Learning Aims

- Describe the characteristics and purpose of high-level and low-level programming languages

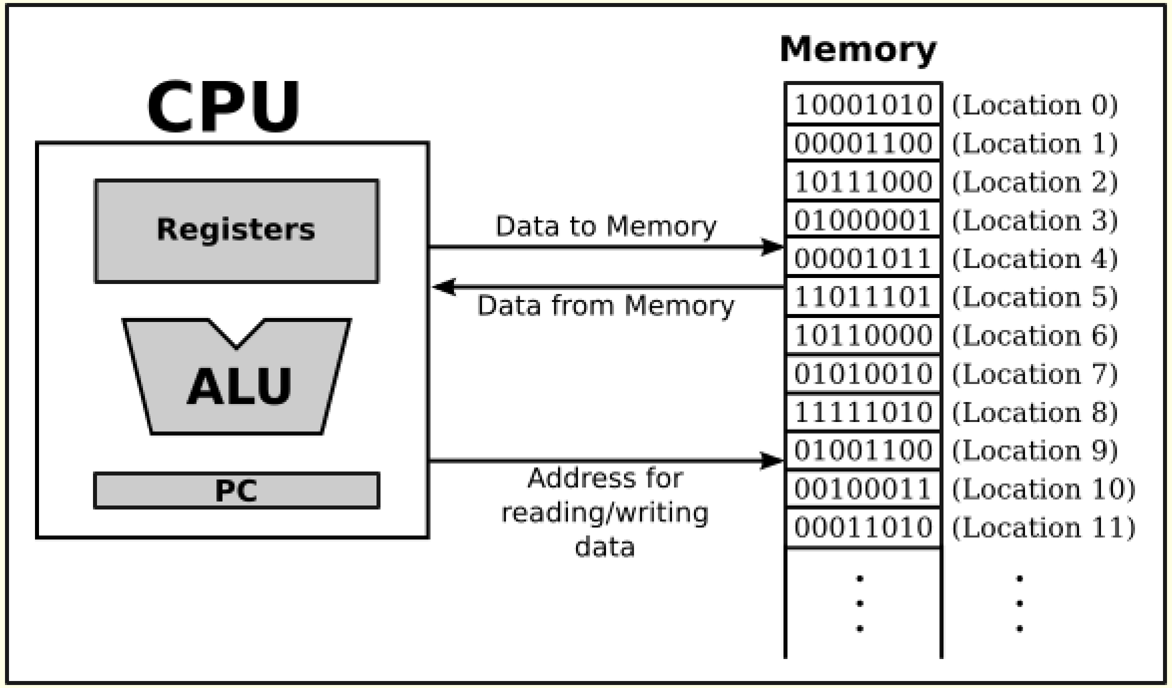
Think and Share

- ____________ and _______ are stored in **memory**.
- Instructions are _____________ one at a time into the processor.
- The instructions are ____________ by the control unit.
- The instructions are executed, sometimes using the _______.
- The Central Processing Unit (CPU) contains the **Arithmetic/Logic Unit (ALU)**, **Control Unit (CU)** and _____________.
- _________ __________determines processor performance, i.e. the number of instructions per second.
- _______________ improves performance by overlapping parts of the fetch-decode-execute (F-D-E) cycle for multiple instructions.

Assembly language	A language that replaces machine code with mnemonics and operands to make them easier to read/write.
Assembler	An assembler translates assembly language into machine code.
Compiler	A compiler creates an executable file for a program by translating a high-level language to machine-readable code.
Execute	To carry out the instructions for a computer program.
High-level language	A human-readable language written in formal, structured English.
Interpreter	An interpreter translates and executes code line by line. It translates the code into machine-readable code.
Low-level language	Quickly executed by a computer, written in either machine code or assembly.
Machine code	A program written using 1s and 0s. A computer can execute this directly.
Mnemonic	A code to help us remember something.
Operand	A piece of data that can be changed.
Translator	Executes the programs that programmers write in high-level languages.

Main memory Instructions and data

- Instructions and data are stored in RAM until they are needed by the CPU.
- Each instruction and item of data is stored in a location in memory.
- Each element of the memory has a unique address.
- Instructions a CPU executes are stored in memory as binary numbers.



10001010	(Location 0)
00001100	(Location 1)
10111000	(Location 2)
01000001	(Location 3)
00001011	(Location 4)
11011101	(Location 5)
10110000	(Location 6)
01010010	(Location 7)
11111010	(Location 8)
01001100	(Location 9)
00100011	(Location 10)
00011010	(Location 11)
⋮	⋮

A history lesson …

The first ever programs were written solely in **machine code** (0s and 1s).

Each instruction was entered by hand before being **executed**.

Operation code tables were used to help with this, but it was still very time consuming.

To speed things up, programmers developed an **assembly language**.

This made the **instructions** easier to read by using a **mnemonic.**

Each line of **assembly language** was equivalent to one line of **machine code**.

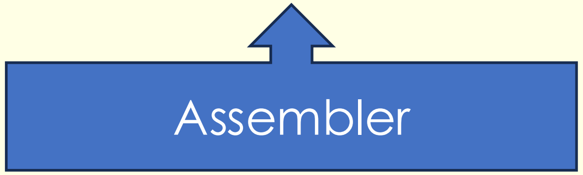
Assemblers were created to automatically translate the assembly language into machine code.

```
100010110101010111111100

100010110100010111111000

111010000

100010010100010111110100
```

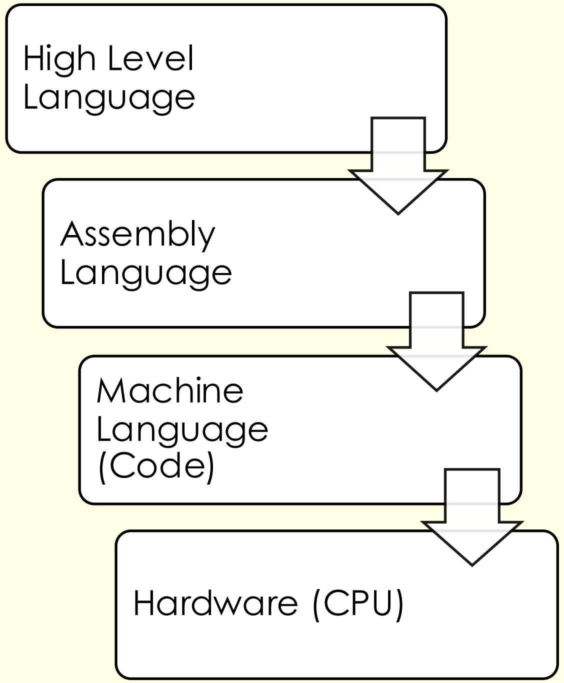

Assembler

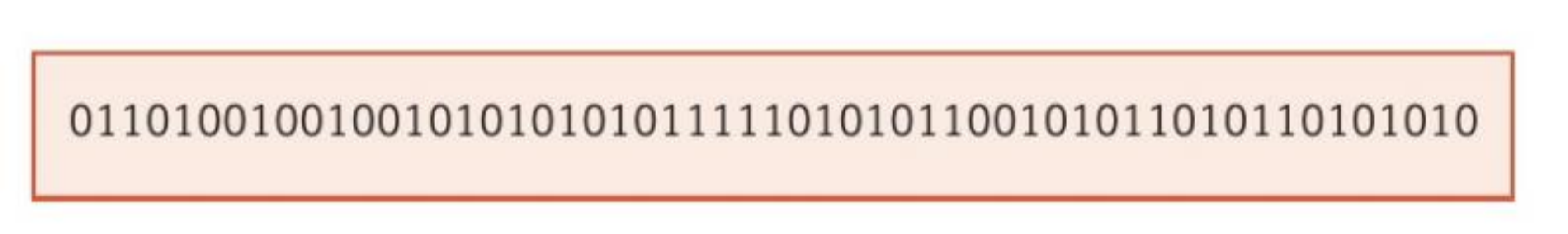
```
1 LDA  4
2 ADD  5
3 STA  6
4 30
5 60
6
```

Machine code

Machine code is written in binary so instructions can be processes directly by the CPU and do not need to be translated.

- An instruction set relates to a specific processor and is written in machine code.
- The central processing unit (CPU) understands machine code directly and can act upon the instructions.
- A program written in machine code consists of 0s and 1s only.
- Machine code is **very difficult to learn, write and debug**.
- Even a very simple program could have thousands of 0s and 1s in it.

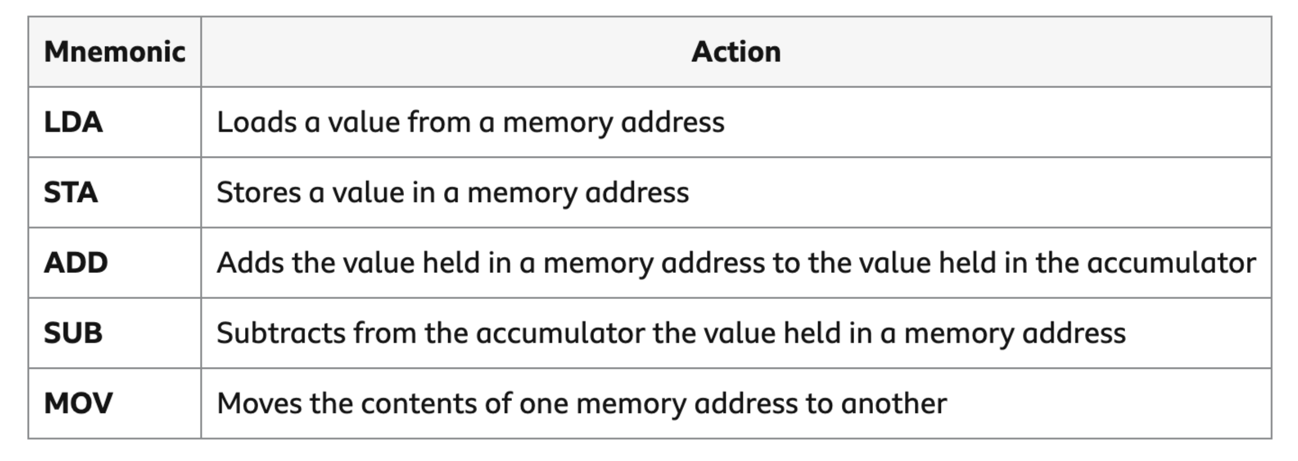




0110100100100101010101011111010101100101011010110101010

What is assembly language?

- Assembly language is a low-level language.
- Designed for **specific processor architecture**
- Can manipulate the **hardware directly.**
- Instruction is written as a short, keyword called a **mnemonic.**
- Each mnemonic directly corresponds with a single machine code instruction.



Mnemonic	Action
LDA	Loads a value from a memory address
STA	Stores a value in a memory address
ADD	Adds the value held in a memory address to the value held in the accumulator
SUB	Subtracts from the accumulator the value held in a memory address
MOV	Moves the contents of one memory address to another

+ Mnemonics are much **easier to understand** and debug than machine code, giving programmers a simpler way of directly controlling a computer.

- But you have to decide and manage where data is stored in memory.

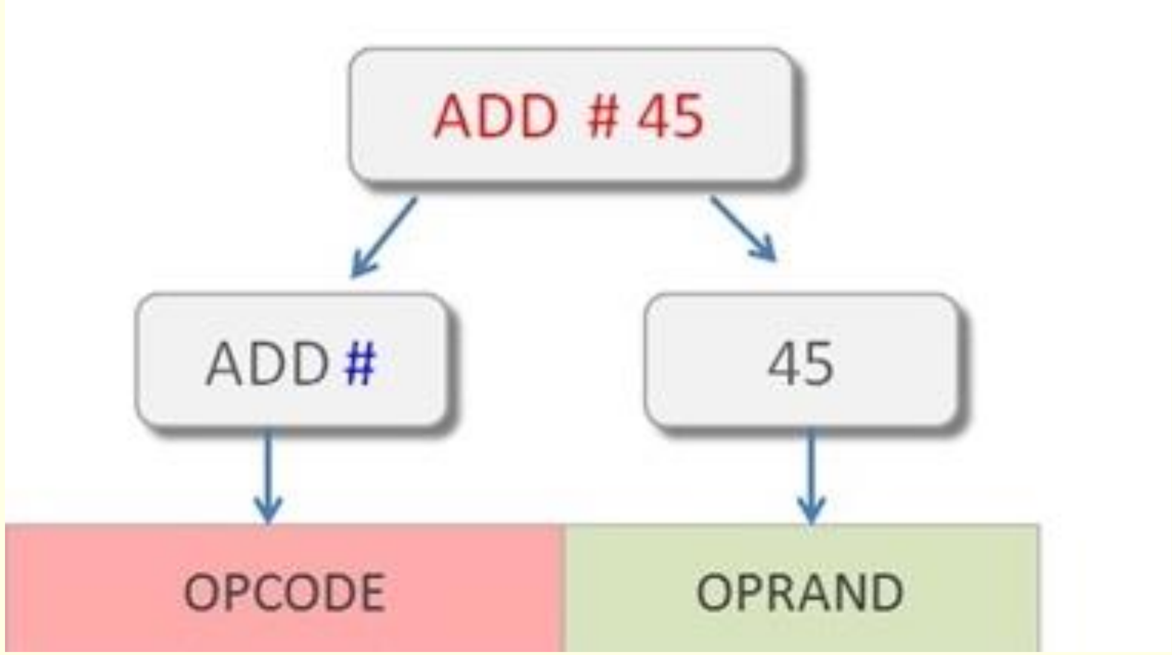
- Debugging can be more difficult than in high level languages.

Opcode and Operand

The instruction is split into an **Opcode** and an **Operand**

Opcode is which operation to carry out.

The **operand** specifies the data that needs to be acted on.



Example of Assembly Code

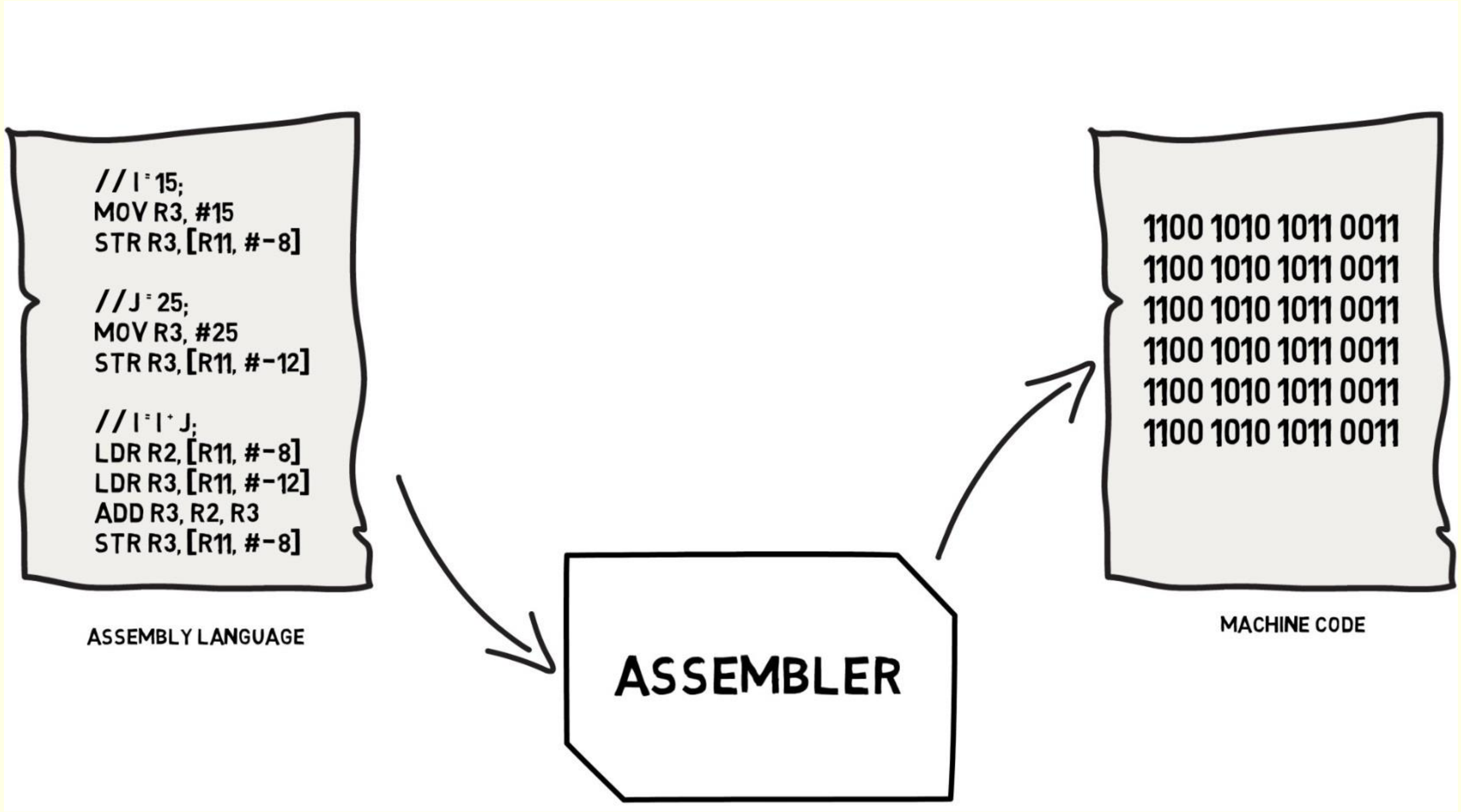
```
1 LDA  4        Load the value from memory location 4 into the accumulator

2 ADD  5        Add the value from memory location 5 to the accumulator

3 STA  6        Store the value from the accumulator into memory location 6 (90)


4 30
5 60
6
```

An **Assembler** to needed to translate assembly code into machine code



```
//I=15;
MOV R3, #15
STR R3, [R11, #-8]

//J=25;
MOV R3, #25
STR R3, [R11, #-12]

//I=I+J;
LDR R2, [R11, #-8]
LDR R3, [R11, #-12]
ADD R3, R2, R3
STR R3, [R11, #-8]
```

```
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
```

Assembler replaces each **mnemonic** with the appropriate binary machine code

Why use Assembly Code

- Code takes up **less memory** and **runs faster**
- To use the hardware more efficiently
- To write software for **embedded systems** because there is no need for code to be used on other architecture and it keeps memory usage to a minimum.
- To write software for device drivers.
  - **Device Drivers** are loaded into memory by the Operating System and used to control the operation of a Hardware Device e.g. Graphics Card Drivers, Printer Drivers.

Drawbacks of Assembly Languages

- A very limited range of instructions is available. Every task, even the simplest, has to be built up from the smallest steps.
- You have to decide and manage where data is stored in memory
- Debugging can be more difficult than in high level languages.

Knowledge Check

1. What is machine code?

2. What is the difference between an opcode and operand? LOAD 14

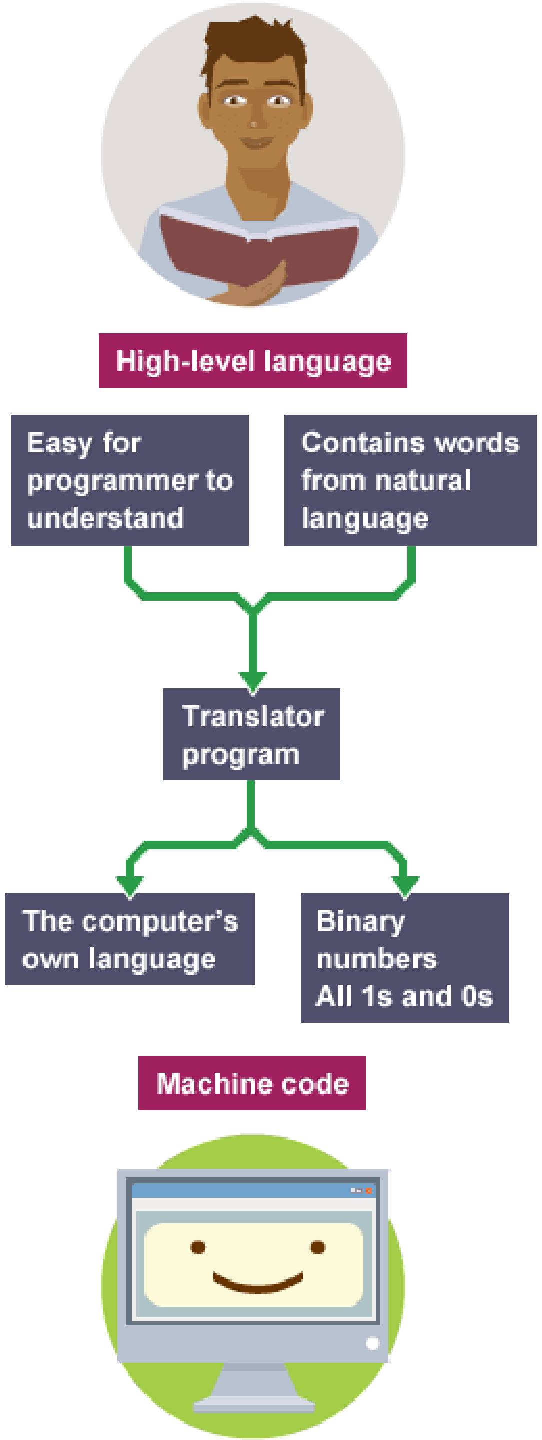
3. An assembly language is a high level programming language. True or False?

4. A assembly language needs a ________________ to translate the code into
___________ __________

What are high level languages?

- High level Language is easier to learn because it uses **English-like statements** which can be read by programmers.
- Example: Python uses 'print', ‘if’, 'input' and 'while' statements - all words from the English language - to form instructions.
- Faster to write programs
- Easier to debug programs
- Many types of high-level language exist and are in common use today, including: Python, Java, C++, C#, Visual Basic.NET, PHP and Ruby

- Translated using:
  - –Compiler
  - –Interpreter



High-level language

Easy for programmer to understand

Contains words from natural language

Translator program

The computer’s own language

Binary numbers All 1s and 0s

Machine code

Comparison

These **high-level languages** could write one line of ‘plain English’ code that would then be **translated** to dozens of lines of **machine code**.

Here is the code written in the programming language, **python**

This line of code is four lines long in assembly language, and four lines long in machine code.

```
c=a+b
```

```
mov edx,DWORD PTR [rbp-0x4]
mov eax,DWORD PTR [rbp-0x8]
add eax,edx
mov DWORD PTR [rbp-0xc],eax
```

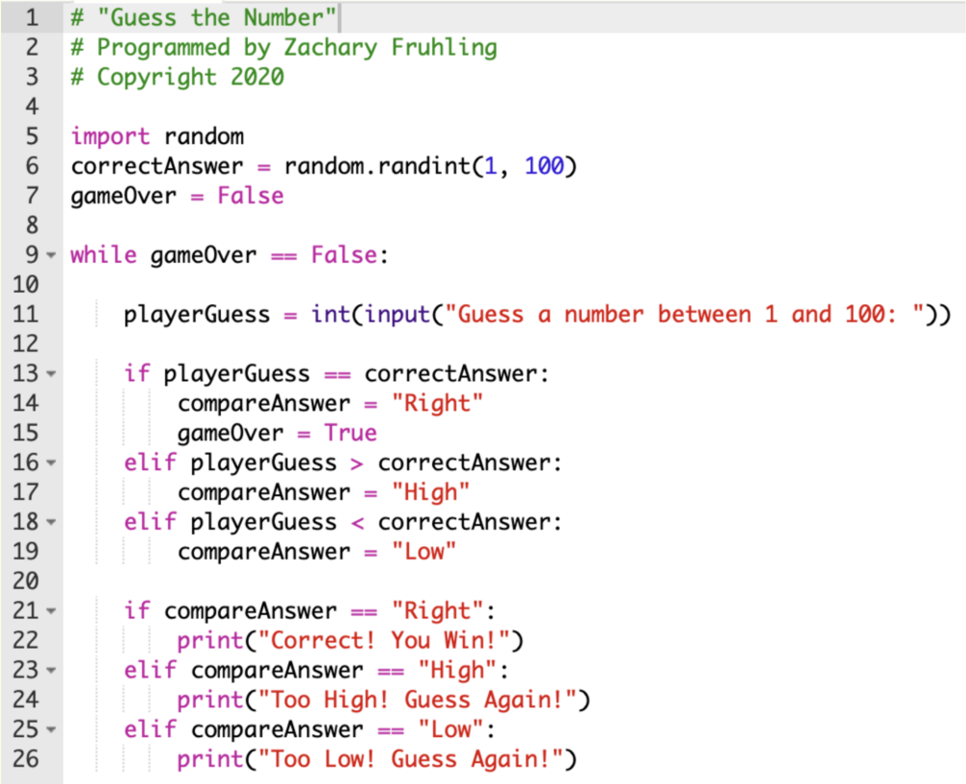
```
100010110101010111111100
100010110100010111111000
111010000
100010010100010111110100
```

Source code

- *Source code* is the term given to a *set of instructions* that are written in human-readable programming language.

- The Python program shown before is an example of source code.

- This code must be *translated* into machine code before the computer can understand and execute it.



```
# "Guess the Number"
# Programmed by Zachary Fruhling
# Copyright 2020

import random
correctAnswer = random.randint(1, 100)
gameOver = False

while gameOver == False:

    playerGuess = int(input("Guess a number between 1 and 100: "))

    if playerGuess == correctAnswer:
        compareAnswer = "Right"
        gameOver = True
    elif playerGuess > correctAnswer:
        compareAnswer = "High"
    elif playerGuess < correctAnswer:
        compareAnswer = "Low"

    if compareAnswer == "Right":
        print("Correct! You Win!")
    elif compareAnswer == "High":
        print("Too High! Guess Again!")
    elif compareAnswer == "Low":
        print("Too Low! Guess Again!")
```

Comparison of types of programming languages

High -Level	**Low-level**
It uses **English-like statements** which can be easily read by programmers. Designed for quick programming. Can be translated for multiple machine architectures. Need to rely on compiler to optimise the code.	Microprocessor/CPU/Machine specific and can control the hardware directly. Can be highly optimised to make efficient use of the hardware and execute more quickly. Each line of code is one instruction only Hard to read and learn. Only works for one type of machine architecture.

Identify the generation

```
Dim Num1, Num2, Tot as Integer
Num1 = Console.Readline()
Num2 = Console.Readline()
Tot = Num1 + Num2
Console.Writelein(“Total is: “ & Tot)
```

```
01010101010100101010100101
01010101001111100100010000
1010100101
```

```
LOAD r1, c
LOAD r2, d
ADD r1, r2
```

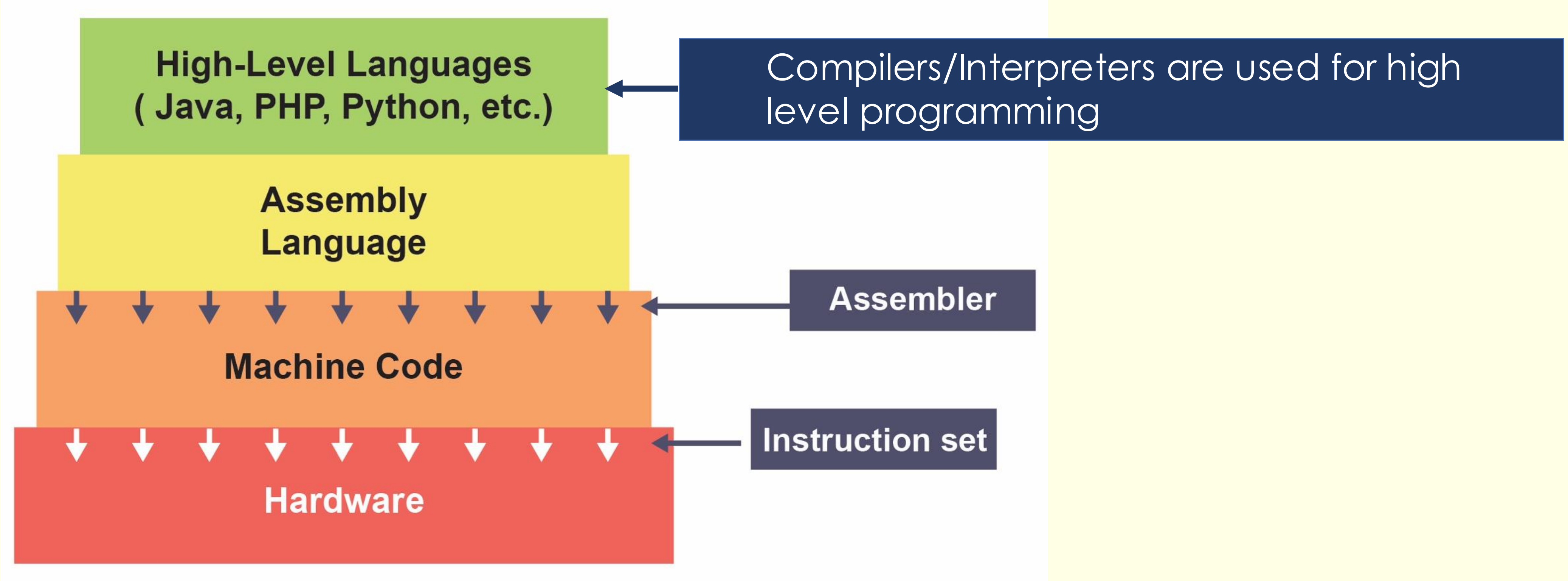
Learning Aims

- Describe how an interpreter differs from a complier in the way it translates high-level code into machine code.

Assembly language	A language that replaces machine code with mnemonics and operands to make them easier to read/write.
Assembler	An assembler translates assembly language into machine code.
Compiler	A compiler creates an executable file for a program by translating a high-level language to machine-readable code.
Execute	To carry out the instructions for a computer program.
High-level language	A human-readable language written in formal, structured English.
Interpreter	An interpreter translates and executes code line by line. It translates the code into machine-readable code.
Low-level language	Quickly executed by a computer, written in either machine code or assembly.
Machine code	A program written using 1s and 0s. A computer can execute this directly.
Mnemonic	A code to help us remember something.
Operand	A piece of data that can be changed.
Translator	Executes the programs that programmers write in high-level languages.

Translators

- Language translators are used to translate a language into machine code.



High-Level Languages (Java, PHP, Python, etc.)

Compilers/Interpreters are used for high level programming

Assembly Language

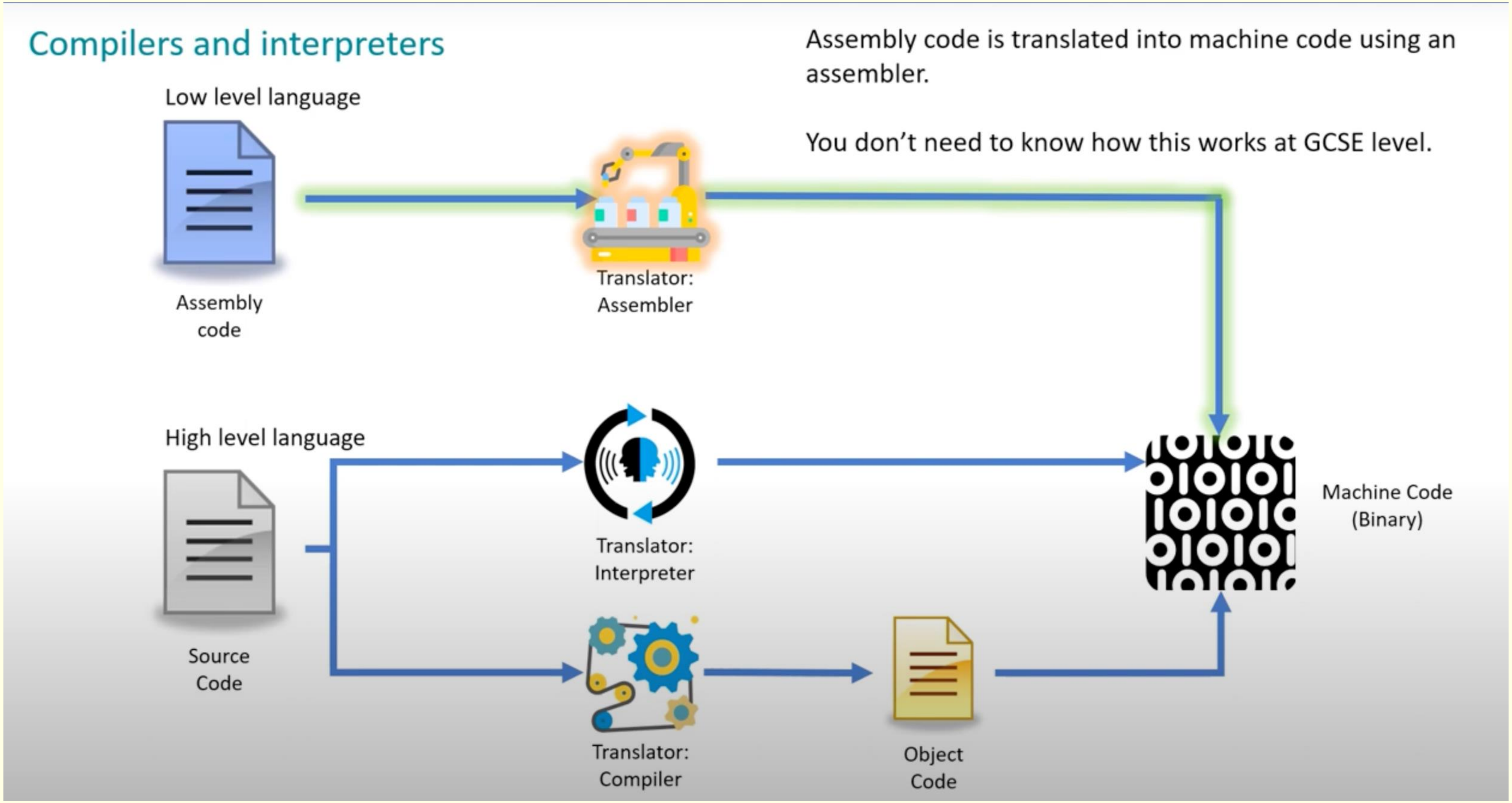
Assembler

Machine Code

Instruction set

Hardware

Processes associated with language translation



Compilers and interpreters

Assembly code is translated into machine code using an assembler.

You don’t need to know how this works at GCSE level.

Translators for high level languages

Complier

Translate **entire source** code all in one go into Machine Code and creates executable file (exe file) known as the **object code**

Interpreter

Translate and execute source code **Line by Line**

Comparison of Compilers and Interpreters

Compilers	**Interpreters**
Translate **entire source** code all in one go into Machine Code and creates an executable file (exe file)	Translate and execute source code **Line by Line**
Object code is faster processing	Slower processing
Reports all errors at the end	Reports errors as they occur
Whole program must be loaded into memory	Memory is only needed as each line of code is run
Used at the end of development (ready for shipping)	Used for development (aide debugging)
Source code hidden so it protects the code from being copied.	Access source code
Machine dependant only run on a computer with a particular machine architecture.	Platform independent as executes the source code just needs a suitable interpreter