MYSTERY OF COMPUTERS?

Computers are a bit of a mystery to most people. It not like trying to understand how the car engine works, as most things about inner workings of the computer are “invisible”.

Our need for computers has a long history. Ancient civilizations too invented special purpose computing devices from time to time, to predict the motion of stars and for navigation. These included both **“analogue”** and **“digital”** devices, dedicated to just one job that they often did quite well. However, the **concept of general-purpose “programmable” computers is entirely new.**

If you look at the books in your local library, you are left just as mystified. They talk about logic gates, binary numbers, CPU, memory, Windows, Unix etc., but fall short of explaining how it all works. If you were given whatever you ask for (e.g. infinite supply of logic gates, microprocessors, keyboards, screen etc), could you put it all together to make a computer? **Just as a collection of body parts do not make a living creature, a computer is more than the sum of its parts.** Software may be the missing "soul", but that too is not enough**. It's the "organisation" that matters the most.** As you will soon see, the computer is a marvel of organisation.

Alan Turing (1912-1954) first proposed the concept when trying to break the Enigma Code.

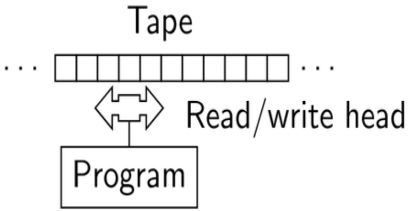


Fig 1 – The Turing Machine

Turing's concept, Fig.1, is a theoretical device to manipulate symbols on a strip of tape according to a table of rules. In modern programmable computers, the tape is replaced by **Memory** (RAM and the Hard Disk), table of rules by the **ALU** (Arithmetic Logic Unit) and the **Instruction Set**, and orderly feeding of the tape by the **Programme Counter, Address Bus** and other hardware.

To top it all, **computers understand only 0s and 1s –** fast they may be, but morons nevertheless. **Well organised morons!**

Fig 2 shows the general arrangement of various parts of a computer. When you turn on the computer, what happens? The average person does not care too much about it, as long as the Windows logo and the Desktop appears soon and a click on the Desktop loads and runs his favourite programme.

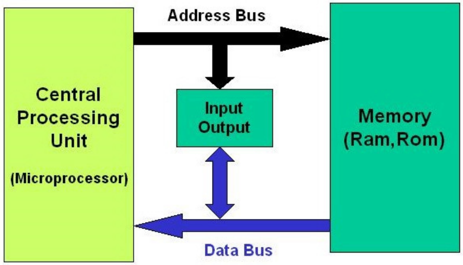


Fig. 2 – The Schematic for a modern computer.

Generally speaking, everything hangs between the **Address Bus** and **Data Bus**. Two main thoroughfares, which carry all information.

As stated earlier, computers are actually a **marvel of organisation**. Modern operating systems, like Windows, Unix, Apple OS and Android **are too large to fit the available memory and are stored**

usually on the hard disk. By itself, the computer hardware can't perform complex tasks like reading the operating system from the hard disk, and then run your favourite programme. All computers have a small, important, programme stored in a **ROM BIOS** chip – the **Read Only Memory Basic Input Output System**.

The **BIOS "chip"** is an integrated circuit, ***sometimes with a transparent window through which you can actually see the microchip***. Most **BIOS** are actually very complex, but all have a **Loader** programme, do a quick computer check up, look for the graphics and sound cards, screen, mouse, printer and keyboard. The **Loader** programme then reads the **"Master Boot Record"** on the hard disk to discover the type of hard disk and how it has been structured.

COMPUTERS UNDERSTAND 10 THINGS?

Yes, computers understand only two (binary 10) things, 0 and 1. How can a **block of 0s and 1s in the BIOS, Hard Disk, Memory and elsewhere do anything at all?**

All digital computers basically work in a similar manner. Microprocessors have a built-in **Instruction Set.** Intel produced the first most successful microprocessors – staring from the **4-bit 4004**, **8-bit 8080** (1973), followed by more complex Pentiums and later **x86** versions in use today. Intel 8080 is still **"source code compatible"** with modern x86 family of microprocessors used in PCs. Intel microprocessors have **"Complex Instruction Set (CISC)"** as opposed to **"Reduced Instruction Set (RISC)"** microprocessors widely used in mobile phones and tablet computers. We will not discuss the merits of **CISC** vs. **RISC** here, however they all work in a similar manner and ***everything hangs between the Address Bus and Data Bus as mentioned earlier.***

To understand it, we will now look at a small programme, written in **"Machine Code"** (i.e 0s and 1s) for the **Intel 8080 microprocessor.**

Memory	Code	Hex	Mne
0000	1101 1011	DB	IN
0001	0000 0111	07	
0002	0011 0010	32	STA
0003	1111 0000	F0	
0004	0000 0000	01	
0005	0111 0110	76	HLT

Fig 3 - Block of 0s and 1s are organised in groups of 8 **"bits",** called a **"byte",** and use the **"hexadecimal"** notation. The computer performs all operations in a **"Step-by-Step"** fashion. Each step is called a **"Machine Cycle",**

controlled by accurate internal clock(s) that tick away at millions of times every second.

At the start, the **Address Bus** points to the first memory location 0000 (in this case). During the first machine cycle, the computer reads the **first row** (**1101 1011**) from memory location 0000. Whatever it finds there is **regarded as an instruction,** called the **Op Code**. This group of 0s and 1s translates as the **"Input Read" or IN** instruction. The computer then asks for **Input Port number** – input ports are usually configured as memory locations. It finds this information by reading the next memory location (0002), reads **0000 0111** (hex 07) as the **Input Port Number.** It has taken two machine cycles so far. In the next cycle, it sends **0000 0111** to the **Address Bus** as the **Input Port a**ddress and **"enables**" it for a short time to **"read"** the data. The **Input Port** could be a block of switches, output of an **analogue-to-digital** converter or something else. The data then appears on the **Data Bus** and goes straight into the (first) **Accumulator**. The computer so far has **read just two memory locations** and has taken **three Machine Cycles**. The actual sequence of operations, of course, depends on the microprocessor used. Some microprocessors have several accumulators, wider **Address** and **Data Bus, Index Pointers and Multi-phase Clocks** requiring more machine cycles to execute instructions.

Having completed the first instruction, the **Address Bus** then points to the memory location **0003** to read the next **Op Code**, and so on. The **Op Code** in memory location **0003** is **0011 0010** (hex 32), which translates to **"Store Accumulator to Memory (STA)".** As this microprocessor uses a 16-bit Address Bus, it reads the next two bytes in memory locations **0004** and **0005** as **1111 0000** (hex F0) and **0000 0001** (hex 01). These combine to form the 16-bit memory address **1111 0000 0000 0001** (hex F001). To store the data in this memory location, the computer then sends **F001** (hex) to the **Address Bus** in the next memory cycle and applies a momentary **"Write Memory"** control signal to memory to write the data at the selected address (F001). This instruction has taken a total of four machine cycles, three to read and one more to execute. ***The contents of the Accumulator are not destroyed in the Store Accumulator to Memory (STA) instruction*** and can still be used for other purposes. It then reverts to reading the memory, at location **0005** for the next instruction, **0111 0110** (hex 76). This is the **HLT** (Halt) instruction and it halts as soon as the instruction is read (one machine cycle). So, the 8080 microprocessor requires 1 to 4 machine cycles in this example, however there are more complex instructions. Here is the list of **step-by-step machine cycles** to run the programme:

Cyl No.	Data Bus	Add Bus	Description
1	1101 1101	0000 0000	Read **Op Code** (hex DB IN)
2	0000 0111	0000 0001	Read **Input Port No** (hex 07)
3	Input Data	0000 0111	**Input Read**, data on Port 7
4	0010 0010	0000 0010	Read **Op Code** (hex 32 STA)
5	1111 0001	0000 0011	**Hi Memory Add** (hex F1)
6	0000 0000	0000 0010	**Lo Memory Add** (hex 01)
7	Data read	F1 00 (hex)	**Write data to memory**
8	0111 0110	0000 0101	Read **OP Code** Execute **HLT**

The number of machine cycles required to read and perform the instruction depends on the operation required.

IT'S ALL TOO COMPLICATED?

If you want to understand how computers **"really"** work, it is best to start with 8-bit microprocessors. You can easily **"breadboard"** or ***build an 8-bit microcomputer yourself*** using the **"asynchronous" Intel 8080** (8085 or Z80) microprocessor. Alternatively, use ready-to-use trainers such as the **Microtutor MPT8080K-1** for learning.

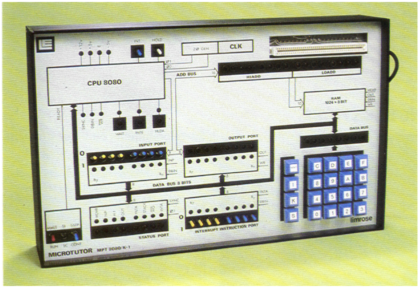


Fig.4 – Limrose Microtutor MPT8080.- It has numerous LEDs on the front panel to show what is happening on the Data and Address bus, Input and Output Ports and control signals.

You can **“see”** the operations described on the proceeding pages, using **LEDs** on **Address, Data and Control** buses if the microprocessor is run in the **“Single Cycle mode”.**

You can also **“simulate”** the operation of a simple computer on a **Windows PC** (WinXP and later) and observe **“Step-by-Step”** operation. The ***“beta”*** version of the **Limrose Small Computer** simulator **(LSC)** is due to be released in Jan 2015 and a full version a bit later. None of this is as complicated as it sounds and can be great fun. **You will actually learn a lot more about computers if you build your own on a breadboard.**

LSC Simulator can be downloaded from www.limrosegroup.com (Jan 2015 onwards) or contact Limrose for a free copy on a CD.

SO, WHERE DO WE GO FROM HERE?

As computers understand only 0s and 1s, the very **first programmes must be written using just 0s and 1s**. No **Python** or **Visual Basic** as yet. Programming using the machine code, at the basic level of 0s and 1s, is not for everyone but **is a must if you wish to become a computer designer of tomorrow?**

Although all microprocessors have the necessary logic, they require some software to kick start it. The **BIOS** chip does this job for PCs. The **BIOS** (or **micro-programme)** is automatically run when the computer is powered. Some support chips used with microprocessors too are **"programmable" a**nd the first few lines in **BIOS** are often used to programme such chips (e.g. USART or USB chip), before loading the remaining system.

ASSEMBLER AND THE “C” LANGUAGE

All microprocessors offer an **“Assembler”** and the **“C”** programming language. **Assembler** is the closest you can get to **“machine code”** programming. Thus, the **Assembler** code for the example in Fig.3 would look like:

```
IN 07
STA F1, 01
HLT
```

Assembler is a lot easier to use. The next language, **“C”,** comes in various broadly similar versions and **runs on most operating systems.** **However, C too is difficult for beginners**.

PYTHON – CODING FOR ALL?

Python is a good first programming language. It can be a great experience on Day 1. As Python is an interpreter, you can type:

>>> print (“Hello World”) and press the **Enter** key.
You will immediately print
Hello World on the printer connected to your computer**.**
>>>

Inexpensive computers like **Raspberry Pi (**which runs on Linux, and comes pre-installed with Python) have re-vitalised **Python**. **Django is a rapid development** web framework for **Python** professionals. **Google Maps** and **Windows Live** are probably written in **Python**.

But, coding for all using Python is probably a bridge too far?

JAVA – REAL WORLD PROGRAMMING

We are now entering the complex world of **“real”** programming and you will come across concepts such as **Object Oriented, Class and Inheritance.** Here is the **Java** programme to print **“Hello World”:**

```
/* comment - Hello World Applet in Java
public class HelloWorld {
            public static void main (string [] args) {
                      system.out.println (“Hello World”);
            }
}
```

More complicated than Python. Note the use of brackets, lower case coding and semi-colon in Java. Java is more difficult to learn and reputed to be embarrassingly slow!

PROGRAMMING LANGUAGE FOR YOU?

Actually, there are **too many programming languages. "C"** is practically a **"universal"** programming language and **runs on PCs, Unix, Android, Apple, Mobile Phones and Tablets**. But, you have to grapple with **memory management, internal pointers and so many other things yourself. Can you cope with all that?**

Java too is a very important language (coding looks similar to C), and is the **main language for the Android** operating system. It is perhaps appropriate to mention at this stage that **Java** and **JavaScript** are two different things – **Java** is a fully developed programming language to create **"stand alone"** applications. **JavaScript** is used within web browser's **HTML code**, interpreted line by line, and cannot produce stand-alone applications.

Beginners should avoid C and Java – too difficult to master.

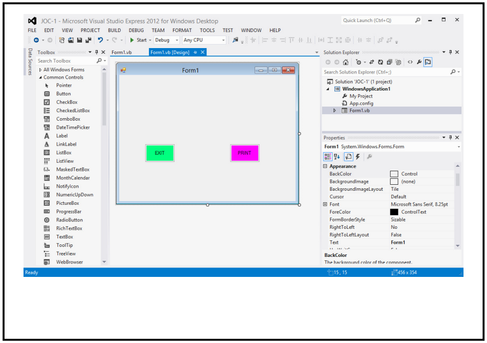
Another programming language, created by two former Apple engineers especially for kids, called **SIMPLE** for Windows, is worth a look. You can download it **free** from www.simplecodeworks.com. **SIMPLE** is easy to learn as it uses only 23 keywords, but you can still write some interesting games for Windows and the Internet.

However, if you are serious about programming, sooner or later you will learn a **"professional"** language like **Visual Basic, Ruby, HTML, PHP or Java**. **TIOBE** index for programming languages (www.tiobe.com) lists **Visual Basic** (VB6 and VB.Net) quite high up. Some say that ***VB is more than just a programming language***; it includes the **complete architecture for integrating programming with the Windows Operating System**. Visual Basic is probably still the language loved by the largest block of developers in the world.

VISUAL BASIC (Visual Studio)

Visual Basic (part of Microsoft's Visual Studio) is **both simple and complex at the same time**, which explains its meteoric rise as one of the most popular programming language. It is simple to get started, yet is comprehensive enough to handle large projects and the Web. **Visual Basic Express 2012** (free download) has everything you need for **Windows Desktop GUI Applications**. However, if your computer is old (WinXP etc), you may have to use **VB Express 2008, VB Express 2005** or even older **VB6. VB6** still has a large installed base and millions of devoted followers.

The example that follows uses **VB Express 2012,** and other versions may work a bit differently. Start **VB** by **clicking the Desktop icon**. Then, **New Project** and **Windows Form Application**. After starting the project, you should see a (blank) **Form**, Fig.5.



Form1 System.Windows.Forms.Form

Appearance	
BackColor	Control
BackgroundImage	(none)
BackgroundImageLayout	Tile
Cursor	Default
Font	Microsoft Sans Serif, 8.25pt
ForeColor	ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	Form1

BackColor
The background color of the component.

Fig 5 - Visual Basic, Development Environment (IDE). To view the **Toolbox**, click the **VIEW** menu bar and select **Toolbox**. You can "dock" *the toolbox to left of Form1.*

Visual Basic is **"Event Driven". Events** usually occur by, but not limited to, user inputs – such as **clicking a button.** The blank opening screen has to be first loaded with **"Controls",** such as **buttons** and **textboxes** using the **Toolbox** menu. Then, insert the necessary code for their **"click"** events. To add a **Button**, ***drag the Button control from the toolbox onto the Form and draw it on the***

Form. Relocate and size to suit. In the **Properties** window on the right, alter the **(Name)** of the button as **cmdPRINT** and **Text** as **PRINT.** Similarly, add the **EXIT** button and **(Name)** it is as **cmdEXIT.** To add the code, click anywhere on the **Form** and go to **Form1.Events** window (ignoring code for Form1_Load event) :

```
Public Class Form1
          .........
          Private Sub cmdPRINT_Click( )
                    Messagebox.show ( “Hello World”)
          End Sub

          Private Sub cmdEXIT_Click( )
                    END
          End Sub
End Class
```

Add code for **cmdPRINT** and **cmdEXIT** buttons between **Private Sub** and **End Sub** (as shown in **blue** above) to print **Hello World** in a **MessageBox.** Sending it to a printer requires a bit more coding.

To run the programme, press **Function Key [F5].** Every time you **click** the **PRINT** button, the computer will print **Hello World** in the **MessageBox**. Clicking on the **EXIT** button would stop the programme. Excellent tutorials are available on the Internet for all versions of **Visual Basic** and other programming languages.

Some **VB-based programmes can be very large indeed.** Limrose's **VProgen4GL/Agile System** has over **400,000 lines of code**. It automatically **"generates"** robust solutions for **business applications** (Manufacturing, Libraries, Care Homes etc.) ***without writing any application code*. The same "VProgen4GL engine" manages all applications without any coding**. **No coding!!**

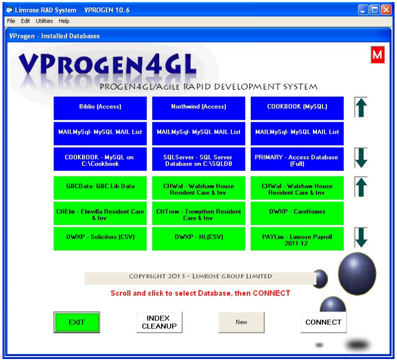

Limrose RAD System VPROGEN 10.6
File Edit Utilities Help
VProgen - Installed Databases
VPROGEN4GL
PROGEN4GL/AGILE RAPID DEVELOPMENT SYSTEM
Biblio (Access)
Northwind (Access)
COOKBOOK (MySQL)
MAILMySql- MySQL MAIL List
MAILMySql- MySQL MAIL List
MAILMySql- MySQL MAIL List
COOKBOOK - MySQL on C:\Cookbook
SQLServer - SQL Server Database on C:\SQLDB
PRIMARY - Access Database (Full)
GBCData- GBC Lib Data
CHWal - Walshaw House Resident Care & Inv
CHWal - Walshaw House Resident Care & Inv
CHElm - Elmvilla Resident Care & Inv
CHTrew - Trewythen Resident Care & Inv
DWXP - CareHomes
DWXP - Solicitors (CSV)
DWXP - HL(CSV)
PAYLim - Limrose Payroll 2011-12
COPYRIGHT 2013 - LIMROSE GROUP LIMITED
Scroll and click to select Database, then CONNECT
EXIT
INDEX CLEANUP
New
CONNECT

Fig. 6 – Opening screen for **Vprogen4GL**, showing multiple objects and buttons, textboxes etc. The “arrows” too are “clickable” objects. The entire code, with minor exceptions, has been written in Visual Basic.

Can’t imagine writing 400,000 lines of Desktop code in Python, or Java? Contact Limrose (Tel. 01978 855555 or limrose@aol.com) for a free copy of **Trial CD for VProgen4GL** to see how you can develop complex business applications **without writing code at all.**