F453

Advanced Computing Theory

Revision Notes

Alicia Sykes

June 2012

Functions of Operating Systems

What the Specification Says

Describe the main features of operating systems, for example memory management, and scheduling algorithms;
Explain how interrupts are used to obtain processor time and how processing of interrupted jobs may later be resumed,
(typical sources of interrupts should be identified and any algorithms and data structures should be described);
Define and explain the purpose of scheduling, job queues, priorities and how they are used to manage job throughput;
Explain how memory is managed in a typical modern computer system (virtual memory, paging and segmentation should be described along with some of the problems which could occur, such as disk threshing);
Describe spooling, explaining why it is used;
Describe the main components of a typical desktop PC operating system, including the file allocation table (FAT) and how it is used, and the purpose of the boot file.

Alicia Sykes

Notes

Operating systems must:

- Provide and manage hardware resources
- Provide an interface between the user and the machine
- Provide an interface between application software and the machine
- Provide security for data on the system
- Provide utility software to allow maintenance to be done

Why OS's were developed:

- If a program needed an input or an output the program would have to contain the code to do this. There were often multiple occurrences.
- Short sub-routines were developed to carry out these basic tasks, like get the key pressed.
- The joining together of these sub-routines lead to the input-output-control-system (IOCS). As this started to get more complex it turned into the OS.
- Also as programs started to be written in higher-level languages translators were needed, these were incorporated into the OS

Types of Interrupts

- I/O interrupts – generated by the I/O devices to signal a job is complete or an error has occurred
- Timer interrupt – generated by internal clock to signal that the processor should attend a time critical activity
- Hardware error – for example power failure, where the computer must try to shut down as safely as possible.
- Program interrupt – generated due to an error in a program, such as violation of memory usage.

Dealing with interrupts

- Interrupts are stored in queues
- The position in the queue is determined by its importance to the user, system and job – priority.

Objectives of Scheduling

- Maximise the use of the whole computer system
- Be fair to all users
- Provide reasonable response time to all users – weather online or batch processing users
- Prevent the system failing or becoming overloaded
- Ensure the system is consistent by giving similar response times to similar

Deciding which order to schedule jobs:

- Priority – where some jobs have higher priority than others
- I/O bound jobs – if a job is bound to a device, and is not served first it may prevent the devise being served efficiently.
- Type of job – batch processing and real time jobs require different response times

- Resources used so far
- Waiting time – the amount of time a job has been waiting to use the processor

Ready, running and blocked queue
- A job may be in any one of three states
- Ready
- Running
- Blocked – e.g. waiting for a peripheral devise

What is a scheduler?
The scheduler is just a program, a set of instructions, used by the OS to decide where each of the jobs which are in its control should be and in what order to manipulate them.

Types of schedulers
- High-level scheduler (HLS) – places job in the ready queue, makes sure that the system is not overloaded.
- Medium-level scheduler (MLS) – swaps jobs between the main memory and backing store.
- Low-level scheduler (LLS) – moves jobs in and out of the ready state, and decides which order to place them in the running state

Pre-emptive and non-pre-emptive schedulers
- Non-pre-emptive - puts a job in the running state then leaves it there until the job does not need to run any more but needs to go to one of the other states, for example it may wait for an input or may have finished executing altogether
- Pre-emptive - has the power to decide that the job that is running should be made to stop to make way for another job

Methods of deciding what order jobs should go in
- First com first serve (FCFS) – the first job to enter the running queue is the first job to enter the running state. Favours long jobs.
- Shortest job first (SJF) – sort the jobs in the ready queue in ascending order of time needed
- Round robin (RR) – gives each job a maximum length of processor time, it will then go to the back of the queue again. Once its finished, it leaves the queue.
- Shortest remaining time (SRT) – the ready queue is sorted on the amount of expecting time still to do. Favours short jobs, and there's a danger long jobs may never be started.
- Multi-level feedback queues (MFQ) – involves several queues of different priorities with jobs migrating across

Memory management
In order for a job to be processed, it must be stored in the memory – obviously. If several job are stored in the memory, their data must be protected from the actions of other jobs.

How free memory can be allocated
If a small gap in the memory becomes free, but it is not big enough for the next job, there are a number of options:

- All the jobs could be moved upwards, so that all the bits of free space are together, and the next job can fit. Although this would use a lot of processing power, as all the addresses would have to be recalculated.
- Another solution would be to split up the new job into available free memory. Although doing this could cause the jobs to all get split up into many tiny parts.

Linkers and Loaders

- The loader – small utility program that loads jobs and adjusts addresses.
- The linker – links parts of the program together

Linkers and loaders are integral to the management of memory space

Paging and Segmentation

- Pages are equal sized sections, where a job will be allocated a number of pages to store itself in. They may be in order, or out of order.
- Segments are of variable size to match the size of the job

Virtual Memory
When data is not needed to be accessed immediately, or programs have been 'minimised' for a long time, they may be transferred to virtual memory so as not to fill up the main memory. Virtual memory is on the hard disk, although it behaves exactly like the RAM.

Disk Thrashing
This is where the code contains many jump instructions so the processor spends most the time switching between the virtual memory and main memory. It involves the disk continuously searching for pages.

Spooling
This is where an I/O devise is a lot slower than the processor, so the job is moved into another storage location while it is being used. It is kept track of in a spool queue. A spool queue is only a reference to the jobs, and not a proper queue. A spool queue also allows for priorities, more important jobs can push into the queue and be done first.

Multi-tasking OS
There are two main types of OS, command driven e.g. MS DOS, and GUI. All OS's allow the user to do basic things like rename, delete and move files stored in a hierarchical structure on the disk. Most GUI's also allow for much more than this, they let the user multi-task, where several programs are running.

File Allocation Table (FAT)

- This is a table that uses a linked to point to the blocks on the disk that contain files.
- In order for this to be done, the disk must first be formatted, which involves dividing the disk, radially, into sectors and into concentric circles called tracks. Two or more sectors on a single track make up a cluster.
- To find a file, the OS looks for the file name, and gets the corresponding cluster number, which can be used to find the file.
- When a file is deleted, the clusters that were used to find the file can be set to zero.
- The FAT table is usually loaded into the RAM to speed things up.

Loading and OS

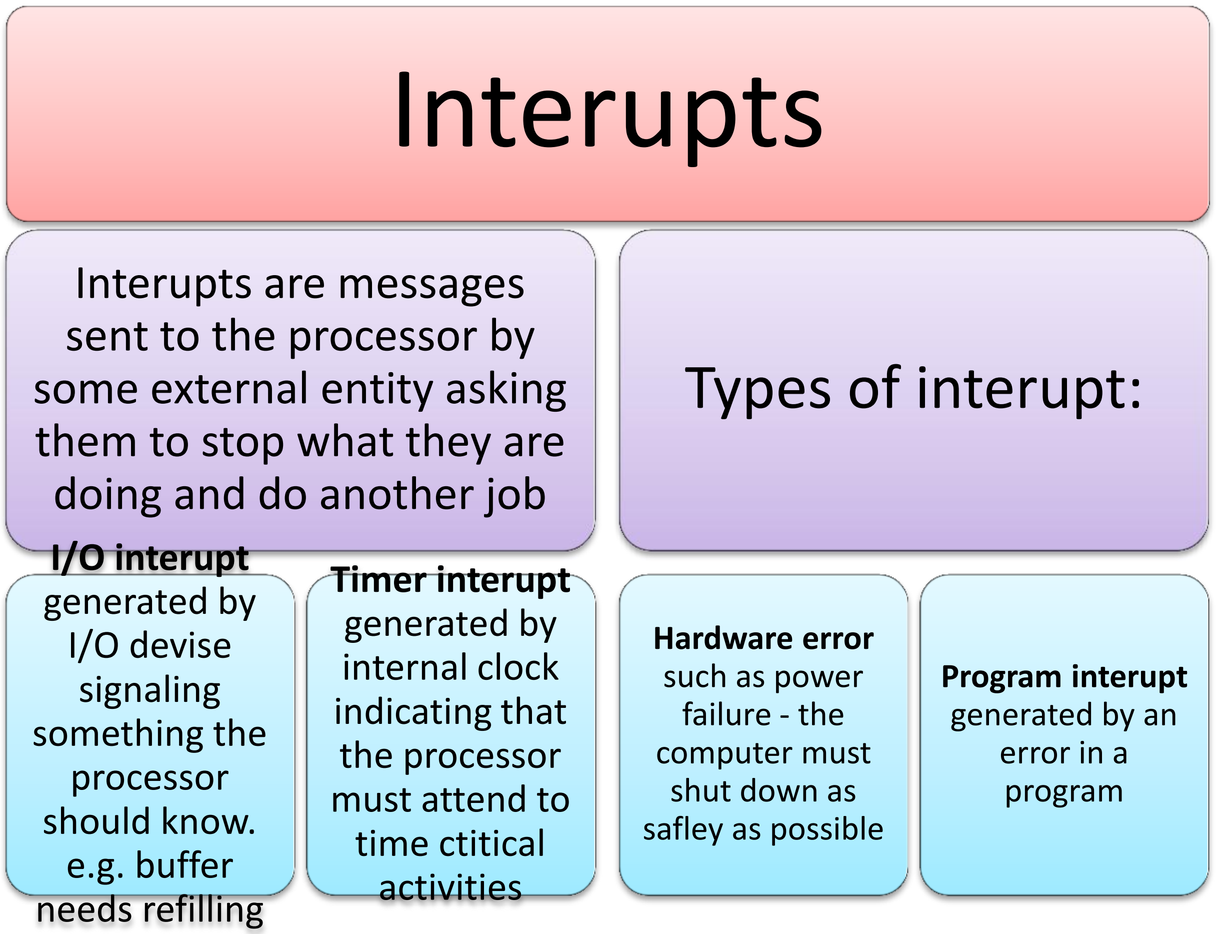
- PC is switched on
- Contains very few instructions
- Runs the power on self-test (POST) which resides in the permanent memory
- The POST clears the registers, loads the address of the first instruction in the boot program into the program counter
- The boot program is stored in the read-only memory and contains the skeleton of basic input output system. The BIOS structure is stored in ROM.
- The user-defined parts of the BIOS are stored in the CMOSRAM.
- The CPU then sends signals to check that all the hardware is working correctly
- This includes checking busses, system clock, RAM, disk drives and keyboard.
- If any of these devises contain their own BIOS then this is incorporated into the system's BIOS
- The PC is now ready to load the OS
- The boot program checks if a disk is present for drive A, if so it looks for an OS on that disk. If no OS is found, an error message is produced.
- If there was no disk in drive A, the boot program will check drive C.
- Once found the OS, if it is Windows, the boot program will look for MSDOS.SYS, and IO.SYS which holds the extensions to the ROM BIOS and contains a routine called SYSINIT which controls the rest of the boot procedure.
- SYSINIT then takes control and loads MSDOS.SYS which works with the BIOS to manage files and execute programs
- The OS searches the root directory for a boot file such as CONFIG.SYS which tells the OS how many files may be opened at the same time and instructions on how to load the necessary devise drivers.
- The OS tells MSDOS.SYS to load a file called COMMAND.COM, which is in three parts. The first part is a further I/O extension which joins with the BIOS to become part of the OS. The second part contains resident OS commands such as DIR and COPY.
- The files CONFIG.SYS and AUTOEXEC.BAT are created by the user so that the PC starts up in the same configuration each time it is switched on.

Keywords and Definitions

- **Operating System (OS)** – A piece of software that provides a platform on which the applications software can run. It controls the hardware, and provides communication with the outside world.
- **Interrupt** - messages sent to the processor by some external entity asking them to stop what they are doing and do another job
- **Scheduler** - a program, or set of instructions, used by the OS to decide where each of the jobs which are in its control should be and in what order to manipulate them
- **High-Level Scheduler (HLS)** – scheduler that places the jobs in the ready queue, and ensures that the system never gets overloaded.
- **Medium-Level Scheduler (MLS)** – Scheduler that swaps jobs between main memory and backing store.
- **Low-Level Scheduler (LLS)** - moves jobs in and out of the ready state, and decides which order to place them in the running state.
- **Pre-Emptive Scheduler** – scheduler that is allowed to move jobs out of the running state and into the ready queue. Has the power to move jobs.
- **Non-Pre-Emptive Scheduler** – scheduler that does not have the power to move jobs until they are finished.
- **First com first serve (FCFS)** – a priority method where the first job to enter the running queue is the first job to enter the running state. Favours long jobs.
- **Shortest job first (SJF)** – a priority method where jobs are sorted in the ready queue in ascending order of time needed
- **Round robin (RR)** – a priority method which gives each job a maximum length of processor time, it will then go to the back of the queue again. Once its finished, it leaves the queue.
- **Shortest remaining time (SRT)** – a priority method where the ready queue is sorted on the amount of expecting time still to do. Favours short jobs, and there's a danger long jobs may never be started.
- **Multi-level feedback queues (MFQ)** – a priority method which involves several queues of different priorities with jobs migrating across
- **Memory Management** - One of the key jobs of the OS, managing memory and allocating it accordingly to jobs.
- **The Linker** – a utility program included in the OS which links parts of the program together through the memory.
- **The Loader** – a utility program which loads jobs into the memory and adjusts addresses accordingly.
- **Pages** – equal sized sections in the memory, which are fixed and jobs are allocated a number of pages.
- **Segments** – similar to pages, although the size can be variable to match the job
- **Virtual Memory** - is part of the hard disk, but has faster speeds, it is slower than the RAM, but is usually considerably bigger. Acts in the same way as RAM

- **Disk Trashing –** caused by many jump instructions in the code, requiring main memory and virtual memory to be repeatedly switching. The hard disk will have to continuously search for pages.
- **Spooling -** It is a method used to place input and output on a fast access storage device, such as a disk, so that slow peripheral devices do not hold up the processor. It allows for queues when several jobs want to use peripheral devices at the same time. It stops different input and outputs becoming mixed up.
- **Command Driven OS –** an operating system without a GUI for example MSDOS
- **GUI OS** – An OS with a GUI, like most modern ones e.g. Windows 8
- **Multi-tasking OS** – most GUI OS's support multi-tasking, where multiple jobs can be carried out simultaneously.
- **File Allocation Table (FAT) -** a table that uses a linked lists to point to the blocks on the disk that contain files, the actual table is usually stored in the RAM.
- **Power on self-test (POST) –** a routine the resides in the permanent memory, it clears the registers, loads the address of the instruction into the program register. It then sends signals to the necessary hardware to check it's all there and working properly.
- **Basic Input Output System (BIOS) -** instructs the computer on how to perform a number of basic functions such as booting and keyboard control.
- **CMOS RAM** – stores the user-defined part of the BIOS. The BIOS is customisable.
- **Boot Program –** gets the system ready to accept an operating system. Unalterable and stored in the ROM.
- **Boot File** - contains some basic parameters to which the system will operate. It can be altered, and is stored in the CMOS RAM.

Key Points Posters



Interupts

Interupts are messages sent to the processor by some external entity asking them to stop what they are doing and do another job

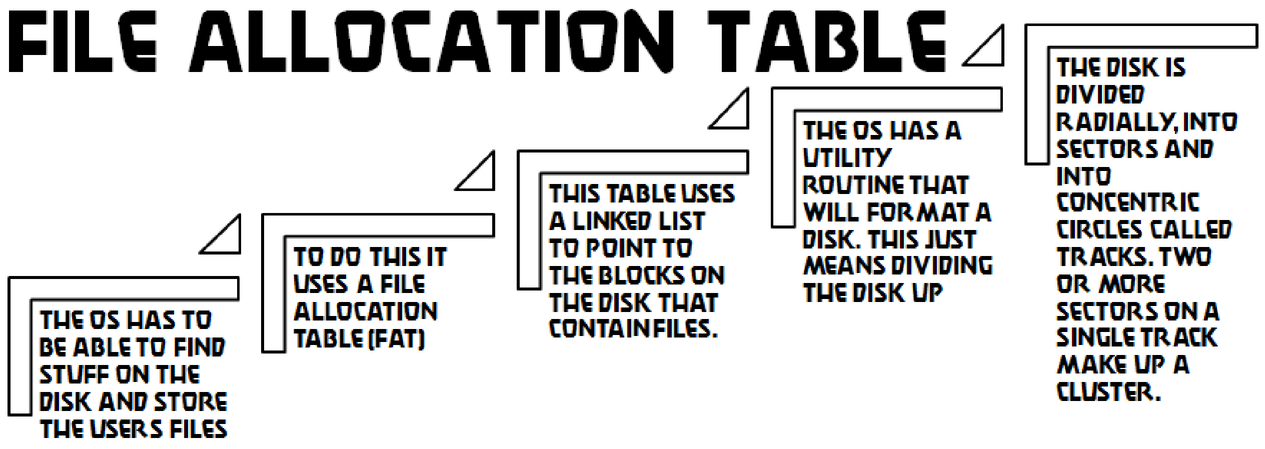
Types of interupt:

I/O interupt generated by I/O devise signaling something the processor should know. e.g. buffer needs refilling

Timer interupt generated by internal clock indicating that the processor must attend to time ctitical activities

Hardware error such as power failure - the computer must shut down as safley as possible

Program interupt generated by an error in a program



FILE ALLOCATION TABLE

THE OS HAS TO BE ABLE TO FIND STUFF ON THE DISK AND STORE THE USERS FILES

TO DO THIS IT USES A FILE ALLOCATION TABLE (FAT)

THIS TABLE USES A LINKED LIST TO POINT TO THE BLOCKS ON THE DISK THAT CONTAIN FILES.

THE OS HAS A UTILITY ROUTINE THAT WILL FORMAT A DISK. THIS JUST MEANS DIVIDING THE DISK UP

THE DISK IS DIVIDED RADIALLY, INTO SECTORS AND INTO CONCENTRIC CIRCLES CALLED TRACKS. TWO OR MORE SECTORS ON A SINGLE TRACK MAKE UP A CLUSTER.

SCHEDULING POLICIES

FCFS

- First Come First Served
- The first job to enter the ready queue is the first to enter the running state.
- This favours long jobs because once in the running state there is nothing to stop them carrying on running

SJF

- Shortest Job first
- Jobs are sorted in the ready queue in ascending order of times expected to be needed by each job. New jobs are added to the queue in such a way as to preserve this order

RR

- Round Robin
- gives each job a maximum length of processor time (called a time slice) after which the job is put at the back of the ready queue and the job at the front of the queue is given use of the processor. If a job is completed before the maximum time is up it leaves the system

SRT

- Shortest Remaning Time
- The ready que is sorted by the expected remaining time to complete a job. Long jobs may never get started

MFQ

- Multi level feedback ques
- Involves several queues of different priorities with jobs migrating downwards.

Spooling

What is Spooling?

- It is a method used to place input and output on a fast access storage device, such as a disk, so that slow peripheral devices do not hold up the processor.
- It allows for ques when severaljobs want to use peripherial devices at the same time
- It stops different input and outputs becoming mixed up

An example of spooling

- If two or more jobs are sent to a printer at the same time, the jobs are sent to a spool que where they wait for the printer to be free.
- The spooll que only stores reference to where the jobs are stored on a hard drive.

Benifits of spooling

- Keeps output of different ques seperate
- Saves the user having to wait for the processor
- Lets the processor get on with something else while the jobs are queued

Loading an OS

POST

- When the computer is powered on it only has access to the ROM, which is only big enough to hold a few instructions
- The computer then carries out the power on slef test (POST), which checks that everything needed for the computer to coe to life is availible.
- It also clears the registers of the CPU and loads the address of the first instruction in the boot program into the CPU

Boot Program

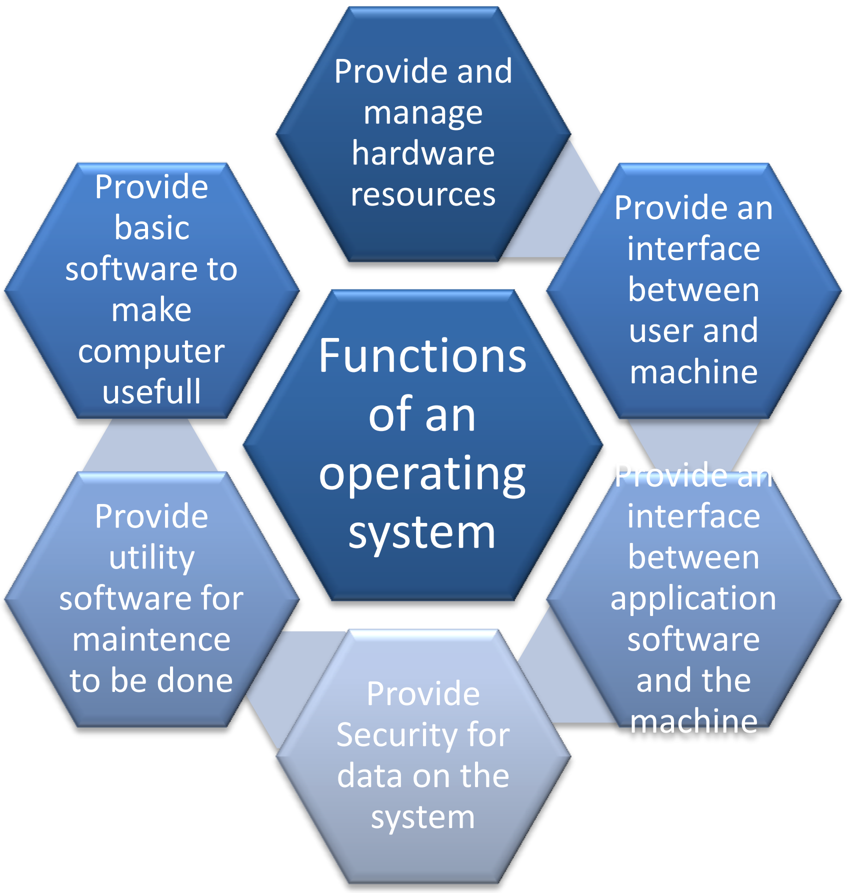
- The boot program is stored in the ROM, and contains the skeleton of the BIOS (only the main structure of the BIOS, because it contains user-defined options, which need to be modifiable, and ROM is read only) The user-defined parts of the BIOS would be stored in the CMOS RAM
- The CPU then sends signals to check all the hardware is working. Hardware containing it's own BIOS will be incorporated into the systems BIOS.

Ready to load the OS

- Once all the initial checks are done the PC is ready to load the OS. It will first check drive A, and if any bootable medium is present it will boot from it. If not it will check for drive C or return an error message to the user
- If the OS is Windows the boot program will look for IO.SYS and MSDOS.SYS, and load them.
- IO.SYS holds extensions to the ROM BIOS and contains a routine called SYSINIT which controls the rest of the boot procedure.
- SYSINIT now takes control and loads MSDOS.SYS which works with the BIOS to manage files and execute programs

Loading the OS

- The OS searches the root directory for a boot file such as CONFIG.SYS which tells the OS how many files may be opened at the same time., and may also contain instructions to load various device drivers.
- The OS tells MSDOS.SYS to load a file called COMMAND.COM which is in three parts. The first part is a further extension to the I/O functions and it joins the BIOS to become part of the OS. The second part contains resident OS commands, such as DIR and COPY
- The files CONFIG.SYS and AUTOEXEC.BAT are created by the user so that the PC starts up in the same configuration each time it is switched on.


Functions of an operating system
Provide and manage hardware resources
Provide an interface between user and machine
Provide an interface between application software and the machine
Provide Security for data on the system
Provide utility software for maintence to be done
Provide basic software to make computer usefull

Linkers and Loaders

The **Loader** is a small program that loads the jobs and adjusts the addresses into necessary places. It helps the OS with memory management

The **Linker** is the program that links all the different memory locations and parts of the program together.

Paging and Segmentation

Paging is when the memory is divided into equal sections, each section being a page. Jobs are then allocated a number of pages. The pages for each job may be in a logical order, or they may be scattered about where ever there is a free page.

Segmentation is similar to paging although the segments that the memory is split up into can be of a variable size, this makes better use of the memory, as less is wasted although is more complex and difficult to control and more complex to predict.

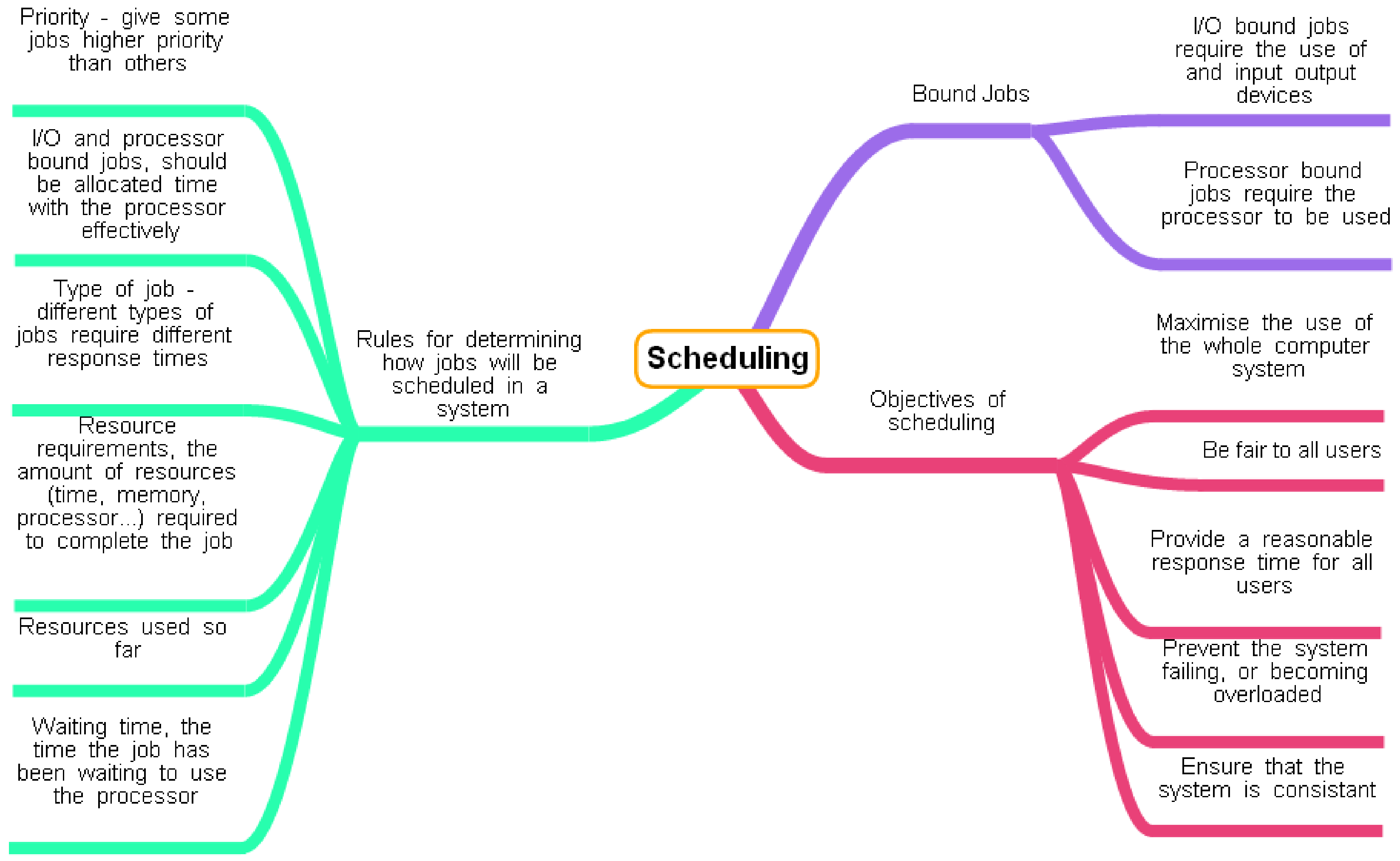
Virtual Memory

When a program is running, only the pages that contain the necessary data are needed to be stored in the RAM, as there is limited amount of space. The rest of the data that may be needed later on in the program or will be needed in a minute, (like a minimised window) will be stored in virtual memory.

Virtual Memory is part of the hard disk, but has faster speeds, it is slower than the RAM, but is usually considerably bigger.

If the memory is nearly full, more of the data will be stored in virtual memory, the OS will have to spend a long time both loading and saving pages.

Disk thrashing occurs when the OS has to spend a considerable proportion of its time swapping data between virtual and real memory.


Scheduling
Rules for determining how jobs will be scheduled in a system
Priority - give some jobs higher priority than others
I/O and processor bound jobs, should be allocated time with the processor effectively
Type of job - different types of jobs require different response times
Resource requirements, the amount of resources (time, memory, processor...) required to complete the job
Resources used so far
Waiting time, the time the job has been waiting to use the processor
Bound Jobs
I/O bound jobs require the use of and input output devices
Processor bound jobs require the processor to be used
Objectives of scheduling
Maximise the use of the whole computer system
Be fair to all users
Provide a reasonable response time for all users
Prevent the system failing, or becoming overloaded
Ensure that the system is consistant

Past Exam Questions with Mark Scheme Answers

Why interrupts are used in a computer system

to obtain processor time...
for a higher priority task
to avoid delays
to avoid loss of data
as an indicator to the processor…
that a device needs to be serviced

State some sources of interrupts

(imminent) power failure/system failure
peripheral eg printer (buffer empty)/hardware
clock interrupt
user interrupt eg new user log on request
software

Why is scheduling used?

maximise number of users…
…with no apparent delay
maximise number of jobs processed…
…as quickly as possible
obtain efficient use of processor time / resources…
…dependent upon priorities
…to ensure all jobs obtain processor time/long jobs do not monopolise the processor

Why are jobs given different priorities in a job queue?

some jobs are more urgent than others
priorities are used to maximise the use of the computer resources

Why is memory management necessary?

to allocate memory to allow separate processes to run at the same time
to deal with allocation when paging/segmentation
to reallocate memory when necessary
to protect processes/data from each other
to protect the operating system/provide security
to enable memory to be shared

Why is virtual memory needed?

to allow programs to run that need more memory than is
available

How is virtual memory used?

use of backing store as if it were main
memory/temporary storage
paging/fixed size units
swap pages between memory & backing store…
…to make space for pages needed

What's the problem of disk threshing?

occurs when using virtual memory/moving pages
between memory & disk
disk is relatively slow
high rate of disk access
more time spent transferring pages than on processing

Methods of scheduling

round robin
each user allocated a short period of time/in a sequence
or
system of priorities
highest priority first
or
length of job

shortest job first
or
first come, first served
jobs processed in order of arrival

What is spooling and when is it used?

output data to disk drive/storage device
for printing at another time
to allow sharing/on a network
job references stored in a queue/buffer
avoids delays / avoids speed mismatch
as printers are relatively slow
jobs can be prioritised

How are paging and segmentation similar?

ways of partitioning memory
allow programs to run despite insufficient memory/used for virtual memory
segments and pages are stored on backing store
segments and pages are assigned to memory when needed

How are paging and segmentation different?

segments are different sizes but pages are fixed size
segments are complete sections of programs, but pages are made to fit sections of memory
segments are logical divisions, pages are physical divisions

What problems may occur from paging and segmentation?

disk threshing
more time spent swapping pages than processing
computer may ‘hang’

When is the boot file used?

while the operating system is loading
when the computer is switched on
after POST

What is the purpose of the boot file

provides personal settings

What's virtual memory

use of backing store…
…as additional memory
uses paging / swapping pages (between memory & backing store)
holds part of the program not currently in use
allows large programs to run (when memory size is insufficient)

What's the purpose and use of the file allocation table?

a map of where files are stored…
…in backing store/hard disk
provides addresses/pointers to (start of) files
stores file names, and stores file sizes
stores access rights,
identifies free space
is updated by the operating system when files are saved/deleted
Is used by the operating system when files are accessed

Further Recourses

VRS(http://vrs.as93.net)

- Prezzi Presentation
- More revision posters
- More past exam questions
- Links for further reading
- Key words list
- Revision Quizzes
- This document
- User contributed content

Revision Quizzes (http://RevisionQuizzes.com)

- Jobs the OS must do quiz
- Loading an OS quiz
- Functions of the OS summary quiz

The Function and Purpose of Translators

What the Specification Says
Describe the need for, and use of, translators to convert source code to object code;
Understand the relationship between assembly language and machine code;
Describe the use of an assembler in producing machine code;
Describe the difference between interpretation and compilation;
Describe the purpose of intermediate code in a virtual machine;
Describe what happens during lexical analysis;
Describe what happens during syntax analysis, explaining how errors are handled;
Explain the code generation phase and understand the need for optimisation;
Describe the use of library routines

Notes

When computers were first invented, the only way to run programs on them, was to code them in binary. This is what the computer can understand. However it is very time consuming, with lots of repetition, resulting in inefficient programs with limited functionality and often full of errors.

Assembly Languages
A low level languages is a computer language which is very close to what the computer understands, but uses words rather than binary. Each binary instruction is given a word to represent it. Assembly language is a low level language. There are two key features to assembly language, it uses mnemonics and labels.

A mnemonic is a group of letters or keyword representing a particular operation. For example ADD could represent 01101000 which means add this number. Labels work in a similar way, they use a short word to represent the binary address, then store this information in a look-up table so it can be replaced when the program is run. Assembly language is translated by the assembler into machine code.

Assembler
This is the piece of software that translates assembly language into machine code. Machine code is all binary. The assembler must be machine specific, which means that a different assembler is needed for each different make of computer, as the machine code is also specific.

High Level Languages
A high level language is less like what the computer understands, and easier for the programmer. Each instruction gives rise to a series of machine instructions, so it is a one-to-many language. This means it has more functionality and it takes less code to compete each step in a program. Also high level languages are more portable between machines; it is not machine specific.

High level languages are written in source code and then is translated into object code. It contains keywords, which tell the computer what instruction to do and variables which store the addresses of data locations. There are two main methods of translating high level languages.

Interpreter
This is a translator which takes one line of source code, translates it, lets the computer run it, then takes the next line. This is very useful for finding errors, because when the program fails due to something like a logic error, the interpreter knows exactly where the error is. This makes the interpreter very useful for developing code.

Compiler
The compiler is a translator that takes source code and translates it into object code before allowing it to be run. They run more quickly that interpreted programs, as they don't have to be translated as they are being run.

Intermediate Code

Because each language has a different translator and every computer requires different machine code there would need to be a lot of additional software. A way round this would be far more efficient if the compiler or interpreter only translated halfway into intermediate code. A virtual machine will then translate if further into machine code.

Stages of Translation

When a high level language is translated with a compiler there are many stages, each done in parse with each other.

Lexical Analysis

Each of the keywords is looked up in a look up table and replaced with it's binary token. If the keyword is not recognised an error will be returned. It will then get rid of any superfluous characters like additional spaces, lines or tabs which made the code easier for the programmer to read. It will get rid of any comments which the programmer may also have added. Next it will recognise the variables and create a look up table for them called the symbol table containing the values for the variables being used, and the location.

Syntax Analysis

The compiler will use the keyword table to decide what to do with each instruction. It will compare what it gets with what it is expecting. It will return an error if it doesn't get what it's expecting.

Code Generation

The compiler takes each statement which is now just a string of binary, and converts it to low level/ intermediate code. It is a one-to-many process, as each high level instruction is translated into many low level ones. Code optimisation is then done, where the unnecessary instructions are removed.

Library Routines

Library routines are the pieces of code for carrying out a particular process which recurs many times throughout the running of a larger program. They are pre-written, pre-compiled and pre-tested. They are loaded into the memory with a utility program called the loader, and linked to the necessary parts of the code with a utility program called the linker.

Key Points

The use of the Translator

- Machine code is the very simple instructions written as a string of binary digits that the computer can understand
- Programs used to have to be written in machine code, which took a very long time, and made them prone to errors.
- When other languages were developed which were closer to English than machine code, there was a need for them to be translated into a form the computer could understand. This is what translators are for

Machine Code

- A form of language based on binary numbers, and using different combinations of digits to stand for different things.
- The codes are machine-specific, which means that they will only run on the type of machine they were written for.
- Each binary statement can be split in two, the first part represents the operation, and is called opp-code, the second part represents the data, or location of data to which the operation is to be applied to.

Assembly Languages

- Low-level language, because they are close the language used by the computer.
- Uses mnemonics, which are groups of letters or keywords that represent the opp-code part of the instruction.
- The references to the locations are also given alpha-numeric representation to make them easier to use and understand. These are called labels.
- One-to-one relationship with the machine code, meaning one assembly language instruction translates to one machine code instruction.

The Assembler

- The code written in assembly language is now impossible for the computer to understand. In order for it to be of any use, it must be translated to machine code, and the program used to translate assembly language to machine code is called the assembler.
- To convert the mnemonics to their binary tokens, the assembler has a look up table, which it searches, and then makes the replacement accordingly.
- The labels are done in a similar way, although the values are populated as the assembler goes.

High-Level Languages

- Closer to the language spoken by the person writing them, i.e. it's in English not binary
- Each instruction gives rise to a series of machine code instructions, meaning they are one-to-many languages.
- They are also more portable between machines
- There are two ways of translating a high-level language to machine code, using a compiler or an interpreter.

The Compiler

- Converts a program written in a high-level language into machine code.

- The high-level language is called the source code, and the machine code is called object code.
- Uses a lot of computer resources, because it has to be loaded into the memory at the same time as the source code, and have sufficient space to store the intermediate results.
- When an error occurs it is difficult to pin-point where it has occurred
- Converts code all at the same time, as a unit. The first instruction cannot be run until it is all converted

The Interpreter

- Takes one line of the source code translates it, lets the computer run it, then moves on to the next line, and so on through all the code.
- Very useful for finding errors, because it knows what line it got to when it failed. Often used for debugging code.
- This system was developed because early personal computers lacked the power and memory needed for compilation

Intermediate Code in a Virtual Machine

- Different designs of computer have different versions of machine code.
- This would mean that every computer would need a different compiler for each high-level language
- An alternative would be to use a compiler to do most of the translating and end up with a version of the program which is close to all the different machine codes. This is called intermediate code. It is halfway between high-level and machine code.
- It is not machine specific, but can be translated into particular machine code needed.

Stages of Translation

Translation of high-level is a one-to-many process, so it's quite complicated. As a result there are three main stages. Each stage is called a parse. The three stages are lexical analysis, syntax analysis and code generation.

Lexical Analysis

- The lexical analyser uses the source program as input and turns the high level language code into a stream of tokens for the next stage of the compilation
- Tokens are normally groups of 16-bits, and each group of characters in the code is replaced by a token.
- Single characters, which have a meaning in their own right, are replaced by their ASCII values.
- Variable names will need to have extra information stored about them. This is done by creating a symbol table. This table is used throughout compilation to build up information about names used in the program. Only their name is stored in this parse.
- The lexical analyser may output some error messages and diagnostics.
- The lexical analyser also removes redundant that the programmer may have added to make the code more understandable for example spaces, tabs, extra lines and comments
- Often the lexical analysis takes longer than the other stages of compilation. This is because it has to handle the original source code, which can have many formats.

Syntax Analysis

- The code generated in lexical analysis is checked to see if it is grammatically correct.
- Vague error messages can be given if something like a keyword is not recognised
- During syntax analysis certain semantic checks are carried out. These include label checks, flow of control checks and declaration checks.
- The syntax analyser verifies all variables and updates the symbol table with necessary information like type, size and scope.

Code Generation

- All the errors should have been removed by now, and the source code is just a string of binary digits that the compiler can understand.
- The addresses of the variables are calculated and stored in the symbol table.
- The intermediate code is then produced.
- When ready the compiler can produce machine code from this intermediate code by looking each binary token up in a look-up table.

Library Routines

- Many short pieces of code for carrying out a particular process recur many times in larger programs
- It would be a waste to go through rewriting and compiling them each time
- Library routines can be called whenever task is necessary to be done
- Pre-written, pre-compiled and pre-tested.
- Loaded into the memory by a utility program called the loader
- Linked to the relevant places in the existing code by a utility routine called the linker

High Level Languages

High level languages are computer languages that are closer to English and further away from what the computer understands.

They are one-to-many languages, as one high level instruction gives rise to a series of machine code instructions.

Source code is the code written by the programmer in the high level language. It is then translated, either with a compiler or interpreter into object code – that the computer can understand.

Linkers and Loaders

The **Loader** is a small program that loads the jobs and adjusts the addresses into necessary places. It helps the OS with memory management

The **Linker** is the program that links all the different memory locations and parts of the program together.

Compiler

- Takes the source code and translates it into object code
- The compiler, the source code and the object code must all be stored in the memory, there must also be additional memory for intermediate results
- When an error occurs it is hard to pin point where it is in the source code

Interpreter

- Each instruction is taken in turn and translated into machine code, it is then executed before the next instruction is translated.
- It was developed because early personal computers lacked the memory and resources needed for compilation.
- It can produce error messages as soon as the error is encountered. This makes it very useful for developers.
- Slower compared to a compiled program because the original program has to be translated every time it is executed. Instructions inside a loop have to be translated each time the loop is entered.

Assembly Language and Machine Code

Machine Code

- The language that the computer can understand, that uses different binary digits to stand for different instructions and locations.

Assembly Language

- a basic language which uses mnemonics to stand for instructions and labels to represent data locations. Assembly language has a one-to-one language with machine code. It is translated by the assembler.

Assembler

- the piece of software that translates assembly language to machine code. It converts the mnemonics to their binary representation through finding them in the look-up table. It builds a look-up table for the labels as data locations as it translates the program.

Stages of Translation

Lexical Analysis

Uses the source code as input and replaces the key words with their binary tokens from the look-up table.

A symbol table is created for the variables.

It will output error messages if it finds an error. For example if a keywork is not in the look up table, or a variable is undefined.

It will remove all additional lines, spaces tabs and comments.

Syntax Analysis

The code is checked to see if it is gramatically correct.

Further errors may be picked up, these won't be as accurate as the error logging in the lexical stage.

Semantic checks are carried out, including label checks, flow of control checks and Declaration checks.

Must ensure that certain control constructs are used in the right places.

Code Generation

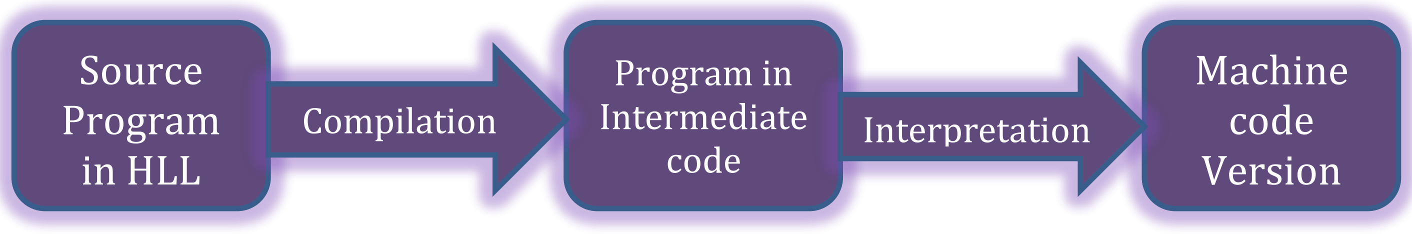
By now, all errors have been removed or reported and the code is all binary.

Complier takes each statement and translates it into low level/ intermediate code. One-to-many process

Optimisation is when the compiler gets rid of any lines which are not strictly necissary, this makes the program shorter so it takes up less memory and funs faster.

INTERMEDIATE CODE IN A VIRTUAL MACHINE

Different designs of computer have different versions of machine code, so different instructions mean different things. This would mean every computer would need a different compiler for each high level language.
Alternatively a compiler could be used to do most the translation, and end up with a program which is close to all the machine codes. This is the intermediate code.
Intermediate code is halfway between the high level language and machine code. It is not machine specific but it is now translated into the particular machine code needed by an interpreter specific to that machine.
This means the program written in a high level language will be able to run on different machines, and is portable.



Key Words

- **Translator** – piece of software that converts one form of code to another form more understandable by the computer. Three type, assembler, interpreter and compiler
- **Assembly Language** – basic low-level language with a one-to-one relationship with machine code, developed in the late 1940's. Uses mnemonics and labels.
- **Mnemonics** - keywords or groups of letters representing basic operations.
- **Labels -** are alpha-numeric representations of data locations.
- **Assembler** – piece of software that translates assembly language to machine code.
- **Machine Code** – the binary code that the computer can understand. Machine-specific, meaning that different computers need different machine code.
- **High-Level Language** – languages closer to English. One-to-many language, meaning each high-level instruction gives rise to a series of machine code instructions. More portable between machines.
- **Source Code** – the high-level code written by a programmer
- **Object Code** – after the source code has been translated, it becomes object code.
- **Keyword** – special word used in high-level languages that is associated with a statement that has its own syntax.
- **Interpreter** – translator program that translates one line of code at a time. Especially useful for debugging and testing as can return accurate error message. Was developed because it uses less computer resources than compiling, but slower.
- **Compiler** – translator program that translates the whole program as a unit. Quicker, but requires a lot of memory, and error diagnosis are vague.
- **Intermediate Code** – half translated language, that is not machine-specific but can be translated the rest of the way.
- **Virtual Machine** – this is the piece of software required to run intermediate code.
- **Parse** – a look through, or stage of translating a program.
- **Lexical Analysis** – the first stage of translation, where each keyword is replaced with its binary token, that's been found in the look-up table. Variables are added to the symbol table, and all superfluous characters are removed.
- **Syntax Analysis** – uses the keyword table to decide what the instructions for that particular keyword is and what rules to apply.
- **Code Generation** – the final stage of translation, where the code is actually generated/ converted to machine code.
- **Optimisation** – this is done during code generation, just removes the unnecessary parts.
- **Library Routines** – pre-written, pre-tested and pre-compiled sub-routines
- **Loader** – utility program that loads library routines into the memory
- **Linker** – utility program that links library routines to the relevant places

Past Exam Questions and Answers

These are questions that have appeared in past papers relating to the function and purpose of translators, and the mark scheme answers.

Describe assembly language

a language related closely to the computer being programmed/low level language/machine specific
uses descriptive names (for data stores)
uses mnemonics (for instructions)
uses labels to allow selection
each instruction is generally translated into one machine code instruction
may use macros

Describe machine code

binary notation
set of all instructions available
to the architecture/which depend on the hardware design of the processor
instructions operate on bytes of data

What tasks are performed by the assembler when producing machine code?

reserves storage for instructions and data
replaces mnemonic opcodes by machine codes
replaces symbolic addresses by numeric addresses
creates symbol table to match labels to addresses
checks syntax/offers diagnostics for errors

What are the features of the interpreter?

translates one line/statement…
…hen allows it to be run before translation of next line
reports one error at a time…
…nd stops

What are the features of a compiler?

translates the whole program as a unit
creates an executable program/intermediate program
may report a number of errors at the same time
optimisation

Describe lexical analysis

source program is used as the input
tokens are created from individual symbols and from…
…he reserved words in the program
a token is a fixed length string of binary digits
variable names are loaded into a look-up table / symbol table
redundant characters (eg spaces) are removed
comments are removed
error diagnostics are given
prepares code for syntax analysis

What is the purpose of a translator?

convert from source code…
…to object code
detect errors in source code

Why may intermediate code may be more useful than executable code?

can run on a variety of computers
same intermediate code can be obtained from
different high level languages
improves portability

What additional software is needed to run intermediate code?

interpreter / virtual machine

What is a disadvantage of using intermediate code?

the program runs more slowly/has to be translated each
time it is run / need additional software

What does code optimisation do?

makes code as efficient as possible
increases processing speed
reduces number of instructions

Describe syntax analysis

accepts output from lexical analysis
statements/arithmetic expressions/tokens are checked...
...against the rules of the language/valid example given eg matching brackets
errors are reported as a list (at the end of compilation)
diagnostics may be given
(if no errors) code is passed to code generation
further detail is added to the symbol table…
…eg data type /scope/address

What's intermediate code, and it's use?

simplified code / partly translated code…
…which can be run on any computer/virtual machine/improves portability…
…using an interpreter
sections of program can be written in different languages
runs more slowly than executable code

What software converts source code into object code?

translator

What is source code?

the original code/code written by the programmer…
…often in a high level language
may be in assembly language
source code can be understood by people…
…but cannot be executed (until translated)

Why might library routines help programmers, and when are they used

Library routines:
routines are pieces of software…
…which perform common tasks…
…such as sorting/searching
routines are compiled
Why library routines help programmers:
routines are error-free/have already been tested
already available/ready to use/saves work/saves time
routines may be used multiple times
routines may have been written in a different source language
allows programmer to use others' expertise
How routines are used:
linker is used…
…to link routine with program
loader handles addresses…
…when program is to be run

Further Resources

Resources on the VRS (http://vrs.as93.net)

- This hand out
- The presentation that goes with it
- More revision posters
- More past exam questions
- User contributed documents

Quizzes (http://revisionquizzes.com)

- High-Level Languages
- Low-Level Languages
- Library Routines
- Stages of Translation
- Translators summary

Computer Architectures

What the specification says

describe classic Von Neumann architecture, identifying the need for, and the uses of, special registers in the functioning of a processor;
describe, in simple terms, the fetch/decode/execute cycle, and the effects of the stages of he cycle on specific registers;
discuss co-processor, parallel processor and array processor systems, their uses, advantages and disadvantages;
describe and distinguish between Reduced Instruction Set Computer (RISC) and Complex instruction Set Computer (CISC) architectures.

Von Neumann Architectures

John Von Neumann realised that the programs and their data were indistinguishable, and therefore can be stored in the same memory. He created a new architecture that contained a single processor which follows a linear sequence of the fetch-decode-execute cycle.

In order to do this it has a few special registers. A register is just an area to store data. The individual locations in the memory are registers, but as they have no special purpose they are not special registers. There are five special registers that are used to control the fetch-decode-execute cycle. They are outlined as below:

- PC - Program Counter
- CIR - Current Instruction Register
- MAR - Memory Address Register
- MDR - Memory Data Register
- Accumulator

The specification requires you to know the fetch-decode-execute cycle and how it links in with the special registers in some detail

Fetch

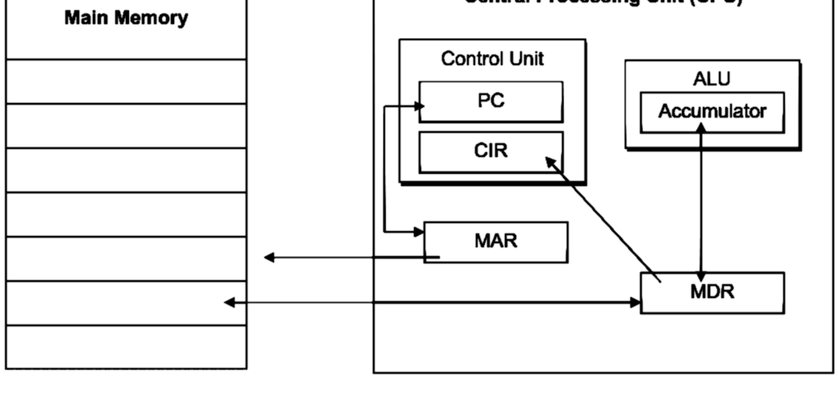
1. The PC stores the address of the next instruction which needs to be carried out
   As instructions are held sequentially in the memory, the value in the PC is incremented so that it always points to the next instruction.
2. When the next instruction is needed, it's address is copped from the PC and placed in the MAR
3. The data which is stored at the address in the MAR is then copied to the MDR
4. Once it is ready to be executed, the executable part if the instruction is copied into the CIR

Decode

1. The instruction in the CIR can now be split into two parts, the address and the operation
2. The address part can be placed in the MDR and the data fetched and put in the MAR.

Execute

1. The contents of both the memory address register and the memory data register are sent together to the central processor. The central processor contains all the parts that do the calculations, the main part being the CU (control unit) and the ALU (arithmetic logic unit), there are more parts to the central processor which have specific purposes as well.
2. The ALU will keep referring back to where the data and instructions are stored, while it is executing them, the MDR acts like a buffer, storing the data until it is needed
3. The CU will then follow the instructions, which will tell it where to fetch the data from, it will read the data and send the necessary signals to other parts of the computer.



Fetch-Decode-Execute Cycle

All instructions on the computer for nearly all types of architecture follow the fetch-decode-execute cycle. It is a simple principle developed in the 1940's. The specification requires you to know it in some detail, and know how each stage makes use of the special registers. It is outlined below:

1. Load the address that is in the program counter (PC) into the memory address register (MAR).
2. Increment the PC by 1.
3. Load the instruction that is in the memory address given by the MAR into the MDR
4. Load the instruction that is now in the MDR into the current instruction register (CIR).
5. Decode the instruction that is in the CIR.
6. If the instruction is a jump instruction then
    a. Load the address part of the instruction into the PC
    b. Reset by going to step 1.
7. Execute the instruction.
8. Reset by going to step 1.

Reduced Instruction Set and Complex Instruction Set Computers

When you used binary to represent instructions, there were two parts to each byte, the instruction and the data. For example 00110100, if the firs 3 bits represented the instruction, like ADD, SUB, DIV, MOD… or whatever, then the last 5 bits would represent the data address. There would only be nine possible instructions available though. This would be a very simple set, the more bits allowed for the operation, the more operations available and therefore the more complex the set becomes.

A complex instruction set computer (CISC) is designed to have operations and operation code for all things they will need to do. It needs to have slightly more complex physical architecture to allow for this wide range of possible instructions. It is easier to program, and requires less code. Because the instructions are already in existence, there is no need to store intermediate results, this uses less RAM. However the processor may need to use more cycles per instruction because the instructions are more complex.

Some processors however are designed to reduce the number of operations which saves space. These are called restricted (or reduced) instruction set computers (RISC). Just because they have fewer instructions doesn't mean they can do fewer things, it just means that the programmer has to be a bit cleverer. The physical architecture is therefore simpler because there are only a few assembly instructions which can be used. Also because each job will be made up of quite a lot of basic instructions, it is necessary to store intermediate results, which is half processed data, as a result more RAM will be required. Because each instruction is simple, it only takes one cycle by the processor to complete the instruction. Recently this has become a more preferred method, as overall it is more efficient, because the only two draw backs are using more RAM, which doesn't matter as it is getting increasingly cheaper, and that it is harder to program, but more tools are being developed which makes the programs easier to write.

Parallel Processing Architectures

The Von Neumann architecture is an example of serial processing, where one instruction is fetched, decoded and executed and then the next is fetched, decoded and executed and so on. Parallel processing however allows many instructions to be carried out at the same time. There are a number of different ways of doing this, depending on the use.

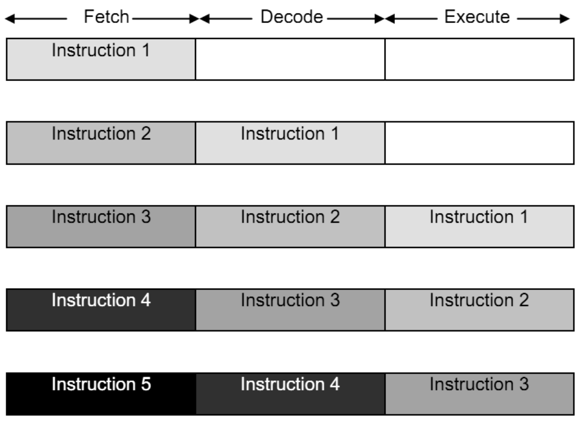
Array or Vector processing
This is a simple processor used for data that can be processed independently of one another – that means that a single instruction can be applied to multiple *bits* of data all at the same time, and none of the results of that data will be needed to process the next bit of data.

These types of processors are normally used for things like controlling input or output devices. They can be used to track the point of the mouse on a screen, or display the data on the monitor.

They are sometimes called array processors, because the data is stored in an array, similar to that used in programing, the array can have one to many dimensions. Array processors are an extension to the CPU's arithmetic unit. The only disadvantage to array processing is that it relies on the data sets all acting on the same instruction, and it is impossible to use the results of one data set to process the next, although this does not matter in their use.

It is also important to know, that an array processor is a single instruction multiple data (SIMD) processor, it should be clear what this means, lots of bits of data get processed with a single instruction.

Pipelining
Pipelining is another method of parallel processing. There are several processors each one does a different part of the fetch decode execute cycle, so the fetch-decode-execute cycle is staggered. This can be best illustrated with the diagram on the right.



As long as the pipelines can be kept full, it is making best use of the CPU. This is an example of single instruction single data (SISD) processor, again it should be quite clear why, the processor is processing a single instruction to a single bit of data.

Multi-core processors
This is where there a number of CPUs being applied to a job, with each part carrying out different parts. Each CPU is likely to be in effect a serial processor, although as there are many processing multiple instructions to multiple data it becomes a parallel processor when viewed as a whole.

These are likely to be used on a large scale in supercomputers, but also many personal computers have multiple cores. The limitation of multi-core processors would be that they are dependent on being able to cut jobs down into chunks, this can make it harder for the programmer to write code for them.

Processor Examples

This does not come into the specification, although it puts the above information into context and makes it easier to understand, it's also very interesting.

Vector processing
The Cray supercomputers all work with vector processors to an extreme, although they are much more expensive than alternative processing methods, and have limitations of what type of data they can process. Array processing is also used on a smaller scale for some I/O devices and games graphics.

Difference between code for serial processors and array/ vector processors

Serial Processor

```
execute this loop 10 times
  read the next instruction and decode it
  fetch this number
  fetch that number
  add them
  put the result here
end loop
```

Array/ Vector processor

```
read instruction and decode it
fetch these 10 numbers
fetch those 10 numbers
add them
put the results here
```

Multi-core supercomputer - Blue Gene
Blue Gene is one of the most powerful computers in the world, IBM started developing it in 1999, costing $100 million in research, and taking 5 years to complete, it became the world's most powerful supercomputer in 2004 (it is now 5th ranked by processing power). It is mainly used by universities and government research departments. Blue Gene is a good example of a multi-core computer to an extreme; it has 1496 cores (most home computers have about 2 cores). Supercomputers work basically the same was as normal multi-core processors, just to a much much larger scale, allowing for very big and accurate calculations to be mad quickly.

Top most powerful multi-core processor computers in the world since 1993

- Fujitsu K computer (Japan, June 2011 – present)
- NUDT Tianhe-1A (China, November 2010 - June 2011)
- Cray Jaguar (United States, November 2009 - November 2010)
- IBM Roadrunner (United States, June 2008 – November 2009)
- IBM Blue Gene/L (United States, November 2004 – June 2008)
- NEC Earth Simulator (Japan, June 2002 – November 2004)
- IBM ASCI White (United States, November 2000 – June 2002)
- Intel ASCI Red (United States, June 1997 – November 2000)
- Hitachi CP-PACS (Japan, November 1996 – June 1997)
- Hitachi SR2201 (Japan, June 1996 – November 1996)
- Fujitsu Numerical Wind Tunnel (Japan, November 1994 – June 1996)
- Intel Paragon XP/S140 (United States, June 1994 – November 1994)
- Fujitsu Numerical Wind Tunnel (Japan, November 1993 – June 1994)
- TMC CM-5 (United States, June 1993 – November 1993)

Von Neumann Architecture

- The most commonly used computer architecture, is serial, and so only processes one job at a time with one set of data
- All the instructions and the data in Von Neumann architecture are stored together in the same memory.
- In order to do this it makes use of the special registers, which are described on the next page.

Advantages	Disadvantages
▪ Nearly all types of data can be processed with the VN architecture ▪ Data that relies on the result on the previous operation is fine Cheaper than alternative methods of processing	▪ Can be slower than alternative methods ▪ Can be limited by the bus transfer rate ▪ Doesn't always make maximum use of the CPU ▪ Poorly written programs can get their data mixed up, because both programs and data share the same memory

PROCESSOR COMPONENTS

PC → MAR → MDR → CIR → ACC

PROGRAM COUNTER (PC)

- KEEPS CHECK OF WHEREABOUTS THE NEXT PROGRAM IS IN THE MEMORY.
- AFFTER ONE INSTRUCTION HAS BEEN CARRIED OUT, THE PC WILL BE ABBLE TO TELL THE PROCESSOR WHERE ABOUTS THE NEXT INSTRUCTION IS.
- THE INSTRUCTIOIONS ARE ALWAYS STORED IN ORDER IN THE PC.

MEMORY ADDRESS REGISTER (MAR)

- THIS IS WHERE THE ADDRESS THAT WAS READ FROM THE PC IS SENT.
- STORED HERE SO THAT THE PROCESSOR KNOWS WHERE ABOUTS IN THE MEMORY THE INSTRUCTION IS.

MEMORY DATA REGISTER (MDR)

- THE MEMORY IS SEARCHED TO FIND THE ADDRESS BEING HELD IN THE MAR, AND WHAT EVER IS UNDER THAT ADDRESS, MUST BE THE INSTRUCTION.
- THE INSTRUCTION IS THEN COPPIED, INTO THE MDR.

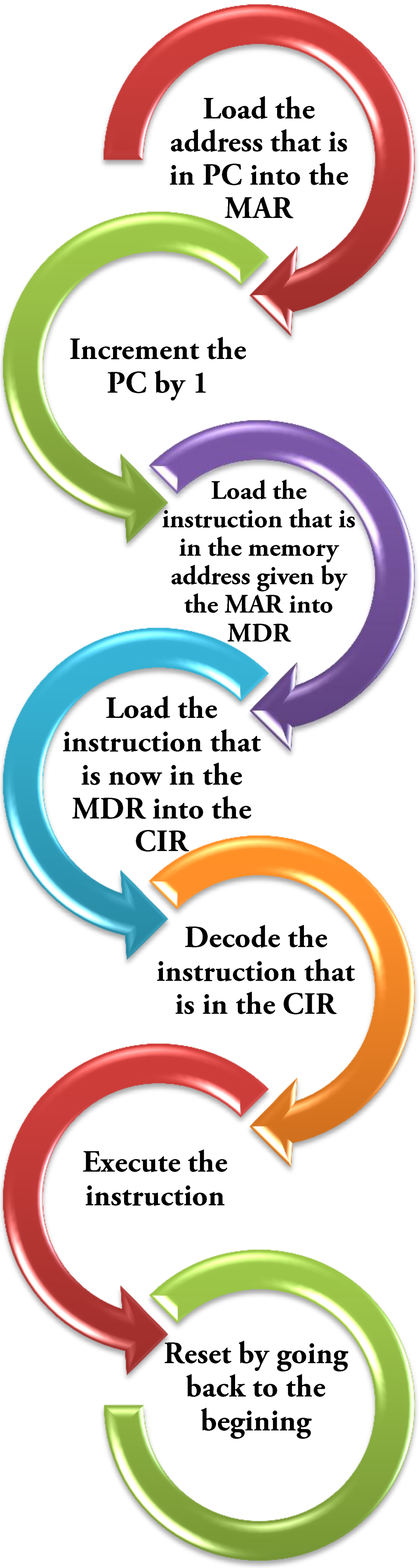
CURRENT INSTRUCTION REGISTER (CIR)

- THE INSTRUCTION THAT IS NOW IN THE MDR IS COPPIED INTO THE CIR.
- IT WILL BE SPLIT INTO TWO PARTS:
  - ONE PART WILL BE SENT TO THE COMPUTER TO BE DECODED SO THAT THE PROCESSOR KNOWS WHAT SORT OF INSTRUCTION IT IS, AND CAN SEND SIGNALS TO THE RELEVANT PARTS.
  - THE OTHER ART TELLS THE PROCESSOR WHERE ABOUTS IN THE MEMORY THE DATA THAT NEEDS TO BE USED IS.

THE ACCUMALATOR (ACC)

- THE ACCUMALATOR IS USED TO ACCUMALATE RESULTS, IT IS WHERE THE RESULTS FROM OTHER OPERSAIONS ARE STORED TEMPORARILY BEFORE BEING USED BY OTHER PROCESS'S.

The Fetch Decode Execute Cycle


Load the address that is in PC into the MAR
Increment the PC by 1
Load the instruction that is in the memory address given by the MAR into MDR
Load the instruction that is now in the MDR into the CIR
Decode the instruction that is in the CIR
Execute the instruction
Reset by going back to the begining

Pipelining

- Multiple CPUs carrying out a different part of the fetch-decode-execute cycle.
- As long as it is kept full, makes maximum use of CPU
- Single Instruction Single Data Computer (SISD)

Array Processing

- Where data is arranged into arrays and all acted upon by the same instruction
- Relies on the fact that all the sets of data are being acted on by the same instruction
- Single Instruction Multiple Data Computer (SIMD)

Multi-core processing

- Where several serial processors operate simultaniously on different parts of a job
- Dependant on being able to cut problems down into chunks
- Multiple Instruction Multiple Data Computer (MIMD)

SERIAL AS OPPOSED TO PARALLEL

Advantages

- Nearly all programs can run on with serial processors, there is no additional complex code to be written.
- All types of data are suitable for serial processing
- Data can use the previous result in the next operation
- Data sets are independant of each other
- Cheaper than parralel

Disadv

- The Von-Neumann bottle neck can slow data transfer in VN architectures.
- Both the program and data share the same memory, it is possible that trashing can occur with poorley written programs

PARALLEL AS OPPOSED TO SERIAL

Advantages

- Faster when handling large amounts of data
- Not limited by bus transfer rate
- Can make maximum use of the CPU
- Programs and data are stored seperatley usually

Disadv

- Only certain types of data are suitable for parralel processing
- Data that relies on the result of the previous operation can not be made parralel
- Each data set must be inderpendant of each other
- Usually more expensive

Past Exam Questions

Below are all the exam questions relating to computer architectures since the GCE computing from 2008 specification was released. The mark scheme answers and explanations are below.

Question 1

(a) Describe the effects of the fetch-execute cycle on the program counter (PC) and the memory address register (MAR).

[5]

(b) (i) State three features of a Complex Instruction Set Computer (CISC) architecture.

1

2

3

[3]

(ii) Explain one disadvantage, other than cost, of a CISC architecture compared with a Reduced Instruction Set Computer (RISC) architecture.

[2]

Question 2

In classic Von Neumann architecture, a number of registers are used.

(a) (i) Explain the term register.

[2]

(a)(ii) Give the correct names for two of the special registers used. (Do not use abbreviations.)

1

2

[2]

(b) Explain the advantages and disadvantages of parallel processor architecture compared with Von Neumann architecture.

[5]

Question 3

(a) State the three stages, in order, of the machine cycle in classic Von Neumann architecture.

[2]

(b) Two computer architectures are Reduced Instruction Set Computer (RISC) and Complex Instruction Set Computer (CISC) architectures.

(b)(i) Complete the table to show how the statements apply to these architectures.

	RISC only (✓)	CISC only (✓)	both RISC and CISC (✓)
Has many addressing modes			
Many instructions are available			
Uses one or more register sets			
Uses only simple instructions			

[4]

(b)(ii) Compare the number of machine cycles used by RISC and CISC to complete a single instruction.

[2]

(c) An array processor is used in some systems.

(c)(i) Explain the term array processor.

[2]

(c)(ii) Give one example of the type of task for which an array processor is most suitable.

[1]

Past Exam Question Answers

Question 1

(a)

- PC holds address of next instruction
- PC passes this address to MAR…
- MAR holds address of instruction/data
- Instruction/data from address in MAR is loaded to MDR
- PC is incremented (in each cycle)
- PC is changed when there is a jump instruction…
- …by taking address from instruction in CIR

(b)(i)

- uses (complex) instructions each of which may take
- multiple cycles
- single register set
- instructions have variable format
- many instructions are available
- many addressing modes are available

(b)(ii)

- programs run more slowly…
- …due to the more complicated instructions/circuit

Question 2

(a)(i)

- a location in the processor
- used for a particular purpose
- (temporarily) stores data/or control information
- explained example of contents held by named register

(a)(ii)

- program counter
- memory address register
- memory data register/memory buffer register
- current instruction register
- index register
- interrupt register
- accumulator

(b)

advantages:

- allows faster processing
- more than one instruction (of a program) is processed at the same time
- different processors can handle different tasks/parts of same job

disadvantages:

- operating system is more complex...
- ...to ensure synchronisation
- program has to be written in a suitable format
- Program is more difficult to test/write/debug

Question 3

(a)

- fetch, decode, execute
- correct order

(b)(i)

	RISC only (✓)	CISC only (✓)	both RISC and CISC (✓)
Has many addressing modes		✓	
Many instructions are available		✓	
Uses one or more register sets			✓
Uses only simple instructions	✓		

(b)(ii)

- RISC: each task may take many cycles
- CISC: a task may be completed in a single cycle
- ...as instructions may be more complex than individual instructions in RIS

(c)(i)

- a processor that allows the same instruction to operate
- simultaneously...
- ...on multiple data locations
- the same calculation on different data is very fast
- Single Instruction Multiple Data (SIMD)

(c)(ii)
(accept any example of a mathematical problem involving large number of similar calculations)

- eg weather forecasting / airflow simulation around new aircraft

Further Resources

Resources on the VRS (http://vrs.virtutools.com)

- This hand out
- The presentation that goes with it
- More revision posters
- More past exam questions
- User contributed documents

Quizzes (http://revisionquizzes.com)

- Serial Processors
- Parallel Processors
- Co-processors
- CISC and RISC
- Computer Architectures summary

Class resources (email me for copy)

- Team quiz
- Videos
- Presentation
- Further notes on computer architectures

Information Sources

The following sources were used in writing the above:

- AS/A2 Computing textbook
- Harvard website
- Wikipedia
- The sites mentioned in the links section

None of the information in this hand-out, the presentation, the quizzes or any on the revision notes was copied directly from another source without being noted.

Links for further reading

How Von Neumann Architecture works

http://bugclub.org/beginners/history/VonNeumann.html

Video presentation on busses in Von-Neumann

http://www.youtube.com/watch?v=eWhnMNjxYDQ

Interesting article about John Von-Neumann

http://www.maa.org/devlin/devlin_12_03.html

Simple Von-Neumann Vs Harvard architecture

http://www.pictutorials.com/Harvard_vs_Von_Nuemann_Architecture.htm

Data Representation

What the Specification Says:
Demonstrate an understanding of floating point representation of a real binary number;
Normalise a real binary number;
Discuss the trade-off between accuracy and range when representing numbers.

Number Systems

Introduction
When we count we usually use the denary number system, which is base 10. We count from 0 to 9, and then put a 1 in front and start again. Computers find it easier to work with binary, which is base 2, because it is only 0s and 1s, which can represent on or off, yes or no, or true or false,

Converting from Denary to Binary
To turn a denary number into a binary number, just put 1s in each column which is needed to make the number, using the column headings below. We usually work with 8 bits at a time, which is about 1 byte, so if the number is smaller; we need to put in leading zeros.

128	64	32	16	8	4	2	1

For example: 75_{10} in binary would be equal to 01001011_2 in binary, because 75 is the same as no lots of 128, one lot of 64, no lots of 32 or 16, one lot of 8, no lots of 4, and one lot of 2 and 1.

Converting from Denary to Octal
Octal is base 8, the principle for converting from denary to octal is similar as to that of denary to binary, although the column headings are obviously different, and you can have up to 8 lots of a number in each column.

512	64	8	1

For example 75_{10} in denary would be no lots of 512, one lot of 64, one lot of 8 and three lots of 1, so it would be 0113_8 in octal.

Converting from Denary to Hexadecimal
Some information is stored in computers as numbers in base 16, or hexadecimal sometimes just called hex. The principles are exactly the same for binary, octal, denary or any other base. But if you have to be able to count from 0 to 15, we would have to have 16 digits, as we only have ten (0 to 9) we have to use letters after that.

256	16	1

Hex Letter	Denary Representative
A	10
B	11
C	12
D	13
E	14
F	15

For example; 75_{10} in denary will be four lots of 16 and eleven lots of 1, so it is equal to $4B_{16}$ in hex.

Binary Coded Decimal
Some numbers look like numbers, but don't behave like numbers, for example you can not add together barcodes. Binary Coded Decimal (BCD) just represents four different digits in the number separately, using four binary digits for each denary digit.

8	4	2	1

For example; 7_{10} would be equal to 0111_2 because it is no lots of 8 and one lot of 4, 2 and 1. 5_{10} would be equal to 0101_2. So 75 in BCD would be 01110101 (just put the two together).

+/-	64	32	16	8	4	2	1

Negative Binary - sign magnitude

In sign magnitude method of storing negative binary numbers, the first bit, where the 128 was becomes a +/- bit, if there is a 1 here it is negative, whereas a 0 indicates that it is a positive binary number.

For example -75_{10} would be = to 11001011_2.

There are two problems with this method, firstly the biggest number that can be stored in one byte is half what it was, and secondly it is now storing two different types of data, a sign and a value, this makes it harder to do arithmetic operations.

Negative Binary - Twos complement

This gets round the problems with the sign magnitude method, in 2s complement the first bit of the bye becomes -128. 2s complement is very useful for addition and subtraction, because the negative part will do its self, and all we need to do is add it.

For example: -75_{10} would start with a 1, because it is a minus, we then work upwards adding more digits

-128	64	32	16	8	4	2	1

to get it up from -128 to -75. We need 53 to get up to -75 which is equal to 1 lot of 32, 16, 4 and 1. So -75 denary is equal to 10110101 in 2s complement binary.

Addition of Binary numbers

Addition in binary is much more simple than addition in denary, so computers can do it much faster, there are only four possible sums in binary:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0, carry 1

For example 75 + 14 in denary:

	-128	**64**	**32**	**16**	**8**	**4**	**2**	**1**	
75 =	0	1	0	0	1	0	1	1	
14 =	0	0	0	0	1	1	1	0	
	0	1	0	1	1	0	0	1	**= 89**
Carry				1	1	1			

Subtraction of Binary numbers

We use 2s complement when doing subtraction of binary numbers, (because 75 – 14 is the same as 75 + (-14)). When we do subtraction of binary nun umbers, we never do any subtraction; it's all addition

For example, 75 – 14

	-128	**64**	**32**	**16**	**8**	**4**	**2**	**1**	
75 =	0	1	0	0	1	0	1	1	
14 =	1	1	1	1	0	0	1	0	
	0	0	1	1	1	1	0	1	**= 61**
Carry	1					1			

Floating Point Binary

This text presumes we already know how to convert to and from binary numbers, and can represent both negative and positive.

Floating point binary is used to represent fractions.

The model for a whole number byte: 128 64 32 16 8 4 2 1
Where each sub heading will have either a one or a zero under it depending on what the value of the number.

Effectively there is a decimal point at the end of this number. (e.g. 139 = 139. = 139.0)

Also each heading it the heading before divided by two.

If we had a two byte representation of a binary number, with the second byte being the decimal point it would look like:
128 64 32 16 8 4 2 1 || 0.5 0.25 0.125 $\rightarrow$
And so on. (the arrow is because there wasn't room on the page to carry on, and it's kind of obvious what's going to come next).

However a major restriction of this is that the decimal point can't be moved.

So instead we put the decimal point at the beginning of the binary byte and add put the point to the power of something. Like in base 10, we represent big numbers by doing for example $1.64 \times 10^{9} =$ 1,640,000,000 and $3.29 \times 10^{-4} = 0.000329$.

In binary we can have for the value of 2.75, $10.11 = .1011 \times 2^{10} = .001011 \times 2^{100}$.
The point is moving.

The computer cannot cope with the point being in a different place from number to the next, because it has no way of representing the point.

As a result the decimal point is usually put before the first 1. There are however other options, but everyone's got to use the same, in order for all computers to understand it.

So 2.75 which is equal to 10.11 would be represented as $.1011 \times 2^{10}$. The x2 just shows how many decimal places it should be moved by in order to give the original number.

When the value is written like this, it is normalised, which means expressed in the same form as all the others.

With the number, 0.1011×2^{10}, there are two parts. The 0.1011 is called the mantissa. The second part, after the x2 is 10 and is called the exponent.

So..	Mantissa	||	Exponent
	01011	||	010

Trade of between Accuracy and Range
If you increase the number of bits for the exponent (and subsequently decrease the number available for the mantissa), then the accuracy is decreased and the range (size of numbers that can be stored) is increased. This is important to remember.

Key Words

- **Mantissa** – The part of the representation that contains the actual values. i.e. the first part.
- **Exponent** – The part of the representation that shows how many places the decimal point has to be shifted in order to return the original number.
- **Normalised** – When the number is written in standard form.

Example

A number is represented as an 8 bit floating point number. 5 bits are used as the mantissa and 3 bits are used as the exponent. Both the mantissa and the exponent are in twos complement.
The floating point number is 01100010

a) Show the floating point in its denary equivalent
b) What is the number in decimal?

Step 1. Break the floating point number into its two parts and place a decimal point in the mantissa between the most significant bit and the next one:
Mantissa is 0.1100 because the question states it uses the first 5 bits.
Exponent is 010
Both are positive binary numbers because they have a leading zero. So the mantissa in denary form is 1100 -> 12 denary
The exponent is 010 -> 2 denary

Step 2. Slide the decimal point of the mantissa by the exponent value i.e. 2 places to the right
The floating point numbers represents 11.00 binary
or +3.0 decimal

Past Paper Questions and Answers

4 In each part of this question, all working must be shown.

A real binary number may be represented in normalised floating point binary notation using 5 bits for the mantissa and 3 bits for the exponent, both in two's complement binary.

(a) Convert each of the following binary numbers to denary.

(i)

0	1	1	0	1	0	1	0
mantissa					exponent		

(ii)

0	1	0	0	0	1	0	1
mantissa					exponent		

Question			Expected Answers	Mks
4	(a)	(i)	• exponent 010 represents 2 • mantissa 0.1101, move point 2 places right so becomes 011.01 • value is 3.25 *or* • exponent 010 represents 2 • mantissa 0.1101 represents 13/16 or 0.8125 • value is 13/16 multiplied by 2^{2} = 13/4 = 3.25 **[max 3]**	**[3]**
4	(a)	(ii)	• exponent 101 represents -3 • mantissa 0.1, move point 3 places left so becomes 0.0001 • value is 1/16 or 0.0625 *or* • exponent 101 represents -3 • mantissa 0.1 represents ½ or 0.5 • value is ½ multiplied by 2^{-3} = 1/16 or 0.0625 **[max 3]**	**[3]**

Question number four of all exam papers will involve a floating-point binary calculation

Data Structures and Data Manipulation

What the Specification Says:
Explain how static data structures may be used to implement dynamic data structures;
Describe algorithms for the insertion, retrieval and deletion of data items stored in stack, queue and tree structures;
Explain the difference between binary searching and serial searching, highlighting the disadvantages and disadvantages of each;
Explain how to merge data files;
Explain the differences between the insertion and quick sort methods, highlighting the characteristics, advantages and disadvantages of each.

Static and Dynamic Data Structures
- Static data structures are those which do not change in size while the program is running. Most arrays are static, once you declare them, they cannot change in size.
- Dynamic data structures can increase and decrease in size while the program is running.

Advantages of Static Data Structures
- Compiler can allocate space during compilation
- Easy to program
- Easy to check for overflow
- An array allows random access

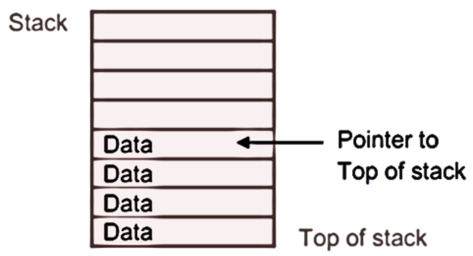
Disadvantages of Static Data Structures
- Programmer has to estimate maximum amount of space needed
- Can waste a lot of space

Advantages of Dynamic Data Structures
- Only uses the space that is needed at any time
- Makes efficient use of the memory
- Storage no longer required can be returned to the system for other uses

Disadvantages of Dynamic Data Structures
- Difficult to program
- Can be slow to implement searches
- A linked list only allows serial access

Static Structures holding Dynamic Structures
A static data structure (like an array) can hold a dynamic structure. The static structure must be big enough.

Stacks
A stack is a last in first out (LIFO or FILO) data structure. The head pointer will point to the most recent item of data which will be at the top. There are only two operations that can be applied, inserting and deleting/reading.

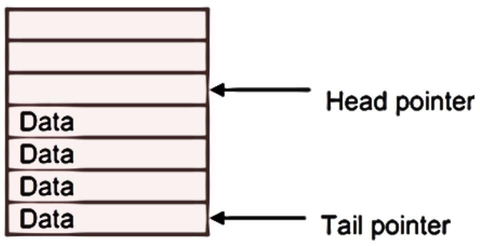


Inserting Data into a Stack
- First check that the stack is not full. If it is stop, and return an error.
- Next, increment the stack pointer, so it will now be pointing to the next empty data location.
- Finally insert the data into the location pointed to by the stack pointer.

Deleting and Reading from a Stack
- Check to see if the stack is empty. If it is stop and return an error.
- Copy the data item in the cell pointed to be the stack pointer.
- Decrement the stack pointer and stop.

Queues

A queue is a last in last out (LILO or FIFO) data structures. It has a head pointer, like that of a stack which points to the next empty data location, and a tail pointer which points to the last data item. Again, there are only two operations that can be done to a queue.



Inserting Data into a Queue

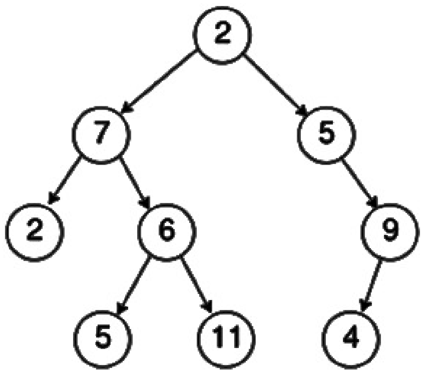
- Check that the queue is not full, if it is report an error and stop.
- Insert the new data item into the cell pointed to by the head pointer.
- Increment the head pointer and stop.

Deleting and Reading from a Queue

- Check that the queue is not empty, if it is report an error and stop.
- Copy the data item in the cell pointed to by the tail pointer.
- Increment the tail pointer and stop.

Binary Tree's

- A binary tree is a data structure, where each item of data points to another two items, and a rule is needed to determine the route taken from any data item.
- The data items are held in nodes.
- The possible routes are called paths.
- Each node has two possible paths.
- The nodes are arranged in layers.
- The first node is called the root, or root node.



Inserting Data into a Binary Tree

- Look at each node starting from the root
- If the new value is less than the value of the of the node, move left, other wise move right
- Repeat this for each node arrived until there is no node
- Then create a new node and insert the data.
- This can be written as:

```
1.  If tree is empty enter data item at root and stop.
2.  Current node = root.
3.  Repeat steps 4 and 5 until current node is null.
4.  If new data item is less than value at current node go left
else go right.
5.  Current node = node reached (null if no node).
6.  Create new node and enter data.
```

Deleting Data from a Tree

Deleting data from a tree is quite complicated, because if it has sub-nodes, these will also be deleted. There are two options.

- The structure could be left the same, but the value of that node set to deleted.
- The tree could be traversed, the value removed, then put back into a binary tree.

Serial Search

- Expects data to be in consecutive locations (such as, an array). Doesn't expect the data to be in any particular order.
- To find the position of a value, look at each value in turn, and compare it with the value that you are looking for.
- When the value is found, it's position must be noted. If it gets to the end and has not found the value, it is not in the array.
- Can be slow, especially for a large amount of data.
- The algorithm for this is:

```
1.  If n < 1 then report error, array is empty.
2.  For i = 1 to n do
a.  If DataArray[i] = X then return i and stop.
3.  Report error, X is not in the array and stop.
```

Binary Search

- Where the list is arranged in a particular order.
- The list is split in two, and compared to be either higher or lower than the value being searched for.
- The list is continually split further halving each time until the value is found.
- The algorithm for this is:

```
1.  While the list is not empty do
a.  Find the mid-point cell in the current list.
b.  If the value in this cell is the required value, return the cell
position and stop.
c.  If the value to be found is less than the value in the mid-point
cell then
  Make the search list the first half of the current search list
  Else make the search list the second half of the current
  search list.
2.  Report error, item not in the list.
```

Sorting

Sorting is placing values in order, such as numeric or alphabetic. There are two main types we need to know about. Insertion sort and quick sort.

Insertion Sort

Where the data of the files is copied into a new file, but copied into the correct location. The result is that the new file is in the correct order, although it's very time consuming.

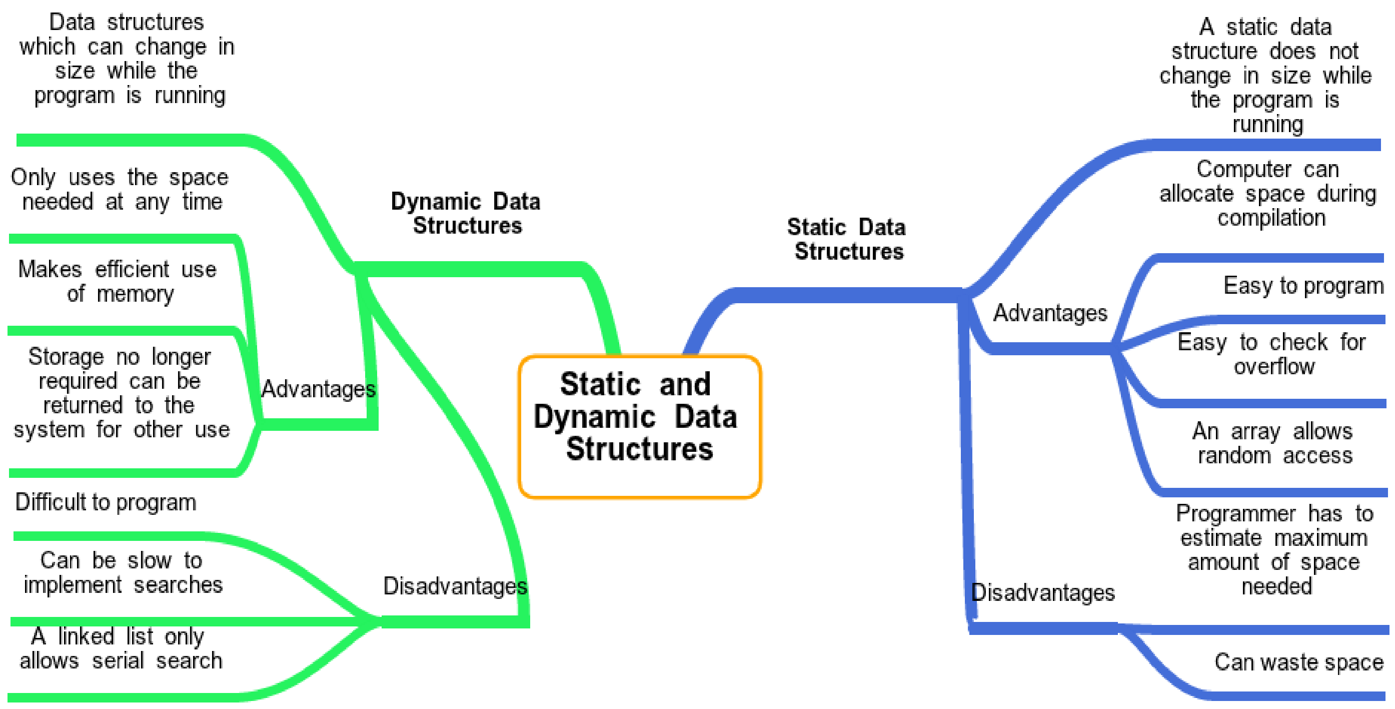
Quick Sort

- First the data is placed in a row with an arrow under the first and last values, pointing at each other, one is fixed where as one is movable.
- If the two values are in the correct order then move the movable arrow towards the fixed arrow, else swap the items and the arrows.
- Continue to repeat this until the arrows collide.
- Continue to repeat this process until the files are of a length of one.

Key Words

- **Data Structure** – Method of storing a group of related data.
- **List** – A simple one dimensional array
- **Pointers** – The numbers after the data, they point to the next data item.
- **Linked List** – A dynamic data structure similar to an array.
- **Queue** – A fist in first out data structure, containing a head and tail pointer.
- **Tree** – A data structure where each item of data points to two others.
- **Nodes** – These hold the data items.
- **Paths** – These are the routes between the nodes in a tree.
- **Layers** – Binary trees are arranged into layers, by the different levels.
- **Root** – The first node in a tree.
- **Insertion Sort** – A method of sorting data where all the items are copied to a new file, and put into the correct order.
- **Quick Sort** – A faster method of sorting data, involving two pointers which move towards each other, and swap values if data pointed at are in the wrong order.

Static and Dynamic Data Structures Summary:


Static and Dynamic Data Structures
Dynamic Data Structures
Data structures which can change in size while the program is running
Advantages
Only uses the space needed at any time
Makes efficient use of memory
Storage no longer required can be returned to the system for other use
Disadvantages
Difficult to program
Can be slow to implement searches
A linked list only allows serial search
Static Data Structures
A static data structure does not change in size while the program is running
Advantages
Computer can allocate space during compilation
Easy to program
Easy to check for overflow
An array allows random access
Disadvantages
Programmer has to estimate maximum amount of space needed
Can waste space

Past Exam Questions and Answers

Showing the steps of serial search
e.g. Find York
Aberdeen, Belfast, Cardiff, Oxford, York

1. start at 'Aberdeen'…
2. look at each word in turn/then 'Belfast', 'Cardiff' etc…
3. …until 'York' is found

Showing the steps of binary search
e.g. Find York
Aberdeen,Belfast, Cardiff, Oxford, York

look at middle/'Cardiff'/'Glasgow'
'York' is in second half of list
repeated halving…
…until 'York' is found

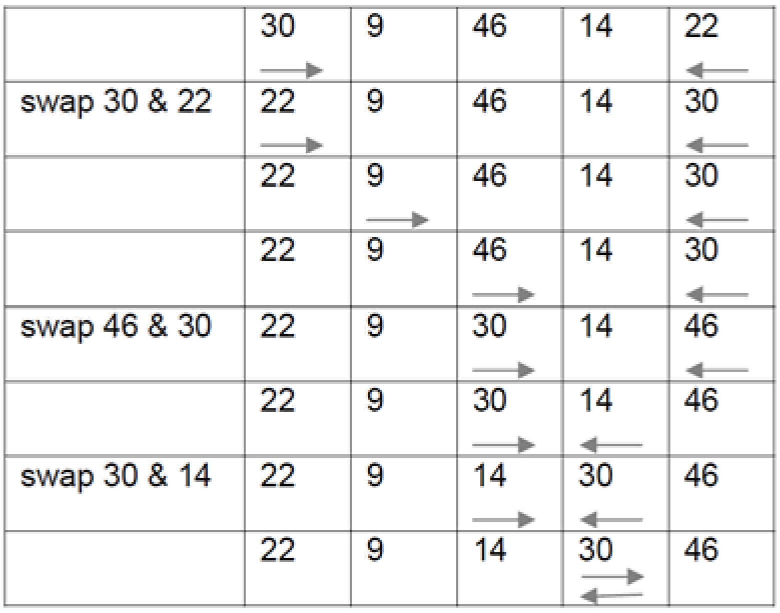
What are the advantages of binary search over seial search?

(usually) faster because…
…alf of data is discarded at each step/fewer items are checked

How do you add a data item into a stack?

if stack is full…
…report error and stop
increment pointer
add data item at position 'pointer

How can quicksort be used to put a set of numbers in ascending order?
e.g. 30 9 46 14 22



	30 →	9	46	14	22 ←
swap 30 & 22	22 →	9	46	14	30 ←
	22	9 →	46	14	30 ←
	22	9	46 →	14	30 ←
swap 46 & 30	22	9	30 →	14	46 ←
	22	9	30 →	14 ←	46
swap 30 & 14	22	9	14 →	30 ←	46
	22	9	14	30 →←	46

**

highlight first number in the list (the 'search number')
pointer at each end of list
repeat:
compare numbers being pointed to…
…f in wrong order, swap
move pointer of non-search number
until pointers coincide so search number in correct position
split list into 2 sublists
quick sort each sublist
repeat until all sublists have a single number
put sublists back together

Name another method or sorting

insertion sort or bubble sort

What is a static data structure?

size is fixed when structure is created/size cannot change during processing

What would be an advantage of static data structures over dynamic structures?

amount of storage is known/easier to program

How would you merge two files?
e.g. File A: Anna, Cleo,Helen, Pretti
Fie B: Billy, Ian, Omar, Rob, Tom

(Anna, Billy, Cleo, Helen, Ian, Omar, Pritti, Rob, Tom)
You must: get correct order, use all names used once

State the algorithm for merging two sorted files

```
open existing files
create new file
check existing files are not empty
use pointers/counters to identify records for
comparison
repeat
compare records indicated by pointers
copy earlier value record to new file
move correct pointer
until end of one file
copy remaining records from other file
close files
assume common key
assume if 2 records are the same…
…only 1 is written to new file
```

What is a dynamic data structure?

size changes as data is added & removed/**size** is not fixed

What would be a disadvantage to the programer of using dynamic data structures over static ones?

more complex program to write

State a data structure which must be static

array/fixed length record

What steps need to be taken to add a new item to a binary tree

start at root
repeat
compare new data with current data
if new data < current data, follow left pointer
else follow right pointer
until pointer is null
write new data
create (null) pointers for new data

How can insertion sort be used to arrange numbers in order?
e.g. 17 2 3 26 5

Original set	17	2	3	26	5
Insert 17	17	2	3	26	5
insert 2	2	17	3	26	5
insert 3	2	3	17	26	5
insert 26/no change	2	3	17	26	5
insert 5	2	3	5	17	26

**

list of sorted numbers is built up…
…ith one number at a time being inserted into correct position
plus 1 mark per correct row *[max 4 rows]* *

What features of quick sort are not used in insertion sort?

set of numbers broken into multiple sets
uses pivots

What steps are needed to be taken to pop a data item from a stack?

if stack is empty…
…eport error and stop
output data(stack_pointer)
decrement stack_pointer

What is the meaning of a dynamic data structure

size changes as data is added & removed/size is not fixed

What's the main disadvantage of a dynamic data structures over static one?

more complex program to write

What data structure must be static

array/fixed length record

How would you add a new item to an existing binary tree?

```
start at root
repeat
  compare new data with current data
  if new data < current data, follow left pointer
  else follow right pointer
until pointer is null
write new data
create (null) pointers for new data
```

High Level Programing Paradigms

What the Specification Says
Identify a variety of programming paradigms (low-level, object-oriented, declarative and procedural);
Explain, with examples, the terms object oriented, declarative and procedural as applied to high-level languages, showing an understanding of typical uses;
Discuss the concepts and, using examples, show an understanding of data encapsulation, classes and derived classes, and inheritance when referring to object-oriented languages;
Understand the purpose of the Unified Modelling Language (UML);
Interpret class, object, use case, state, sequence, activity and communication diagrams;
Create class, object, and use case and communication diagrams;
Discuss the concepts and, using examples, show an understanding of backtracking, instantiation, predicate logic and satisfying goals when referring to declarative languages.

Notes

Programing Paradigms
A programing paradigm is just a method, or *a way* of programing. This unit will cover the 4 main paradigms(low-level, object-orientated, declarative and procedural)

Low Level Paradigms

- Assembly language is a low level language.
- Assembly language was developed to make early programing easier, by replacing machine code functions with mnemonics and addresses with labels.
- It is a second generation language. Because it came after using strait machine code.

Procedural Languages

- Third generation, also called high level languages
- Problem orientated
- They describe exactly step by step what procedure should be followed to solve a problem.
- They use the constructs sequence, selection and repetition.
- May use functions and procedure, but they always specify the order instructions must be executed in, in order to solve the problem.
- Can be difficult to reuse code

Object Orientated Languages

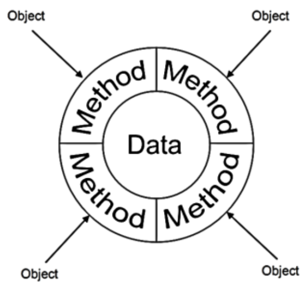
- Where data and the methods for manipulating the data are kept in a single unit, called an object.
- The only way the user can access the data in the object is through the objcts methods
- Once an object is fully working, it cannot be corrupted by the user – because they cannot change any of its parameters.
- Also the internal workings of the object may be changed without affecting the workings of the object.
- A class is a construct that is used as a blueprint or template for an object.
- An object is an entity that can be manipulated by the commands of the class.

Declarative Programing

- The computer is told what the problem is, not how to solve it.
- It searches for a solution from a database
- Very useful for solving problems in the real world, like medical diagnostics.
- The user inputs a query to the search engine, which searches a database for the answers, and returns them to the user.
- The programmer does not have to tell the computer how to answer the query, the system just consist of a search engine of facts and rules.
- Backtracking is going back to a previously found successful match in order to continue a search.
- Instantiation is giving a variable in a statement a value.
- Predicate logic is a branch of mathematics that manipulates logical statements that can be either True or False.
- A goal is a statement that we are trying to prove whether or not it is True or False

Encapsulation

- Data encapsulation is the concept that data can only be accessed via methods provided by the class.
- This ensures data integrity.
- Data encapsulation is the combining together of the variables and the methods that can operate on the variables so that the methods are the only ways of using the variables.



Inheritance

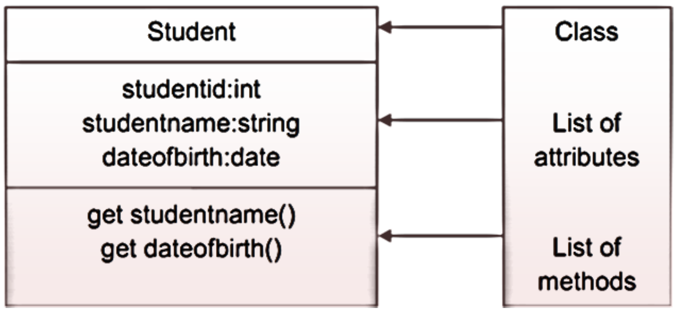
- Inheritance allows the reuse of code and the facility to extend the data and methods without affecting the original code.
- A derived class can be create which inherits its structure from the super class, but has added data.
- The derived class can be accessed through its methods, or the superclass's methods.
- A class describes the variables and methods appropriate to some real-world entity.
- An object is an instance of a class and is an actual real-world entity.
- Inheritance is the ability of a class to use the variables and methods of a class from which the new class is derived.

Unified Modelling Language (UML)

The Unified Modelling Language consists of a set of descriptive diagrammatic representations to describe the stages to produce effective object oriented programs. For the exam, we need to know 7 types.

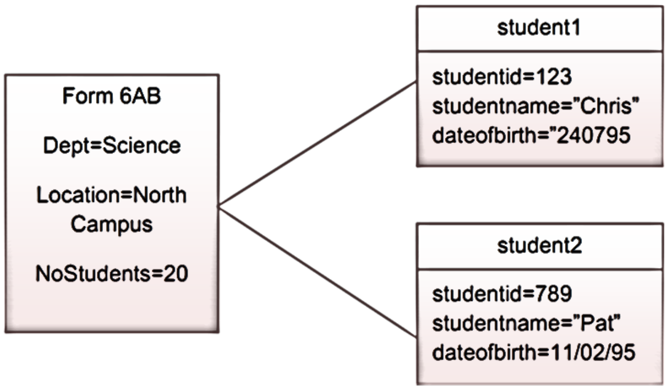
Class Diagrams

- Object oriented programming uses classes.
- A class is 'an entity of a given system that provides an encapsulation of the functionality of a given entity.'
- A class diagram is a rectangle, divided into 3 parts. The first gives the name of the class, the second gives the facts that should be known about any element belonging to that class and the third gives methods that can be used to look at the facts that are stored in the class.



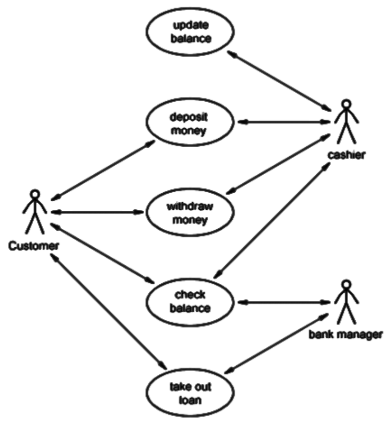
Object Diagrams

- If classes are groups of real things, with each class representing at least one entity
- Each object will have specific data to match each of the attributes.



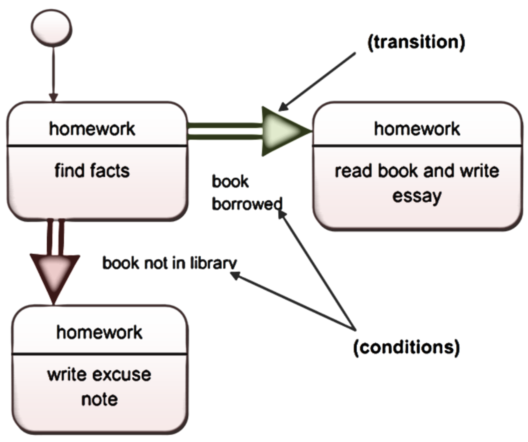
Use Case Diagrams

- Like business-orientated diagram show what should be going on in a system.
- Shows the users, or people.



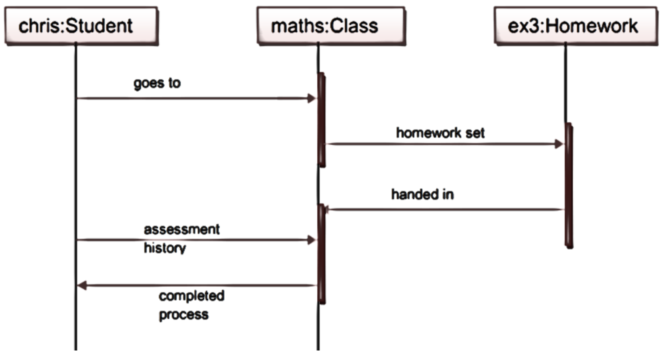
State Diagram

- A state diagram shows how an object may behave through the various processes of a system. It is a bit like a cross between a DFD and a flow diagram.
- It starts with a shaded in circle to show the initial state before anything has happened. Arrows are used to show the flow of activity to different outcomes for the object as it goes through the system



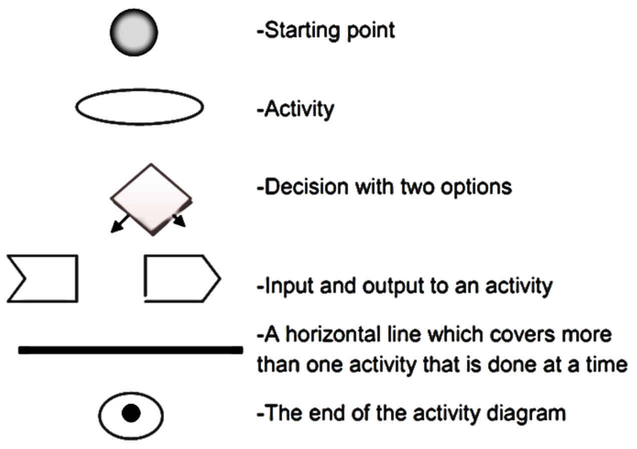
Sequence Diagrams

- The show how the objects in a class interact with one another.
- The classes go along the top of the diagram, with a dotted line underneath it which shows how long that object is said to exist, called it's lifeline. Where the dotted lines sometimes turn into thin boxes is where the classes methods have been called to do something.

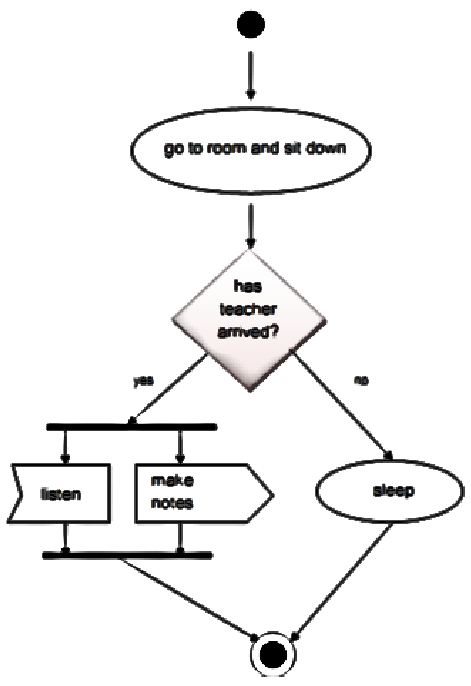


Activity Diagram

- Similar to a flow chart in procedural programing, although it shows the activities necessary to get an object into a particular state, rather than the development of the logic behind the code.

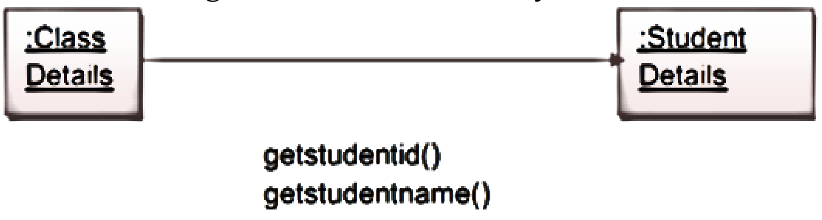


-Starting point
-Activity
-Decision with two options
-Input and output to an activity
-A horizontal line which covers more than one activity that is done at a time
-The end of the activity diagram



Communication Diagrams

- Used to show how different objects combine to pursue a common purpose.
- Each object is contained in a rectangle and the data that they share with one another is placed beside arrows showing in which direction the data is flowing.



Key Words

- **Programing Paradigm** – method of programing (e.g. low-level, procedural, declarative or object-orientated)
- **Low Level Language** – 2nd generation language with a one-to-one relationship with assembly code, makes use of mnemonics that represents assembly operations and labels that represent data locations.
- **Procedural Language** – where instructions are comprehensive, sequential and specify how to solve a problem.
- **Declarative Languages** – a problem-orientated language where the computer is given a set of facts and a goal, which it uses to interrogate a data base and find the solution. The computer is not told how to solve the problem, just what the problem is.
- **Goal** – this is that is trying to be found (in declarative programing)
- **Instance** – an existence of a piece of data (in declarative programing)
- **Backtracking** – where the computer goes back and takes another route while trying to find solutions to the problem (in declarative programing)
- **Predicate Logic** – is the process of different facts being applied to rules to produce a result that can be considered true or false.
- **Object-Orientated Programing –** paradigm that relies on objects in the real world being classified.
- **Class** – a group of data (in OOP)
- **Data Encapsulation** –data can only be accessed through methods in the class
- **Derived Classes** – classes that have inherited data from a super class.
- **Unified Modelling Language (UML)** - descriptive diagrammatic representation that describe stages required to produce object orientated programs.
- **Object Diagram** – shows the attributes for a specific object from a class.
- **Use Case Diagram** – shows what is happening in a system, rather than how it is done.
- **State Diagram** – shows the state of an object through the process.
- **Activity Diagram** – show how the logic behind the program was developed.
- **Communication Diagram** - shows how different objects combine to pursue a common purpose.
- **Sequence Diagram** - shows how the objects in a class interact with one another.
- **Class Diagram** – represents the classes, showing their methods and data.

Declarative Programing

- The computer is given a **set of facts and a goal**.
- It does not need a set of instructions - it is capable of **deciding how to solve the problem** itself.
- When it is run, the computer **applies the definitions to the facts** and supplies the result.
- The **goal** is what is being searched for.
- The value found is an **instance**
- When it has either found a result, or failed to find a result on one path, it will go back and check the other possible routes. This is called **backtracking**.
- Declrative languages use **predicate logic**.

Object Orientated Programing

- This is another programing paradigm, also called OOP.
- It relies on objects in the real world being classified.
- It can show information about a group of things that must have the same characteristics - the group is called a **class**
- The objects can provide data, which can only be accessed through the objects methods, this is called **data encapsulation**.
- **Derived classses** have **inherited** data from a **superclass**
- The methedology for planning and developing object orientated code is called **Unified Modeling Language (UML)**
- UML consists of a number descriptive diagramtic representations that decribe the stages required to produce OO programs.

Past Exam Questions and Answers

Explain backtracking

after finding a solution (to a goal)
go back and follow an alternative path…
…to attempt to find another solution

What is the need for BNF?

to unambiguously
define the syntax of a computer language

Why is UML used?

a standard way to present (information)…
…the design of a system…
…which is visual, so easy to understand
allows systems analysts, programmers and clients to communicate
makes system maintenance easier…
…when modifying a system

Describe the features of a procedural language

imperative language
uses sequence, selection & iteration
program states what to do…
… & how to do it
program statements are in blocks
each block is a procedure or function
logic of program is given as a series of procedure calls

Programing Techniques

What the Specification says
Explain how functions, procedures and their related variables may be used to develop a program in a structured way, using stepwise refinement;
Describe the use of parameters, local and global variables as standard programming techniques;
Explain how a stack is used to handle procedure calling and parameter passing;
Explain the need for, and be able to create and apply, BNF (Backus-Naur form) and syntax diagrams;
Explain the need for reverse Polish notation;
Convert between reverse Polish notation and infix form of algebraic expressions using trees and stacks.

Notes

Stepwise Refinement
This is where a complex problem is broken down into smaller and smaller sub-problems until all the sub-problems can be solved easily.

Functions and Procedures
- A procedure is a small section of code designed to perform a specific definable task, and may or may not return a single value.
- A function is a block of code which performs a single task or calculation and returns a single value. They use local variables.

How Functions and Procedures can Develop Programs in a Structured way
- Each module can be written as a functional procedure
- Modules can be tested individually
- Library routines can be used
- The code is reusable
- Main program consists of calls to functions/procedures which may be nested

Parameters
- A parameter is (information about) an item of data supplied to a procedure or function which may be passed by reference or by value, and is used as a local variable.

Local Variables
- Local variables exist only in the block which they are declared, they can only be accessed in that part.
- The data contained in the variable is lost when the execution of that part of the program is complete
- The same variable names can be used in different modules
- This means that different programmers do not have to worry about variables overwriting themselves.

Global Variables
- A variable that is defined at the start of a program and exists throughout program including functions/procedures.
- Allows data to be shared between modules
- Overridden by local variables with the same name

Stacks
- When a procedure or function is called the program needs to know where to return to when the execution is complete. The return address must be known.
- Also these functions and procedures may call more functions and procedures, all of these will have return addresses, which must be stored, along with the order.
- This is done using a stack
- When values are read, they are popped of the stack, but they remain in the stack.
- The stack pointer can be moved then items are popped of or pushed on.

The Purpose of the Stack

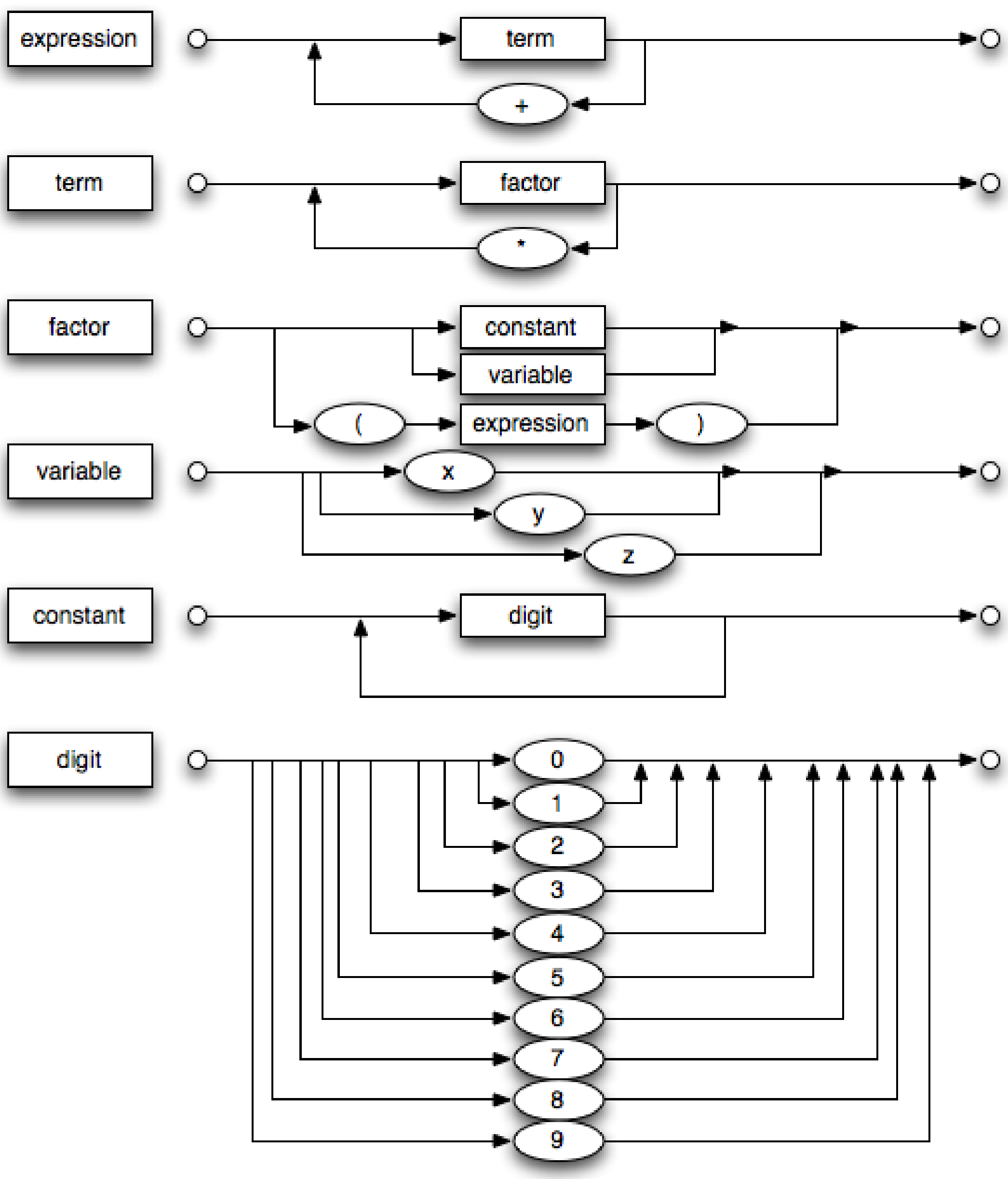
- So program can return correctly when procedure has been completed/store return address
- Allows data to be transferred

Backus-Naur Form

- BNF is to unambiguously define the syntax of a computer language

BNF and Syntax Diagrams Examples

```
<expression> ::= <term> | <expression> "+" <term>
<term>       ::= <factor> | <term> "*" <factor>
<factor>     ::= <constant> | <variable> | "(" <expression> ")"
<variable>   ::= "x" | "y" | "z"
<constant>   ::= <digit> | <digit> <constant>
<digit>      ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```



Past Paper Questions and Answers

What type of traversal should be used to obtain reverse polish from a binary tree?

post-order (traversal)

State a data structure that may be associated with reverse polish notation

stack
binary tree

What symbol used in mathematical expressions is not required in reverse polish?

bracket

How can functions and procedures develop a program in a structured way?

each module can be written as a functional
procedure…
…which can be tested individually
library routines
code is reusable
main program consists of calls to
functions/procedures…
…which may be nested

Compare the use of local and global variables and parameters

local variables
a variable defined within one part of program…
…& is only accessible in that part
data contained is lost when execution of that part of
program is completed
the same variable names can be used in different modules
global variables
a variable that is defined at the start of a program…
& exists throughout program…

…including functions/procedures
allows data to be shared between modules
overridden by local variables with the same name
parameters
information about an item of data…
…supplied to a function or procedure
can be passed by reference or by value
used as a local variable

What data structure is used when procedures are called during program execution?

stack

What is the purpose of using a stack?

so program can return correctly when procedure has
been completed/store return address
allows data to be transferred

State the need for BNF

to unambiguously define the syntax of a computer language

What's the use of functions, procedures and step-wise refinement when developing a program?

function:
block of code…
…which performs a single task/calculation…
returns a single value
uses local variables
procedure:
block of code…
…which performs a task
…which may or may not produce a single value
uses local variables

stepwise refinement:
breaks a problem into sections…
…which become progressively smaller…
…until each module can be written as a single procedure/function
each module can be tested separately
library routines can be used

What data structure is used to handle procedure calling and parameter passing?

stack

What's a parameter?

(information about) an item of data…
…supplied to a procedure or function
may be passed by reference or by value
used as a local variable

What is the purpose of a syntax diagram?

to define terms unambiguously (for a computer language)

What is an advantage of reverse polish over infix notation?

any expression can be processed in order (left to right)
no rules of precedence are needed/no brackets are needed/unambiguous

Low Level Languages

Explain the concepts and, using examples, demonstrate an understanding of the use of the accumulator, registers, and program counter;
Describe immediate, direct, indirect, relative and indexed addressing of memory when referring to low-level languages;
Discuss the concepts and, using examples, show an understanding of mnemonics, opcode, operand and symbolic addressing in assembly language to include simple arithmetic operations, data transfer and flow-control.

Notes

- Opcodes is the part of the binary string that represent the operations that the computer can understand and carry out. They are easier to remember.
- They can be represented by mnemonics which are the pseudo names given to the different operations that make it easier. E.g. ADD.
- The operand is the data to be manipulated, there's no point telling the computer what to ADD if there's no data to apply it to. It can hold the address of the data, or just the data.
- Address Labels are used as a symbolic representation of the operands
- Symbolic addressing is the use of characters to represent the address of a store location

There are three different operations that can be done in assemble language, which have different effects on the processor. These are: arithmetic or logic operations, data transfer and flow control.

Arithmetic Operations

- When opcode is decoded, data is collected and placed in the memory data register
- It is then manipulated in the ALU. Part of this is the accumulator, where results are temporarily stored until they are needed for the next operation.

Data Transfer

- Some operations (like GET and GTO) just move data in and out of the memory.

Flow Control

- This is where the order which instructions are executed may be changed by a jump instruction or a conditional jump instruction.

The special Registers

The program counter (PC) is used to keep track of the location of the next instruction to be executed. (This register is also known as the Sequence Control Register (SCR)). It is used so that the processor always knows where the next instruction is. Note that the content can be changed by some instructions as well as simple incrimination.

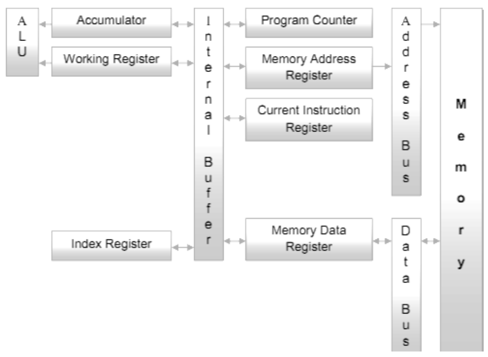
The memory address register (MAR) holds the address of the instruction or data that is to be fetched from memory. This address can come directly from the PC if the data to be fetched is the next instruction, or it can come from the CIR when the address of the data to be used is found in the decoding process.

The current instruction register (CIR) holds the instruction that is to be executed, ready for decoding. This instruction will consist of (at least) an operation code which will be looked up in the table of codes so that the processor knows what actions are necessary and the address which will be sent to the MAR.

The memory data register (MDR) holds data to be transferred to memory and data that is being transferred from memory, including instructions on their way to the CIR. The computer cannot distinguish between data and instructions. Both are held as binary numbers. How these binary numbers are interpreted depends on the registers in which they end up. The MDR is the only route between the other registers and the main memory of the computer.

The accumulator is where results are temporarily held and is used in conjunction with a working register (ALU) to do calculations.

The index register is a special register used to adjust the address part of an instruction. If this is to be used then the simple case of two parts to the instruction which was detailed above, must become three parts because there must be a part of the instruction which defines which sort of addressing must be used.



Addressing

Each instruction in machine code is represented by a series of binary digits. There are two parts to each instruction, an operation (the opcode) and the data. The data is what the operation is being applied to, there are a number of different ways in which this data can be represented, and this is known as addressing. Usually the address of the data is given rather than just the data.

Direct Addressing
This is where the address part of the instruction holds the address of where the data is. For example if an instruction was 00110111 and the first 3 bits were the instruction e.g. ADD and the last 5 bits were the address, then the computer would know to add whatever is in the data location 10111 to the accumulator. Most bytes are bigger than 8 bits, more like 32 or 64 bits. However it does not give access to enough memory, and needs supplementing with other methods.

Immediate Addressing
This is when the value in the instruction is not an address at all but the actual data. This is very simple, although not often used because the program parameters cannot be changed. This means that the data being operated on can't be adjusted and only uses constants.

Indirect Addressing
This is where the real address is stored in the memory so the value in the address part of the instruction is pointing to the address of the data. This method is useful because the amount of space in a location is much bigger than the space in the address part of the instruction. Therefore we can store larger addresses and use more memory.

Relative Addressing
This is direct addressing that does not commence from the start of the address of the memory. It begins from a fixed point, and all addresses are relative to that point.

Indexed Addressing
This is where the address part of the instruction is added to the value held in a special register. The special register use is the Index Register (IR). The result of this is then the required address.

Addressing

Direct Addressing

The address in the instruction is the address to be used. It is very simple, although does not make best use of memory

Immediate Addressing

This is where the value to be used is stored in the instruction. The program parameters can't be changed.

Indirect Addressing

Where the real address is stored in the memory and so the value in the address part of the instruction is pointing to the data. This method can store bigger addresses.

Relative Addressing

This is like direct addressing, except it doesn't begin from the start of the memory. It starts from a fixed point.

Indexed Addressing

The address part of the instruction is added to a value held in a special register. This register is called the index register.

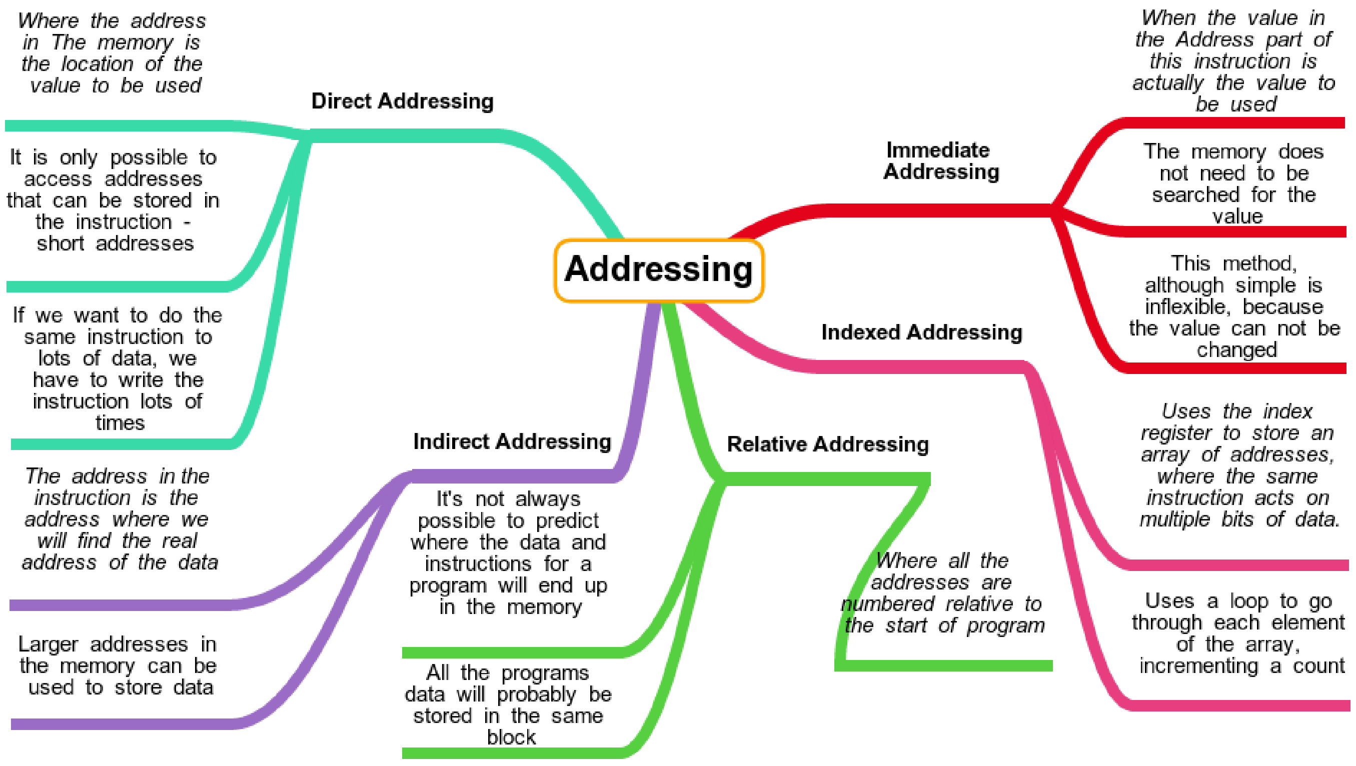
Features of Low Level Languages

Computers work with machine code instructions which are written as a series of binary digits, each instruction is in two parts, as operation and an address.

Each computer has a different operation set, which means different computers understand different operations – the machine code is unique to that machine.

For example, if instructions were 8 bits, and there were 3 bits used to stand for the operation code (8 possible combinations) and 5 bits used to represent the identifier of the address, where the data is stored. (Note: this is very simplified, and the binary instructions are usually considerably longer that 8 bits). If 001 means ADD then 00101001 would mean add whatever is in location 01001. This value will then be added to the accumulator.

Next, read about addressing, there are five different types in the specification.


Addressing
Direct Addressing
Where the address in The memory is the location of the value to be used
It is only possible to access addresses that can be stored in the instruction - short addresses
If we want to do the same instruction to lots of data, we have to write the instruction lots of times
Indirect Addressing
The address in the instruction is the address where we will find the real address of the data
Larger addresses in the memory can be used to store data
Relative Addressing
It's not always possible to predict where the data and instructions for a program will end up in the memory
All the programs data will probably be stored in the same block
Where all the addresses are numbered relative to the start of program
Immediate Addressing
When the value in the Address part of this instruction is actually the value to be used
The memory does not need to be searched for the value
This method, although simple is inflexible, because the value can not be changed
Indexed Addressing
Uses the index register to store an array of addresses, where the same instruction acts on multiple bits of data.
Uses a loop to go through each element of the array, incrementing a count

Past Exam Questions and Answers

Explain the term opcode

the mnemonic part of the instruction/that indicates what it is to do/code for the operation

What is symbolic addressing?

the use of characters to represent the address of a store location

What is the purpose of the accumulator?

temporary storage (within ALU)
holds data being processed/used during calculations
deals with the input and output in the processor

What is indexed addressing?

uses an index register/IR
…and an absolute address…
…to calculate addresses to be used

What is direct addressing?

the instruction gives the address to be used

What is relative addressing?

allows a real address to be calculated…
…from a base address…
…by adding the relative address
relative address is an offset
can be used for arrays

can be used for branching

What’s immediate addressing?

used in assembly language
uses data in address field…
… as a constant

Why is it not possible to use only direct addressing in assembly languages?

number of addresses available is limited…
…by the size of the address field
code is not relocatable/code uses fixed memory locations

Explain Mnemonics

a code that is easily remembered…
…used to give the opcode/instruction
e.g. ADD

Explain flow controll

the order in which instructions are executed
the order may be changed by a jump
instruction/conditional jump instruction

How and why is the the index register (IR) used?

used in indexed addressing
stores a number used to modify an address…
… which is given in an instruction
allows efficient access to a range of memory locations/by incrementing the value in the IR
eg used to access an array

What are the differences between machine code and assembly language?

Machine Code:
written in binary or hex
no translation needed
very difficult to write
Assembly language:
includes mnemonics
includes names for data stores
translated by an assembler
easier to write than machine code, but more difficult than high level language

What's the use of an operand, in an assembly language instruction?

address field (in an instruction)
it holds data…
to be used by the operation given in the opcode
eg in ADD 12, "12" is the operand

What's the difference between direct and indirect addressing?

direct:
the simplest/most common method of addressing
uses the (data in) the address field…without modification
eg In ADD 23, use the number stored in address 23 for
the instruction (accept any valid example)
limits the memory locations that can be addressed *
indirect:
uses the address field as a vector/pointer… to the address to be used
used to access library routines
eg In ADD 23, if address 23 stores 45, address 45 holds the number to be used increases the memory locations that can be addressed

Databases

What the Specification Says
Describe flat files and relational databases, explaining the differences between them;
Design a simple relational database to the third normal form (3NF), using entity-relationship (E-R) diagrams and decomposition;
Define and explain the purpose of primary, secondary and foreign keys;
Describe the structure of a DBMS including the function and purpose of the data dictionary, data description language (DDL) and data manipulation language (DML);
Use SQL to define tables and views, insert, select and delete data and to produce reports.

Notes

Databases

- A file is a collection of sets of similar data, called records.
- An item is what each item of data within a record is called.
- The items are stored in fields.
- Records all have the same sort of contents, but relates to a different object, they are usually represented as a row in a table.
- A database is a collection of data arranged into related tables. How it is arranged depends on its normal form.

Flat Databases

Originally all data was held in files, which consisted of a large number of records, each containing a larger still number of fields. Each field has its own data type and stores a single item of data.
This lead to very large files that were difficult to process, and quite inflexible not making the best use of computer resources. The disadvantages of it are:

- Separation and isolation of data.
- Duplication of data.
- Data dependence.
- Incompatibility of files.
- Fixed queries and the proliferation of application of programs.

A solution to this is to use a relational database.

Relational Databases

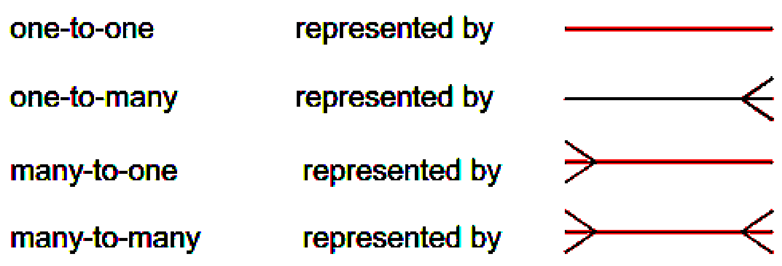
- Instead of individual unrelated files, the data is stored in tables which are related to each other.
- Each table has a key field by which their values in that table are identified.
- The records (or entities) in the tables can be related to entities in other tables by sharing keys as attributes within the entities
- There is less data duplication, as relationships are used to link matching fields.

Arranging Data

- A normal form is the name given to how data in a database is arranged
- First normal form (1NF) is where each table has no repeating groups
- Second normal form (2NF) is where the values of the attributes are all dependant on the primary key
- Third normal form (3NF) is where no attributes are predictable because of one of the other attributes

Entity Relationship Diagrams

These are used to illustrate the relationships between entities. If 3NF is used, there will be no many-to-many relationships, and one-to-one relationships can be combined. Many-to-many relationships can be replaced by a link entity.



Keys

- The primary or key field is a unique field used to identify a record
- A foreign key is a primary key in one table used as an attribute in another to link tables though providing relationships
- A secondary key is a field in a table that can be used to access the data in different ways. It is used to search for a group of records.
- The candidate keys are all the possible primary keys, i.e. all the unique fields

Database Management System (DBMS)

- Contains the data definition language (DDL)
- Contains the data manipulation language (DML)
- Allows data to be amended and controlled
- Includes a data dictionary which is a file of descriptions of the data and structure.

Data Definition Language (DDL)

- Part of the DBMS
- Used by the database designer to define the tables of the database
- Allows the designer to specify data types and data structures as well as any constraints on the database
- Cannot be used to manipulated the data
- Just creates tables and structures that hold information about the data that will be put in the tables, such as validation checks.
-

Data Manipulation Language

- Allows data to be manipulated
- Users of the database will have different rights

Views of Data

The DBMS can be made to present various views of the data held in the database.

- Internal level, this is the view of the entire database store in the system. It is hidden from the user by the DBMS.
- Conceptual level, gives a single usable view of all the data on the database.
- External level, where data is arranged according to user requirements and rights. Different users will get different views of the data.

Key Words

- F**ile** – a is a collection of sets of similar data called **records**
- **Table -** another name for a file
- **Tuple** - another name for a record
- **Item** - an item of data within a recorded
- **Field** - the area where items are stored
- **Attribute -** another name for a field
- **Database** - a series of related files, called **tables**
- **Primary** or **key field -** a unique field in a record
- **Foreign key** - the field which contains a link to another field
- **Secondary key** - a field in a table that can be used to access the data in different ways
- **Normal Form** - the name given to how data can be arranged in a system
- **1NF –** where there are no attributes that have multiple data in them
- **2NF –** where the values of the attributes are all independent of on the primary key
- **3NF –** where no attributes are predictable because of any other attributes
- **Link Entity –** additional table used to join many-to-many relationships in 3NF
- **DBMS –** database management system, the piece of software that allows databases to be easily viewed and edited and ensures that the rules remain unbroken. Includes DML, DDL and the data dictionary.
- **DDL –** data description language, used to define tables in the database as well as the data types and data structures to be used, and any validation checks required.
- **Schema –** The database design produced by the DDL.
- **DML –** data manipulation language, allows different users to carry out operations on the data. (e.g. amend, delete, insert data). Different users will have different permissions.
- **Data Dictionary** - a file maintained by the DBMS which includes the descriptions of the data and structure of the storage of the data.
- **Internal Level –** a view of the entire database as it is stored in the system. The data is organised as it is stored, usually hidden from the user by the DBMS.
- **Conceptual Level –** gives a single usable view of all the data on in the database.
- **External Level –** where the data is arranged according to the user requirements and rights. Different users will get different views of the data.
- **SQL –** structured query language, a language that allows the user to set up their own queries on the database.

Past Exam Questions and Answers

What is the use of the primary key?

unique identifier

What is the use of the foreign key?

primary key from one table…
…used as an attribute in another…
…to link tables/provide relationship between tables

What is the use of the secondary key

used to search for a group of records

What is a data dictionary?

a file containing descriptions of data in database
used by database managers…
…then altering database structure
Uses metadata to define the tables

What are the benefits of a relational database compared with a flat file database?

avoid data duplication/save storage
data consistency
data integrity
easier to change data
easier to change data format
data can be added easily
data security/easier to control access to data.

Explain why a foreign key is also a primary key, but a primary key need not be a foreign key

foreign key links tables (to represent many to one relationship)…
…so that only one record is accessed/to avoid duplicate data
eg primary key from B used as foreign key in C from (a)
primary key is in a table that may contain data not required in another table
eg primary key from C is not used in B and hence cannot be a foreign key

What's the difference between char and varchar data types

CHAR is fixed length
VARCHAR is variable length

Why are different views of data made available to users in a database?

so users can access the data they need
users do not need specialist knowledge
to protect data
to prevent unauthorised access.

What are the consequences of a many-to-many relationship?

not allowed/not in 3NF
needs another table between Student & Subject…
…to avoid duplication of data/to change to 3NF

What is a report, and what are the key features?

presentation of selected data…
…usually in the form of a table/specific layout
may be defined in advance…
…so the user does not need to set it up
Features of report definition: a query, a display order