



HARDWARE & OPERATING SYSTEMS

TUTORS

These notes are designed to assist teachers of the course and are in a condensed format. Teachers should also consult the syllabus for this module and adapt these notes accordingly by using extra examples and by filling out this material with detail. Students will be expected to apply the material in a BUSINESS environment and with reference to the particular situation specified in the examination question. Repeating these or any other notes in a generalised form is unlikely to satisfy the required answers for any question.

TYPES OF COMPUTERS

PC (personal computer) is a desktop, single user computer which could be a terminal to a network. It will usually have a private hard disk and keyboard entry. Additional features could be an attached printer, mouse, built-in modem, variety of CD/DVD drives, USB ports for add-on peripherals...

Laptop is a portable PC with rechargeable battery supply and flat LCD screen. A mouse can be attached but is normally fixed and position next to the keyboard. Data transfers can be made to a normal PC and internet connection is also possible.

Definitions

Mb (megabyte)	= refers to the storage size of memory or hard disc and is 1 000 000 bytes
DVD ROM	= is external Video disc for large user files
SD RAM	= Static Dynamic RAM for temporary user programs and data
CD-ReWriter	= device to record output on CD
Tower	= Processor unit built as a vertical stack to reduce desk space needed
Cache	= fast memory used for regularly used routines
ROM	= non volatile memory for running the system start up
RAM	= volatile memory for user programs/temporary data
EPROM	= ROM but can be reprogrammed with an EPROM blower after erasing with U-V light
PROM	= ROM which can be programmed <u>once</u>

Specifications of the latest models

These notes will not attempt to define the characteristics of the latest state-of-the-art computers because the specifications are continually changing. Candidates must have access to advertisements from computer suppliers for the latest figures.

FETCH-EXECUTE CYCLE (see Notes for Information Processing)

REGISTERS

These are special purpose single address locations used for specialised purposes and they feature in the FETCH and EXECUTE.

Accumulator – used to hold current data value during the running of a program and features in input/output, data movement, arithmetic and comparison operations. Used only in the EXECUTE phase.

CIR – Current Instruction Register holds the latest instruction read in from memory. Its contents are decoded by the control unit decoder (in the simplest format) into operation code and data address. It features in both FETCH and EXECUTE phases.

MAR – Memory Address Register holds the address where data can be found in memory so that it can be transferred in or out via the address bus. FETCH and EXECUTE

MDR/MBR – memory data register/memory buffer register. Holds data / instructions brought in from or memory and data going back to memory. Features in both FETCH and EXECUTE phases.

PC (or SCR) – program counter holds the memory address where next logical instruction in the current program can be found in memory. As soon as the instruction is loaded, PC is incremented by 1 ready for the next Fetch. If the instruction is a branch/jump instruction, then the next instruction may be altered during EXECUTE if the conditions tested are true (i.e. accumulator = 0) and the address in the PC is overridden – amended after the accumulator value is tested. Features in both FETCH and EXECUTE phases.

INSTRUCTION DECODER

The Control Unit contains an instruction decoder which operates on the binary coded instruction fetched from memory and held in the Current Instruction Register. The purpose of this decoder is:

1. Separate the operation code and address parts of the instruction.
2. Determine the nature of the operation code – an ADD instruction would require a signal to the ALU to perform an addition when it receives the data from memory to be added to the accumulator. A STORE/MOVE instruction would signal for the data in the accumulator to be copied to memory. A conditional BRANCH/JUMP instruction would require a signal for the ALU to compare the contents of the accumulator with ZERO.
3. Determine the address of the current instruction which would be placed in the MAR so that the correct part of memory could be accessed. The ADD instruction would require the contents of the memory at the given address to be placed on the Data Bus and into the MBR so that the data could be made available to the ALU for adding this value to the accumulator.

RISC INSTRUCTIONS (Reduced Instruction Set Computer)

Most of the time, the actual operation codes used in programs are limited to a subset of those available. Far more LDA (Load number to accumulator), STO (Store accumulator contents to memory) and ADD instructions are used than any others. Some more complex operation codes may not be used at all in a particular program.

RISC computers use a shorter set of operation codes which has the effect of making the computer

faster in operation. If one of the more complex instructions is needed, it is made up by using several of the other codes.

MEMORY MAP

While it is convenient to think of memory as 2-dimensional, in fact it is effectively a continuous line of addresses numbered from zero upwards. At anytime, there will be several programs in memory occupying their own spaces and usually starting on distinct boundaries e.g. 1KB. Some of these programs will be operating system routines loaded at booting time. Programs deleted release their space and can be taken up by other routines. If a program is too big to fit into a space, it can still use that space with a transfer of control at the end of it to another part of memory where the rest of the program resides.

Relocatable code – is program code which can be loaded into ANY space in memory and any internal jumps within the program are catered for by the use of an OFFSET address.

Arrays – A 2-dimensional array is a concept devised by humans to represent data in table form. However, in memory, it will occupy the 1-dimensional space described above. So an array with 3 rows and 4 columns, each needing 1 byte and beginning in memory address could actually be located as follows:

Memory address	1000	1001	1002	1003	1004	1005	1006
Array Row,Column	1,1		1,2		1,3		1,4
	2,1	2,2	2,3			

The 12 element of the array would occupy addresses 1000 to 1011. Of course, the array could also be held in the order below which holds the first column first (three rows of data) and then other columns:

	1,1	2,1	3,1	1,2
2,2	3,2		

In which case only the first and last of the array elements would occupy the same position in the two arrangements. A formula for the first arrangement to locate the exact memory address of element with row R and column C starting at 1000 would be $1000+4x(R-1) + (C-1)$. Students should check this is correct and devise a formula for the other arrangement.

MEMORY ADDRESSING

There are FOUR major memory addressing modes. In each of the examples below, assume memory location P has the value 20. The operation codes in these examples are not taken from any specific assembly language but used here to illustrate their operations.

Direct - Address gives the location address where data is stored in memory.
 e.g. LDA P results in 20 → accumulator

Indirect – This is a two-stage access. The address part gives a location in memory where the address of data can be found
 e.g. LDI P means look in location P, and use this as the address to find the data to load. So, if memory address 20 contains 66, the contents of memory 66 will eventually be loaded into the accumulator. If memory address 66 holds the value 15, then LDI P results in 15 → accumulator.

Immediate – The Address part gives a NUMERIC value that will be placed in the accumulator.
 e.g. LDN 23 results in 23 → accumulator.

Indexed – To the address given, the value in an index register (N) is added to give address where the data is actually stored.
 e.g. LDA P,N results in loading the contents of memory address 24 if N holds the value 4.

This is equivalent to providing a subscript for accessing an array. If the first element of the array is in memory address 20, then by increasing the value in N from 0 upwards in stages, each element of the array will be accessed in turn.

ADDITIONAL NOTES

1. Instructions can be more complex than just operation code and address. Some instructions might reference a register or a second address. For the purpose of the examination, only simple types will be considered.
2. Assembly language mnemonic codes can differ quite considerably across different machines. Candidates will always be asked to define some of the codes they use. This means explaining FULLY, not only the meaning of the operation code (e.g. ADD) but also the effect this has on the address part quoted with the instruction. Too often, this is taken for granted.

SHIFTS INSTRUCTIONS

These are instructions that MOVE the bits within the accumulator register. For all of the examples below,

- a) Assume the accumulator initially contains the value 1111 0011 1100 1001
- b) Bits lost in the operation are shown in ().
- c) Extra bits added are underlined

Logical – moves bits left(L) or right(R) losing some bits at one end and filling with ZERO at the other end.

e.g. LSR 2 gives 00 1111 0011 1100 10 (01) - 00 gained at the left, 01 lost at the right

LSL 3 gives (111)1 0011 1100 1001 000 - gain at the right, loss at the left
 USE: If a particular bit is moved to the extreme LEFT, it will occupy the position of the sign bit. If the accumulator is now tested to see if it contains a NEGATIVE number, this will tell whether that bit was originally set to “1” or not.

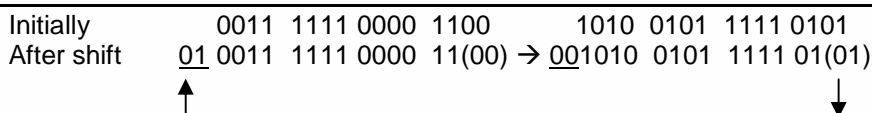
Circular – bits lost at one end are replaced at the other end.

e.g. CSR 2 gives 01 1111 0011 1100 10 (01) - CSL bits moved off left end and added to right end.

Arithmetic – is a logical shift BUT the sign bit is preserved.

e.g. ASR 2 gives 11 1111 0011 1100 10 (01). The original number had a “1” in the sign bit so when all the bits move right by two places, the gap left at the left hand end is replaced by 1’s.

Double Length Shift – effectively considers TWO registers to be positioned one after the other. Each of the above 3 shift operations could apply but with reference to two registers. So, CDR 2 (Circular Double Right 2) applied to registers X and Y would have the following effect:



- and NOT item ordered with identifiers of each customers ordering it. What about those customers who do not order a item? The program would not go through 25000 items and consider whether each customer wanted each.

How does the computer handle the need for the program to return back to the start of a loop and how can it cope with two loops as above?

SOLUTION : When the program reaches the point A, the address of the instruction at A is placed on the stack. Shortly after, the address of B is placed on the stack. When the execution of this program reaches C and there is a decision to repeat for another item, the address B on the top of the stack is **POPPED** to find out where to return in the program. In returning control to B, the address of B will be **PUSHED** back onto the stack. When eventually there are no more items ordered, the program execution passes onto D. C's address will have been **POPPED** off the stack and so the top of the stack will only hold the address of A. If there are more customers, control returns to A. The address B is **PUSHED** back onto the stack to deal with the second customer's items. Only when all customers have had all their orders processed will the stack be empty - there will be no need to return to the start of either loop.

RECURSION – is the process whereby a program routine can call itself. This can simplify some complex programming problems. The simplest example is when calculating $4 \times 3 \times 2 \times 1$. This is known as **factorial 4** and is often written $4!$. It is not difficult to see that $4! = 4 \times 3!$. If we know $3!$, we can calculate $4!$. But what is $3!$? Simple $3! = 3 \times 2!$ and $2! = 2 \times 1!$. Each time the process is effectively reusing itself but at a lower level. Fortunately, $1! = 1$ so here is the end of the chain.

The program routine **FACTORIAL(n)** to calculate $n!$ has the number n given to it as a parameter. It would consist of two parts:

```
A FACTORIAL (n)
    IF n>1
        then F ← n x FACTORIAL (n-1)
        else F ← 1
    ENDIF
B END ROUTINE
```

The routine continues to call itself but has an exit clause when eventually it is being asked to calculate $1!$. When this occurs, it is no longer calling itself. Note that in the first use of the routine, control does not pass out of the bottom of the routine but goes off to enter it again, this time with a small factorial value to calculate. **NOTE:** This is **NOT** looping, it is effectively re-entering another version of the routine. How are stacks affected by this?

At each call of the routine, the number whose factorial is needed and the address of the start of the routine are placed on the stack. To calculate $4!$ where A is the address of the start of the routine, the stack would gradually build up to:

4, A , 3, A, 2, A, 1, A

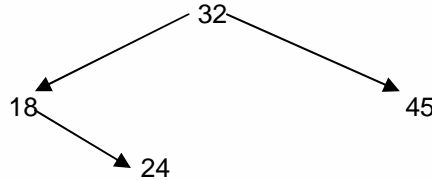
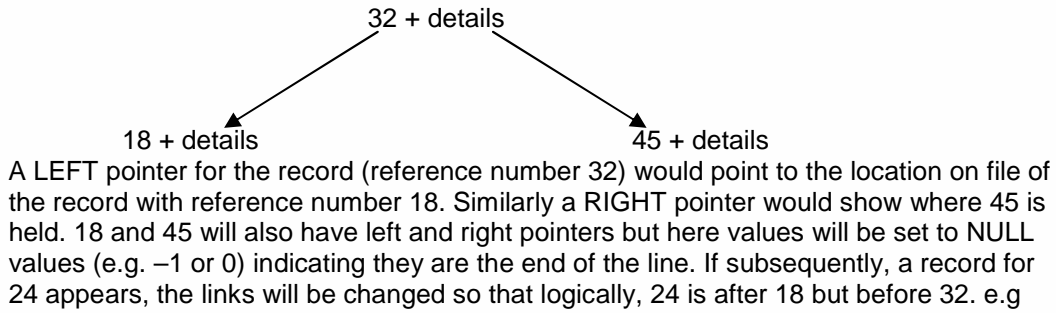
Which is saying use the routine to calculate $4!$, then (within the routine calculating $4!$, call it again for $3!$, then again for $2!$ and finally for $1!$ This last time, the answer is known $1! = 1$ so there is no need to call the routine any more and the routine “unwinds” by reaching the point B for the first time. Control passes back into the middle of the routine for the point B to be reached a second time. This would happen four times. The information would be taken out of the stack with a series of **POPs** until eventually the stack is empty and $F = 4 \times 3 \times 2 \times 1$. Note recursion means that the routine is entered 4 times before eventually leaving it 4 times at the end.

Summarising (using F for **FACTORIAL**) the process is:

(i) into F (for $4!$) – within this, (ii) re- enter F (for $3!$) - within this, (iii) re- enter F (for $2!$) - within this, (iv) re- enter F (for $1!$) - $1!$ is known so its value is set as 1 and F can be exited (iv) – there is a value for $1!$ so F can be exited having calculated $2!$ (iii) – a value for $2!$ is known so F can be exited (iv) – a value for $3!$ is known so F can be exited (ii) – a value for $4!$ is known so exit F for the last time. The stack is empty.

RECURSION can be used with **trees** of data where **randomly** entered data is organised (perhaps in numerical order) by linking each item to others by the use of left and right points.

e.g. Consider records each with a reference number (here 2 digits)



Now reference number 18 will have a right pointer value which is not NULL. To find any particular record (target) given the TOP (sometimes known as the **datum**) of the tree, it is only necessary to follow the left and right links down the tree. At each NODE, three different outcomes are possible

- the correct record has been found – data found. No need to go any further down the tree
- the record found has a reference number < target – therefore use LEFT pointer
- the record found has a reference number > target – therefore use RIGHT pointer

The repeated accesses to the tree can be implemented using a recursive process and therefore a stack could hold the current position and historical path if needed. Adding to a tree or deleting from a tree requires the path to be identified (see Linked List below)

LINKED LIST is similar to a tree where items perhaps are held in memory in an array in totally random order as they are entered but are ordered by having a pointer which shows where the next logical one item is. It is necessary with a dynamic linked list to have

- two linked lists – one holding the data and the other linking all the free spaces
 - DATUM – the address of the first item in the list – the one with lowest value
 - FREE – the address of the first space in the free list.
- When a new item is added, the following processes occur:
1. The first spare location (X) is taken from the free space list – FREE must be adjusted
 2. The data is placed in X
 3. The list is traversed from DATUM onwards to locate between which existing two items (P and Q), X belongs LOGICALLY.
 4. The pointers for P, Q and X must be adjusted accordingly.
- If an item has a value less than that of the datum, then this item would be marked as the new datum with a pointer to the location of the old datum.
- To remove an item from the list, the reverse occurs – the space it occupies is returned to the free space list, and the two items between which it was located logically are adjusted so that they now point to each other.

Queue – is a structure sometimes known as FIFO – First In First Out. It resembles the physical queue that people form when lining up in order for some particular reason. Queues would clearly be important anywhere in a computer system where priorities applied. e.g. queuing of jobs for the network printer. In memory, a queue would also be held in an array and would require further information:

- a) Address of first item in queue
- b) Count of the number of items in the queue

The problem with a queue is that the queue shortens at one end as the first items are processed and lengthens at the other end as new items join the queue. In a human queue, the problem does not arise because all the people in a queue advance as the people at the front leave. This could be achieved on a computer but it involves unnecessary movement of data. Instead, a CIRCULAR queue is usually used.

Position	1	2	3	4	5	6	7	8	9
Item	23	14	28	37	19	62	19	33	46

Let Start of queue = 7

Number in queue = 6 i.e. 6 numbers starting at position 7

From this data, the current queue in priority order is 19, 33, 46, 23, 14, 28

After the last position in this table, the queue is assumed to continue from the start of the array so 46 is followed by 23. All the other numbers in this queue represent items that have been in the queue but have since left the queue. The fact that position 5 (no longer in the queue) is also 19 means that this item WAS in the queue earlier, left it and has now REJOINED.

INPUT DEVICES

Keyboard – This has the standard typewriter keyboard layout with additional keys as follows:

- Function keys F1 to F12
- Various control keys – ESCAPE, CONTROL, ALT, PRINT SCREEN, BREAK, NUM LOCK, INSERT
- Number Pad for pure number entry
- Cursor Keys – UP, DOWN, LEFT, RIGHT, PAGE UP/DOWN, HOME/END

When a key is pressed, it activates the touch pad below and a circuit is completed. A 2-D grid identifies the particular character which is then converted to byte code characters. These queue in a buffer.

USE: correcting data, online ordering, entries to spreadsheet/WP programs

A **concept** keyboard is designed for a limited type of entry and has a totally different layout.

e.g. Use in restaurants for one-key selection of a particular meal.

Touch Screen - This should not be confused with VDU. Items on the screen can be selected by “touching” an icon. The device works because the finger breaks two infra-red beams just in front of the screen, one vertical and one horizontal and this determines the x,y positions selected on the screen. Strictly, it is not a touch screen.

USE: Restaurant to select set meals so the operator does not need to know or enter prices and the order goes directly to the kitchen. Tourist information can be provided by these screens without the need for staff attendance.

Magnetic Ink Character Recognition Device (MICR) – It was developed for banks and by banks solely for the processing of cheques. The data occupies ONE line only of stylised characters printed in magnetisable ink. The font gives a reasonable additional security against forgery. The PRE-printed data is bank account number, bank sort code (to identify which bank issued the cheque) and cheque number. When the cheques are collected and batched, the one missing data item (cheque amount) is manually keyed in on a special device printing using the same ink. The cheque is run through a magnetic device to magnetise the ink. The cheques are then run through a cheque reader which separates the data into batches and this data is sent electronically to each clearing bank. Very large number of cheques can be processed quickly. The paper cheques are sometimes returned to the owner’s own bank but this is becoming less common.

MICR devices are NOT used for other applications because - the size of paper and data on it is limited, the data needs to be formatted in the E13B font, only 14 different characters are available, it requires the additional magnetising process AND it is probably very difficult to obtain such a device because of the banks have a monopoly on their uses.

Optical Character Recognition Device (OCR) – This is a device to read standard characters as printed by printers and enables what was an output document to be re-input. The text is scanned by laser light and the reflected light pattern converts into a binary value to identify the character. There two types of common uses:

- Turnaround documents** – An Utility company sends out printed bills to customers who pay by returning a cheque and the payment slip at the foot of the bill. Lining-up characters on the edge of the slip enable it to be read when inserted into an OCR

reader. The clerk handling the payment has no need to enter any data because the account number and value are both pre-printed. Only if the value of the cheque does NOT tally with the bill amount is there any need to use the keyboard for data entry. The data read can then be directed to the appropriate files to signal payment.

- b) **Text input** – Printed text documents can be re-entered into word processing systems. Whole or part Documents received by an organisation can be scanned into the computer. A binary bit map image is converted by OCR software by comparison with standards character patterns.

Bar Code Reader – Codes consisting of dark and white lines of varying thickness are scanned by laser light and the reflected light detected is converted into electrical signals and then to binary. The readers are capable of detecting the codes at any angle. A line-up mark is at the centre of the code and usually the number it represents is printed below. The reader normally beeps when a successful read has occurred. If the read fails, the operator can try again at a different angle or if the read continues to fail, keyboard entry is used. The failure rate is below 1%. Candidate answers often refer to this device as a scanner without being specific enough. Often the words “bar code” never appears. See SCANNER below.

USES:

1. **Supermarket checkout** - The sole purpose of the code is to identify the country of origin and the product code. Barcode reading has the advantage for the supermarket that:
 - a) items do not need to be individually priced saving manpower costs,
 - b) codes captured at source enable automatic stock control to take place identifying stock running out,
 - c) customer spending patterns can be determined,
 - d) staff do not need to know prices and hence faster throughput at the checkout benefits also the customer. Staff therefore need less training,
 - e) there are no errors at the checkout,
 - f) customer receipts are itemised in detail

Note: prices are NEVER included in the bar code because:

- a) the supermarket can set its own price. This is achieved by holding the price on the centrally held product file accessed by product code and is entered just once.
 - b) special offers enable the price to be changed ONCE on the central file
 - c) linked items can be offered at special prices – e.g. buy A and get B at half price.
 - d) products are international and a price in one country would be inappropriate anywhere else.
2. **Library** – Books issued can be automated. The Borrower has a single bar-coded ticket (instead of 1 for each book borrowed) and every book has a bar code. To issue books, the ticket is scanned to bring up the borrower’s record on the screen from the borrower file. Each book is then scanned and a record placed on the Issues file linking borrower code with book code. When books are returned, scanning each book enables the book to be cleared from the Issues file. The automation also enables late returns to be identified immediately and the fine calculated. Candidate answers often fail to mention that bar codes are read from the ticket AND from the book.
 3. **Hotel** – Guests are issued with a bar-coded card on arrival. At each purchase including meals, drinks, purchases in the hotel shop and services, the card is scanned to enable the account to be updated. At the end of the stay, the final invoice is printed, activated by the card, which the guest then pays.
 4. **Event tickets** – Tickets for sporting and other types of events can be issued and paid for in advance. This method enables quick entry to the event itself, identification of wrong entry into a particular part of the ground and detection of forged tickets.

Optical Mark Recognition Devices (OCR) – These are useful where a limited number of data answers are possible. Pencil marks are placed in pre-printed boxes and scanned by laser light – the lack of reflected light indicates a particular box has been selected. Changes are possible. Line-up marks are needed at the edge of the paper to ensure the reader reads the correct positions of marks.

USES:

1. **Lottery** – to select a limited number of numbers in a line.

2. **Multiple-choice examination papers** – to read answers selected by the candidate from a range of 4 or 5 possibilities. The reader only reads the selected answers and does NOT mark the papers. The correct answers are entered into a computer file BEFORE the answer sheets are read. Answer sheets are batched and the candidates' answers are compared with those on file – the results being recorded on the results file.
3. **Meter reading** – meter readers for utility companies visit individual houses, read the meter and mark the latest value into boxes against the house owner's details on a pre-printed sheet. When the reader returns to the company offices, the sheets are batched and read by an OMR device to enable charges to be calculated and invoices printed. Hand-held computers are now replacing this method where the reader keys in the actual meter numbers. These computers can be connected to a main computer and the actual keyed data transferred across more readily.
4. **Stock Audit/Checking** – Warehouse shelves from time to time need checking to ensure that recorded stock levels tally with the actual stock held. The stock clerk counts the number of items on the shelf for each product and against the stock numbers on pre-printed sheets, marks the number on the shelf. As for meter reading, hand-held devices are now replacing OMR methods.

Mouse – Is a hand held device for positioning the screen cursor. Three small balls underneath rotate as the mouse is moved over a flat surface and this movement turns small axles - their relative positions identifies the new position of the cursor. Various buttons can be pressed to indicate a selection has been made of the icon where the cursor is positioned or to provide a selection of options.

It is better to avoid describing the mouse as a DATA input device. Examination questions more often restrict the input to BUSINESS DATA, a fact often ignored by candidates.

An important point to note is that the mouse is NOT normally used as a data input device. It serves two main purposes:

- a) Selection of options
- b) Repositioning of the cursor perhaps in a word processing document or at a spreadsheet cell.
The mouse COULD be used for data input but only of a specialist type. Compared with the keyboard, it would be difficult to input a number or name.
- a) Drawing/Designing is effectively the entry of graphical data. To draw a rectangle, the user SELECTS the icon for a box but then marks where the box should be positioned, stretching it appropriately in horizontal and vertical directions. The box can be moved, turned as required.
Candidate answers need to be clear on this application where the mouse is used as data entry and where it is being used to select (which standard shape, line thickness, colour). Any answer to a question which requires an explanation of different DATA entry applications but which only describes the use of a mouse in selecting will score NO marks.
- b) It could be possible to "type" using a mouse if a standard keyboard was displayed on screen and each character selected in turn using the mouse. There seems no point in this method when a keyboard is always available. However, a number of music-composing programs do have a piano keyboard displayed on screen and notes can be selected. This method of music input would only be used for small-scale operations or perhaps for minor editing of music already entered by other means.

Voice Recognition – Audio sounds picked up by a microphone are converted to electrical signals and then to binary before being compared with standard patterns in memory. This method of input is limited and highly susceptible to errors. It could be useful in a factory or dirty environment to give limited commands such as controlling the movement of an unmanned trolley. Its restrictions are:

- May not recognise regional accents
- Variations in voice due to colds
- It must be calibrated first and often to particular individual voices

Word Processing – There is potential here although early efforts do not seem to have been continued. It would be beneficial in business if a letter/report could be dictated and automatically converted to a document

file in word processing format. It would then require a read through and minor editing to correct and improve the layout. The Spell-Checker and Grammar-Checker could perform the first stage of this.

Scanner - Laser light shone onto a picture/text receives the reflecting light and converts the signal to electrical and then a binary pattern held as a bit map in memory. It could then be saved en-bloc to a disc file.
The scanner resolution is measured in dots per inch (dpi). The device is ideal for archiving old documents. Picture scanning software could be used to repair minor faults (tears in the picture) as well as re-size, reposition, chop and colour. The use of OCR software could enable text to be edited within a word processing program.

OUTPUT

A range of output methods and devices should be considered. Candidates should be aware of the types of output possible:

Printed – on standard paper. A4 and A5 are the standard sizes although some printers will accept envelopes. The output can clearly be taken away and used in different places without the need for any equipment. Several people can study it at the same time. Some printers can handle continuous stationery driven on rollers and held by sprocket holes, which needs to be removed before sending out to business organisations.

Plotted – on large paper either as large sheets or part of a lengthy roll. The use is for large/complex designs/charts/diagrams.

Filmed – onto roll or microfiche. This is a compact method of archiving but needs a viewer to see it. It may take a long time to locate a particular item. It can be magnified.

Displayed – on a screen. This gives instant response and is ideal for checking/alterations but is lost when the machine is turned off or the next application is run. Suitable for only one person to view.

Sound – via speaker where internal signals are converted to speech. There could be many errors and the voice sounds artificial.

DEFINITIONS

pixel = dot on screen representing smallest possible display and identifying the colour

resolution = number of pixels available to display whole screen

scrolling = moving up and down the screen to reveal text/graphics previously off screen. It enables other material to be viewed without clearing the screen and losing the current contents

wordwrap = moving a word to the next line if it will not fit fully. This is an automatic process in word processing. A built-in hyphenation dictionary allows some words to be split over the lines with the use of hyphens.

bi-directional printing = printing alternate lines in opposite directions (dot matrix printer). This saves the time by not requiring the print head to move back to the start of every line. This is possible because the whole line of print is first held in a buffer and so the buffer can be read from the end instead of the beginning for the return line.

PRINTERS

Notes for Information Processing treats output devices in some detail. Candidates should be aware of the difference in operation of the following printers and the characteristics and advantages/disadvantages each possesses. The range of printers is now decreasing as early types are replaced by laser and ink-jets.

1. **Dot-matrix**
2. **Laser**
3. **Ink-jet**
4. **Barrel/Drum**
5. **Chain**

PLOTTERS

1. **Flatbed**

2. Drum

DATA STORAGE

This refers to magnetic or the use of laser light. Comprehensive notes are included with NOTES for INFORMATION PROCESSING.

PAST PROBLEMS

1. A major concern is that candidates frequently refer to a **file** when **record** is the appropriate. This is overlooked in some questions but NEVER in questions specifically related to filing systems. The correct terminology should be used in the teaching process and students corrected when they misuse technical terms. e.g. Candidates sometimes state that something “sorts out” some particular problem. The word SORT has a quite specific meaning relating to arranging data into a definite order. It does not mean, as it seems the candidate intends, “to put right”.
2. Candidates often have no real appreciation that there are many different types of files:
 - a) **Program** file – holds coded instructions and when loaded and activated, will run a particular application.
 - b) **Data** file – holds pure data usually highly structured. A customer order may have many fields, each with its own particular data types and usually of fixed length. e.g. Customer number, Product Code, Quantity ordered, Date Ordered Databases rely on fixed length records. If all records are 100 bytes long, then the 40th record will begin after 3900 bytes from the start. . A data file will have strict possible lengths. e.g. $n \times 100$ bytes using the example - n is the number of records on it of 100 bytes each.
 - c) **Parameter** file – holds defining details on how a particular application/device etc is to be interpreted. A file could hold codes for alternative LOCAL characters which can be printed by a printer in a particular country. e.g. £ used in UK for currency while other countries have a different symbol.
 - d) **Document** file – as used by other word processing programs. Here the data is continuous text and does not have the structure of data file described above. The concept of record does not have a meaning in this type of file. A document file can be any length unlike the data file. It is possible to re-save it after adding just ONE character.
3. Answers often refer to the need for the user to “save a file”. This certainly applies to word processing document files and other general purpose software but not to data files. If a payroll clerk enters details of a employee’s new address or additional hours worked as overtime, as soon as the details are keyed, they will automatically be recorded onto a file and only THIS employee’s details would be affected. This is very different from a word processing document where the user is advised to save the WHOLE file at perhaps ten-minute intervals to avoid possible loss of the words already keyed in. It may appear that the document is saved in exactly the same space as the previous version but in fact the computer creates a completely new file area and only releases the old area when the new file is complete.
e.g. A large document file which initially occupies more than half of a diskette CANNOT be re-saved onto that same diskette when additional material is keyed in because there is less than half a diskette left in which to save it. IF, and this is risky, while the new version is still in memory, the old file is deleted from the diskette, then the whole diskette is now available and the file CAN now be saved.

BUFFERS

A buffer is a sizeable piece of memory into which data is stored when it is read into an external source or when it is waiting to be output. The reason why buffers were devised in the earliest machines was to compensate for the big difference in speed between internal

operations and the slower data transfers to and from the outside. Speeds can be classified into three types:

1. Very Fast – internal operations – transfers from memory to registers.
2. Slower – transfers between memory and storage devices e.g. diskette to memory
3. Much slower – transfers between slow peripherals and memory – e.g. bar code reader to memory.

Clearly, each of these categories has its variations. It is far quicker transferring data into memory from CD than from diskette.

When a program issues a READ RECORD command, there are two types of buffers involved:

1. **System** Buffer – this holds one block of data transferred to/from the storage device. If the system buffer is 1Kb and records are 300 bytes long, then 3 records will be read at a time into the system buffer – a 4th record would require a buffer 1200 bytes long.
2. **Program** buffer – this holds just one record.

USE OF BUFFERS IN DATA TRANSFER

Consider, as above, a system buffer of 1000 bytes and a program buffer of 300 bytes.

READING

1. Program requests a read of 1 record.
System buffer – 1 block read = 3 records (A, B and C)
Program buffer – record A moved from system to program buffer. Record A is then processed
2. Program requests a second read.
System buffer – already 3 records there - no need to read another block.
Program buffer – move B from system buffer to program buffer
3. Program requests a third read.
As above, record C moved from system to program buffer
4. Program request a fourth read.
System buffer – all records in it are now used so read another block from file (records D, E and F)
Program buffer – D moved from system to program buffer.

This continues. In this example, the disc is only accessed every 3 program requests. The program is unconcerned HOW the records appear as long as a new record appears in the program buffer at each read request. Moving data internally from one buffer to another is much faster than reading a block into the system buffer. Reading a block of 3 data records in one block from disk is not noticeably slower than reading one record.

WRITING

In the reverse process, data is accumulated into the program buffer and transferred across to the system buffer one record at a time. NO write to the disc occurs until the system buffer is full. In this case, after 3 “write to disc” commands, the system buffer is still not full (100 bytes left). Consider the fourth write command:

System buffer – has 3 records (900 bytes). The fourth record would over-fill it. So, system buffer is written to the disc and then emptied.

Program buffer – 4th record moved across to the system buffer.

Unless there is a multiple of 3 records to be written to disc, there could be 1 or 2 records left in the system buffer when the processing ends. The program should issue a “Close File” command and the remaining partly filled system buffer is then written across to the disc and an **End of File marker** added.

DOUBLE BUFFERING

In very slow peripheral operations, two buffers might be used. When the program issues a “Read data” command, data is read into buffer-A. While this data is being processed, a second read is made into buffer-B automatically. When the second “Read Data” command is issued, the data is already in memory or at least on its way. The second command actually triggers a third read into buffer-A whose data has already been used. Double buffering effectively anticipates the reading of data and allows the slow transfer from slow peripheral to occur in the background.

In the reverse process such as printing lines, Buffer-A is filled and moved to the printer but the program can continue preparing for line 2 by filling Buffer-B. It cannot reuse Buffer-A until a



signal has been received from the printer than the data has been successfully printed. This is necessary should a fault occur on the line and the data has to be requested a second time.

PARITY

This is probably the most poorly answered topic across all modules. The reason is that candidates' answers are not precise enough. Questions always ask for actual numeric examples to help explanation but this does seem to make much difference. Candidates can often choose which form of parity is NORMAL (i.e. ODD or EVEN).

A common problem is that candidates confuse EVEN PARITY with EVEN NUMBERS and there is no connection. Similarly, they always seem to assume the error is in the parity bit itself – it could be in any of the data bits.

A **Parity check** is an automatic check, when data is transferred from one part of the system to another, to check that the transfer was satisfactory. i.e. disc to memory. Assume EVEN parity applies and consider the transfer of ONE BYTE. One byte holds 8 bits but ASCII code only needs 7 bits. The 8th bit is used in the parity check. The data to be sent is 1001001 and will be sent as P1001001 where P is the parity bit and will be 0 or 1.

In this case, counting up, there are 3 bits which are set to 1 (an odd number) so parity is set to 1 to make the number of 1's into an even number. i.e. 11001001 is sent.

NOTE this represents an ODD binary number and hence the comment above – parity has nothing to do with odd-ness or even-ness of the actual value of the number.

At the receiving end, the number of 1's is counted and if it is EVEN (in this system), the data is accepted. If the number of 1's is odd, then there was definitely a fault in the transmission and a request to re-send will be made.

In the very unlikely event that 2 bits transferred were wrong i.e. 11111001, this would still be accepted because there are 6 1's here and this is even.

Odd Parity behaves similarly with the parity bit set to 1 make the total number of 1's equal to an odd number.

The above notes refer to **Transverse** Parity. **Longitudinal** Parity is parity applied at right angles.

If more than one byte is sent, a parity check could be applied separately across every corresponding bit of each byte.

Suppose 4 bytes of data are sent using Even parity. Here the 5th byte is a longitudinal parity check. Suppose the following is received after transmission.

```

1000 1000
0101 0011
→ 1011 0000
0101 1111
0011 0000

```

Even parity is applied vertically. 1st column parity = 0 because there are already 2 1's.

^

It can be seen that in the third byte, the parity check fails because there are 3 1's. Similarly in the 6th column, there is only one 1 and so longitudinal parity fails here. The conclusion is that probably, the error in the data is in byte 3 column 6. If the 0 is switched to a 1, both parity checks would pass and effectively, the data could be corrected.

Candidates will not be expected to provide all this but only by covering it in class will they fully understand parity and its purpose.

LOGIC & BOOLEAN ALGEBRA

Definitions

Boolean Algebra – A form of algebra which handles logical statements and which either take the value TRUE or FALSE.

AND – Used to combine two logical statements (e.g. A AND B). The combination is only true if BOTH A and B are true. A decimal point is often used as an abbreviation. (e.g. A.B)

OR – Used to combine two logical statements (e.g. A OR B). The combination is true if EITHER or BOTH A and B are true. "+" is often used as an abbreviation. (e.g. A + B)

NOT – Used to reverse the state of a logical statement. NOT A will have the value FALSE if A is TRUE and vice-versa. "~" is often used as an abbreviation. (e.g. ~A). Alternatively, a line over a logical expression means apply a NOT to that expression (known as **A bar**). Note $\sim\sim A = A$. "~" is used in these notes to avoid possible errors if the "bar" moves away from its subject during transmission.

LAWS OF BOOLEAN ALGEBRA

The laws of Boolean Algebra apply in many situations where the equivalent law in normal algebra has exceptions (shown below). Apart from NOT, there are only two operators in Boolean algebra (AND and OR) whereas normal algebra has many (+, -, x...etc)

Commutative Law – states that the order of logical statements items can be reversed.

$$A \text{ AND } B = B \text{ AND } A \qquad A \text{ OR } B = B \text{ OR } A$$

This applies for some normal algebraic expressions but not for

- i. division. $X \div Y$ is not the same as $Y \div X$
- ii. subtraction $X - Y$ is not the same as $Y - X$

Associative Law – States that the order of executing 3 logical statements connected by the SAME operator does not matter. $A.(B.C) = (A.B).C$ $A + (B+C) = (A+B) + C$

In normal algebra this still applies separately for addition and multiplication

Distributive Law – Shows how to expand an expression with MIXED operators.

$$A.(B+C) = A.B + A.C \qquad A.(B+C) = A.B + A.C$$

This applies for normal algebra when "multiplying out" brackets but $X+(YxZ)$ is not $X+Y \times X+Z$

De Moivre's Law – Enables the inverse of two logical expressions to be determined by breaking up the single statement into two separate statements. The NOT outside the bracket switches the operator inside the bracket (AND/OR) and applies a NOT to each of the components.

$$\sim(A+B) = \sim A . \sim B \qquad \sim(A.B) = \sim A + \sim B$$

In normal arithmetic, NOT can be likened to MINUS but totally different results are obtained.

$$-(X+Y) = -X - Y \qquad -(XxY) = -XxY$$

DeMoivre's Law is particularly useful in some logical circumstances where it might be simpler to devise a situation which is NOT true and then to apply a NOT to that.

Identity – $A + \sim A = I$, the identity which is always true. This is equivalent to saying "it is raining or it is not raining"- a statement which is unnecessary. It is useful in simplifying expressions because, e.g. $(A + \sim A).B = B$

TRUTH TABLES

All the component statements are listed in the first columns of the table. If there are 2 components, there are 4 possible situations (00 01 10 11 - using the convention 1=TRUE and 0=FALSE). If there are 3 components, then there 2^3 possibilities etc. To avoid missing one of the possible combinations, always write them out in an increasing BINARY order. e.g. Prove $A.(B+C) + A.\sim B. \sim C = A$

*** NOTE- the following table MUST all be on the same page - AND OTHER TABLES AND DIAGRAMS BELOW ***

A	B	C	~B	~C	B+C	A.(B+C)	A. ~B.~C	LHS	RHS
0	0	0	1	1	0	0	0	0	0
0	0	1	1	0	1	0	0	0	0



0	1	0	0	1	1	0	0	0	0
0	1	1	0	0	1	0	0	0	0
1	0	0	1	1	0	0	1	1	1
1	0	1	1	0	1	1	0	1	1
1	1	0	0	1	1	1	0	1	1
1	1	1	0	0	1	1	0	1	1

Columns 1,2 and 3 hold the binary number patterns 0 (0000) to 7(111). Columns 4 and 5 are the inverse of columns 2 and 3. Column 6 determines the intermediate expression (B+C). Column 7 is the first term in the left hand side (LHS) of the expression which is combined (by an OR) with the second term found in column 8. Combining columns 7 and 8 produces column 9, the left-hand side of the expression. Normally, the RHS will be a more involved expression than the above and will require some columns of working before the RHS is determined.

Candidates should finally state that LHS = RHS and thus the expression is proved – this vital statement is often omitted. Candidates also miss out one or more horizontal sets of conditions. This is less likely to happen if, as advised, the combinations for component signals (A,B,C) are listed in binary ascending order.

KARNAUGH MAPS

This is particularly useful when 4 logical units are involved (A,B,C and D). Use a standard layout ALWAYS with the shape:

CD \ AB	00	01	11	10
00				
01		x		
11		x		
10				

The top line (and first column) convention is to label in the order as shown so that moving across (or down), only ONE of the components changes at each move - 00 to 01 to 11 to 10

The two boxes marked “x” represent the combined expression $\sim A.B.D$ because the AB column header is 01 (A=FALSE, B=TRUE). The 2 rows marked are 01 and 11 which means D must be TRUE. C does not feature in the expression and can be TRUE or FALSE.

Consider a problem where it is required to prove that 5 terms on the LHS = 2 terms on RHS. A good way to explain HOW the proof is reached is to number the terms 1 to 7 from left to right. Mark a “1” in any box where the first term is TRUE. Similarly place a “2” in boxes where term 2 is TRUE. At the end, the combined boxes marked 1 to 5 should be the same as the combined 6 and 7’s. This effectively shows terms 1 to 5 correspond to terms 6 and 7.

A logical expression with all 4 components (A,B,C,D) is identified by 1 box.

An expression with only 3 components needs 2 boxes ($\sim A.B.D$ example above)

An expression with 2 components needs 4 boxes.

An expression composed of just 1 component needs 8 boxes. (2 columns or 2 rows)

USING BOOLEAN ALGEBRA LAWS

Examination answers to logical proofs should show the transformation of the LHS of the expression, one step at a time, into the expression given by the RHS. The laws used should be quoted.

The problem above is solved here to show how an answer should be set out.

e.g. Prove $A.(B+C) + A.\sim B.\sim C = A$

LHS = $A.(B + C + \sim B.\sim C)$	Distributive law (in reverse)
= $A.(B + C + \sim(B+C))$	de Moivre’s Law (in reverse)
= $A.(I)$	Identity. $X + \sim X = I$ here $B+C = X$
= A	Identity
= RHS	Hence the condition is proved.

NAND and NOR

These are logical operators, not in the original Boolean algebra, but introduced for the simplification of electronic circuits. Any logical expression can be represented by all NAND or all NOR components.

NAND = NOT AND $\rightarrow \sim(A \cdot B)$

NOR = NOT OR $\rightarrow \sim(A + B)$

NAND (A,B) means perform a NAND where the two input signals are A and B and then reverse the result.

Note NAND (A) is effectively $\sim(A \cdot A) = \sim A$ and NOR(A) is effectively $\sim(A + A) = \sim A$

So, operating on a SINGLE element, both act as NOT operators.

By the use of de Moivre's Law, an expression can readily be changed to NAND or NOR only operators. The use of double NOTs may be needed in the working.

e.g. Change $A \cdot (B + \sim C)$ to NAND only components. Use the convention NAND(X,Y) meaning X and Y are inputs into a NAND component. To perform this conversion, since NAND contains the "." in its definition, then any "+" in the original expression should be retained while a "+" needs converting using de Moivre's Law. The stages are:

- i. $A \cdot (B + \sim C) = A \cdot \sim \sim (B + \sim C)$ Introduce $\sim \sim$ which leaves the bracket unchanged so that one of the NOTs can be used to create a NAND.
- ii. $= A \cdot \sim (\sim B \cdot \sim \sim C)$ Applying De Moivre's Law. $\sim \sim C = C$
- iii. $= A \cdot \sim (\sim B \cdot C)$ There are no "+" symbols left. Replace any $\sim(X \cdot Y)$ structure with NAND
- iv. $= (A \cdot \text{NAND}(\sim B, C))$ Now introduce a $\sim \sim$ again for the same reason as above
- v. $= \sim \sim (A \cdot \text{NAND}(\sim B, C))$ and use one of these for a NAND
- vi. $= \sim \text{NAND} [A, \text{NAND}(\sim B, C)]$ Finally replace solitary \sim 's by NAND
- vii. $= \text{NAND} (\text{NAND}(A, \text{NAND}(\text{NAND}(B), C))$

4 NAND logic units would be needed whereas the original had 3 components but all different (AND, OR, NOT)

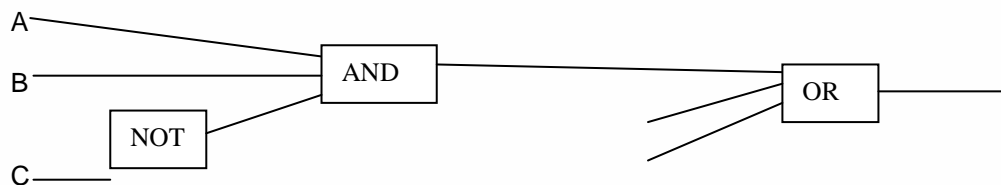
In an examination question and probably in a classroom, it will be simpler to use the Overline for NOT (mentioned above).

LOGIC CIRCUITS

This is best illustrated by an example. Devise a logic circuit to detect whether a 3-bit input has ONLY TWO of the bits set.

Draw a truth table for A, B and C showing the 8 possibilities from 000 to 111. From the table, it can be seen that an expression for 2 bits set is $A \cdot B \cdot \sim C + A \cdot \sim B \cdot C + \sim A \cdot B \cdot C$. In words, this means a "1" output should be produced if (A and B are set and C is not) OR (A and C are set and B is not) OR (B and C are set and A is not).

The full diagram is NOT given here but the first term is shown. Three similar circuit components would then feed into an OR to produce a single output which represents the whole logical expression.



Candidates may be asked to simplify the logical expression (by truth table, Karnaugh map or Boolean algebra). Without this simplification, the circuit could have a large number of unnecessary components.

Standard symbols are used for components as shown below. Candidates can either use these symbols or the rectangular boxes in the above diagram BUT in this latter case, the type of component MUST be written into the box. Unlabelled boxes will gain no marks.

*** SYMBOLS HERE for scanning in.

HALF-ADDER – This is a circuit to “ADD” two bits together and would form part of an adding circuit in the ALU. This circuit is called half adder because it only does half the job – it produces the bit value for the column of addition without taking into account any possible carry from a previous column. Possible outcomes are $0+0=0$ $0+1=1$ $1+0=1$ $1+1=0$ (with a carry)

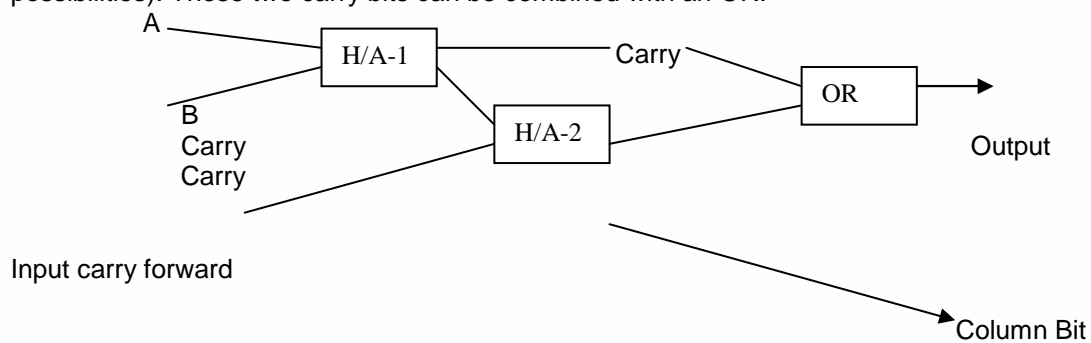
Column bit. It can be seen that the resulting bit is 0 if the two inputs are the same and 1 if they are different. A logical expression for this is $A \oplus B$

Carry forward. The carry to the next column is 1 only if both bits are set. $A \cdot B$

So a circuit can be drawn with two inputs (A and B) and two outputs (column and carry)

FULL-ADDER – A full adder takes in any previous carry. It can be constructed from TWO half adders.

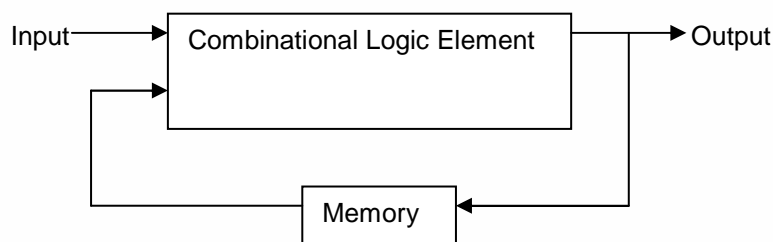
Half adder-1 has inputs A and B. The outputs are column bit and a carry to the NEXT column. Half adder-2 takes the column bit output from half adder-1 and combines it with the carry from the previous column. Since there are two additions, each half adder will generate a carry forward signal (but only one will be a 1 - students should check this by considering all possibilities). These two carry bits can be combined with an OR.



FLIP -FLOP CIRCUITS

The above circuit is merely to add two bits and deal with any carry in or out. For 16-bit addition, there would need to be an array of 16 full adders with outputs from one going in as inputs to the next. This is known as **combinational** circuitry. An alternative solution would be to use a single full adder for the first addition and then reuse it for subsequent bits, the output from the first being input in its second use. This clearly requires synchronisation with a time delay.

There are circuits where the outputs are dependent not only on the current input but also previous inputs. The output from the circuit is fed back in as a later input using a holding memory. This can be shown in the following diagram.



This is known as **sequential** circuitry of which **flip-flop** (also known as **bi-stable**) is a simple form. The output can remain in one of two stable states indefinitely despite the input changing because it remembers previous states. Some sequential circuits are **synchronous** because they rely on a clock to update their states at intervals. **Asynchronous** circuits do not use a clock regulator but are more difficult to design although they consume lower power.

USES:

- Storage during serial addition
- Shift registers where bits are transferred onwards
- Devices which require the output to “toggle” between two possible states

A full analysis of S-R and J-K flip-flops is not required.

OPERATING SYSTEMS

DEFINITIONS

Parallel ports – Connection channels using many wires with simultaneous transmission which are synchronised to give high speed data flow and extending the internal data buses. Used for printers.

Handshaking – A method for two devices to communicate where one checks that the other is ready (such as with serial printers). This is achieved either by hardware having separate wires for the signals or by software using control characters. Communications (WAN) MUST use software checks because there is no extra wire.

Interrupt – A signal sent to the processor that an event has occurred which requires the processor's attention.

A momentary halt allows an Input/Output operation to take place. Different priorities are allotted to different interrupts depending on their importance. A complete failure of the system requiring as much information to be saved as possible immediately is far more important than output of a line of print which could easily wait a fraction of a second. Tasks can always continue in the background and the processor only needs to be notified when that task is completed or more data is needed.

Device drivers – This is a software/parameters to provide the operating system with all it needs to access a specific devices such as the disc controller hardware, screen mappings or a particular type of printer. These drivers are activated when the system boots.

Protocol – This the standard for exchanging information between computers or devices.

Clipboard – A section of memory which holds recently marked data/text so that it can be moved or copied to another file/part of a document.

Icon – This is a graphic picture on the screen representing a data file, program or folder/directory.

It can be selected using a mouse in GUI for quick selection or activation. Icons can be moved around a desktop.

Desktop display – This is the screen “front page” representing a working area and holding icons for frequently used processes. Also it displays date/time and processes currently shelved.

Registry – This is a protected file area describing main system settings. e.g. which files relate to which

programs, shortcut definitions and information about OLE enabling information to be copied

between applications, personal preferences such as colours, logon details and plug-and-play features available. Only experienced technicians should make changes to the registry.

PDF – This is the Printer Definition File. When new printers are added, the computer needs to know the specific codes used by that printer when it is installed so that a user can use the full options available on the printer.

Multi-tasking – This is the ability of a computer to permit several processes to be running at the same time

in RAM usually with one in the foreground the others run without active control. The size of RAM limits this.

GUI – (Graphics User Interface) provides an alternative to commands by the use of icons and mouse selection. It reduces the need to learn commands and provides many ways of performing the same task leading to some confusion between different users. As a result it needs a lot of memory.

Pull-down menu – This is a list of items at top of screen which when selected with mouse or ALT+underlined letter a list of subsidiary options is displayed which can be selected by the mouse or keyboard.

OPERATING SYSTEM CONTROL

icon selection – using a mouse to click on a picture representing a required process/data file. It suits a

beginner/non-computer-literate because it guides him/her through each stage by providing on-screen help and reporting errors and how to overcome them.

menu selection – is list of numbered/lettered options from which the user can select by number/letter, first letter, cursor or mouse.

typed commands – require the user to know the exact requirements to instruct the computer program what to perform next. This was the basis of the DOS operating system. It is still preferred by expert users because it can shortcut the number of stages but GUI systems are taking away the opportunities to use commands.

e.g. To rename a file in a folder on a memory stick may require the following stages in GUI.

- a) Activate Windows Explorer (click icon)
- b) Change drive
- c) Select drive letter (H: ?)
- d) Select folder containing the file (folderX)
- e) Select file name (fileY)
- f) Bring up options (right click mouse)
- g) Select rename
- h) Change the name to new value (fileZ)
- i) Press ENTER

In DOS, a single command to the operating system would have been:

```
REN H:\folderX\fileY fileZ
```

The DOS method is only keyboard entry whilst GUI requires switching between mouse and keyboard.

However, make one mistake in keying the DOS command and an error will be reported. The command may have to be re-typed.

GUI CONSIDERED

Advantages

- Provides a common front for all applications
- Simplifies some operations like moving text/pictures
- Displays all options available (menus) where commands hide
- Has a recovery procedure (e.g. UNDO last operation)
- Allows separate windows for different applications or two views within the same (e.g. two documents)
- Copies office processes (e.g. recycle bin)
- Provides on-screen help – reduces the need for consulting manuals
- Suits beginners who do not have to memorise or understand commands
- Links can be made to other software

Disadvantages

- Some old functions lost from the DOS system
- Can be more time-consuming moving through menus
- Programs tend to be much bigger – overheads. e.g. A one-sentence document can take up 1Kb memory.
- Users do not understand what they actually doing
- Software providers are dictating future policy of how software should operate

GENERAL OPERATING SYSTEM FEATURES (across all computers)

Multi-programming – Many programs apparently running in parallel. This makes better use of CPU time because it allows a new application to run when another is temporarily stopped. In practice, only ONE program runs at any given time but for a given small time slot. When an interrupt alters the situation, the scheduler switches to another program with the highest priority. This maximises the use of peripherals. Details of the state of each last mini-run (e.g. Program counter) are saved.

Automatic job scheduling – All jobs entered into a multiprogramming environment are given a priority. The operating system picks which job to run next dependent on priorities or whether a job is runnable (see later).

Spooling – To offset differing speeds of devices/slowness of printers and allow for queuing in a network environment, all print output is directed to a disc file. When the printer becomes available, the file is output to the printer as a background job in a continuous run. High priority print jobs can override the normal order priority.

- Utilities** – These are additional programs supplied with the O/S to perform common tasks e.g. SORT a file. Often the user cannot distinguish between utility programs and OS commands. New utility programs can be added as required.
- Disc House Keeping** – These are routines enabling the user to manage files and include copying, renaming, deleting, moving, sorting. Make a folder/directory, format a disc, inspect disc usage are other features.
- Memory management** - Memory is allocated to user program. The boundaries are strictly controlled so that no program can interfere with another program's memory space.
- Peripheral control** – Input and output is controlled and any problems reported. Violation between different programs is prevented. e.g. two programs cannot both be sending data to a printer thus mixing up the two outputs.
- Communication with operator/user** – All communications with the operator/user need to be interpreted and appropriate action taken. Any problems that arise are reported back to the operator usually on the screen. It is essential that the operator knows precisely when a job is complete – it would be inappropriate to remove a disc while there was still output due to go to it.
- Accounting** – Work done is logged. In a business environment, it is essential that there is an audit of what actually happened during a particular session – an operator could not run a program of his own to perform a fraudulent task. In some situations, time and storage space used may need to be charged to particular departments/users.
- Booting** – This is the start-up process where hardware is checked, user preferences are loaded and the basic operating system is loaded into RAM
- Virtual memory** – This is the use of disc to provide additional memory. Memory is **paged** into fixed sized blocks and program sections are **swapped** in and out, as they are needed.
- Security** – Some security is provided with different access levels for different users by the use of passwords. The system will also remind users to change their passwords at regular intervals and prevent easy-to-guess passwords from being used.

MULTI-USER ENVIRONMENTS

Locking. In a multi-user environment there is a risk that two users could try to access the same record at the same time. e.g. Suppose that two on-line queries are being handled by different sales staff. Customer-1 wants 5 of product-A and customer-2 wants 8. There are 12 on the shelves so only one of the orders could be satisfied. However, if both staff requests "look" at the stock record at the same time and see that 12 are available, they could both accept the order despite the fact that both orders cannot be satisfied.

A solution is that before any data is used, the first user's access (and it may only be nano-seconds earlier) automatically **locks** the record. Any subsequent user must wait until the lock is released. Above, customer-1 may get the 5 items wanted reducing the stock level to 7 and then unlocks the record. A fraction of a second later, Customer-2 then is granted access to reveal insufficient stock available.

Similarly **deadlock** could occur if two users (A and B) but access different records (X and Y) but then both need access to the other. The records X and Y will both be locked preventing both users progressing – both are waiting for each other. The solution is to release all locks but then queue the two users allowing one to progress (re-locking records as appropriate) while the other waits. The system decides which user is given the priority of going first.

Database management systems can check for circular deadlocks using a matrix and releasing one of the requests which must then be informed of the failure.

Scheduling - Aims

- To maximise processor use
- To maximise peripheral use
- Produce faster responses
- To aim for maximum throughput
- To cater for uneven loading
- To give users get equal/fair share

- To provide a reasonable turn-around for background/batch work
- To avoid deadlock

Job Status

A job can be in one of three states:

Runnable = The job could use the processor but it is not free to do so.

Running = The job is using the processor at this moment.

Suspended = The job is unable to use the processor perhaps because it is waiting for an input/output operation to finish

What Decides the Priority Of A Job?

- The priority allocated to the job externally by users. Some jobs could run in the background and therefore be given low priority.
- The memory the job requires. A high memory job might never gain access because other smaller jobs would take priority.
- The peripheral requirements
- The estimated run time
- Previous waiting time – priority rises with waiting
- Resources already used

SCHEDULERS

High Level Scheduler – This controls the initial job entry – All jobs are placed in a queue on disc until entry.

If there are insufficient resources, the job cannot be accepted yet.

A process might decide the order of entry (FCFS=First-Come-First-Served)

Short/low resource jobs first could be given priority to increase throughput and get them out of the system.

Middle Level Scheduler - If the processor workload peaks, a job in memory not running is swapped out and reloaded later.

Low Level Scheduler - This deals with fractions of a second. Its aim is to identify which job is to run next after an interrupt or a time slot expires. It decides the next job using one of the scheduling processes:

Round Robin - Each job is given a time small **time slice**. Jobs are run in turn for a small fraction of a second provided they are runnable. In a networked system, this process is used by a multiplexor to check each input/output channel and to decide which message to process next.

FCFS – This is appropriate provided resources are available for the job selected.