Data types and representation

- ## Machine code and instructions

Machine code is in binary and is composed of raw instructions that the CPU carries out (**1st generation language**).

All the instructions are written in machine code (which uses binary). They consist of an opcode which gives the instruction required and the operand which gives the data (value or memory address).
E.g. Real world operators examples are +,-,*, / Real world operand examples are variables, values

Opcode also sometimes called Operator

More commonly we are given 8-bit machine code. The first 4 bits show the opcode and the last 4 bits show the operand.
For example, 10011011 would be split into Opcode (1001) and the Operand (1011)
The opcode might represent say "Add the contents of the operand to the accumulator"

- ## Instruction sets

An instruction set = the complete set of all the instructions in machine code that can be recognised and executed by a central processing unit to carry out tasks.

For example, an instruction might look like: 'Add ACC, 3'. This instruction tells the CPU to add 3 to the accumulator.

Every model/architecture of CPU has its own instruction set.

- ## Assembly code

Assembly language is a low-level programming language written in mnemonics that closely relates to the machine instructions of the CPU.

One lien of assembly code relates to one line of machine code.

It is a second generation language as the mnemonics are translated directly to machine code by an assembler.
Assembly languages are unique to the architecture and so are not universal across devices.

Each command in assembly language is called a 'mnemonic'.

What are the advantages of assembly language?

It is easier for a human user to interpret than binary since it is not formed of long strings of numbers like machine code.

Assembly language is more efficient than a higher generation language as it is smaller (uses mnemonics).

Less overall code compared to higher level languages making it easier to locate errors.

How does a computer/CPU distinguish between instructions and data?

The CPU/Computer can't distinguish between data and instructions and so simply deals with what it expects to find. It will distinguish between data and instructions from the context. It will find any data if it is told to find data (likewise with an instruction).

How does a computer/CPU know whether the operand contains an address or actual value?

The computer can only operate on the basis of what it is expecting to find and will interpret the next binary value as either an address or a value depending on the opcode of the instruction.

Primitive data types

Primitive data type	Definition	Example
Integer	Stores positive or negative whoel numbers	17
Real/Float	Stores numbers that can contain decimal places/values and can also store integers	17.65
Character	Stores a single character which can be a letter, number or symbol	%
String	Stores alphanumeric combinations or text. The string is a group of characters stored as one (a tuple). Numbers used in calculations should not be stored this way.	Name
Boolean	Stores values of only two states (TRUE or FALSE, 1 or 0) only.	TRUE, 1

The data structure of a string

A String is regarded as a data type but can also be considered as a data structure since: It is actually an array of separate characters that make up the alphanumeric combination.

When all characters are joined up they will make the string.

Arrays only store data in a list if the data is of the same data type.

File	Size
One character of text	1 byte
A full page of text	30 KB
One small digital colour photograph	3 MB
Music CD	600 MB
A DVD	4.5 GB
Hard disk	1 TB