A2 Computing Revision

Data Representation

Number Bases

You might want to see http://saps2.mine.nu/school/ASA2/Computing/Numberbases.ppt for an interactive PowerPoint on number bases.

Comparing the Bases

Denary – or base 10 – is the system we use naturally, we have 10 different digits from 0 to 9. Each digit in a denary number has a place value, calculated in powers of ten from the right hand side.

10^4	**10^3**	**10^2**	**10^1**	**10^0**
10,000	**1,000**	**100**	**10**	**1**
5	2	4	5	1

For example, the number 52,451 can be expressed as:

= (5 × 10000) + (2 × 1000) + (4 × 100) + (5 × 10) + (1 × 1

= 52,451

Other number bases work in exactly the same way, except rather than having powers of ten; we have powers of different numbers. In binary we have powers of two, and are restricted to two digits, 1 and 0.

2^7	**2^6**	**2^5**	**2^4**	**2^3**	**2^2**	**2^1**	**2^0**
128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
1	1	0	0	1	1	0	1

For example, the number 11001101_2 can be evaluated in denary using the place value idea in the same way as for a denary number:

= (1 × 128) + (1 × 64) + (0 × 32) + (0 × 16) + (1 × 8) + (1 × 4) + (0 × 2) + (1 × 1)

= 205

In hexadecimal, we use base 16 – with place value incrementing in powers of 16, and 16 possible digits. Since we have no more digits after 9, we use the letters A-F instead, whereby A is the digit which comes after 9, B is the digit which comes after A, etc.

10^4	**10^3**	**16^2**	**16^1**	**16^0**
65,536	**4,096**	**256**	**16**	**1**
0	0	0	F	6

Again, the number $F6_{16}$ can be evaluated in denary using the idea of place value:

= (F × 16) + (6 × 1)

= (15 × 16) + (6 × 1)

= 246

The computer uses binary because it models the behaviour of a switch, it is either on or off, and the processor is only able to manipulate binary numbers. Hex is used because it is a very efficient system of storing large numbers, consider the number 255, this requires eight bits to store in binary (1111 1111), three to store in denary (255), but only two in hex (FF).

Converting Between Bases

You can use the place value system described in the previous section to convert numbers from any number base back into denary, and to convert from denary into other number bases there is a reverse process to use:

For example, to convert the number 176 into binary:

$176 \div 2^7$	$= 176 \div 128$	= **1** remainder 48
$48 \div 2^6$	$= 48 \div 64$	= **0** remainder 48
$48 \div 2^5$	$= 48 \div 32$	= **1** remainder 16
$16 \div 2^4$	$= 16 \div 16$	= **1** remainder 0
$0 \div 2^3$	$= 0 \div 8$	= **0** remainder 0
$0 \div 2^2$	$= 0 \div 4$	= **0** remainder 0
$0 \div 2^1$	$= 0 \div 2$	= **0** remainder 0
$0 \div 2^0$	$= 0 \div 1$	= **0** remainder 0

The binary number is the whole part answers of the modular division.
In this case 10110000

Whilst you can apply this method to any number base, including hexadecimal, it is usually easier to convert from denary to hexadecimal using binary as a go-between. Since the conversion from binary to hexadecimal is much easier.

To now convert our 10110000 into hexadecimal, we just split it into blocks of four bits (known as a nibble) and convert these into hex digits. So 1011 becomes (8 + 2 + 1 = 11 = B) B, and 0000 is of course 0. So in hexadecimal 176 is equal to B0.

Negative Binary

To represent a binary number as being negative, we use a system knows as “two’s compliment”. This can be thought of as like a car’s odometer, if the car drives a mile forwards, the odometer reads 000001, however, if it were to be forced back a mile, it would read 999999, which is like -1 mile. A negative binary number will always start with a 1 – this is known as the sign bit.

To convert a number to its two’s compliment, starting from the left we invert all the bits up to and *not* including the last 1. So, for example,

	-	0	0	1	1	0	0	1	1
=		1	1	0	0	1	1	0	1

To convert a two’s compliment number back to denary, consider the place value of the first bit to no longer be 128, but *negative* 128. So in our above example:

= -128 + 64 + 8 + 4 + 1
= -51

Binary Sums

You can add binary numbers in just the same way you add denary numbers. Remembering the simple rules that 1 + 1 = 10, 1 + 0 = 1 and 1 + 1 + 1 = 11.

For example, 10110011 + 00111010 = 11101101

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

To do binary subtraction, we have to convert the number to be subtracted to its two's compliment form, and then treat the operation as an addition. This may seem obscure, but the same thing is true in denary, for example: 5 – 3 = 5 + -3

Binary Fractions

Fixed Point

Just as our place value scale continues indefinitely to give increasing powers of the base number, the scale continues in the opposite direction with negative powers. These allow us to have fractions.

10^3	**10^2**	**10^1**	**10^0**	**.**	**10^{-1}**	**10^{-2}**	**10^{-3}**
1,000	**100**	**10**	**1**	**.**	**1/10**	**1/100**	**1/1000**
0	1	2	4	.	2	5	0

In denary, we can consider 124.25 on the place value scale with the .25 representing two tenths and 5 hundredths, and the same is true of binary fractions except in base two.

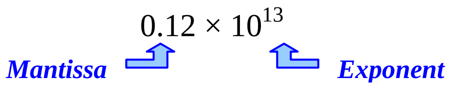
2^3	**2^2**	**2^1**	**2^0**	**.**	**2^{-1}**	**2^{-2}**	**2^{-3}**
8	**4**	**2**	**1**	**.**	**1/2**	**1/4**	**1/8**
0	1	0	1	.	1	0	1

Here, we can show that 0101.101 = 4 + 1 + $\frac{1}{2}$ + $\frac{1}{8}$ = 5 $\frac{5}{8}$ = 5.625

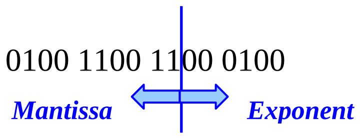
With fixed point notation the position of the decimal point is fixed at a given bit number. In the example above there are four bits before the point, and three after. This is reasonably to work with, we can quickly calculate the fractions, however the downside of fixed point notation is that to store a number such as 0.00000000000001, we require a great deal of storage bits, just to store a single bit of data. The way around this is with floating point notation.

Floating Point

Floating point numbers are the equivalent of scientific form in binary. Just as we can express 1,200,000,000,000 as 0.12×10^{13}, we can express binary numbers as multiples of a power of two.



In our denary example of 0.12×10^{13}, 0.12 is the mantissa and 13 is the exponent. The mantissa is the coefficient, the mantissa holds the digits and the exponent defines where to place the decimal point.



With a binary floating point fraction, it might be that a 16bit word is made up of a 10 bit mantissa and a 6 bit exponent. Just as with a normal binary number, the first bit (most significant bit) of each part will indicate whether that part is positive or negative.

To evaluate our example of 0100 1100 1100 0100 there are a number of steps to follow.

- Firstly, we identify 0100110011 as being our mantissa, which is positive since it starts in a zero. Similarly 000100 is our exponent, which is also positive. The decimal point can be located after the first (sign) bit.
  $= 0.100110011 \times 2^{000100}$

- Now we evaluate the exponent in denary. 000100 = 4.
  $= 0.100110011 \times 2^{4}$

- Next we move the decimal point *exponent* times to the right of its current location. If the exponent is negative we move to the left, not the right.
  = 01001.10011

- Lastly we can now convert this number to denary just as we did fixed point numbers.
  $= 8 + \frac{1}{2} + \frac{1}{16} + \frac{1}{32}$
  = 8.59375

Normalisation

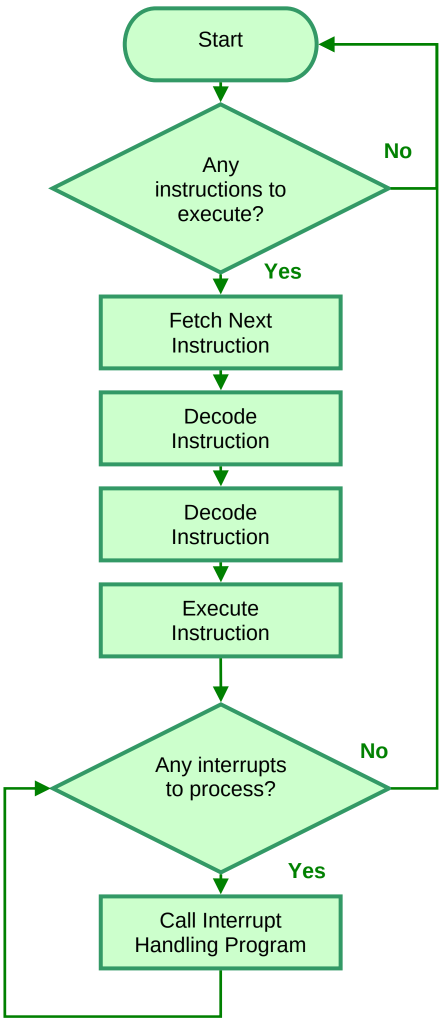
Normalisation is designed to give maximum precision for a given number of bits. A number in floating point notation could be written in many ways, for example the number 1200 could be written as 1200, 1.2×10^{3}, 12×10^{2}, etc. In normalised form however there is only one correct way to represent any given number – and that is so that the mantissa lies between ½ and 1 (or -½ and -1 in the case of negative mantissas). In binary this is achieved by altering the exponent such that the mantissa starts with “01” if it is positive or “10” if it is negative (i.e. we have removed all leading zeros.)

Machine Level

The Processor

The Fetch-Execute Cycle

The processor processes instructions; these instructions are dealt with in the manner described by the Fetch-Execute cycle:



Note: An interrupt is a signal generated by a device when it requires the attention of the processor. The following types of interrupt may occur:

- Interrupts by running process. This might be, for example, that the process needs to perform I/O operations and therefore pause execution until this is complete.
- I/O Interrupts. These are generated by I/O hardware and may be sent when a transfer is complete, or if an error occurred in transfer.
- Timer Interrupts. These interrupt the current job to perform regular tasks which must be performed at set intervals. (For example, in a multi-user OS to allow the other users' jobs to be processed for a period).
- Program Check Interrupts. These are caused by errors in the program.
- Machine Check Interrupts. These are caused by malfunctioning hardware.

Structure of the Processor

A processor will typically contain these three key components:

- The Arithmetic-Logic Unit (ALU). This is where (as the name indicates) all arithmetic and logic operations are carried out in the processor.
- The Control Unit. This co-ordinates all the activities taking place in the CPU, memory and other peripheral devices via the use of control signals. The control unit is where instructions are decoded and executed; it may call upon the ALU or other components to aid in the execution of instructions.
- The System Clock. This sends out a sequence of timing pulses or signals, which are used to step the control unit through its operations.

Additionally, the processor contains a number of registers; these are very fast storage locations inside the processor that are used for temporary storage of binary values. The processor contains a number of special purpose registers, which have dedicated uses, and general purpose registers which may be used for arithmetic function and are a sort of "working area".

There are all kinds of special purpose registers, but some of the most common are details below:

- The **Program Counter** (PC). This holds the address of the next instruction to be executed. Each instruction the processor is going to perform is stored in a memory address; this register is automatically incremented so that it always holds the memory address of the next instruction.
- The **Current Instruction Register** (CIR, or IR). This contains the operator and the operand of the current instruction. (Where in the example instruction `MOVE 80,#13` "MOVE" is the operator, and "80" and "#13" are the operands.).
- The **Memory Address Register** (MAR). This holds the address of a memory location from which data will be read from or written to. Data might be a variable as part of a program, or an instruction for the processor to execute.
- The **Memory Data Register** (MDR, or MBR). This is used to store data which has been read from or is ready to write to memory. All transfers from memory to CPU go through this buffer.
- The **Status Register** (SR). This contains status bits which reflect on the results of an instruction. For example, if there is an error in the operation (such as an overflow) this will be recorded in the status register. The Status Registers also store data about interrupts.

We can now express the fetch execute cycle in terms of the processor's registers:

1. The address of the next instruction is copied from the PC to the MAR.
2. The instruction at that address is loaded into the MDR. Simultaneously the PC is incremented for the *next* instruction.
3. The MDR is copied to the CIR.
4. The contents of the CIR are decoded.
5. The decoded contents are executed.

Interrupt Handling

At the end of each Fetch-Execute cycle, the contents of the interrupt registers are checked. Should there be an interrupt; the following steps will typically be taken:

1. The contents of the PC and other registers will be stored safely in a stack.
2. The highest priority interrupt is identified. Interrupts with a lower priority are disabled.
3. The source of the interrupt is identified.
4. The start address of the interrupt handler is loaded into the PC.
5. The interrupt handler is executed.
6. Interrupts are enabled again, and the cycle will restart with any further interrupts.
7. The PC and other registers are “popped” from the stack and restored.
8. The user’s program resumes with the next step in its cycle.

Interrupts have different priorities; this is so if two are received simultaneously the computer knows which the more important one to execute first is. Typically there are four levels of priority, which are (in descending order) Hardware Failure, Program Interrupts, Timer Interrupts, I/O Interrupts).

When dealing with an interrupt, the computer has to know which interrupt handler to call for which interrupt. One method of doing this is known as the vectored interrupt mechanism. In this approach a complete list of interrupts and the memory address of their handler is stored in a table called the interrupt vector table.

The interrupt supplies an offset number, which identifies the interrupt uniquely. This offset is added to a base term, and the resultant number is the memory address of a *pointer* to the memory location of the handler routine. This is explained in the example below:

Memory Location	Data
5001	6002
5002	6280
5003	7580
…	
6002	;Interrupt Handler for 001
6280	;Interrupt Handler for 002
7580	;Interrupt Handler for 003

} The interrupt vector table

If the interrupt 002 is received, the base number 5000 is added to it, which allows the processor to know that the handler can be found by opening the data stored at address 5002. The address 5002 simply stores a pointer to another memory location, 6280, where the actual handler routine begins.

The advantage of this approach is that each interrupt only needs to give the processor an offset number, such as 002, and the processor can determine from that the correct memory location to use. This is more efficient than the interrupt sending the full memory address itself. This approach also allows the interrupt routines to be stored anywhere in the memory, with the pointer table updated to reflect if a handler routine is moved.

Processor Performance

The traditional processor's performance is affected by these four main components:

- Clock Speed

  As previously mentioned the processor contains a timing device known as the clock. This sends out signals at a given interval, and all processes within the computer will start with one of these pulses. A process may take any amount of time to complete, but it will only start on a pulse. It therefore makes sense that a processor with a faster clock speed will perform faster, since more pulses will be sent out in the same time frame. The clock speed is generally quoted in factors of Hertz, with modern processors typically Gigahertz.

- Word Size

  The word size of a computer is the number of bits it can process at a time. Bits are grouped into words, the length of a word varies but it is typically, 8, 16, 32, 64 or 128 bits. Obviously if the processor is able to deal with more bits at a time then it is going to perform better. Typically home computers are 32bit, but 64bit technology is becoming widespread too.

- Bus Width

  The addresses of data, and the data itself, is transmitted along buses inside the computer. The width of the bus is the number of bits it can store. A wider data bus will allow more data to be sent at a time, and therefore the processor will perform more efficiently. A wider address bus will increase the number of memory addresses a computer may use. For example an 8 bit bus will only allow a value between 0 and 255 to be transmitted at a time.

- Architecture

  The architecture of a processor will affect its performance, a better designed processor will perform better than a different processor.

Instructions in the Processor

Machine Code

The processor obeys instructions in machine code. Machine instructions are binary codes; different processors have different codes for different instructions depending on their application. The complete list of possible codes and their corresponding operations is known as the instruction set.

Generally there are four types of instructions:

- Data Transfer. For moving information from one place to another.
- Arithmetic. For mathematical operations.
- Logical. For logical expressions and operations.
- Test and Branch. To create conditional or unconditional code segments.

Writing in machine code is a very tedious process, with the programmer having to work purely in binary patterns. Because of this we tend to use assembly language, which uses mnemonics instead of binary codes to represent operations. The assembler then translates the assembly language to machine code.

Assembly Language

Whilst assembly language is fairly basic to code with in comparison to a high level language such as Delphi or Visual Basic, it still has its uses. Mostly when the programmer needs to access the processor directly, either to access registers or write to exact memory locations. Device drivers for example are usually coded in assembly, the other use of assembly is where the code must execute as efficiently as possible with minimal storage space (such as the bootstrap loader for an OS).

Commands in assembly consist of an operator (or op code) and then a number of operands. Some examples are shown below:

- `RTS` (Return from a subroutine)
- `LDA #20` (Load the number 20 into the accumulator)
- `MVE #20, R1` (Move the number 20 into the register R1)

In these examples, the three letter mnemonic is the op code, and any values beyond it are operands. Note that RTS is one of few commands to have no operand.

Instructions

Data Transfer

Data transfer instructions involve moving data from one place to another.

```
➢ MOVE #30, R1      ;Move the number 30 into register 1
➢ LDA R1            ;Load register 1 into the accumulator
➢ STA R2            ;Store the accumulator in register 2
```

Arithmetic

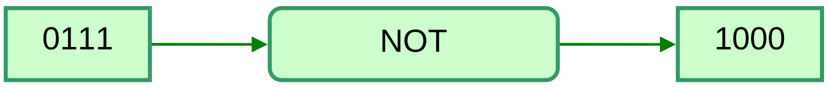
Many processors only support additional and subtraction as basic operations, whilst others include a more comprehensive list.

```
➢ ADC #2      ;Add 2 to the accumulator
➢ SBC #2      ;Subtract 2 from the accumulator
➢ INC         ;Increment the accumulator by 1
➢ ABS R1      ;Absolute (positive) value of register 1
```

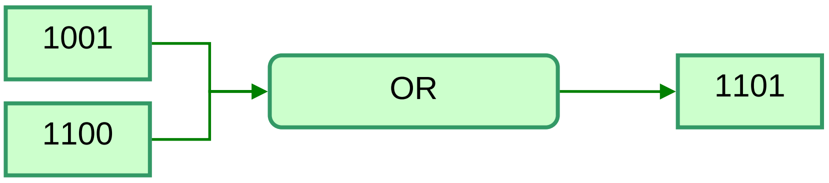
Logical

Logical operations generally take place on bit patterns in the processor. We tend to use these four main commands:

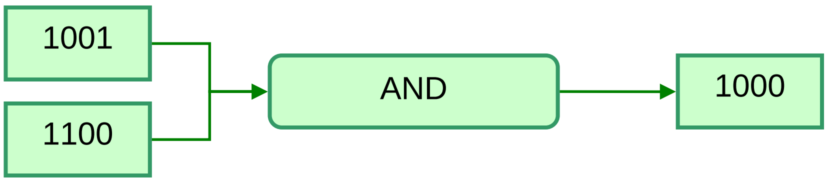
- NOT: This inverts the bit pattern, and can be used as part of an operation to find the two's compliment of a number. *Returns the inverse bit pattern.*



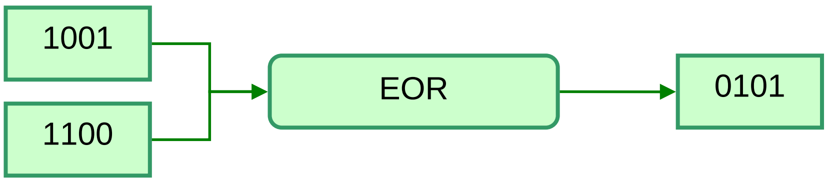
- OR: We can use this to set bits to 1 within a number, without affecting the other bits. *Returns "1" bits which are present in either of the two inputs.*



- AND: We can use this to mask parts of a binary number out. *Returns "1" bits which are present in both input patterns.*



- EOR (or XOR): We can use this to compare two binary patterns. *Returns a "1" where bits in the inputs do not match.*



Shifts

Shifting operations may alter the location of bits within the number. These shifts tend to be used to examine individual bits within a pattern by moving them into the carry bit position and examining that.

Shifts are logical, arithmetic or circular (rotating). In a logical shift (`LSR`), bits are moved with no regards as to the previous value of the number, for example after shifting a negative number on bit to the right, a zero is now prefixed to the number, and the number now reads as positive – when in fact it is negative but shifted to the right. The solution to this is an arithmetic shift (`ASR`), which would in this case prefix a one, to ensure the shifted number is also negative.

In a rotating shift (`ROR`), the number chomped from the right end of the pattern goes into the carry bit position, and then on the next step of the shift, it is put back at the left end, so that after *n* number of steps where *n* is the number of bits in the pattern, the number will be shifted back to its original position.

Branching and Comparing

- Compare (`CMP`). This can be used to compare two values in the processor; they may be registers, memory locations or fixed values. The result of the comparison will be stored in a status bit ready for use at the next stage of the program.
- Branch If Equal (`BEQ`) and Branch If Not Equal (`BNE`) are used to alter the path of executing within a program given the results of a comparison. The operand of this command will be a label or line number to branch (jump) to.
- Jump (`JMP`) will cause an unconditional jump to a defined label or line in the code given in the operand. The same is true of `JSR` which will jump to a defined subroutine in the code.

Example…

```
;Program to calculate multiplication
;Example values of 6x8

MOV #6, X           ;Move 6 into register X
MOV #8, Y           ;Move 8 into register Y

Loop:
LDA X               ;Load X into the accumulator
ADC X               ;Add X to the accumulator
STA X               ;Store content of accumulator in X

LDA Y               ;Load Y into the accumulator
SBC #1              ;Subtract accumulator by 1
STA Y               ;Store accumulator in Y

CMP Y, #0           ;Compare Y, our counter, with zero

BNE Loop            ;Branch to “Loop” if not equal
```

Operands and Addressing

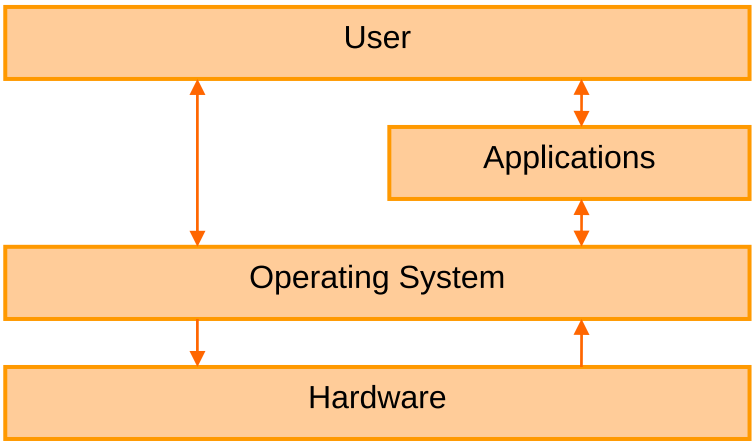
When specifying operands there are different sources from which data might be coming, as well as numerous formats that data may take. For example, if we just had an operand of "10", does this mean 10 in binary, 10 in denary, 10 in hexadecimal, the memory location 10, or something memory location 10 points to? To get around this, we have different addressing modes, and we show these using various special characters, as described below:

`LDA #&10`	**Immediate Addressing.** Loads the value "10" in hexadecimal.
`LDA #10b`	**Immediate Addressing.** Loads the value "10" in binary
`LDA #10`	**Immediate Addressing.** Loads the value "10" in denary
`LDA 10`	**Direct Addressing.** Loads the value of memory location number 10
`LDA (10)`	**Indirect Addressing.** Loads the memory location which is pointed to by memory location number 10
`LDA #5,#100`	**Base Register Addressing.** Loads the memory location formed by adding the first operand to the second (base) operand, so 105. This might be done by means of the index register, X, which is known as **Indexed Addressing**.
`JMP +10`	**Relative Addressing.** Jumps to the instruction located 10 bytes on from the current instruction.

Operating System Level

Concepts

Role of the OS



The operating system is a set of programs which manages the operation of a computer. It sits between the hardware and the user to provide an interface by which to control processes and devices on the computer.

The operating system has four primary functions:

- Process Management
- Memory Management
- I/O Control
- File Management

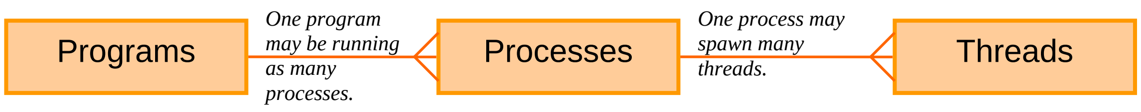
Processes and Threads

A **process** is a **program** in execution. The distinction is that a program is a static bit of code; the process is the instance of that program being executed. For example, if the same program is running twice, the program will be loaded into memory once, but it will have two processes - one for each instance of the program.

The operating system maintains information about all the processes in the process control block (PCB). The contains information such as the process's unique ID, the current state of the process (explained in a moment), the program counter, pointers to allocated memory space and resources, CPU time used so far, and estimated time to completion.

The process can be in any one of three states, running (if it is currently using the processor), runnable (if it is ready to be executed and is just waiting for the processor to become available), or suspended if it is waiting for I/O and could not execute even if the processor was free.

Threads are paths of execution within a process. For example, the same code may have two sections being executed simultaneously. One process may have many threads, which are stored in the thread control block (TCB) which stores information very much like the PCB. An operating system which allows multithreading (i.e. more than one thread-per-process) allows the programmer to create programs which can have two or more parts executing concurrently. This is more efficient than spawning a new *process* because threads share address space and therefore require less resources than new processes.


Programs
One program may be running as many processes.
Processes
One process may spawn many threads.
Threads

Resource Scheduling

Priorities are assigned to jobs (processes) by both the user and by the operating system. This is the job of the scheduler, which obeys a scheduling policy.

The scheduling policy will aim to:

- Maximise the number of jobs being executing in the shortest possible time.
- Ensure all users receive an acceptable response time for their actions.
- Balance the use of resources.
- Ensuring all jobs, regardless of apparent priority, do get executed eventually.
- Enforce user preferences on job priorities.
- Not be so complicated the processor spends all its time deciding which job to execute, rather than actually executing anything…
- Resolve conflicts between its criteria.
- Resolve deadlocks where two or more interdependent processes are left waiting for the other to complete.

The scheduler will make its decisions based upon information such as what resources a program needs, the urgency of the process, the time the process has been waiting and how much time is needed to complete the process.

Three possible strategies a policy may take are:

- Shortest Job First.
- Shortest time remaining first.
- Round Robin. (FIFO within a limited time slice per user).

Memory Management

Address Assignment

The memory manager is responsible for allocating physical memory to different processes. A process cannot function (or indeed exist) without having some memory assigned to it. A good memory manager will aim to allocate space in such a way that as many processes are able to execute concurrently as possible, whilst protecting processes' memory space (or indeed un-protecting it to allow sharing of resources where required). All of this should be clear to the programmer and provide a satisfactory level of performance. Generally memory may be assigned to processes either statically (with an absolute loader) or dynamically (with a relocating loader).

In static assignment, the program is loaded into a fixed area of main memory, and the memory addresses used are "hard coded" into the compiled machine code. The code will therefore only execute if the program is always loaded into the same memory space.

With dynamic or relocating assignment, the program is loaded into any area of main memory, and all addresses within the code are relative to a base register. For example the code might speak of memory address "200", if the base register is "100000", then the address which will actually be used is 100200, however if we change the base register to "100500" the code will not need changing, but the new address used will be "100700". This mapping is controlled by the memory management unit (MMU).

Virtual Memory

Virtual memory is where the hard disk is used to store data which would otherwise be stored in main memory on a temporary basis.

The main memory is divided into blocks, which are occupied by “pages” of a program (segments of its code and data). What happens is that pages of programs which are not currently needed are “swapped” out to the hard disk temporarily, to make way for other pages in the main memory which *are* currently needed. Each process will have a page management table which shows where the pages are at that moment. The file which these pages are put into on the hard disk is known as the pagefile.

Dynamically Linked Libraries

Libraries are not independent programs in themselves, but hold collections of routines used within other programs. These DLLs might be provided by the operating system to access some of its functions or the information it holds, or by software developers to reduce code duplication between two programs with common traits between them.

The DLLs is linked from the developer’s source code, and accessed when the program is *run* (not when it is compiled, the DLL does not become embedded within the application). For example, Microsoft Office has lots of DLLs since it has a number of programs in its suite which share the same functions and features.

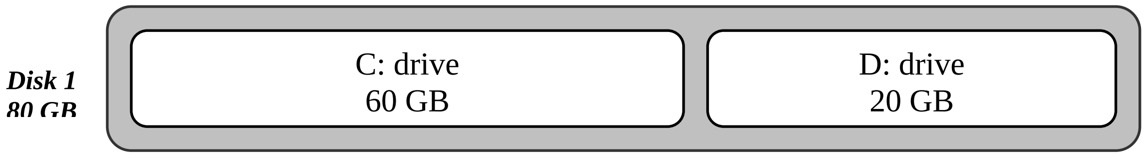
File Management

File Management Systems

The file management system is the system which controls access to the storage devices. Some typical examples are NTFS (Windows NT File System), FAT (File Allocation Table) and EXT (extended File System). The file management system has four main functions:

- To allocate space on storage devices to data. Space is usually divided into slots or blocks called **allocation units**. The system must also deallocate space when a file is deleted.
- To track which allocation units store which files, since a file may be spread across units.
- To control access rights and permissions.
- To map logical addresses to physical addresses.

The File System may allow the creation of logical drives (or partitions). This is shown in the diagram below, where a computer showing two drives (C and D) has in fact one disk which the file system has cut into virtual slices.



I/O Management

The I/O bus allows communications with storage devices; each device on the bus has an address, and via a series of interrupts knows when to send data. A device controller sits between the storage device and the bus, to provide a generic interface for the hardware to connect to. Software known as device drivers will allow the operating system to manage the actions which must take place to correctly communicate with different storage devices.

Buffering

Data is always transferred from the storage device to the processor through a buffer. The buffer is fixed at the size of an allocation unit, and so every time data is required an entire allocation unit is sent to the buffer, then just the required data is read from the buffer.

Classification

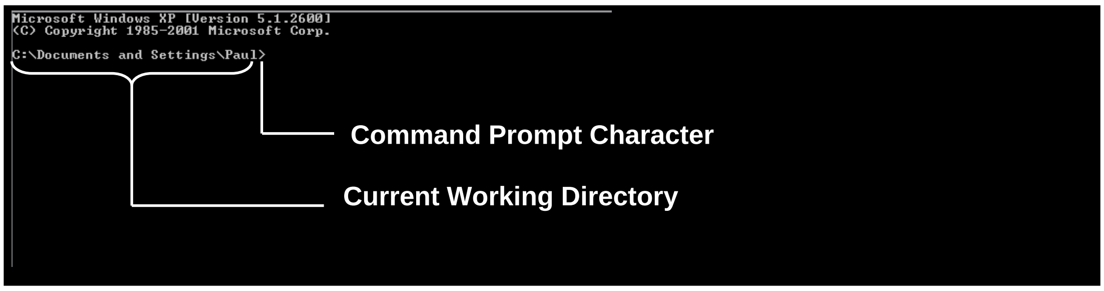
Interface

The interface of an operating system is the way in which it can communicate with the user. The interface may be command line, graphical or take the form of job control language.

Command Line Interface.

Here the user accesses the computer via a terminal which provides a text input and output facility. A command line interpreter (CLI) is responsible for identifying the correct process the user has requested.

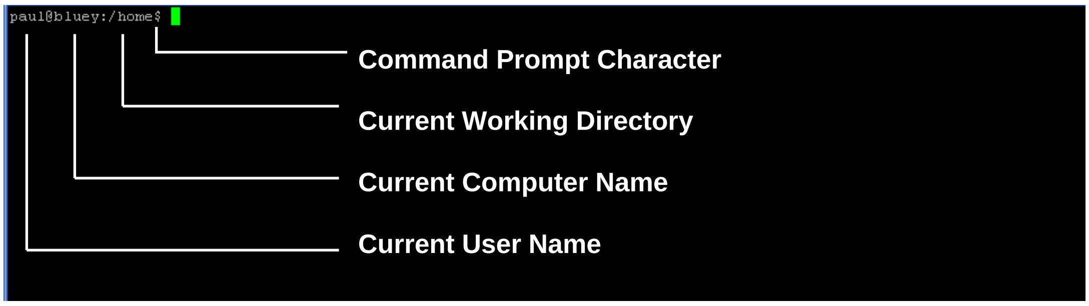
In Windows, a command line interface will look something like:



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Paul>
```

However, in operating systems it will look different. For example in a Unix/Linux system it might look like:



```
paul@bluey:/home$
```

Job Control Language.

(JCL) In a JCL interface a user has no direct way of interacting with the computer; instead they prepare a program using a series of instructions, and allow this program to be executed with any results being prepared for viewing at a later date.

A JCL program will have to specify things such as:

- Who owns the job
- What priority the job has
- The names of any data files used.
- Action to take if the task does not execute properly
- The maximum resources to allocate to the job

Graphical User Interfaces

A GUI allows the user to interact in more ways with the computer. This is often by use of a WIMP (Windows, Icons, Mouse and Pointers) system which is more intuitive than a command line system. GUIs do however use far more resources than a basic interface and can be inefficient when it comes to repeating processes numerous times.

Modes of Operation

- Batch. This is where a job (often defined in JCL) runs from beginning to end without direct user intervention.
- Interactive. This is where the user interacts directly with the system, entering commands and data whilst the program executes.
- Real Time. This is where data is processed with minimal delay so the system can respond instantly to changes in variables.

- Single User. The OS can only allow one user to be using the machine at a given time.
- Single Process. The OS can only execute a single process at a time.
- Multi Programming. The OS appears to be able to process two or more programs concurrently.
- Multi User. The OS allows two or more users to be communicating with the computer simultaneously.
- Multi Tasking. The OS allows the user to be working on two or more (related) tasks simultaneously, and allows these tasks to interact.
- Network. A computer which wishes to communicate with other computers must run network software to enable such communication.

Network Concepts

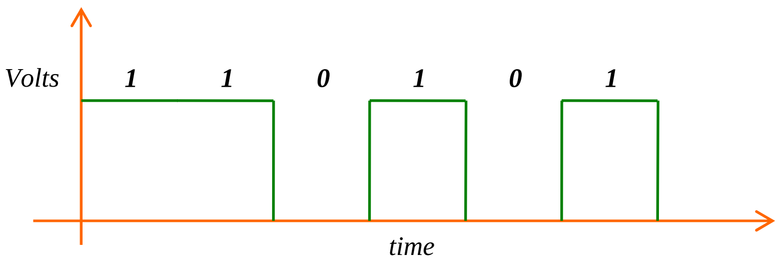
Networking Methods

Terminology

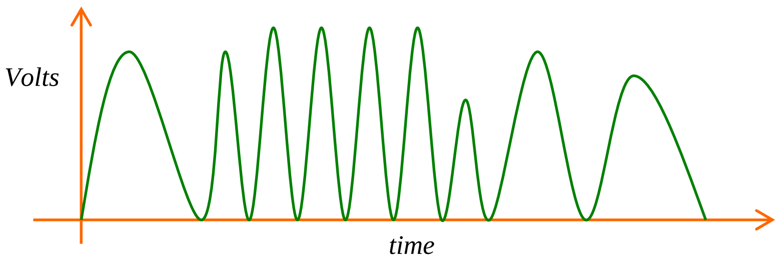
- A **Network** is two or more computers sharing resources and services.
- A **Local Area Network** (LAN) is computers which are close together sharing resources and services.
- A **Wide Area Network** (WAN) is computers which are geographically remote sharing resources and services.

Modes of Operation

- **Baseband** is a system which carries one signal at a time down a cable. The presence or absence of a signal in the cable counts as a binary 1 or 0. This system can operate very quickly – but only operates over relatively short distances. Most LANs operate in this mode.

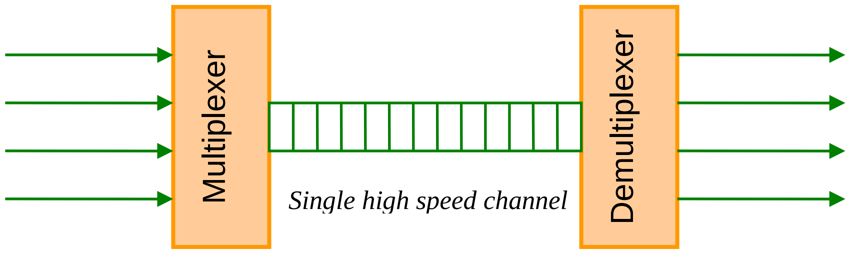


- **Broadband** is a system which can carry multiple signals over one fixed carrier wave. The signals 1 and 0 are sent as variations on the carrier wave. The bandwidth of the line is shared by several "channels" of data. This system is generally used in WANs.



- **Synchronous data transmission** is where timing signals are used to ensure both ends of the transmission line know what data they are receiving when. This is as opposed to start and stop bits. By removing the start and stop bit after each character and just using them at the beginning and end of each block of data, faster transmission speeds are possible. However, this is only reliable over relatively short distances and therefore mainly used in LANs.

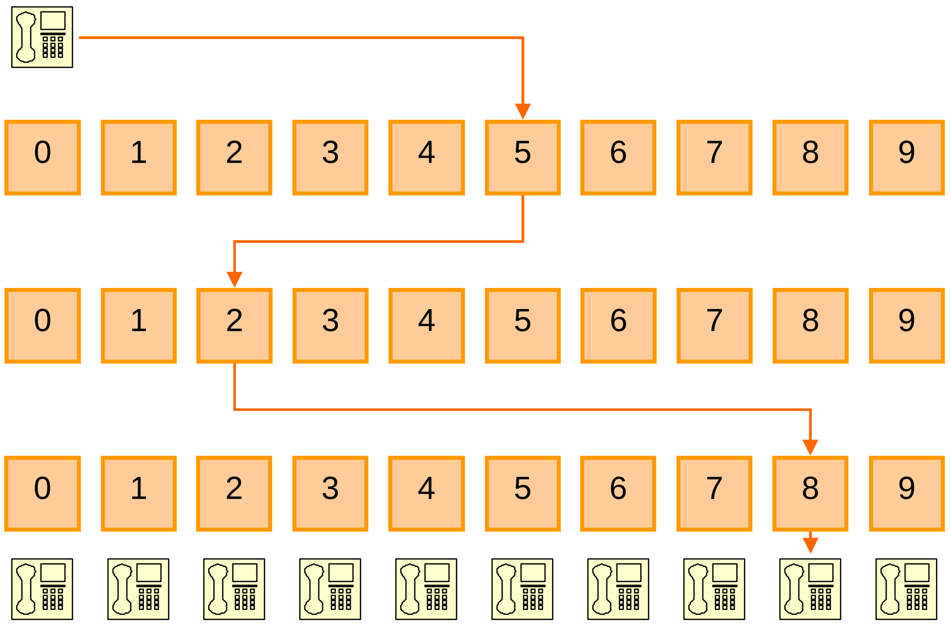
- **Time Division Multiplexing** is a method of combining several data streams into one communication channel. This can save on the cost of communication channels and increase efficiency.



With this method, the use of the high speed channel is split into time slots, each data stream in then has a slot on the high speed channel, before handing over to the next data stream, etc. For this to run effectively the speed of the high speed line must be at least *n* times faster than one of the input data streams, where *n* is the number of input steams.

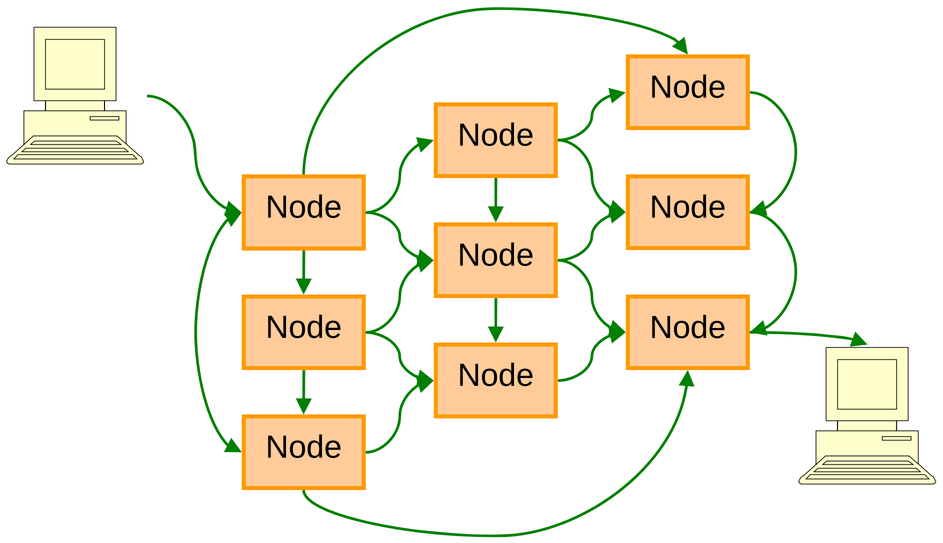
- **Circuit Switching** is where a (single) physical pathway of cabling is established between the sender and receiver for the duration of a data transfer. This is very much like the plain old telephone system (POTS). A complex series of electric switches at telephone exchanges ensures that the senders phone is physically connected to the receivers phone – and so a physical pathway is formed.

As shown in the diagram below, when you dial a phone number, this corresponds to the switches which are to be chosen. So in the example below, the caller dialled 528, to reach one of a possible 999 phones.



- **Packet Switching Systems** (PPS) break down messages into packets (or datagrams). These are fixed length blocks of information which are then all transmitted individually through the network.

  In a system such as TCP/IP, a large computer network directs packets around until they reach their destination. They might not take the same route to reach their destination (although in a **virtual circuit**, there is one dedicated "virtual" pathway formed between source and destination).



  The path a packet should take may be predetermined, as with a **virtual circuit** system, the two computers arrange a route before beginning data transfer. However, it may be that nodes in the network simply redirect the packet around until it reaches the destination it is addressed to, this would be the case in a **datagram** system.

  Packet switching is used in the Internet, and it has many advantages over circuit switching. Firstly it is a more efficient use of lines, and also it is less likely to be affected by things like line failure and faults, since the packets are simply able to use another route instead – with a multitude of different routes possible. Finally security can be seen to be better in a PSS since the packets are sent via different routes and interleaved with other information, which makes intercepting a whole data block very difficult (unlike a circuit switching system where you would only need to tap one line to listen in on all communications between sender and receiver).

- **Asynchronous Transfer mode (ATM)** this is a packet switching system which employs virtual circuits to transfer multiple data channels over a single line, with each channel able to dynamically set the bandwidth it requires. ATM is used over digital lines, and can perform very high transmission rates with virtually no noise or errors.

Wide Area Connection Methods

- **Dial Up** uses a MODEM (MOulator DEModulator) to convert the computer's digital signal into an analogue one that can be sent down a normal phone line.
- **ISDN** (Integrated Services Digital Network) is a system of using the phone line digitally to transmit internet and other services over a regular phone line at high speed.
- **Cable Television** are often capable of connecting users to the internet by transmitting data down their coaxial or fibre optic lines.
- **ADSL** (Asymmetric Digital Subscriber Line) is a technology which uses broadband principles to allow multiple data channels to be modulated into one signal which can be sent down a regular phone line *at the same time* as voice signals from a phone line.

Protocols

A protocol is a predefined system of signals, codes and rules to be used for data transfer between two systems. A standard protocol is one which conforms to a standard laid down by an authority to ensure any computer operating with the protocol can communicate correctly with all others on the same protocol.

TCP/IP Suite

TCP/IP is one of the most common standard protocol suites, and it is what is used on the internet. It is called a protocol suite because technically it is a collection of many protocols which work together.

The protocol suite operates on four distinct levels:

Application Layer	This provides applications with access to the communications environment. Example protocols on this level are HTTP (Hypertext Transfer Protocol), Telnet, FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol) and POP (Post Office Protocol).
Transport Layer	This is responsible for dividing the data from the application layer into packets, ready for sending down the network. The protocols acting on this layer are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).
Network Layer	This layer is responsible for addressing, routing and sequencing data. Protocols acting on this layer are IP (internet protocol) and ARP (Address Resolution Protocol).
Data Link Layer	This layer interfaces directly with the cable handling direct data flow. It can support various connection types and network topologies.

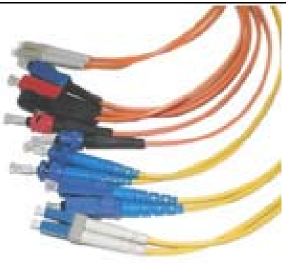
The TCP/IP system allows for what are known as sockets, this is a construct which allows multiple processes to apparently use the network connection simultaneously without interfering with each other. For example, a computer which runs as a web server will listen on port 80 for incoming HTTP requests. The same computer might be listening on port 22 for SSH requests, and 11 for SMTP requests. The combination of port and IP address is known as a socket. This is represented in the form [IP]:[Port], so I.e. 192.168.0.1:80 is the computer known as 192.168.0.1 operating on port 80.

Communication Media

Wired

The type of cabling used can dramatically affect the performance of a network, as well as this cabling must be selected that is suitable for the budget, network type and environment.

- Twisted Pair. This is used in the telephone network and many newer LANs. It can come as STP (shielded twister pair) or UTP (unshielded, which is less bulky).
- Coaxial Cable. This is used in older LANs, as well as being used in aerials. It is generally high quality and well insulated and can transmit data at high speeds.
- Fibre Optic Cable. Through this the principle of total internal reflection (TIR) allows pulses of light, rather than voltage, to send a signal from source to destination.

		
Coaxial Cables	Twister Pair Cables	Fibre Optic Cables

Wireless

Wireless networking may be used on both a local scale, say to provide access to the internal all around a house, as well as on a heavy duty scale to transmit data across continents.

- Radio Waves. These are used in most localised wireless networks (WLANS) where a typical range might be 50m, as well as in the distribution of the internet to remote areas, where ranges might span as far as those of a radio station.
- Microwaves. These are used by mobile telephones and have a range of about 50km.
- Satellite. Data can be bounced around the word by communicating through a geosynchronous satellite.

Network Topologies

Implementations

Server Vs Peer to Peer

The Internet

Programming Concepts

Database Concepts

Systems Development