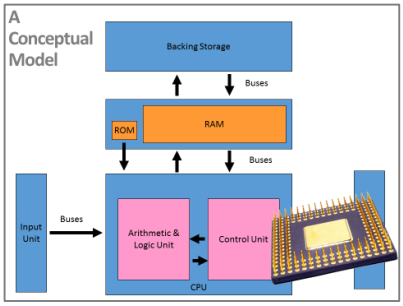
Considers the remarkable development of microprocessor technology and develops a conceptual model of a computer. Introduces two activities. A Human Computer simulation introduces the different components and illustrates the limited nature low level actions. A Little Man Computer simulation build on this to introduce the fundamentals of low level language, a limited instruction set and the fetch-execute cycle.

Preparation required:
Hiccup labels and props produced to facilitate the Human Computer activity.
Hiccup program listings per person.
LMC simulator available on all devices, copy of instruction set and challenges for each person.

Lifting the Lid

So how does a computer actually work? Lifting the bonnet of a desktop computer can be an interesting, hands-on classroom experience for children, but it tells them very little about the principles on which they operate. Before getting out some old computers and the screwdrivers, it helps to have a conceptual model. We need to abstract away the technical detail and focus on the concept of a computational machine.

A computer provides a means to Input data – Process it – Output data. We ought to add the ability to store (and retrieve) data too. Looking 'under the hood', asking how a computer processes information identifies two key components: the CPU and Memory. The data is passed between components using buses. Memory can be divided into RAM (Random Access Memory) and the much smaller ROM (Read Only Memory). Read Only Memory can't be altered, so the data flow is one way, rather than two. ROM stores instructions that allow a computer to start (boot) and check it can receive and send data to peripherals (input / output / storage). Programs (and data they use) are loaded into RAM to run. When the computer is turned off, they are lost so we save data or programs as files on backing storage (often a hard drive).



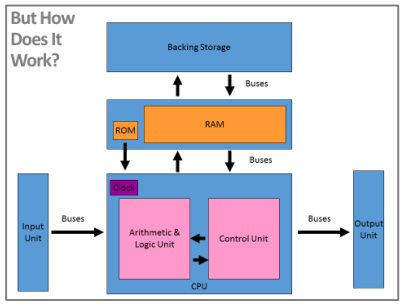
A nice but old 3 minute video (youtu.be/C3dLwtwdEjM) provides a simple introduction for children, making an analogy between the CPU and an unintelligent filing clerk. It also introduces the idea that a program is executed by fetching, decoding and executing each instruction in turn.

If we lift the lid on the CPU, we can identify another two key components: the Control Unit, which decodes the instructions it receives, and the Arithmetic and Logic Unit which performs any manipulation and calculation. Buses inside the chip tend to be referred to as internal, rather than external buses. The processor lies at the heart of a computer. If we lift the lid on a desktop pc, they often look like the photo shown. The chip itself is a tiny piece of silicon sitting inside this housing. You can get a feel for the size from a photo which compares it to a grain of rice. Further photos show the chip wired to the pins of the housing.

The chip is billions of switches, connected to create the sort of logic gates we saw earlier. Made from silicon, many are produced on a single wafer. A video (2min) shows the process: youtu.be/d9SWNLZvA8g. It's difficult for children to make the link between switches and semi-conductor transistors etched onto a wafer. Another picture shows the actual connections on a wafer. There are three layers. The depth is 22nanometres. An Intel video (4min) gives children a sense of the scale involved: youtu.be/YIkMaQJSyP8. To get a sense of the millions of circuit switches involved, www.visual6502.org/ shows the state changes involved in an ARM chip executing a simple program. Designing a chip can be likened to designing a road map for the entire planet, including footpaths!

The point is not to get obsessed about the technology, but encouraging reflection on what 50 years of technological progress has delivered. Key metrics are compared; an improvement by a factor of 50 billion! The quote from Bill Gates puts it in perspective.

A Human Computer



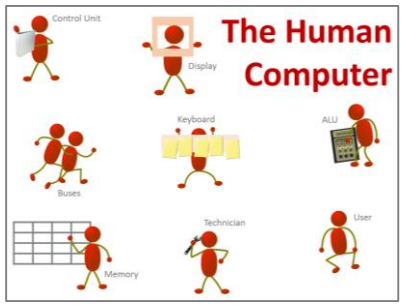
It's one thing to marvel at technological progress, but it doesn't get us any closer to understanding how a computer actually works. Let's just recap our conceptual model. We've added one extra component to complete the picture – the clock. It is the clock that regulates all the switching on/off that happens billions of times per second.

Children act out the roles of components in a Human Computer group activity (7 – 10 per group). It works well outdoors. The idea comes from Steve Baker (goo.gl/Hnc8VE), resources provided in a zipped folder.

We have our conceptual model, so what parts do we need for our human computer? Discuss this before revealing the minimal list of we will use. Each role equates to part of our model. Programs are already loaded into memory, so we have no need for backing storage. Our computer is very slow, so we can dispense with the clock. ROM is optional. It is worth spending the latter half of a lesson allocating children to groups, assigning roles and ensuring they understand what is involved. That allows the following full lesson be given over to executing programs.

Some components are essential, whilst others allow flexibility to accommodate varying numbers.

- The Control Unit is the key role. They hold the program instructions, should read each out in turn and direct operations. Stay close, or have two children working together in this role.
- One child needs to act as the ALU. They could have a calculator or mini white board in case basic calculations need to be performed.
- One acts as an input unit. They hold a card keyboard with letters on
- A child with a cardboard box on their head with a square cut out or writing on a whiteboard represents output; a screen / printer.
- One with several pigeonholes (or small boxes) labelled with A, B, C, D etc. represents memory (RAM).
- Energetic pupils can act as buses, running data according to instructions given by the Control Unit. This role can double up.



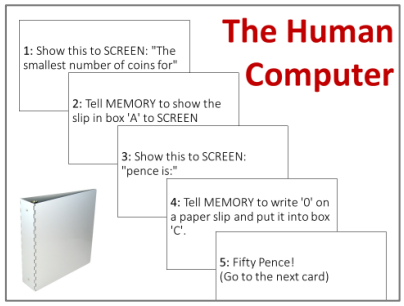
- A user must be identified. They perform the role of operating the keyboard, or reading from the screen.

If you have a child who requires firm control, an extra memory component, ROM can be included. Their job is simply to run around at the start putting the correct labels on the other pupils (booting up). Beyond that, they stay still. The teacher (or an able pupil) can play the role of technician, following the action with a code listing and fixing things if the activity breaks down! Once roles are allocated, give out the role descriptions and ensure the children read them carefully. A list of equipment and preparatory instructions is provided. Between runs of programs, roles can be changed.

They aim is to run programs written in our simple instruction language HICCUP. Two programs are provided to get started. They are best printed out on card and put in a ring binder.

The first program calculates currency required for change. It does not need a User or keyboard. It does needs a value in memory location A. Write 67 on a slip of paper and put it there before you start.

Each instruction is on a separate card. The first five instructions are shown. First the Control Unit writes the message on a slip of paper, gives it to the buses, who show it to the screen. The screen writes it on a display. Second the Control Unit tells Memory to write the value in Box A (67) on a slip of paper, give it to the buses, who show it to the screen, who adds it to the display. You will notice that it takes 3 instructions just to get a message on the screen. Each instruction is executed in sequence. Note instruction 5: Here nothing is done! This is a label or placeholder. It will be used later



when a jump is required to return to this place in the program. They are used to control program flow, implementing selection statements and loops. It is therefore very important to keep the cards in order (a ring binder works best). Children will probably be surprised at how long it takes to work through the program, but if successful the output should say how many of each coin is required to make 67 pence. Encourage simple modifications to test understanding. Display (or hand out) the code listing to aid this. How would we modify the code to calculate change for 85 pence? How could a User input the amount?

The second program is a High / Low guessing game. It introduces user input, so needs the extra role and a keyboard. The keys required would be <, > and =, but a later extension might require digit keys too (some blanks are included in the resources). If pupils have studied search algorithms they should be able to recognise this as a divide and conquer routine. Again, after running it, encourage simple modifications to test understanding. Could we modify the program to restrict numbers between 1 and 100? An interesting extension might consider how to alter the language for a different output device, such as a simple robot / turtle, like Logo.

It is worth drawing out of a discussion what insights the pupils have gained about how a computer works. The most obvious is how small each of the instructions is, in order to do anything, even simple things. The 'low level' instructions computers work with are very limited in what they can do. You may want to get them to consider what is required to use a mouse as input. TED Ed have a very good 4 minute video (goo.gl/uOqhI1) that traces the journey of a mouse click, and introduces a sense of the complexity involved in today's multitasking systems. The sample chapter from an excellent book, Brown Dogs and Barbers (included in the resources) explains the same process and makes good background reading for teachers.

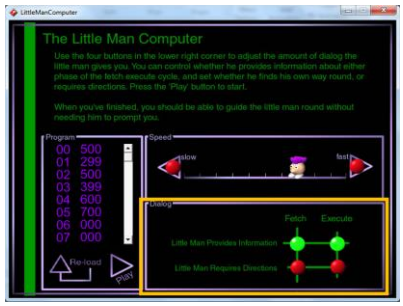
Try to draw out from a discussion the following points. In a 'real' computer:

- Instructions are coded as numbers.
- Program instructions are stored in Memory, not a file.
- Each instruction has to be fetched before it can be read.
- Before reading, must be decoded.

The next exercise addresses these points and helps develop children's appreciation of the process further.

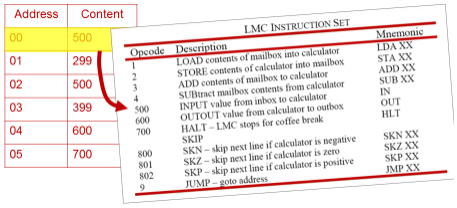
A Little Man Computer

Welcome to the Little Man Computer. The software is included or available from goo.gl/WQKnko. Our Control Unit is no longer a human, but a little character with a crazy hairstyle! The process known as the fetch – execute cycle is illustrated by this activity. Before we start, let's review our conceptual model. The Little Man has the same components as before. We don't need to worry about Backing Storage as our program will be in memory (RAM). The computer is ready to go, so no need to think about the role of ROM. The pupils running the simulation will be the clock, advancing the sequence by clicking the mouse. But there is one more component we need, and that is known as the Program Counter.



Here is the opening screen, when the application loads. The buttons in the bottom right allow you to set whether you watch an animation, or whether the pupil has to provide directions for the little man. It is set initially for the Little Man to provide the student with a commentary.

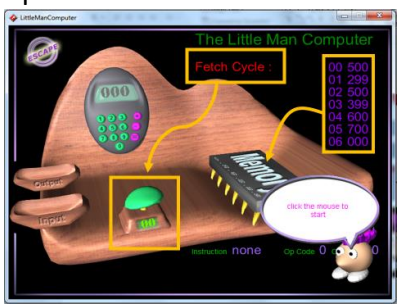
A sample program is already loaded into memory. We don't know what it means unless we have a copy of the instruction set. Hand these out - the following slide talks through how to decode the program.



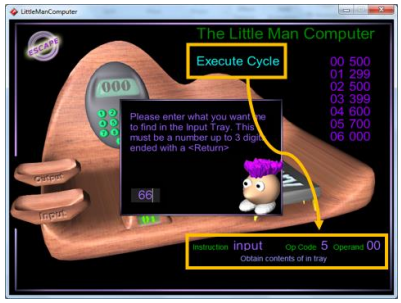
The address column refers to the memory location for each instruction. LMC refers to these as 'mailboxes'. It has 100 mailboxes, from address 00 to 99. The instruction in memory location 00 is 500. What does that translate to? Input a value to the calculator. LMC uses 'calculator' to refer to the ALU.

The second instruction (299) isn't quite as straightforward. The content can be split into two part: the Opcode and the Operand. The Opcode is in the Instructions Set. What does it mean? Store the contents of the calculator (ALU) into a mailbox. But which mailbox? The XX in the Instruction Set means any given in the operand. Armed with this knowledge, can the class translate the remaining program and state what it does? Work through it line by line. The play button allows us to see the Little Man in operation. Before you do, turn the speed down a touch.

Address	Content	
	Opcode	Operand
01	2	99



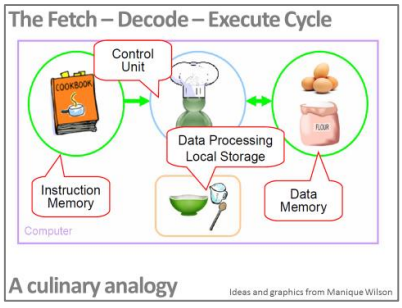
Here is the Little Man ready to go. The program is listed over on the right, but is really loaded into the Memory chip. The ALU is represented by the calculator; input and output trays over to the left. We are at the start of a fetch cycle. The instruction must be fetched from memory before the Little Man can decode it. But how does he know which instructions to fetch? He looks at the value in the Program Counter at the front. As soon as he knows the value, it is increased by 1, ready for the next Fetch cycle.



Click the mouse to watch Little Man talk you through what he is doing. If it is too fast, the Escape button takes you back to the opening screen to alter the speed. Once the Little Man has fetched the required instruction we enter the Execute Cycle. Notice the instruction has been decoded by the Little Man in the lower right. You cannot input a negative number, but it will handle negative values produced by calculations in the ALU. To reinforce understanding, pupils can see whether they can give directions to the Little Man. They may find this surprisingly difficult! Programming challenges are in the resources. These are probably too challenging for KS3, but are included to allow teachers to practice.

The Fetch – Decode – Execute Cycle

CAS member Manique Wilson offers a nice culinary analogy for explaining the Fetch / Execute Cycle to children. Her original PowerPoint is on the CAS Community: goo.gl/hdOabt. Take the 3 key components: The Control Unit, Memory and ALU which processes the data. An analogy with cooking might be the cook (Control Unit), cookbook (instructions) and ingredients (data) in memory. Mixing the ingredients is akin to processing, the final output being the cake of course! The fetch finds the next instruction in the cookbook. The instruction is read and understood (decoded) and the data (ingredients) required found in the store. Finally it is processed – in this example, the egg is cracked into the bowl.



Notice the example makes a distinction between the Instruction Memory and Data Memory. This is an example of alternative architectures. When data and instructions are stored in the same memory space (as in most PC's), we talk of Von Neumann architecture, designed initially by John Von Neumann. When two separate memory spaces are used (as in most mobile phones) we talk of Harvard architecture. Either way, the same Fetch – Decode – Execute cycle is followed.

An old lecture by Richard Feynman the noted physicist (youtu.be/EKWGGDXe5MA) rounds this section off. The basis for many of the metaphors about 'little men', this lecture explains how computers work from the inside out, using a file clerk as his analogy.

Computers use a very small number of 'low level' commands. Programming in 'low level' languages is difficult and soon after the first computers, 'high level' languages also developed. These grouped lots of low level commands together, making it easier for programmers to express their solutions. The languages we use today are high level languages. We will look at how they are translated into machine code in another unit. But a short, interesting article from cs4fn (goo.gl/aOYgVw) notes that Grace Hopper, the programmer who popularised the term 'bug', also developed the first translator for a high level language.