|  |  |
| --- | --- |
| use dataflow diagrams (DFDs) and system flowcharts to represent input, process, storage and output in computational systems | (**W**) Introduce the basic components of flow charts.  (**I**) Learners create flow charts to describe how they prepare for school in the morning.  (**W**) Demonstrate how a flow chart can be used to describe a password box on a website. Cover loops and **If** statements.  (**G**) Groups create flow charts for complex routines such as bubble sort.  (**I**) Using the flow chart learners implement bubble sort in any language. |
| fill memory cells in a rational manner | (**W**) Teacher reminds learners of Assembly language programs.  (**G**) In pairs learners try to find as many different Assembly operation codes (opcodes) as they can. Feedback to the whole class.  (**G**) Take a very simple executable file that adds two numbers together. Disassemble it and try to work out what the assembly is doing. Ask one group of learners to explain to the rest of the class.  (**W**) Introduce them to a small variety of programs.  Watch tracing of code using loops and labels.  (**G**) (**f**) Learners to practise using the Little Man Computer by creating simple programs to:   * Add two numbers together * Doubling a number * Multiplying a number by 7   (**I**) If there is time; provide more complex problems for learners to solve such as loops and selection. |
| understand the concept of addressable memory |
| understand how addressable memory is used in a computer |
| trace a given simple assembly language program |
| show understanding of the various stages in the compilation of a program: lexical analysis, syntax analysis, code generation and optimisation | (**W**) Give learners a description of how a compiler works.  (**G**) Give pairs of learners several pieces of VB.NET code and ask them to find the syntax errors.  (**I**) Learners create code with syntax errors for their peers to find.  (**G**) Give pairs of learners several pieces of VB.NET code and challenge them optimise each piece of code.  (**W**) Ensure learners understand how different languages are interpreted and compiled, including Java, C++, Python, and JavaScript.  (**G**) Given different scenarios learners should debate whether they need a compiler or an interpreter and provide reasons why in each case. |
| show awareness that high-level language programs may be partially compiled and partially interpreted, such as Java |
| summarise the relationship between high level languages and low level languages | (**G**) Learners research the main differences between high and low level languages. Give the uses for both. Ask groups to play argument tennis, one side arguing why low level is better and the other side arguing why high level are better. |
| be aware of the history of programming languages and the drawbacks of assembly-language and machine-code |
| show understanding of the differences between RISC and CISC processors | (**G**) Research the difference between CISC and RISC including references to pipelines and registers, present the pros and cons of both to the rest of the class.  (G) Think pair share, what factors affect the speed of a processor. |
| show understanding of the importance/use of pipelining and registers in RISC processors |
| explain the effect of clock speed, word length and bus width on performance |
| create a website using HTML, CSS and XML | (**W**) Recap on how to build a basic webpage, get learners to list the code parts of a page (Head, Body, etc.). Introduce a simple script to insert today's date.  (**G**) (**f**) In pairs learners create a script to add and delete document sections using tutorials to help them.  (**G**) (**f**) Learners use the Document Object Model and a scripting language to extract data from XML and build a list of products on a webpage when different selection criteria is provided. Get learners to pull the data from the XML and display it on the screen. |