In preparation for a demonstration of programming the Apple 1 computer at the 28 March iMug meeting you may wish to acquaint yourself with some of the very basic stuff that computers consist of. There will be a number of these basic tutorials over the next few weeks.

Lesson 1

ALL Computer Data is a ONE or a ZERO.

At the end of this lesson you should be able to recall that all computer data is represented by a series of 1s and 0s.

Rule 1: The only data inside a computer's chips are zeros and ones.

Rule 2: Refer to rule 1.

Rule 3: Rule 1 is not strictly true. What actually is inside the chips is two different voltage levels (typically +5 volts and 0 volts) that we can tag as 1s and 0s. (+5 volts = 1 and 0 volts = 0).

Now that we can represent 1s and 0s, we need a number system to make computers useful. The binary number system is ideal as it only uses 1s and 0s.

Back to rule 1. Refresher course! There are no colours, no circles or squares, no musical notes and certainly no intelligence. Just 1s and 0s.

Memory is not intelligence. A light switch on the wall always remembers in which position you put it and it stays there … off or on = 0 or 1. Is this intelligence?

Binary numbers

The numbering system, which we commonly use and inherently understand, is one of thousands of numbering systems and is known as the decimal system. The decimal number system is a number system based on powers of 10. And we are dealing with whole numbers, not decimal points. What follows relates the decimal numbering system to another numbering system called the binary numbering system.

binary 0 = decimal 0

binary 1 = decimal 1

binary 10 = decimal 2

binary 11 = decimal 3

binary 100 = decimal 4

binary 101 = decimal 5

Why is this so?

In the decimal number system we have positional values of:
1 10 100 1000 10000, etc. or
100 101 102, 103 104 etc.

Note that 100 is a 1 with no zeros.

This is known as base ten or radix ten.

Thus a decimal number such as
9,721 = 9 x 1000 + 7 x 100 + 2 x 10 + 1 x 1

In the binary number system the positional values are
1 2 4 8 16 32 64, etc or
20 21 22 23 24 25 26 …

This is base two.

So a binary number such as 101101 = 32+0+8+4+0 +1 = 45 decimal.

Best you check out my solution!

Whole Number Conversion Table
(Do not memorise this as your brain has better things to do.)

Binary	**Decimal**
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Test yourself with the following:

Binary to decimal:

1 0000 =
1 0001 =
1 1111 =

Now try decimal to binary:

16 =
17 =
27 =
32 =
33 =
64 =
100 =

Easy as !

Just for fun, here is a number of other numbering systems!

Please note that there are zillions of possible number systems. You may remember the old weight system of tons, hundred weight, quarters, pounds and ounces:

16 ozs = 1 lb, base 16

28 lb = 1 qtr, base 28

4 qtrs = 1 cwt, base 4

20 cwt = 1 ton, base 20

Don't forget our old money system.

12 pence in a shilling, base 12
20 shillings in a pound, base 20
21 shillings in a guinea, base 21.

And of course don't forget about base 12, aka a dozen,
and a score = 20.

Summary:

1. There are many number systems.
2. The number system that we all know and understand is the decimal number system.
3. The decimal number system is based on position values of 10.
4. Because computer data consists exclusively of 1s and 0s, we need a numbering system that uses exclusively 1s and 0s.
5. The binary number system is such a system.
6. Decimal numbers can be converted to binary numbers.
7. Binary numbers can be converted to decimal numbers .

BUT! What you *have* learnt in this lesson - ALL Computer Data is a ONE or a ZERO.

More lessons soon!

Lesson 2

Entering and Displaying Memory Data, the Hexadecimal Number System and ASCII code.

At completion of this lesson you will understand the use of HEX and ASCII coding. There is no need to memorise these codes.
All the data in a computer is a 1 or a 0. When this data is displayed on your computer screen it will look like:

00010111010101010100101010101010101
01010101010101010101010101010101010
11101010101010111101010100010010101
00100011111001010101010101010101010

This is very hard to read and to understand though with practice it is quite easy.

A better way is to encode groups of 1s and 0s into a more readable code is the hexidecimal number system.

The Hexadecimal Number System.

This number system uses base 16. The sequence of numbers is:

0 1 2 3 4 5 6 7 8 9 A B C D E F,

where A =10, B = 11, etc. Just remember that the numerals 1, 2, 3 etc are just shapes that represent numbers and now we use A, B, C etc for 10, 11, 12 etc because they exist on the keyboard.

Table of equivalent values:

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C

13	1101	D
14	1110	E
15	1111	F

When entering and displaying computer data we simply enter each memory byte of 8 binary bits as two HEX digits, such as:

00 04 F4 DD EE 14 D7 87
FE 55 49 8B BB CB BF A5
AA BB CD B6 56 12 34 F5

The hex 00 above is stored in memory as the binary number 0000 0000, etc. This conversion is done by a piece of code installed in the computer.

The Apple 1 uses HEX code for input and output.

ASCII Code (American Standard Code for Information Interchange)

This is the code sent to the computer from the keyboard.

ASCII code is a 7 bit code originally used in teletypes.

ASCII code is an example of encoding data in binary bits that is not a numerical number system like binary and hex numbers.

ASCII can represent the 26 upper case and lower case characters, numbers 0 - 9 and other characters such as: ? / > &) $ # etc.

Key	**ASCII in Binary**	**ASCII in HEX**
0	0011 0000	30
1	0011 0001	31
3	0011 0011	33
4	0011 0100	34
etc		
A	0100 0001	41
B	0100 0010	42
C	0100 0011	43

NOTE: it is the binary code that is stored in memory.

Here endeth Lesson 2.

This is not the last lesson!

Lesson 3

The Basic Computer Cycle of Operation

The digital computer consists of three blocks as shown in the diagram below. The Central Processing Unit, CPU, FETCHES an instruction from memory and then EXECUTES the instruction and then fetches the next sequential instruction. This cycle is repeated forever.

A continuous cycle of FETCH – EXECUTE – FETCH – EXECUTE –

The CPU is also known as a microprocessor and some common ones are: 68000, Pentium, PowerPC and many others.

Memory stores both the code being executed and the data used by the code. Memory consists of byte sized locations (8 bitd) which are numbered or addressed from 0 to 1,000,000s and is usually done in Hex code. e.g., $12FE where the $ sign indicates Hex number system

Memory is of two types: Volatile Random Access Memory, RAM, which losses its data when the power goes off and Read Only Memory, ROM, which retains its data when the power is off. ROM holds software to read the keyboard, mouse and drive the screen and can include code to do graphics etc as it did in the older Macs.

When an application like TextEdit is double clicked to run, it is copied from the hard disk into RAM for execution. The document associated with TextEdit will be created in RAM or copied from the hard disk into RAM. When you save your document it is saved as a file on the hard disk.

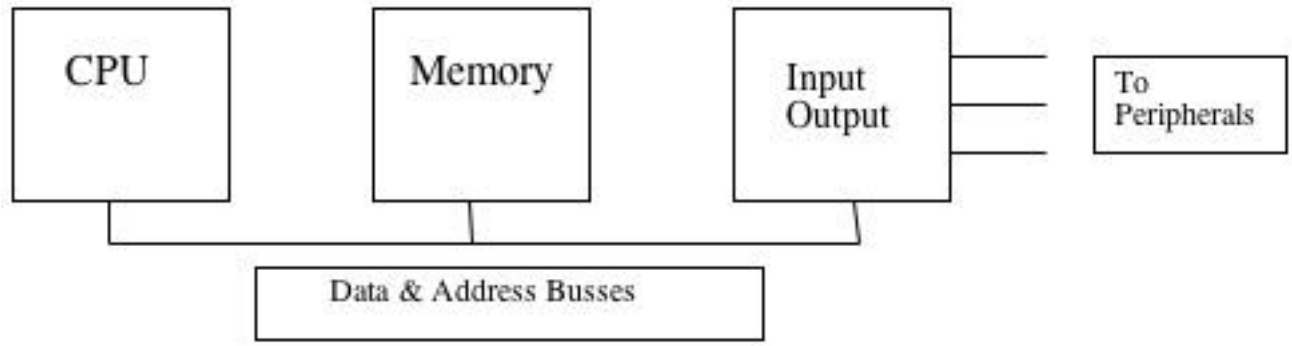
Communication with the outside world of peripherals is via the INPUT – OUTPUT block.

Peripherals consist of: keyboards, disks, screens, mice, printers, modems, etc, etc.

The hard disk is just a mass storage medium for all your applications and data files.

Instructions and data are moved between the blocks via the data, address and control busses. Data and address busses are these days 64 bits each.

Computer Block Diagram



The computer diagram is the same as it was in the 1930s, 1976 and 2006.

Lesson 4

The Control and Processing Unit — CPU

A very basic block diagram is shown below.

The Data and Address busses are connected to the computer's memory and input/output blocks. The busses are used to pass address information to the memory and the input/output blocks and data bus is used for transferring data between each of the blocks.

The Arithmetic and Logic Unit, ALU performs the mathematical and logic operations such as ADD, Subtract, Multiply, etc.

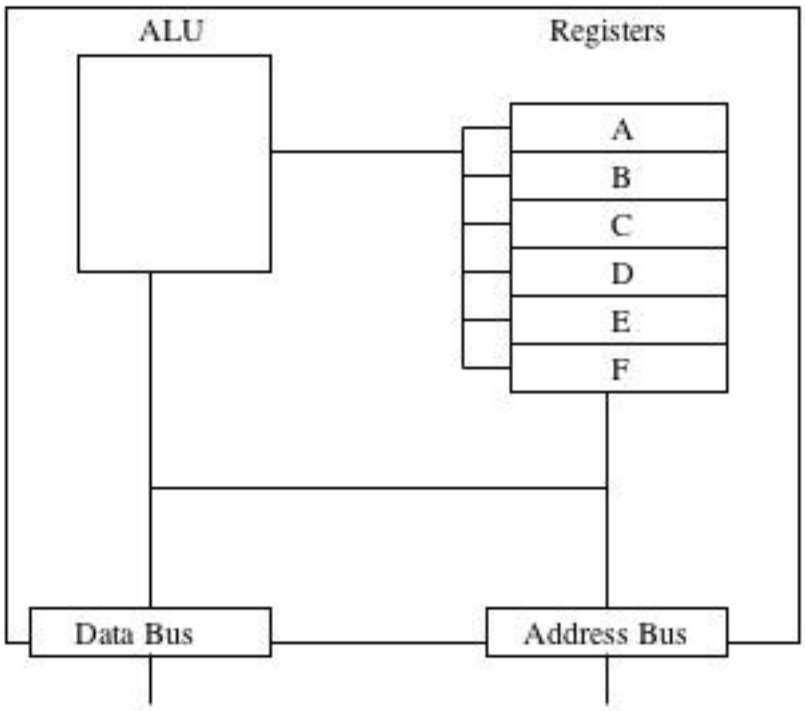
The registers are like memory locations inside the CPU where the CPU can temporarily store data for very fast access. The number of registers can vary from about 5 to several hundred.

The registers are referred to by the letters A, B, C, D, etc.

Data is transferred internally in the CPU on a set of busses.

Typical instructions are:
ADC $1288 Add to register A the contents of memory location $1288
STA $10DD Store the contents of register A in memory location $10DD
LDA $0005 Move the contents of memory location $0005 to register A

All computer programs consist of these very basic instructions—millions of them!



Lesson 5

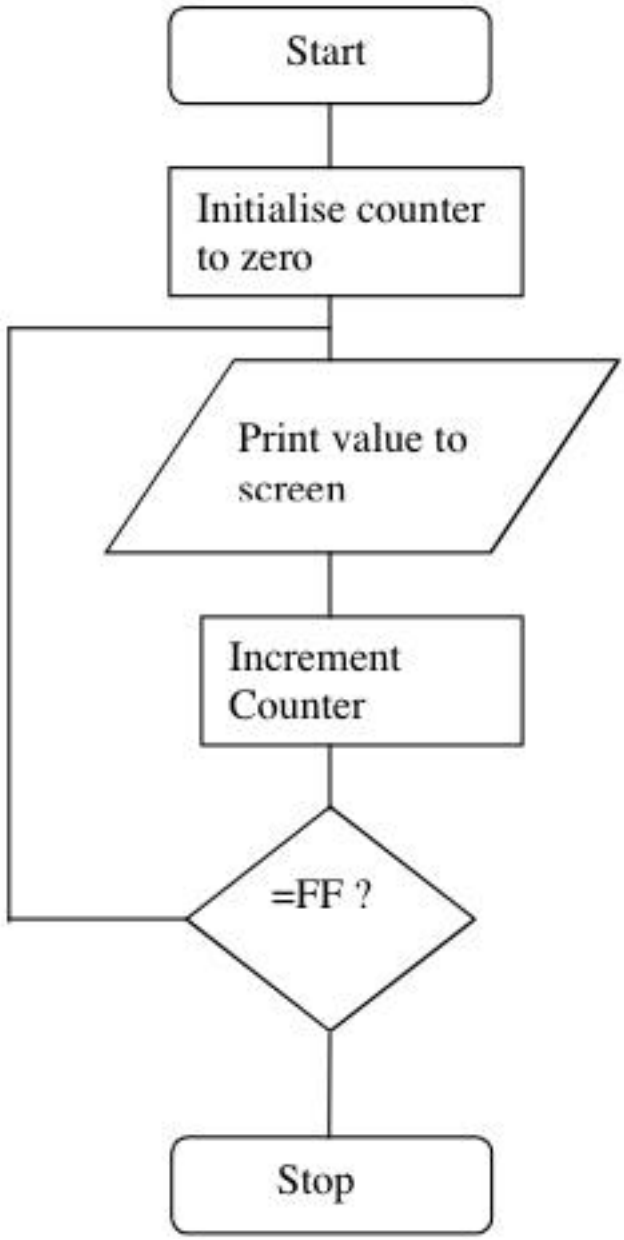
Summary of the talk at the Tuesday 28 March 2006 meeting

The aim is to specify an example program, draw a flow chart and then write an assembly language program that will produce then desired result.

The problem:

Write and test an assembly language program on the Apple 1 replica that will print on the screen a sequence of ascii characters.

Flow chart



Description

The counter uses accumulator or register A in the 6502 CPU. It starts with 0000 0000 which is printed on the screen. We then ADD 1 to the counter and print it until the counter gets to $FF. Then the program stops. So we should see on the screen 0123456789 etc, etc. See the screen shot.

The registers in the CPU, 6502 are a bit like memory locations that hold currently used data.

$FF means hex FF.

The trapezoidal block represents a subroutine which is where the program is directed to some code that exists in the ROM of the Apple 1 replica. When the ROM code has been executed the program returns to the next block in the diagram.

The increment counter block just ADDs 1 to register A, the counter.

Has then program finished ? We do this by testing to see if the counter has reached 1111 1111 binary or FF in hex.

If the counter has not reached $FF the program goes back to the print to screen instruction. If the counter is $FF then the program terminates.

The program with the hex code for each instruction.

Location		**Code**	**Instruction**	**Comments**
0000	START	A941	LDA #$00	Put 0000 0000 in register A
0002	LOOP	20EFFF	JSR PRINT	Execute the code in ROM
0005		6801	ADC #1	Add 1 to the counter
0007		C9FF	CMP #$FF	Is the counter = $FF ?
0009		DOF7	BNE LOOP	If not yet to $FF go to print
000B	SELF	4C0B00	JUMP-SELF	Stop, loop here till reset.
	END			

Test it

It sort of worked OK. There is a bug in the code because the subroutine in the ROM sometimes modifies register A and mucks up the counter. Ah well! Better luck next time!

Typical computer applications have 100s of millions of lines of code like this, but it is written in a high level language like C.