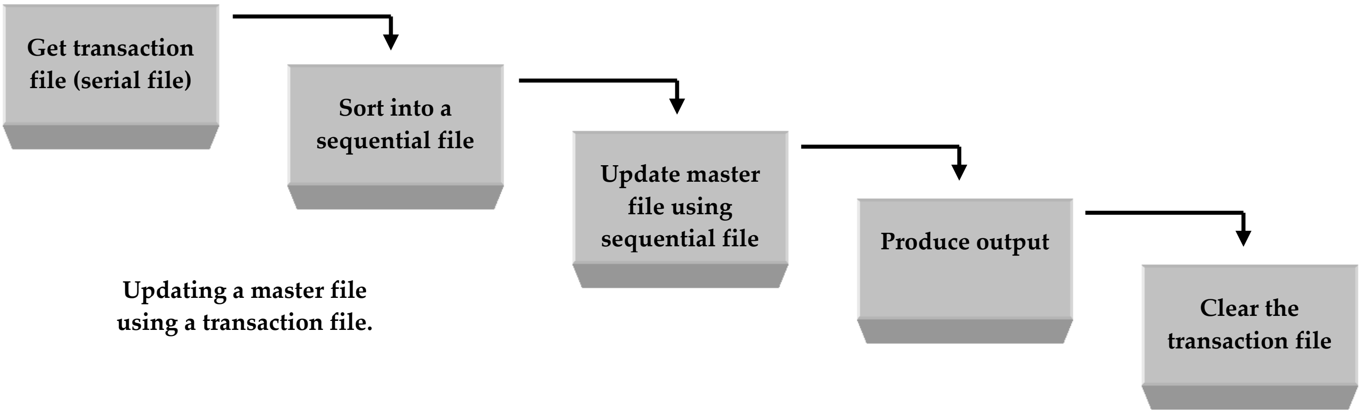
It is also worth noting that updating the master file using the transaction file is done using batch processing. In batch processing systems, all the data is collected together in one place (in the transaction file) before being processed. Once a transaction file has been used to update the master file, the transaction file can be cleared out. It will then be ready for a new transaction period. Master files are permanent. Transaction files exist only for the transaction period, and then they are cleared. It is also worth noting here that while a serial file can be used for this application, a more efficient way to process the transaction file would be to re-organise it into a sequential file first. This is discussed in the sequential file section later in this chapter.

21.2.3 An Automatic Teller Machine (ATM)

An ATM machine keeps a serial file of all the transactions made at the machine during a period of time, perhaps the last 24 hours - it keeps a 'transaction file'. Every time a user withdraws money, orders a new chequebook and so on, it is recorded in time-order. In the small hours of the night, however, the ATM will link to the bank's main computer and will batch process the transaction file to update the master file - everyone's bank details gets updated in one go. In addition, if there is a query from a customer about a transaction, then the serial file can be referred to for details of the transaction.

21.2.4 A payroll system

When a company's employees 'clock-on' or 'clock off', a record of this event is stored in a serial file set up for this purpose. The entry in the file will record the employee number and the time and date they clocked on or off. At the end of the week this serial file (take note that it is also known as a 'transaction file') is sorted into a sequential file - sorting the serial file into a sequential file would collect together each employee's entries, which are scattered throughout the serial file, and put them together and into the order that the employee records are stored on the master file, probably by employee number. The sequential file is then batch processed with the master file (the master file contains an up-to-date record of each employee, what their hourly rate is and a running total of their pay and deductions for the current tax year). The processing in this example means that the number of hours each employee has worked in total for the last week is calculated. The pay due is then worked out and the deductions (tax, NI, pension contributions and so on) made. Pay slips are printed out. The transaction file is then cleared, ready for the next period of record collection. The last job is to back up the master file and store it securely, ideally off-site.


Get transaction file (serial file)
Sort into a sequential file
Update master file using sequential file
Produce output
Clear the transaction file

Updating a master file using a transaction file.

21.3 Sequential files

Serial files are organised by time. There are situations where this may not be the best way to organise data. Consider the above example involving a shopkeeper. Over a 24 hour period, the same products will be bought many times yet the details of each transaction involving the same product will be scattered throughout the serial file. When the serial file is processed, it will clearly need to return time after time to one product's details in the master file every time it comes across a transaction for that product. Having to constantly re-visit master file entries isn't very efficient! It would be better if the serial file, before the master file was updated, were processed to put all of the transactions for each of the products together. Not only that, however, it would be useful if the records were in the same order as the records in the master file. Then a product could be accessed once on the master file and all of the transactions that have taken place involving that product could be processed in one go. And since the sequential file and the master file records are in the same order, the records will be processed together in the most efficient way. You won't get a record from the master file and then have to hunt through the sequential file.

A file that is in some kind of order other than time order is known as a **sequential file**. In the above example, the master file might be a file of product details held in product ID order. We should therefore create a sequential file organised by product ID by processing the serial file, ensuring that all the product transactions for each product are together. We would then batch process the sequential file to update the master file.

21.3.1 What media can be used for serial and sequential files?

Both serial and sequential files are accessed by going to the beginning of the file, reading the first piece of data, then going to the next, and then the next and so on until the end of the file. Serial and sequential files can be stored on magnetic tape as well as media such as CD R/W and hard disks.

21.4 Index sequential files

If you store files on a magnetic tape, for example, to back up a hard drive then the files on the tape will be either in serial or sequential order. You cannot go directly to a data item but have to go through all the other data items to get to it. This means data access can be slow, especially if there are lots of records. There are many occasions when you need fast access to some data. You first of all need to select a storage medium that allows you to access directly areas of data such as a hard disk or CD R/W (but not magnetic tape) and then you need a file structure that allows you to go straight to some data.

21.4.1 An enquiry system

Index sequential is a method that speeds up access to data. It does this by taking a sequential file and splitting it up into areas. Each block of data is stored in its own area. An index is then provided that points to each area. For example, suppose you had to design a database that allowed you to retrieve details about authors. You would get your file of authors and put them in a sequential order. At the beginning of the file, you might create an index, like this:

- For authors beginning with A, go to address 23000.
- For authors beginning with B, go to address 24000.
- For authors beginning with C, go to address 25000 and so on.

When someone wants to get back details of an author, they:

- type in the author's surname
- the first letter of the author is stripped out and examined
- the letter is looked up in the index
- the computer jumps to the address that corresponds to the letter
- a sequential search begins from that address, until the author is found or the end of the file is reached.

Note that the file **must** be in sequential order.

21.4.2 An employee database

You could split all of the employees for a company into blocks, and provide an index in much the same way as the above example. When an employee makes an enquiry that requires their record to be brought up on the screen, this can be done quickly because the file is arranged in an index sequential structure. There may be times, however, when the company needs to use all of the records in the file to batch process, for example, pay slips. The index sequential file is ideally organised for this as well, since all the employees are in alphabetical order. Index sequential files, then, are especially ideal if you:

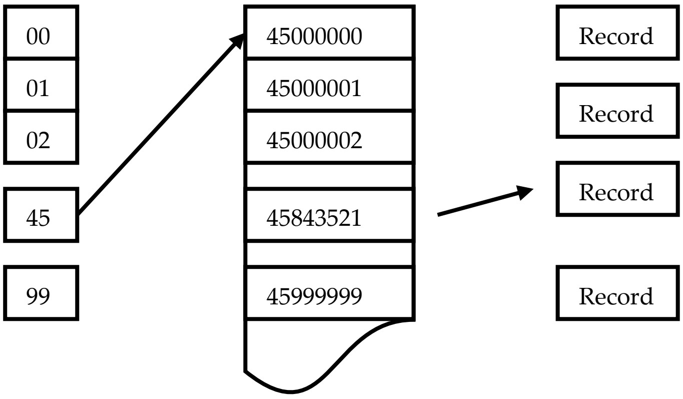
- Sometimes need to access individual records quickly **AND**
- Sometimes need to perform operations on the whole file.

21.4.3 A detailed example of an index sequential file structure

Consider a bank with 100 million customers. They all have their own account number, which is an 8-digit number. Sometimes, **the whole file** holding the customers' details is accessed in batch processing mode, for example to print statements every month and sometimes **individual records** are retrieved, for example, when a customer has a query about their account. Because of this, the index sequential method of organising these records has been selected. The records are organised by putting the account numbers in sequential order. An index is then designed. Imagine we didn't have an index, just one big sequential file. To find a particular account, you would go to the beginning of the file, and search one record after another until you either find the record or come to the end of the file. This is a simple system but with so many records it would take a long time. You could end up checking 100 million records before you found the one you wanted!

21.4.3.1 An index sequential file with one index level

Let's implement an index sequential file. To start with, let's just have one index. The account whose record we want to retrieve is account number 45843521. This is entered into the 'search' field. How is the record retrieved?

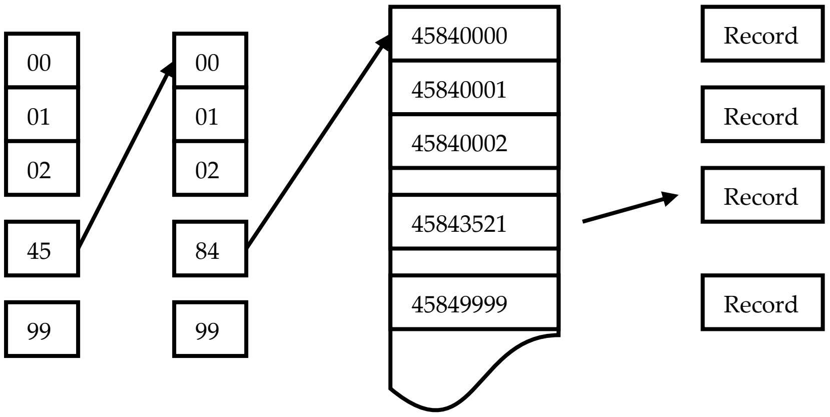


An index sequential file with one index.

In the system illustrated in this example, we have created an index. It is an index of the first two numbers in an account number. It starts with 00 then 01, 02, 03 up to 97, 98 and finally 99. If an account begins with e.g. 45, you go down the index to 45 and get the data held in that location. That data is the first memory address of a block of memory addresses where all the accounts begin with 45 - and the block in this example is 1000000 memory addresses long. You go to that memory address, and then start sequentially searching the block for the account number. You can do that because all of the accounts in that block from that memory location begin with 45. You don't have to search all the other account numbers that begin with 01 or 02 or 03 and so on. How has this helped? When we had just a sequential file, you may have had to check up to 100000000 records before you got to the one you wanted. With one index, you need to sequentially search the index (up to 100 comparisons) followed by a sequential search of 1000000 records. In the worst case, your program would have to make 1000100 comparisons before it found the requested record - a massive improvement on 100000000 records!

21.4.3.2 An index sequential file with two index levels

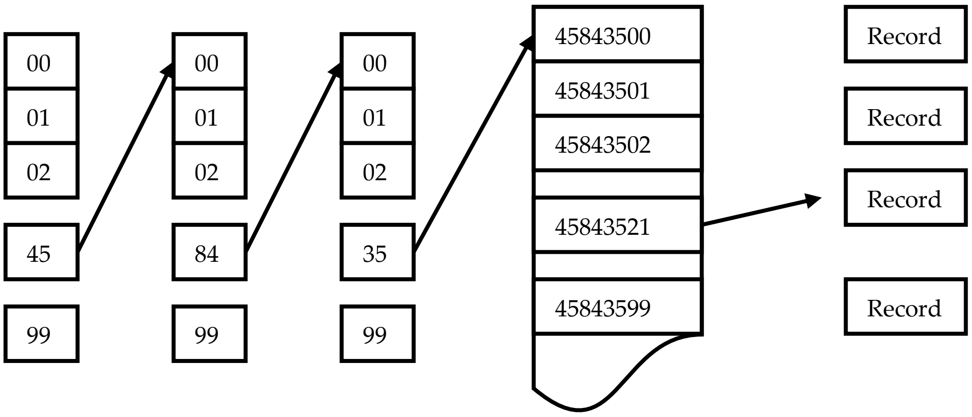
We can improve things further, however. After we have got to the place in memory where all accounts begin with 45, we could examine the next two numbers in an account and use this in another index. The first index pointed us to records that began with 45. The second index points us to records that begin with 4584. If we get to the block of addresses where every account begins with 4584, we can quickly find our target account using sequential searching. You can see this in the next diagram. The first index takes up to 100 comparisons. The second index will take up to a further 100 comparisons, a total of 200. The final sequential search will take only up to 10000 comparisons because the blocks being pointed to are now blocks of 10000 memory addresses! We have gone from having to search 100000000 records, to 1000100 records to 10200 records!



An index sequential file with two indexes.

21.4.3.3 An index sequential file with three index levels

There is nothing stopping us adding a further index! Once we have got to the place where all accounts begin with 4584, we can check the next two digits by searching from that point sequentially. This will point us to an area in memory where all accounts begin with 458435. We then search that block of memory sequentially. This is shown below.

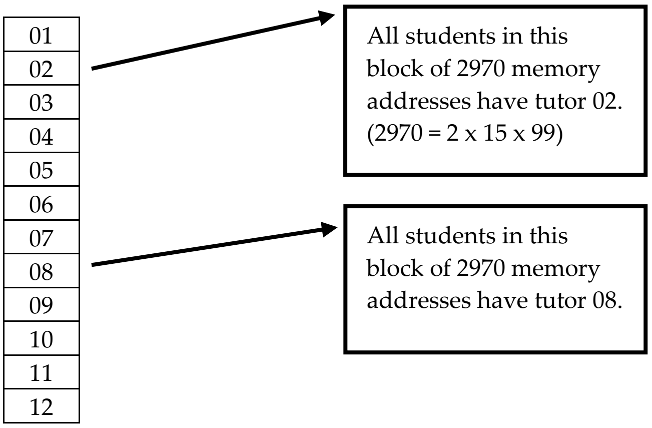


An index sequential file with three indexes.

We can see how many comparisons are needed now before we find a record. We need up to 100 comparisons for the first index, 100 for the second index and 100 for the third index, a total of 300 comparisons. The blocks we will sequentially search to find an account only need to be 100 memory addresses long now. So now, we need only do up to 400 comparisons to find a record (or an average of 200 comparisons) - a far cry from 100 million!!

21.4.4 Another example of index sequential organisation

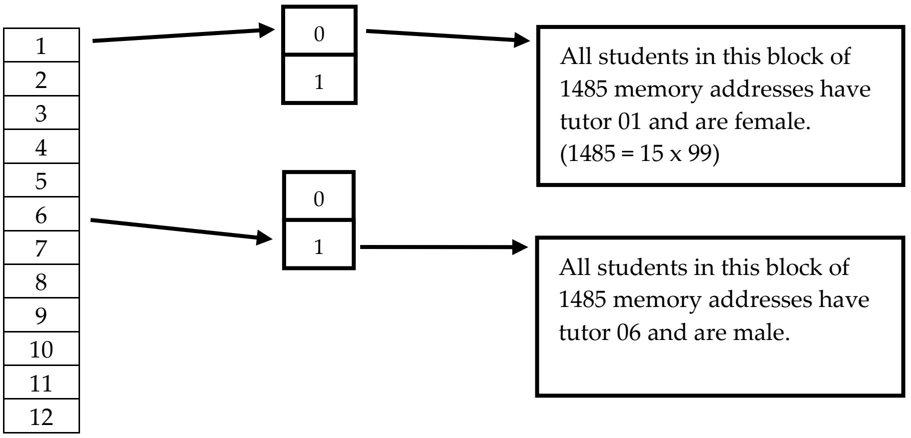
When a pupil joins the Noname University, they are allocated a unique ID code. The first two numbers of the code are based on the tutor that they have been allocated to. There are twelve tutors. The first has an ID of 01, the second 02 and so on up to 12. The third number in the student's unique code is a 0 if the student is female and 1 if they are male. The next 2 digits are a code for the course they are on, from 01 to 15. The final two digits are a unique ID number, ranging from 01 to 99. An example of a code for somebody who was allocated tutor 6, is male, is doing course ID 15 and has the ID 40 is 0611540. We want to design an index sequential file structure so that we can retrieve records quickly. We will represent the structure using diagrams. How would we go about this? The first thing we need to do is have an index with 12 locations, for the twelve tutors. We need a first level index. The diagram would look like this:


01
02
03
04
05
06
07
08
09
10
11
12
All students in this block of 2970 memory addresses have tutor 02. (2970 = 2 x 15 x 99)
All students in this block of 2970 memory addresses have tutor 08.

A sequential file structure with one index level.

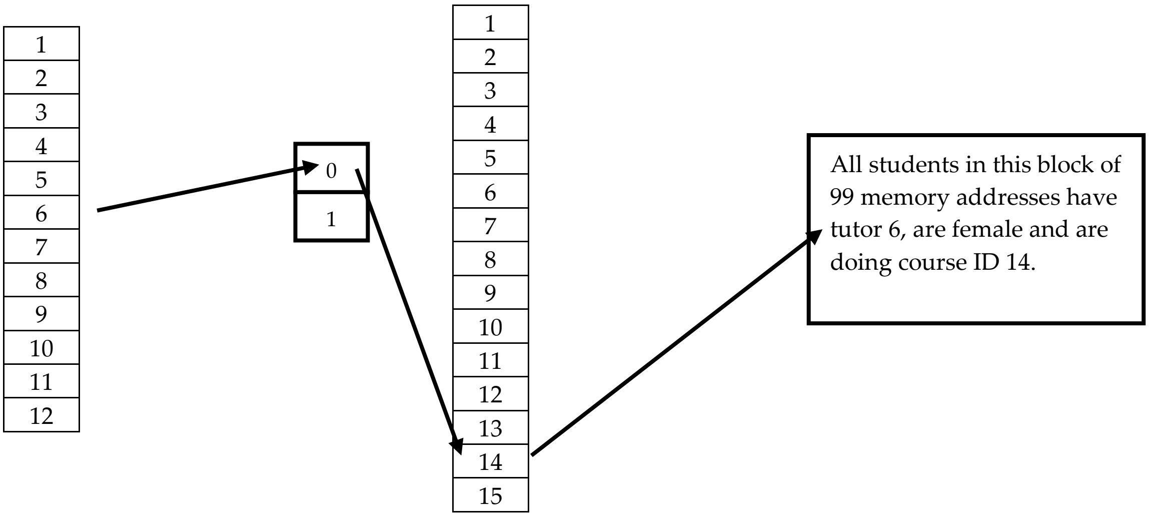
Whenever we want to retrieve a record, we can now strip out the first two numbers in a student's code and use it in a sequential search of our index. This will then point us to a block of memory where all students have that particular tutor. We can then search that block sequentially for any particular student. The worst-case scenario is searching for student 1211599. The first index will take 12 comparisons. Then searching the associated block will take 2970 comparisons, a total of 2982 comparisons. However, these blocks will still be large and will take time to search. (We might of course get lucky and need only 2 comparisons, if we were searching for student 0100101).

We can improve things by having a second index. After we have stripped out the first two numbers and searched for the right tutor, we can then strip out the third digit in the code and search sequentially for the right gender. This will then point us to blocks of memory where, in each block, students have a particular tutor and are a particular gender.


1
2
3
4
5
6
7
8
9
10
11
12
0
1
0
1
All students in this block of 1485 memory addresses have tutor 01 and are female. (1485 = 15 x 99)
All students in this block of 1485 memory addresses have tutor 06 and are male.

A sequential file structure with two index levels.

We can add another level for the course code to improve things even further, like this:


1
2
3
4
5
6
7
8
9
10
11
12
0
1
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
All students in this block of 99 memory addresses have tutor 6, are female and are doing course ID 14.

A sequential file structure with three index levels.

21.5 Random files (also known as hash files)

In the three methods of file organisation we have looked at so far, serial, sequential and index sequential, there involves varying degrees of searching through other records to get to the one you want. With a random file structure, you go straight to the record you want! This is a very fast access method. In a random file, some search data is entered for the system to use to get back the whole record. You then apply a maths algorithm (or formula) to this search data. This transforms the search data into an address. The computer can then go and get the record! Let's look at an example of this.

21.5.1 An example of how random files work

A database of pupils is going to be organised using a random file. This file structure has been chosen because the database will be used to deal with pupil/parent enquiries and so needs to retrieve each record quickly.

The secretary dealing with enquiries will type in a pupil's surname to get their data back. The database designer has selected the following maths formula, known as a ‘hash algorithm’:

> **When a surname is typed in, convert each letter of the surname into a number, and then add the numbers together to give an address. Convert letters to numbers using A=1, B=2, C=3 ... X=24, Y=25, Z=26.**

For example, when the secretary types Jones, this is converted into 10 + 15 + 14 + 5 + 19 = 63. The computer goes to memory address 63 and there's the start of the record!

If you wanted to store ‘Jones’ instead, you would work out the memory address you are going to store the data in by applying the hashing algorithm to the data.

This is a much-simplified example but does illustrate the basic idea. It is a way of converting a request for a file into an address, so the record can be retrieved immediately, without having to go through other records.

21.5.2 Good hashing algorithms

Creating a random file is an excellent way of creating a fast access file structure. You do need, of course, a direct access storage device, not a magnetic tape, for example. You also need to ensure that you have a 'good' hashing algorithm. The one above is very poor indeed because there will be lots of different surnames that give the same memory address! This is called a 'clash' or 'collision'. You need to design a hashing algorithm that minimises clashes because they slow down access times. On the other hand, an algorithm might also spread out the data so much that large areas of storage are used up! Having large areas of storage that aren't used efficiently is known as ‘redundancy’. A further consideration is the hashing algorithm itself. It mustn’t be so complicated that it takes ages to calculate anything!

A good hashing algorithm will:

- Minimise clashes.
- Ensure that the hash codes of data aren't spread too far apart, wasting memory.
- Be quick to calculate.

21.5.3 Dealing with clashes

Some clashes (collisions) are inevitable. When they do happen, there are two techniques that can be used to deal with the situation:

1) The computer can still go to the memory address that was calculated by the hashing algorithm. It then starts searching sequentially from that point to either find the data that it was searching for or to store the data it wanted to store.
2) The computer can set-up an overflow area. When there is a clash, the computer can jump to this special area to either find the data that it was searching for or to store the data it wanted to store.

21.6 Direct-access files

If you have a sequential file, and each record is a fixed length, you can read back the contents of each file if you know its position. For example, if the size of a record is 50 bytes then the records start at position 0, 50, 100, 150 and so on. The start position of any record can therefore be calculated using the formula:

Record start position = (p-1) * 50, where p is the record number you want to read back.

- If you want the first record, (1-1) * 50 = 0 so the first record starts at location 0.
- If you want the second record, (2-1) * 50 = 50 so the second record starts at location 50.
- If you want the tenth record, (10-1) * 50 = 450 so the third record starts at location 450.

Q1. How are serial files stored?
Q2. How do sequential files differ from serial files?
Q3. What is a transaction file?
Q4. What is a master file?
Q5. Describe how a transaction file is used to update a master file.
Q6. Using a clear example, illustrate a typical situation where an index sequential file is appropriate.
Q7. What is a random file?
Q8. Describe the three characteristics of a good hashing algorithm.
Q9. Describe two techniques for dealing with clashes (collisions) in a random file.
Q10. How can you retrieve a record in a direct-access file?

Chapter 22 - Backing up data verses archiving data

22.1 Backing up data

Have you ever lost work you have done on the computer? Do you backup your work onto a pen drive every time you do some work on the computer? If you don't, you should! Given the amount of coursework you probably have to do, it is a wise approach keeping up-to-date backup copies of work that goes towards your final course grades! Your teacher would have told you to do this many times and you will get little sympathy from anyone if you lose work! Companies must also back up their work. If you lose your work, you can start again - not much fun but possible. If a company loses files they could go out of business. People could lose livelihoods. Data is valuable to an organisation. It takes companies years to build up a customer base. Getting data into a computer takes time. Businesses today are run on computers. Examples of records that a company keeps include:

- Employees' details, so they can be paid.
- Details of who owes the company money, so the company can collect payments, pay their bills and make a profit.
- Details of who needs to be paid, so that they don't stop supplying you with raw materials and services because they haven't been paid.
- Designs of products made with CAD.
- Correspondence with suppliers and customers.
- Legal information.
- Web page designs.

22.1.1 What might cause a company to lose its valuable data?

This could happen in a number of ways.

- Hardware failure.
- Corruption of files.
- Attack by a virus.
- Attack by a hacker.
- Theft of data.
- Accidental misplacing of data.
- Theft of the equipment that data is on.
- Sabotage by an employee.
- An honest blunder by an employee.
- Espionage by a rival company.
- A natural disaster.

22.1.2 A backup strategy

An organisation needs to implement a backup strategy so that if data is lost, the activities of the organisation aren't compromised. A sensible organisation will have a backup policy written down. This will deal with the following things:

- The hardware/software that will be used to backup a system.
- The person responsible for backups.
- The frequency that backups will be done.
- When they will be done.
- Where backup copies will be kept.
- How backup copies will be labelled.
- How the company will know the backup system is working as it should.

A backup system needs to use media that is big enough to hold all the data to be backed-up! Magnetic tape is often used for backing-up networks because gigabytes of data can be stored on them. Servers often have a special piece of hardware in them that holds a number of magnetic tapes. The software that comes with it can be configured to automatically backup data in the middle of the night and eject the tape ready for the network manager to put in the fire safe first thing in the morning.

Backing-up networks to cloud storage is also increasingly common. Files are stored off-site, with many different versions being stored going back in time. The cloud storage company takes responsibility for the safe storage of data, ensuring it is protected from hackers or viruses, for example. Of course, you need a reliable Internet connection for this system to work.

USB pen drives, CD R/W or cloud storage are all useful backup storage media to use for personal, day-to-day files.

Backups need to be done, or at least managed, by a named person so that everyone is clear whose job it is to do what and whose job it is to check that the system is working as it should. A backup file quickly becomes out-of-date because the actual system is constantly changing. Therefore, backups need to be taken frequently enough to ensure that only a very limited amount of data is lost. Because a 'rolling program' of backup copies will most likely be made, it is important that everyone is clear how these will be labelled and where they will be kept. Will a copy be kept in a fire safe or off-site, for example? If off-site, how often will it be changed and who will do it? It is very easy for this kind of system not to be done properly because backup copies are rarely needed. Systems generally run smoothly. The problem comes when there is a problem!

22.2 Archiving data

Archiving data has nothing to do with backing-up data!

They are different terms used to describe different things. Computer systems invariably take in data. Consider a school. Each year, new pupils join a school and some pupils leave. If all a school ever did were to input the details of pupils into the system but never remove details, the system's resources such as hard disk space would soon diminish. Therefore, the school should remove pupils who leave. Consider a company. They may keep accounts, orders, invoices and the like on the organisation's network. Once a company has received the goods it has ordered, the order, invoice and any other associated paperwork may be redundant. If you keep all of the computer files a company makes on the system but never remove them, it won't be too long before the system's resources are under pressure. There is a problem, though. You cannot just delete pupils' details from school once they have left. Neither can you destroy invoices. A school is under a legal obligation to keep the records of pupils for many years after they have left. What would happen if a pupil needed a reference? Any company must keep financial records for many years after they have become redundant - the tax office might want to check details of tax returns years after a tax year has finished. Consider what happens after a serious accident involving a plane crash a space shuttle. An accident investigator might want to trace components used in the aircraft or shuttle. They can only do this if records are kept of each component for many years after they were actually made. You want to get rid of data because it is using up hard disk space and slowing down the retrieval of active documents - it takes longer to search a big file than a small one, for example. But for various reasons, you can't! What should the organisation do? The answer is to 'archive' the data.

- There should be a written archiving policy in an organisation stating who is responsible for ensuring data is archived, intervals between archiving and so on.
- Archiving should take place at appropriate time intervals, e.g. every 6 months. Compare this to backing-up files.
- Redundant data could be copied onto a magnetic tape or other mass storage device because you can store more bytes per unit cost compared to other storage devices.
- The data may also be compressed so you can fit more of it onto the tape.
- The archived data is meant to be kept for a long time and doesn't generally need to be easily accessible.
- The original data can be deleted from the system once it is archived.
- The magnetic tape should be labelled up and put in a fire safe or somewhere else secure.

In summary, archived data is generally data that is unlikely to be needed again but you can't get rid of it for legal or other reasons. You want to remove old data from a system because it ties up system resources. Should you need to get back data that has been archived, it will be possible but may take a little bit of effort. You will have to get the correct archived tape, un-compress the data if it has been compressed and then find the data on a serial file, which can take time.

Q1. Why is data valuable?
Q2. State five ways that data might be lost in a company.
Q3. Do some research on the Internet. What are the most common ways for data to be lost?
Q4. State two suitable storage methods for backing up files a network.
Q5. State two suitable storage methods for backing up personal files.
Q6. Describe the advantages and disadvantages of cloud storage.
Q7. Describe a suitable backup policy for a company.
Q8. What is meant by archiving data?
Q9. Why does data have to be archived?
Q10. What is meant by data compression and why is it used in archiving?

Chapter 23 - The CPU in more detail

23.1 Introduction

The CPU, or Central Processing Unit, is the central part of a computer. Because the CPU performs many different functions, you need to divide it up into its components. Only then can you successfully describe what it does. There are 4 component parts that need a mention. These are:

- The Arithmetic Logic Unit (known as the ALU).
- The Control Unit.
- The Registers.
- The Immediate Access Store (also referred to as the IAS, or memory, or RAM, or Primary memory). Some authors treat the IAS as part of the CPU. You can do this, too, or treat it as a separate piece of hardware that the CPU needs constantly to write to and read from. Read widely and then take your pick!

23.2 The ALU

The Arithmetic Logic Unit is that part of the CPU that does all the calculations. It has electronic circuits that can manipulate data in various ways. It has three main functions.

1) It can perform arithmetic calculations on data. For example, it can add and subtract two numbers together or multiply and divide numbers (in binary, of course).
2) It can perform logical operations on data. These are computations that involve, e.g. the use of AND, OR and NOT.
3) The third job of the ALU is to hold data it has already worked on, ready to be sent out, and to hold data that has been fetched, ready to be processed. All data that goes into and out of the CPU goes via the ALU. The ALU acts like a 'revolving door' for data, letting data pass in as well as out of the CPU. In summary, the ALU:

    i. **performs arithmetic calculations**
    ii. **performs logical operations on data**
    iii. **acts as a 'revolving door' for data going into and out of the CPU.**

23.3 The Control Unit

This part of the CPU is responsible for managing how instructions are executed. It has some very important jobs to do! It is in charge of fetching instructions and data from wherever they are stored in the memory unit. It does this by sending control signals at the right time to various parts of the computer that then access the correct memory unit location and retrieve the contents of that location. Its next very important job is to interpret, or 'decode', an instruction so that it knows what has to be done. It has a special piece of equipment called an 'instruction decoder' to do this. Once it has done this, it can then execute the instruction. It can send signals to all the different parts of the CPU telling them what to do and when.

Any program you write or application you run is made up of a sequence of instructions. When you 'run' a program, it is the control unit that is fetching, then decoding and finally executing every instruction, one after another. Unsurprisingly, this is known as the 'fetch-decode-execute cycle'. This is dealt with in another chapter.

23.4 Registers

No discussion about a CPU would be complete without mentioning the role of 'registers'. The registers are an integral part of the CPU. They are a type of memory that can be accessed very quickly compared to other types of memory. The pieces of information they hold are needed very often. They can be used to store data and control information during a fetch-decode-execute cycle or they can be used to hold values that are generated as part of the ALU working on data. There are a number of very special registers that do very specific jobs. We will discuss them in more detail in a later chapter but here are some important ones, just to get you excited about them!

23.4.1 The Accumulator

If a calculation has to be done the data will be moved into a very special register called the 'accumulator'. CPUs will typically have a number of accumulators. Data will be processed and the result held in one of the accumulators, overwriting the old contents. It will then be moved out into memory.

23.4.2 The Status Register

When you do any calculation, you don't just get a result! You also get information about the result. These pieces of information are stored in 'flags' in a special byte called the 'status register'. A flag is simply another name for one bit that holds a Boolean piece of information. For example, if the result of a calculation was negative, the 'negative result' flag gets set to say that the last calculation produced a negative result. If there wasn't an overflow in the last calculation then the overflow flag gets reset to say that the last calculation did not produce an overflow. Flags are important because they can be used to alter the flow of a program, for example, by using IF instructions. Suppose you do the sum 23 - 43. The flag bit that indicates a negative number is 'set' (made a one) to indicate the result was negative. The negative flag can be tested after the calculation using an IF statement. If the flag is a one (as it is here), then PROCEDURE NegativeNumber is called ELSE PROCEDURE PositiveNumber is called.

23.4.3 The Program Counter

We talked about the Control Unit working on a program of instructions, carrying out a fetch-decode-execute cycle on each instruction in turn. The Control Unit knows where the next instruction in memory is because there is a register called the Program Counter that holds the memory address of the next instruction that must be fetched, decoded and executed.

23.5 Immediate Access Store (IAS)

The IAS is the place where programs and the data that is needed by programs are held, ready to be fetched then decoded and executed by the CPU. The CPU may also use this place to store the results of any processing it does. It can be thought of as made up of lots of individual 'memory' locations, each capable of storing a byte of data. You can treat the IAS as being part of the CPU although it is probably easier to visualise it and discuss it as a separate piece of hardware from the CPU.

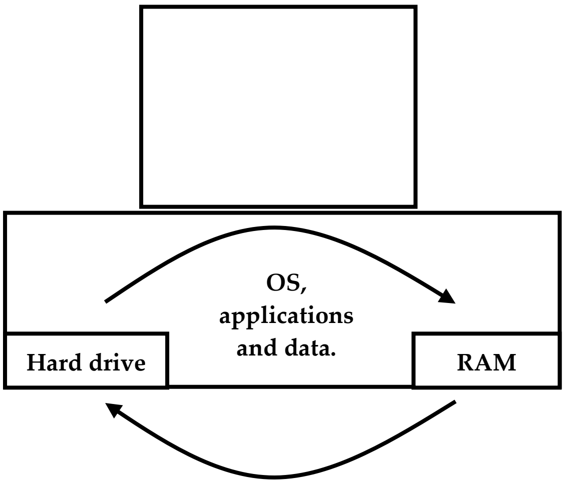
Memory address 1	Data item or program instruction
Memory address 2	Data item or program instruction
Memory address 3	Data item or program instruction
Memory address 4	Data item or program instruction

↓

Memory address 256435454	Data item or program instruction
Memory address 256435455	Data item or program instruction
Memory address 256435456	Data item or program instruction

Diagrammatic representation of 256 Mbytes of RAM.

When you open an application, the application is typically moved from storage on your hard disk into the IAS, ready for the CPU. When you open a file, it is also moved from a storage device into the IAS. When you save work, it's moved from the IAS into a storage device, such as your hard drive. The operating system, programs and data must be worked on from the IAS, **not** from storage.



When the computer is turned on (known as 'booting-up' a computer) it checks itself for problems. Then, before a user can get access to the hardware and use the computer, the operating system must be copied from the hard drive to the RAM.

ALL programs, including the operating system, and any data files you need to use, MUST be copied to RAM before they are used.

The O/S, applications and data must be loaded into RAM for them to be used.

23.5.1 Data or program instruction?

It is worth making the point at this time that each 'box' which has an address in RAM holds a bit pattern. That bit pattern might piece a piece of data, such as a number or a character on the keyboard. It might equally well be an instruction or part of an instruction. As far as the CPU is concerned, it can't tell the difference! It 'knows' whether it is a piece of data or an instruction only because the operating system has kept track of which memory locations are used for data and which ones are used for program instructions. (If you remember, one of the jobs of the operating system was memory management).

23.5.2 - Primary memory and secondary storage

IAS is sometimes made distinct from storage devices by using the terms **primary memory** to describe the IAS whilst a hard disk, pen drive and so on are known as **secondary storage devices**. You may see the words 'memory' and 'storage' used interchangeably in books because they have very similar meanings. However, think of memory as being more **immediately accessible** compared to storage, which has to be **fetched** before it can be used!

23.6 Not all processors are equal

Similar specification processors perform differently. There are a number of reasons for this.

23.6.1 The materials the processor is made from

Different processors, like any product, are made using different materials and different qualities of materials. There are 'good' quality materials and not-so-good ones! The materials used in a processor will affect the reliability, speed and performance of that processor. Importantly, the speed of the slowest component might slow right down an otherwise fast CPU.

23.6.2 Clock speed

Another reason why different processors perform differently is clock speed. Every computer has a 'clock'. This is a crystal that vibrates, generating 'pulses' that are used to control how the different components of a computer system work together. If you have a 800Mhz machine, it means the system clock is generating approximately 800 million pulses every second! The faster the clock, the more fetch-decode-execute cycles the CPU can perform in a second, and (very broadly) the faster your programs should go! (There are lots of other factors that may mean this is not the case, however).

23.6.3 The size of data and address busses

Another factor affecting performance is the size of the address bus and the data bus in a system. A bus is simply a 'highway' used to move binary data around a computer. When the CPU needs to get a piece of data or an instruction from RAM, or needs to write a piece of data to RAM, then it puts the address of the RAM location to use on the address bus. When a piece of data is actually retrieved from RAM, or a piece of data is being sent to RAM, then the data is placed on the data bus. You can have big computer buses and little computer buses, just like you can have motorways and B-roads! If your computer has motorways for an address bus then it can address far more RAM locations than if it had a small address bus. For example, if your address bus was 8 wires wide, 8 bits wide, then you could address 256 different RAM locations (2^8). If, however you had an address bus that was 16 bits wide, you could address 65536 different locations (2^{16}). Similarly, if the data bus was a small size, it might have to collect items of data or instructions a section at a time. Each time it collected part of a piece of data or an instruction, a clock cycle would pass. The more clock cycles, the slower the computer!

23.6.4 Word size

An important characteristic of a processor is its word size. This is the number of bits that the CPU can work with in any one clock cycle. The more bits it can work with in one clock cycle, the faster the computer will go. You will have read about 16-bit processors and the latest 128-bit processors. This is referring to the word size.

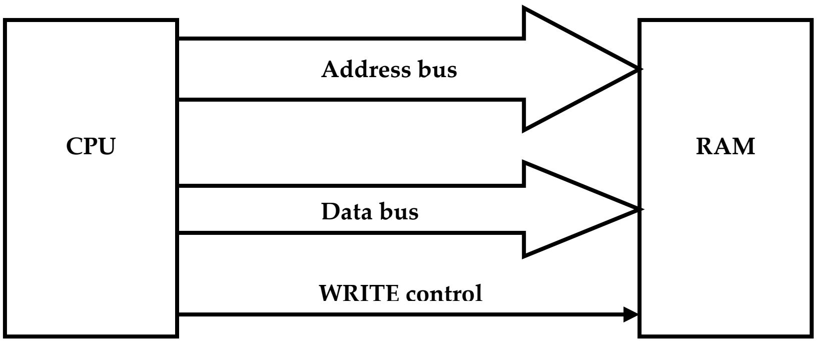
23.7 Control signals

It is worth mentioning at this point that there is also a control bus in a computer. This provides a wide range of control and status signals between the CPU and all of the components in a computer, to ensure that they all work together without clashing and without problems. We are about to meet two such control signals, the WRITE TO RAM and READ FROM RAM signals.

23.8 How does the CPU write a piece of data to RAM?

To store a piece of data in RAM, The CPU must do the following things:

1) Place the address where it wants to store the data onto the address bus.
2) Put the data that it wants to store onto the data bus.
3) Send a **WRITE** control signal. This has the effect of storing whatever is on the data bus and putting it into RAM at the address specified on the address bus.

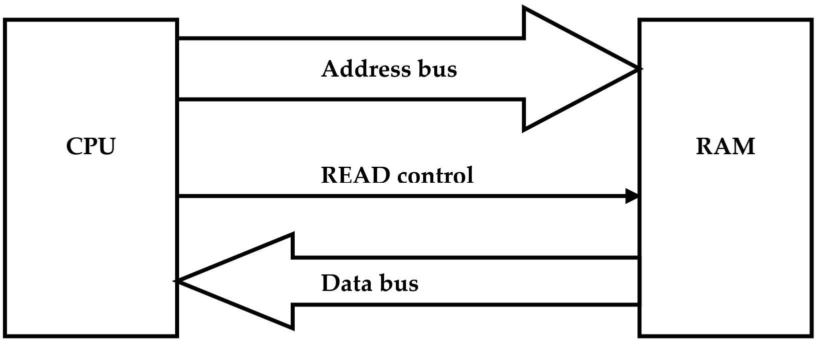


How a CPU writes to RAM.

23.9 How does the CPU read a piece of data from RAM?

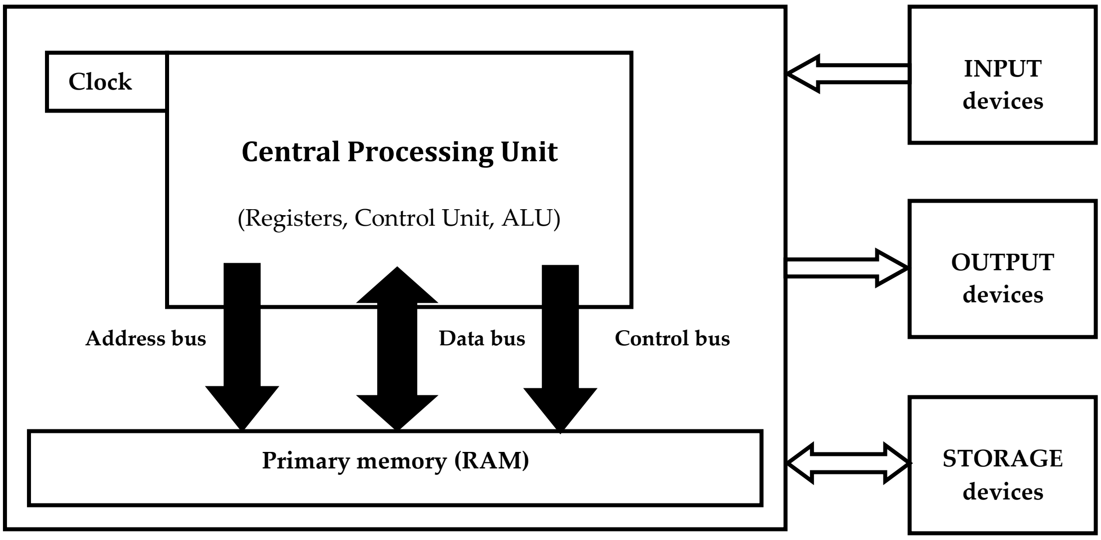
To store a piece of data in RAM, The CPU must do the following things:

1) Place the address where it wants to read the data in RAM from onto the address bus.
2) Send a **READ** control signal. This has the effect of transferring the contents of the RAM address specified on the address bus onto the data bus.
3) The CPU then reads whatever is on the data bus via the ALU.



How a CPU reads from RAM.

We can summarise in a block diagram how the CPU works with input, output and storage devices, the busses and RAM:



Q1. Describe what the ALU does.
Q2. What is the Control Unit's job?
Q3. Describe the function of the Accumulator register.
Q4. Describe the function of the Program Counter register.
Q5. Why is RAM also known as the Immediate Access Store?
Q6. State one piece of hardware that is volatile and one that is non-volatile.
Q7. What is meant by secondary storage?
Q8. Why do processors, which are the same speed, differ in performance?
Q9. What is a bus?
Q10. Draw a block diagram from memory that shows how the CPU works with busses, I/O, storage and RAM.

Chapter 24 - Memory

24.1 Introduction

In the previous section, we saw that there is a subtle distinction between memory and storage. We noted that 'storage' devices suggest a place where we store data and applications that are not needed immediately by the CPU. Such devices would include a hard disk and a CD R/W, for example. Another name for such devices is 'secondary storage devices'. Memory, however, is a place where data and applications needed immediately by the CPU are put. We can refer to memory as the 'Immediate Access Store', the memory unit or 'Primary Memory'. We can even refer to it simply as the RAM or Random Access Memory'. Just to confuse things a little further, there are other types of memory, too! We will look at memory in a little more detail here.

24.2 Some terms used to describe memory

There are a number of terms associated with 'memory'. These include RAM, ROM, registers, cache, battery-backed RAM, firmware and buffers.

24.3 Random Access Memory (RAM)

We have come across RAM before. We have called it by different names, **Immediate Access Store (IAS), the memory unit** and **primary memory**. This type of memory is the memory you are talking about when you say that your computer has got 256 Mbytes of RAM, for example. The number is a measure of how much memory you have got, how many applications you can open at the same time and how much data you can store, ready for the CPU to access. If you have 256 Mbytes of RAM, for example, you have approximately 256 million individual memory locations in which to store applications and data. Each memory location has its own address and you can store an instruction or a piece of data in each location. The more RAM you have, the more applications you can open at the same time and the more data files you can open as well. You can never have enough RAM, although, of course, you have to pay for it and there will be a maximum amount of RAM the motherboard in your PC can hold! RAM is '**volatile**'. That means that the contents of the memory locations disappear completely once power is removed. The contents only remain whilst your computer is switched on. You may have noticed that sometimes your computer stops working as it should. The contents of the RAM may have become 'corrupted', or mixed up. By 're-booting' your computer, you are clearing out all of the applications from RAM (including the operating system) and data, and then reloading them again into RAM. This often clears any problem with the computer and is certainly one of the first things to try if your computer locks up and becomes unusable.

24.4 Read Only Memory (ROM)

Another type of memory is Read Only Memory, or ROM. This type of memory holds part of a special program that starts running when the computer is powered up. It is called the BIOS (Basic Input Output System) and does two things. It checks that the computer hardware is present and correctly working, by carrying out a POST (Power On Self Test). For example, it will check that you have sufficient RAM and that it is working well. It will check that you have a mouse and keyboard attached to your PC and report an error to you if it can't find one. It runs a routine that looks for another special program called the **bootstrap program**. This is usually held in a special place on the hard drive. When it finds it, it loads it into RAM and runs it. The job of the bootstrap program is to locate the operating system on the hard drive and then load it into RAM and run it. Starting up a computer from a power-off situation is known as 'booting up' the computer.

ROM is '**non-volatile**'. That means that the contents of ROM are NOT lost when you turn the power off, unlike RAM. The actual program in ROM is put there by the manufacturers of motherboards, for example, around the time the ROM chip is placed in a motherboard. Although it is known as READ ONLY MEMORY because you cannot generally write to it, It is possible to update the software held in ROM chips through a procedure known as '**flashing**'.

24.4.1 Flashing your BIOS

Most newer motherboards come with a type of ROM known as **flash ROM**. Flash ROM allows a user to install an upgrade for their BIOS should the need arise. This might happen because the existing BIOS cannot support some features that the user wants. Perhaps the existing BIOS doesn't have a password-protection utility or cannot support the latest, very large hard drive that has just come onto the market. Whatever the reason, computer users can download and install another, later BIOS. They simply need to identify their own motherboard and chipset and then identify a company who has written a new BIOS. They then have to download the BIOS and follow a set of procedures for getting it into their motherboard's ROM.

This process should not be undertaken lightly. There are many desperate requests for help on the Internet because somebody has attempted to load the latest version of a BIOS and found that their whole computer doesn't function! The message here is, ‘Don’t try this at home’ unless you can afford to have your computer not work on you for a period of time whilst you work out what went wrong and how to put it right!

24.4.2 Why is the operating system not stored in ROM on a personal computer?

You may have wondered why there appears to be such a roundabout system of loading up your operating system, why you have to run some instructions in ROM that looks for a program called the 'bootstrap' on the hard disk that then loads up the operating system! Why not just put the operating system in ROM and let it load up straight away when you boot up the computer? The answer is 'flexibility'. You should understand that you, the user, can’t normally change the contents of ROM. If you wanted to change operating systems, and the operating system was in ROM, you would be stuck! However, by putting the OS on the hard disk, you can upgrade it anytime you want and you don't need to change the program in ROM.

24.5 Registers

We have come across these before! Registers are part of the design of the CPU. They are memory circuits and are very fast because they are constantly being accessed by the CPU. Examples of registers include the Accumulator, the Status Register and the Program Counter. There are others, however, and you can read about these in a later chapter.

24.6 Cache

Another type of memory is known as the cache and it is provided in computer systems to speed up processing. Like all memory, it is measured in bytes (or Kbytes, Mbytes etc). Programs are made up of instructions. Instructions are fetched from memory using the fetch-decode-execute cycle. The data that instructions need is also fetched from memory and some data might need to be fetched over and over again, for example, a constant that is held in memory and used lots of times in lots of calculations. Fetching data from the IAS takes time. Fetching the same data time after time is a waste of time! Processing can be speeded up by storing constantly-needed data in some very fast-access memory. This will reduce the 'fetch' time. This fast-access memory is called 'cache'. It is much faster than RAM (but not as fast as registers). You only get limited amounts of it in a computer system because it is expensive to make. One question that should always be asked when buying a new computer is "How much cache has it got"? It is always worth getting more cache for a computer because it speeds up processing - at a price!

24.7 Battery-backed RAM (sometimes known as CMOS)

We have already said that RAM is 'volatile' - when you turn the power off, it loses its contents. We have also said that ROM is non-volatile and a user can't change the contents of ROM easily. We also know that ROM is used to store part of the BIOS. The rest of the BIOS is stored in battery-backed RAM. This is RAM, but the computer ensures the power is never removed from it by connecting a battery to it. This takes over when the main power is removed, when the computer is switched off. By putting part of the BIOS in battery-backed RAM, you are giving the user the ability to store their own settings, used when the computer boots up. For example, we have already said that when a computer powers up, the BIOS looks for the bootstrap program. Now it can look for it on a pen drive or CD and if it doesn't find it, it can then try the hard disk. It might not seem that important, but the user can set the order that the computer looks at the different devices! The user can also use this part of the BIOS to set up and store a computer password, to prevent casual access to the computer. Next time you boot up your computer, take note of which storage device is being looked at first. It is probably the CD ROM drive or a USB port. If you ever open up a computer, look for a flat, silver battery on the motherboard, about the size of a penny. You can always reset the contents of the battery-backed RAM by removing this battery for 10 minutes.

24.8 Firmware

Firmware is another term that you will come across as you read widely. A piece of hardware such as a CD ROM is a very complicated device. It has its own processor to control how it works. Just like the CPU on your computer needs an operating system to make the computer do things, so the processor in each piece of hardware in your computer system needs its own ‘operating system’ to make that device work. This ‘operating system’ is known as **firmware**. You will also see authors referring to both the software *and* the hardware for a device as firmware. Read widely and then make your own minds up! Firmware is usually held on flash ROM in each device. We have already discussed this type of ROM earlier in this chapter and we know that we can upgrade the software held in it - we can upgrade the firmware. You can search the Internet for firmware upgrades for a particular piece of hardware that you have. The first place to start looking is the manufacturer’s web site!

The same warning that was made when we discussed upgrading the BIOS applies to upgrading firmware. If something goes wrong, for example, you don’t follow the instructions properly or your computer crashes in the middle of the upgrade, then the peripheral may stop functioning altogether. Don’t upgrade firmware unless you can afford for things to go wrong!

24.9 Buffers

A buffer is an area of RAM that has been reserved for one purpose - to aid the transfer of data between different parts of a computer because those parts work at different speeds. An example of this is the transfer of data between primary memory and secondary storage devices. Primary memory is part of the CPU and works at very fast speeds compared to secondary storage devices such as hard drives, which are very slow. If you didn't have a buffer then the transfer would have to take place at the speed of the slowest device and that is an inefficient use of the CPU. It prevents the CPU doing other more important things because it has to take charge of the management of the data transfer. In fact, buffers are used in many places, wherever there is a speed mismatch between two devices. Other examples of where they are needed include the transfer of data from the keyboard to the CPU and the transfer of data from the CPU to a printer. Whilst buffers can indeed be part of the IAS they can also be separate peripherals. You can buy buffers to connect between a computer and an output device such as a printer so that print jobs are immediately sent out of the control of the computer's CPU and into the control of the buffer. Peripherals themselves will come with a certain amount of buffer. A printer might come with a buffer of 8 Mbytes, for example. This can easily be upgraded so that the printer can accept more print jobs and bigger print jobs without needing lots of the CPU's time.

24.10 How to compare different types of memory (including storage devices)

Memory can be classified in a whole variety of ways, depending upon your focus.

24.10.1 Primary memory and secondary memory (more commonly referred to as 'secondary storage')

You will often read about Primary memory and secondary storage devices in the same sentence as they are sometimes treated together as 'memory'. One way to deal with 'memory' if it is being discussed like this is to classify it into the two types: primary memory and secondary storage. Primary memory is the memory that forms part of the CPU circuitry itself. It's the place where applications and data are held for use by the CPU and where results of calculations are put. Secondary storage devices are devices that are connected to the CPU as peripherals e.g. a hard disk. They store data and applications ***not*** immediately needed by the CPU. Primary memory is also known as RAM, Immediate Access Storage or IAS.

24.10.2 READ devices and READ/WRITE memory devices

Registers, cache, RAM, pen drives, hard drives and CD-R/W are READ/WRITE devices. That means that data in these devices can be accessed and read but also new data can be written to these devices. ROM, DVDs and some CDROMS are READ ONLY. Data is burnt (written to once) onto these devices and cannot then be changed. They are sometimes called **WORM** devices (**W**rite **O**nce **R**ead **M**any times).

24.10.3 Volatile memory and non-volatile memory

Another way to split memory types up is to divide them between volatile and non-volatile memory. RAM is volatile. That means that when the power is turned off, you lose the contents. Motherboards have a small amount of battery-backed RAM known as CMOS. When the main power is turned off, this type of RAM will keep its contents (as long as the battery works!) Battery-backed RAM keeps the details of the BIOS password, for example. If you forget the password to the BIOS, you can empty the contents of the battery-backed RAM by removing the battery for a minute. ROM is non-volatile, as is the hard drive. When the power is turned off, the contents will not be lost.

24.10.4 Other criteria

You can also discuss, for example, the cost of different kinds of memory, how fast they are to read from and write to (their 'read-write speed'), their potential capacity, their reliability and other criteria.

Q1. What is RAM used for?
Q2. What are other names for RAM?
Q3. RAM is described as 'volatile'. What does volatile mean?
Q4. What does the bootstrap program do?
Q5. Describe one type of data that might be found in the cache and justify your answer.
Q6. Why do computers need battery-backed RAM?
Q7. What is a buffer?
Q8. Do some research on the Internet. What is 'firmware'? Give some examples that clearly illustrate your answer.
Q9. Suggest some criteria that could be used to compare different types of memory.
Q10. What is the unit of data storage?

Chapter 25 - Using buffers and interrupts to transfer data

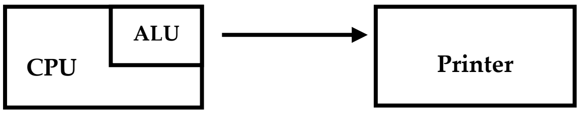
25.1 Introduction

When two devices working at different speeds try to communicate, they have to do so at the speed of the slowest device. This is not good because the CPU gets tied up managing the transfer of a constant stream of data to a printer, for example. That means it can't work on other tasks that may be more urgent than mere printing! However, by using a 'buffer', the problem of working at the speed of the slowest device and slowing the CPU down can largely be overcome. A buffer is simply some memory that improves the efficiency of data transfer between two devices working at different speeds by allowing big blocks of data to be collected together and then sent at once rather than as a stream of data that needs constant CPU management time. A buffer can also be an external piece of memory designed specifically for the purpose of collecting data from a CPU and then taking over the management of the transfer between itself and the device it is connected to.

25.2 An example of the use of a buffer

All devices such as printers come with their own buffers and you can usually upgrade the amount of buffer memory you have in the printer. The bigger the buffer in a printer, the more data it can accept before it tells the computer that is sending the data to stop for a minute because it is full and needs to empty itself (by printing). Large buffers are particularly important on networked printers such as those commonly found in schools and colleges. A lot of people will potentially be sending a lot of work at the same time. This could result in the system grinding to a halt, as the server tries to manage all of the jobs being sent to the printer. If, however, the jobs could be sent to the printer and stored there, the server would be free to get on with other things and the printer could print at its leisure. Consider sending a file to a printer. The file is held in memory (RAM). We know that RAM works very quickly and we know that all data in and out of the CPU goes via the ALU - that is one of the functions of the ALU. We also know that the printer churns out pages much slower than the speed a CPU can work at!

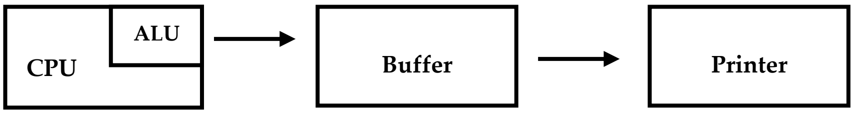
25.2.1 Assume there isn't a buffer for a moment



The transfer of data without a buffer.

The CPU will take charge of the data transfer. The data goes from the RAM, via the ALU and out to the printer. The printer works very slowly compared to the CPU, however. If you send data constantly, byte by byte, the CPU will have to slow down how fast it sends the data to the printer so that the printer can receive the data. This means, however, that the CPU is not being used as efficiently as possible.

25.2.2 If you provide a buffer, however, data transfer can be improved

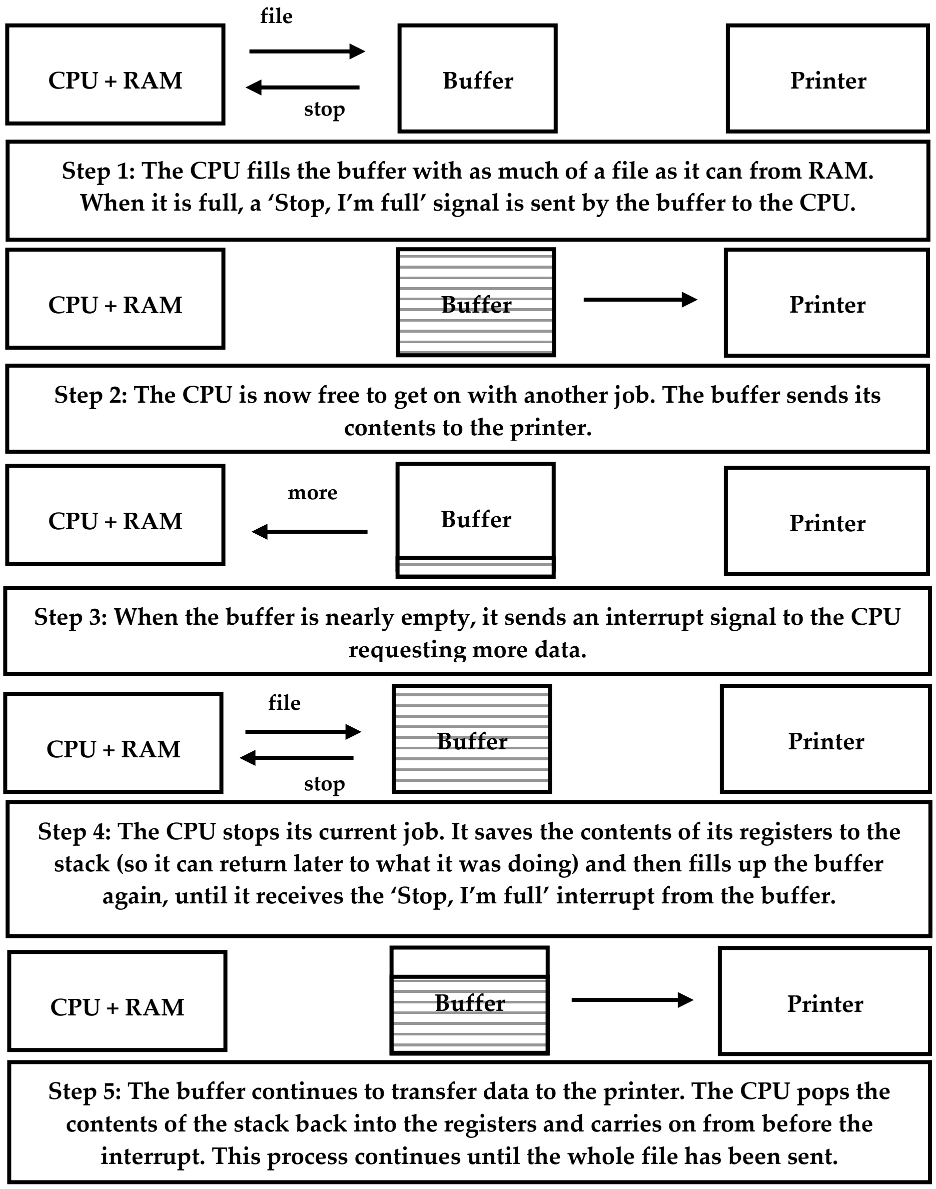


The transfer of data with a buffer.

When the CPU wants to send a file, all of the data can be transferred to the buffer in one go. The buffer works at the speed of the CPU so data transfer between the CPU and the buffer is very fast. The CPU is now free to do other tasks while the buffer can transfer data to the printer. This immediately improves the transfer of data because the CPU is not waiting around, trying to constantly send data. It can send data in one, high-speed burst and then get on with something else!

25.3 Buffers and interrupts

A complication can arise, however, if you need to send a file to a printer but the whole file won't fit into the buffer because the file is too big for it! Firstly, the CPU sends an 'Are you ready' signal to the printer. If the printer is ready, it sends back a 'Ready' signal. Then the CPU, via the ALU, fills the buffer with as much of the file as it can. When the buffer is full, it sends an interrupt to the CPU to say 'Stop'. The buffer then sends the data to the printer. The CPU, meanwhile, is free to get on with another job. When the buffer is nearly empty, it sends an interrupt signal to the CPU, to say 'More data, please'. When the CPU gets this interrupt, it checks its priority. The CPU has to decide if the job it is currently doing is more important than filling up the buffer! If it is, then it carries on with what it is doing until it is finished, and then it services the buffer interrupt. If it isn't more important, then it stops doing what it is doing and 'pushes' the contents of its registers onto the stack, thereby saving where it was before the interrupt happened! The CPU then sends the buffer some more of the file until it gets the 'Stop' signal again. The buffer resumes sending data to the printer whilst the CPU 'pops' the contents of the stack back into its registers and continues from where it left off before the interruption. This process is repeated until the whole file has been sent! The transfer of data between two devices that work at different speeds can be made far more efficient by using a buffer because we can send data in large blocks rather than in lots of small quantities. We have looked in some detail at how you would transfer a file from memory to a printer using a buffer designed for the purpose, but we could equally have used the example of transferring data from memory to a secondary storage device such as a hard disk, for example.



Step 1: The CPU fills the buffer with as much of a file as it can from RAM. When it is full, a 'Stop, I'm full' signal is sent by the buffer to the CPU.

Step 2: The CPU is now free to get on with another job. The buffer sends its contents to the printer.

Step 3: When the buffer is nearly empty, it sends an interrupt signal to the CPU requesting more data.

Step 4: The CPU stops its current job. It saves the contents of its registers to the stack (so it can return later to what it was doing) and then fills up the buffer again, until it receives the 'Stop, I'm full' interrupt from the buffer.

Step 5: The buffer continues to transfer data to the printer. The CPU pops the contents of the stack back into the registers and carries on from before the interrupt. This process continues until the whole file has been sent.

The transfer of large data files.

Q1. Define 'buffer'.
Q2. Define 'interrupt'.
Q3. How many different jobs can a CPU work on at any moment in time?
Q4. Why does RAM work at about the same speed as a CPU?
Q5. Why are hard drives and printers much slower than a CPU?
Q6. What is meant by 'the speed mismatch problem between devices'?
Q7. Define 'stack'.
Q8. What is meant by secondary storage?
Q9. What does 'pop' and 'push' mean when talking about data transfer?
Q10. How are buffers and interrupts used to overcome speed mismatch problems between RAM and a hard disk?