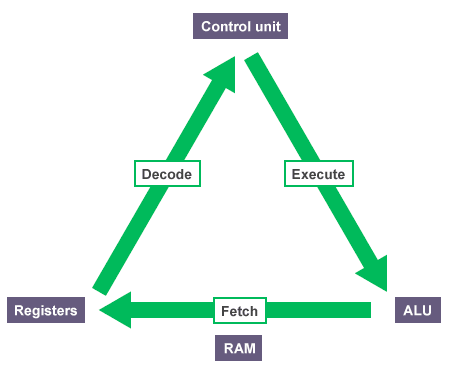
Basics

* Computers are very dumb machines
* They only do what they are told to do
* The basic operations of a computer will form what is known as the computer’s instruction set
* To solve a problem using a computer, you must provide a solution to the problem by sending instructions to the instruction set
* A computer program sends the instructions necessary to solve a specific problem
* The approach or method that is used to solve the problem is known as an algorithm
* So, if we were to create a program that tests if a number is odd or even
* The statements that solve the problem becomes the program
* The method that is used to test if the number is even or odd is the alogorithm
* To write a program, you need to write the instructions necessary to implement the algorithm
* these instructions would be expressed in the statements of a particular computer

Terminology

* CPU (central processing unit)
* does most of the computing work
* instructions are executed her
* RAM(random access memory)
* Stores the data of a program while it is running
* Hard drive (permanent storage)
* Stores files that contain program source code, even while the computer is turned off
* Operating system
* Developed to help make it more convenient to use computers
* a program that controls the entire operation of a computer
* All input an output
* manages the computer’s resources and handles the execution of programs
* windows, unix, android, etc…
* Fetch / Execute Cycle (life of a CPU)
* fetches an instruction from memory (using registers) and executes it (loop)

Fetch/Execute Life Cycle



Higher Level Programming Languages

* High-level programming language make it easier to write programs
* Opposite of assembly language
* C is a higher level programming language that describes actions in a more abstract form
* the instructions (statements) of a program look more like problem solving steps
* do not have to worry about the precise steps a particular CPU would have to take to accomplish a particular task
* Compilers
* a program that translates the high-level language source code into the detailed set of machine language instructions the computer requires
* the program does the high-level thinking and the compiler generates the tedious instructions to the cpu
* compilers will also check that your program has valid syntax for the programming language that you are computing
* finds errors and it reports them to you and doesn’t produce an executable until you fic them
* high-level languages are easier and much more efficient to write programs

Writing a program:

The act of writing a C program can broken down into multiple steps:

1. define the program objective
2. design the program
3. write the code
4. compile
5. run the program
6. test and debug the program
7. maintain and modify the program

(Taken from the “C Primer Plus”)

Steps in writing a program

1. Define the program objectives

* understand the requirements of the program
* get a clear idea of what you want the program to accomplish

2. Design

* Decide how the program will meet the above the above requirements
* what should the use interface be like?
* how should the program be organized?

3. Write the code

* Start implementation, translate the design in the syntax of C
* you need to use a text editor to create what is called a source code file

4. Compile

* translate the source code into machine code (executable code)
* consists of detailed instructions to the CPU expressed in a numeric code

5. Run the Program

* the executable file is a program you can run

6. Test and debug

* just because a program is running, does not mean it works as intended
* need to test, to see that your program does what it is supposed to do (may find bugs)
* debugging is the process of finding and fixing program errors
* making mistakes is a natural part of learning

7. Maintain and MOdify the Program

* Programs are released and used by many people
* have to continue to dix new bugs or add new features

For the above steps, you may have to jump around steps and repeat steps

* E.g. when you are writing code, you might find that your plan was impractical
* Many new programmers ignore steps 1 and 2 and go directly to writing code
* a big mistake for larger programs, may be okay for very simple programs
* the larger and more complex the program is, the more planning it requires
* should develop the habit of planning before coding
* Also, while you are coding, you always want to work in small steps and constantly test (divide and conquer)