Computer Fundamentals: CPUs, Fetch-Execute, Compilation

Dr Robert Harle

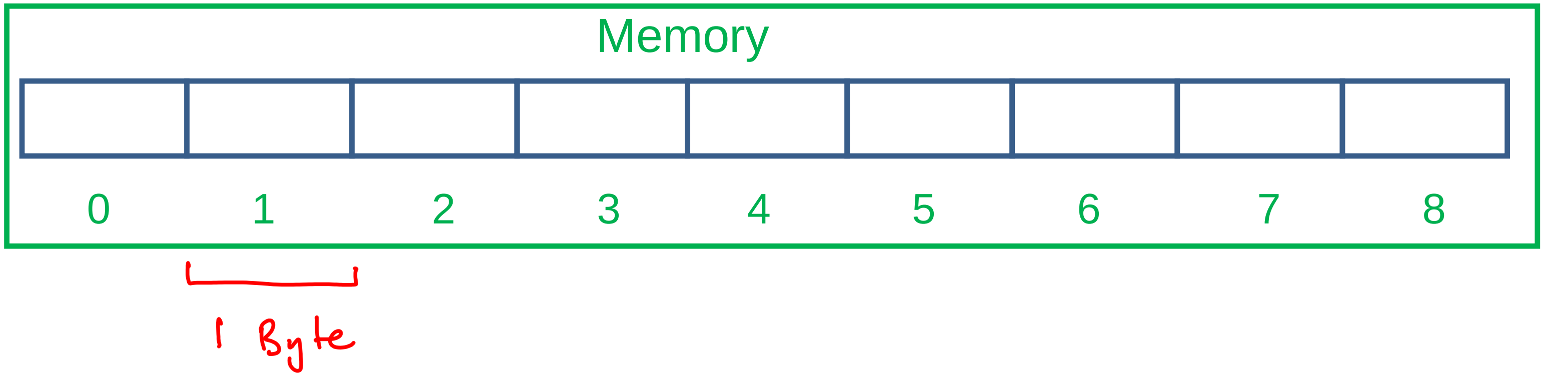
Today's Topics

- Stored Program Models
- The Fetch-Execute Cycle, registers, ALU etc
- Machine code, assembly, higher languages
- Compilers vs. interpreters

A Modern Computer

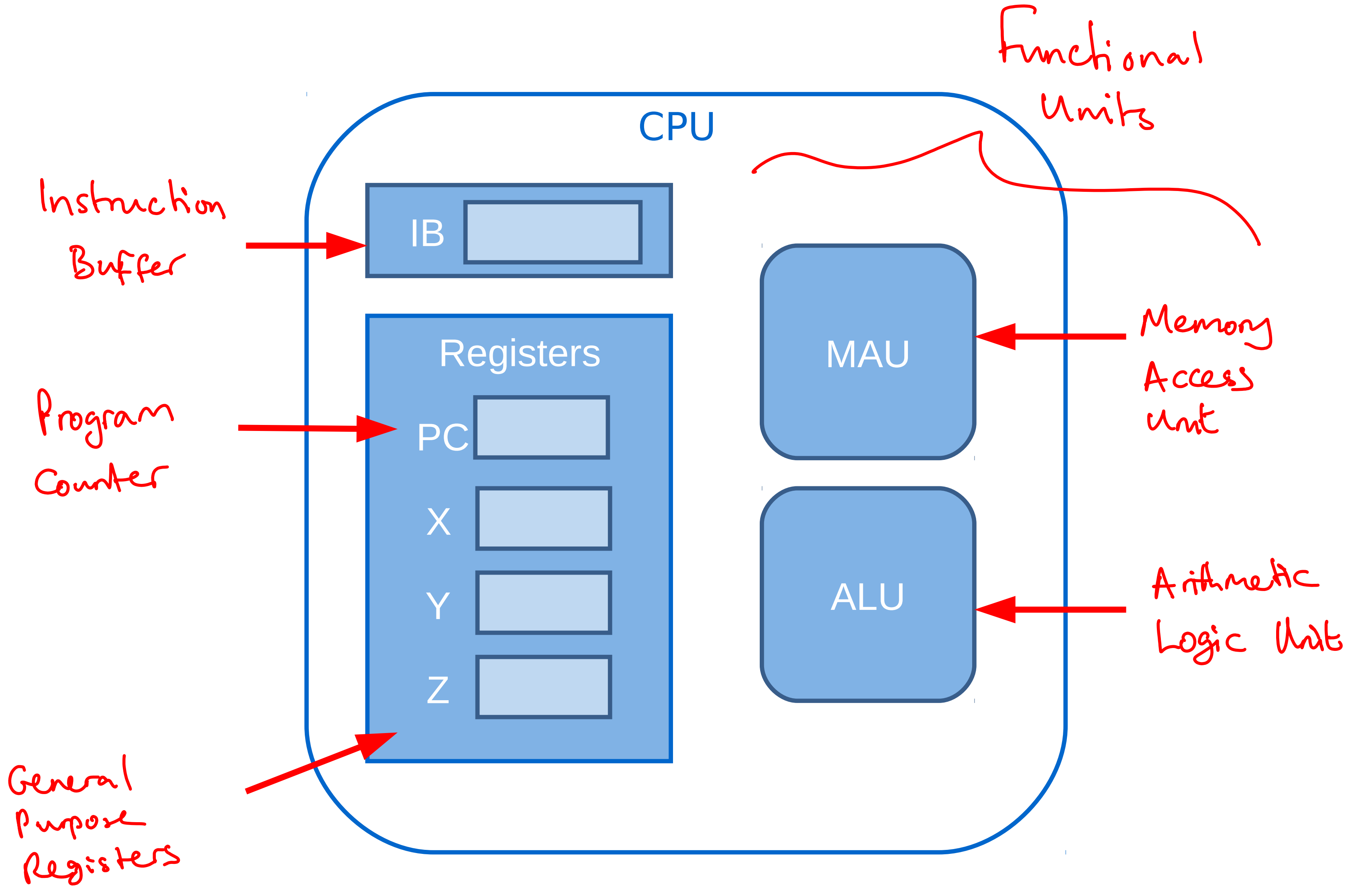
- A modern computer boils down to three fundamental things
  - Storage/memory – giving the ability to hold state (programs & data)
  - Processing unit (CPU) – giving the ability to manipulate state.
  - A program – giving the ability to instruct the CPU how to manipulate state in storage

Simple Model of Memory



- We think of memory abstractly, as being split into discrete chunks, each given a unique *address*
- We can read or write in whole chunks
- Modern memory is big

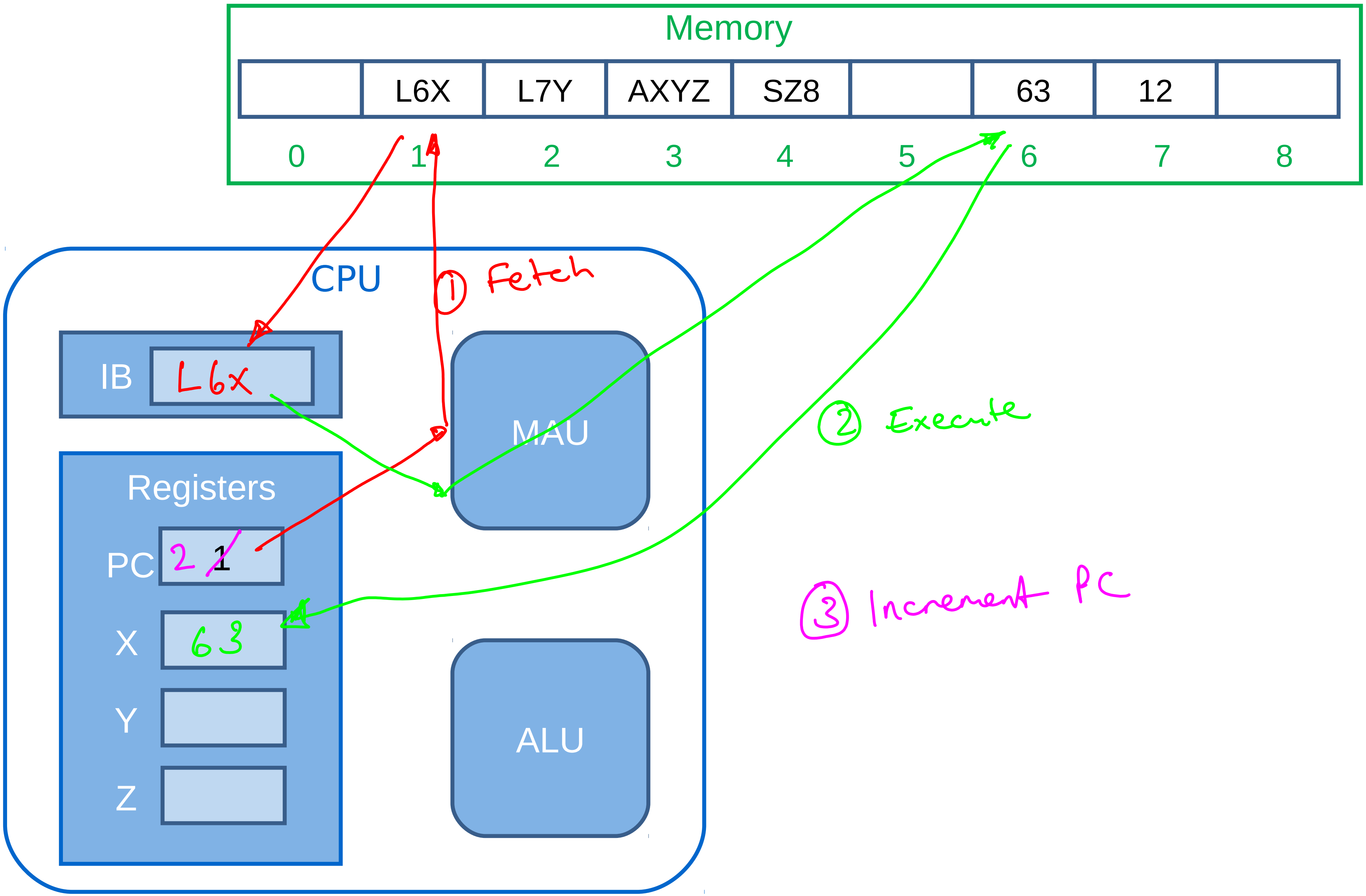
Simple Model of a CPU



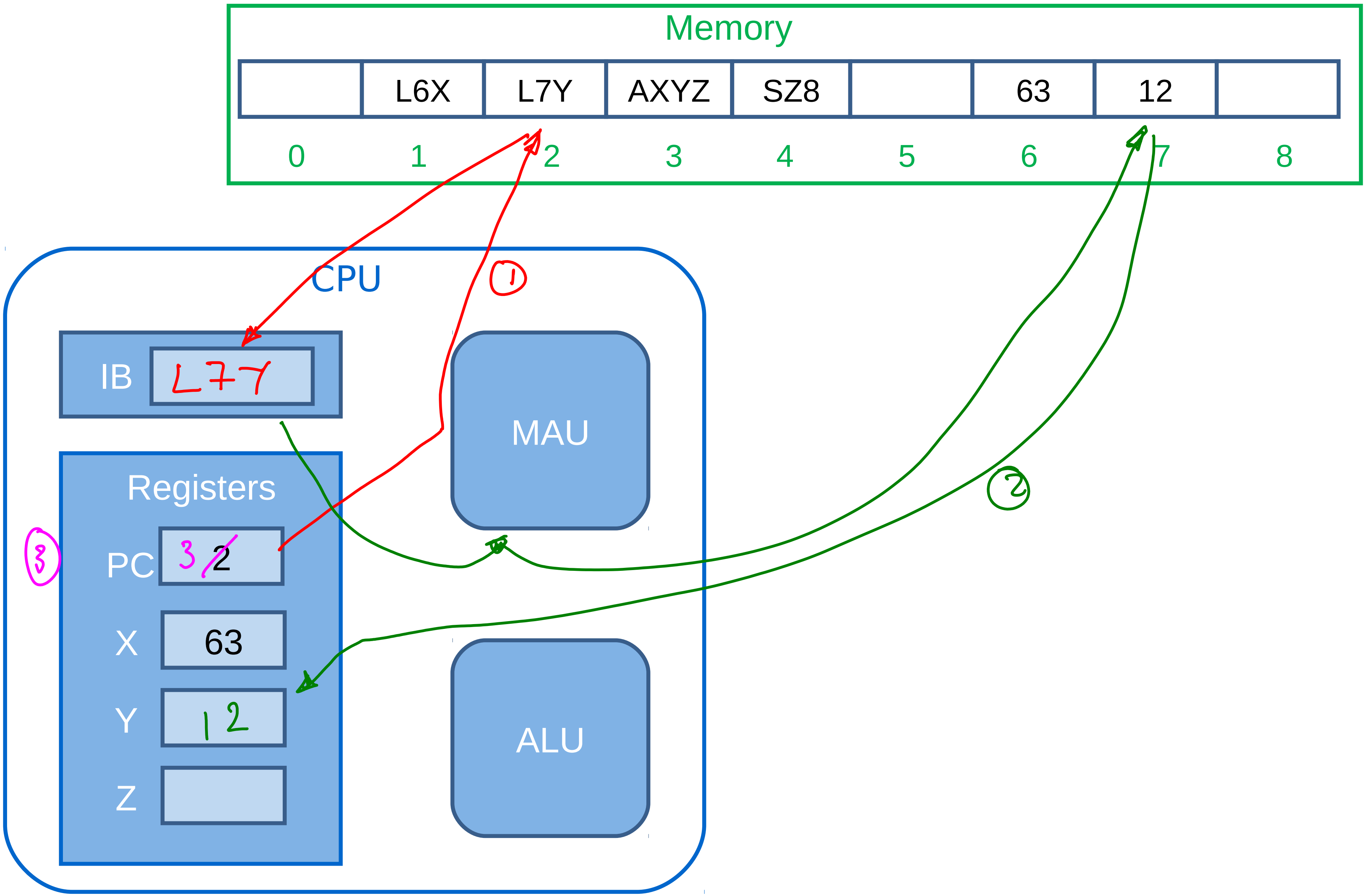
Simple Programming Language

- A program is just a sequence of instructions. The instructions available depend on the CPU manufacturer
- We will make up some very simple instruction labels
  - **L**IJ: **Load** value at memory address I into register J
  - **A**IJK: **Add** register I to J and put the result in register K
  - **S**IJ: **Store** register I in memory address J

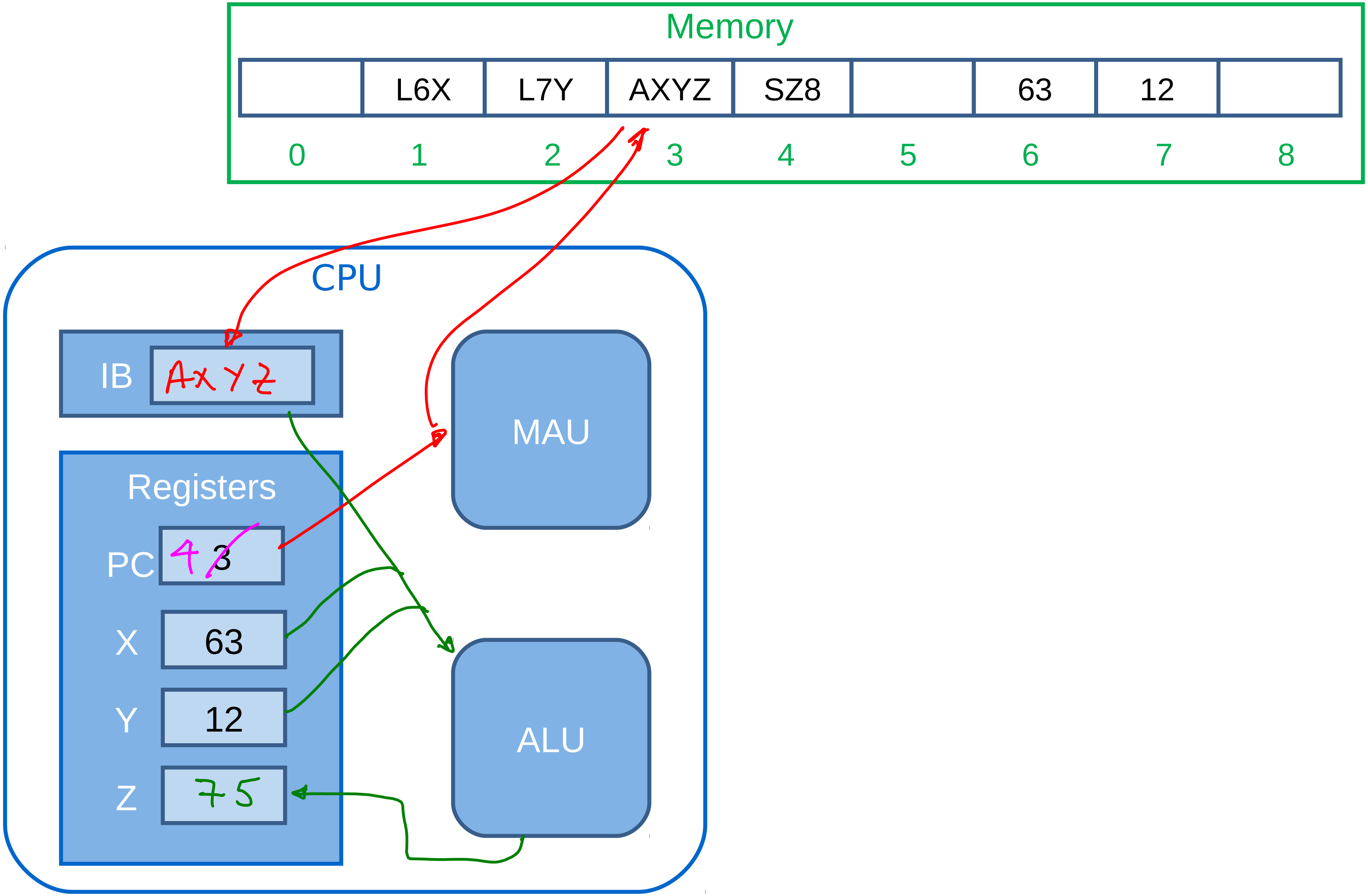
Fetch-Execute Cycle I



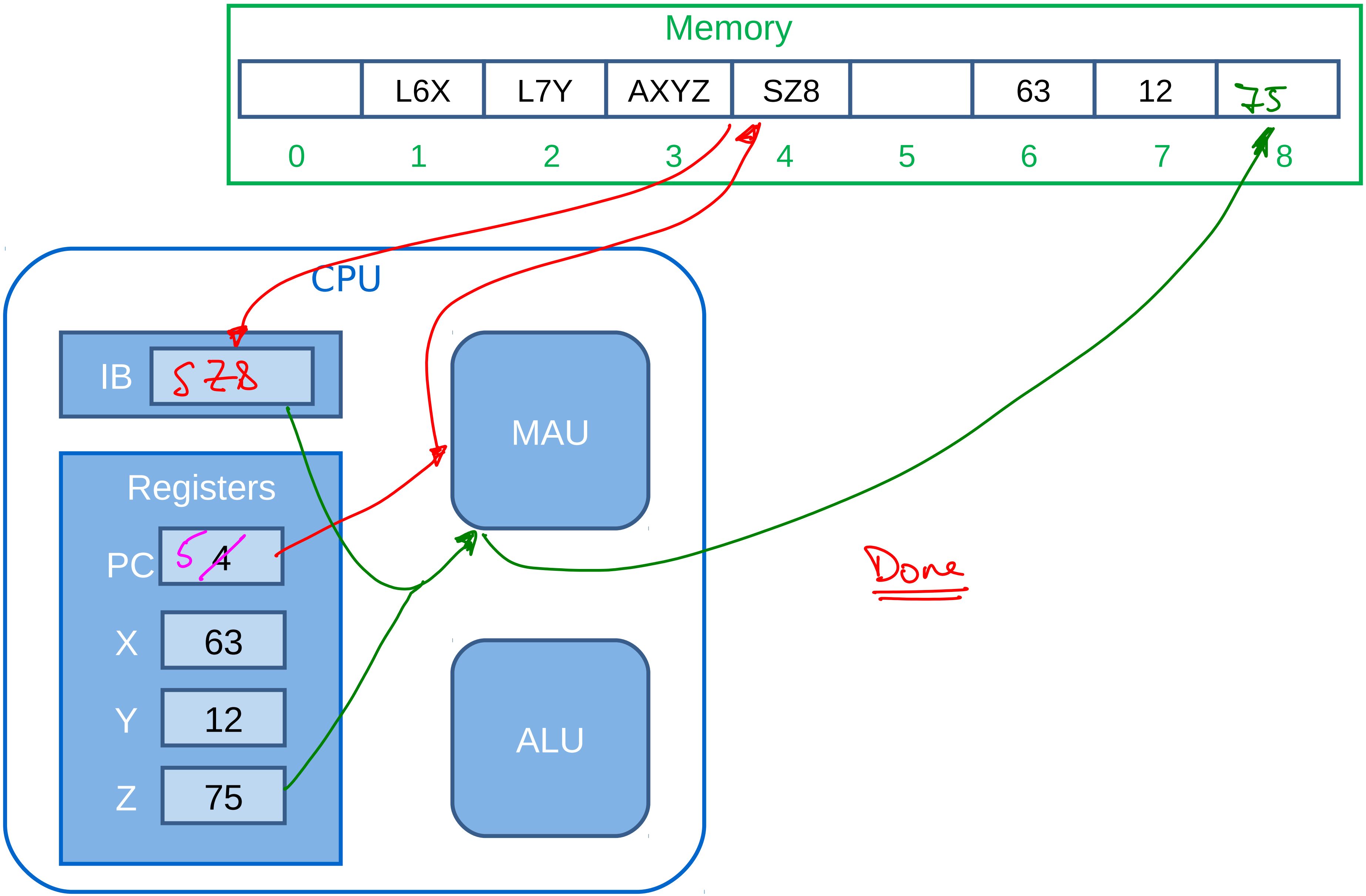
Fetch-Execute Cycle II



Fetch-Execute Cycle III



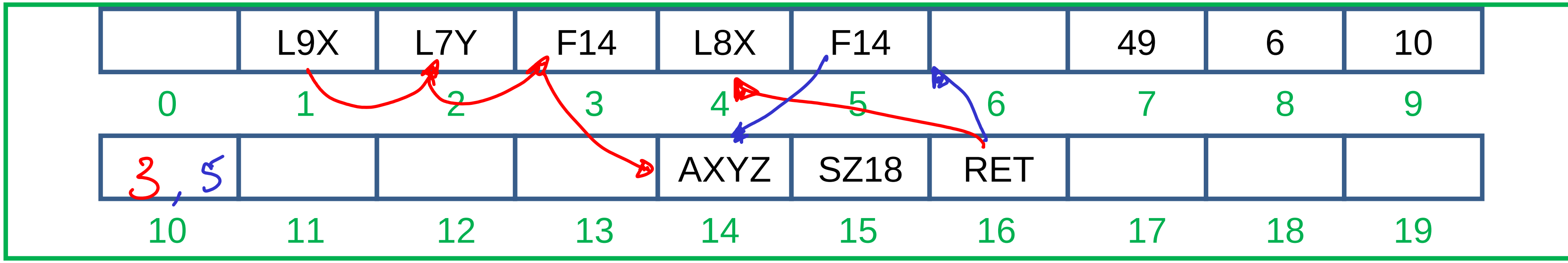
Fetch-Execute Cycle IV

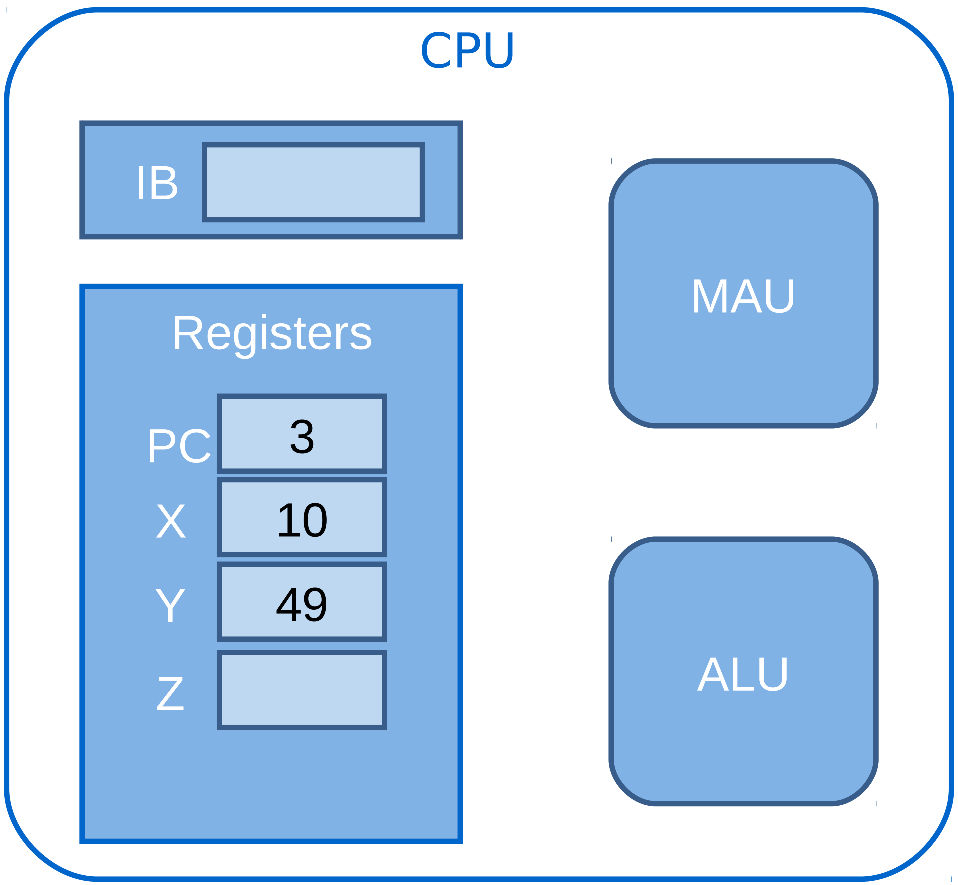


Add Functions

- **F**x: **Jump** to address x and run code from there
- **RET**: Jump back to where we left off

Functions

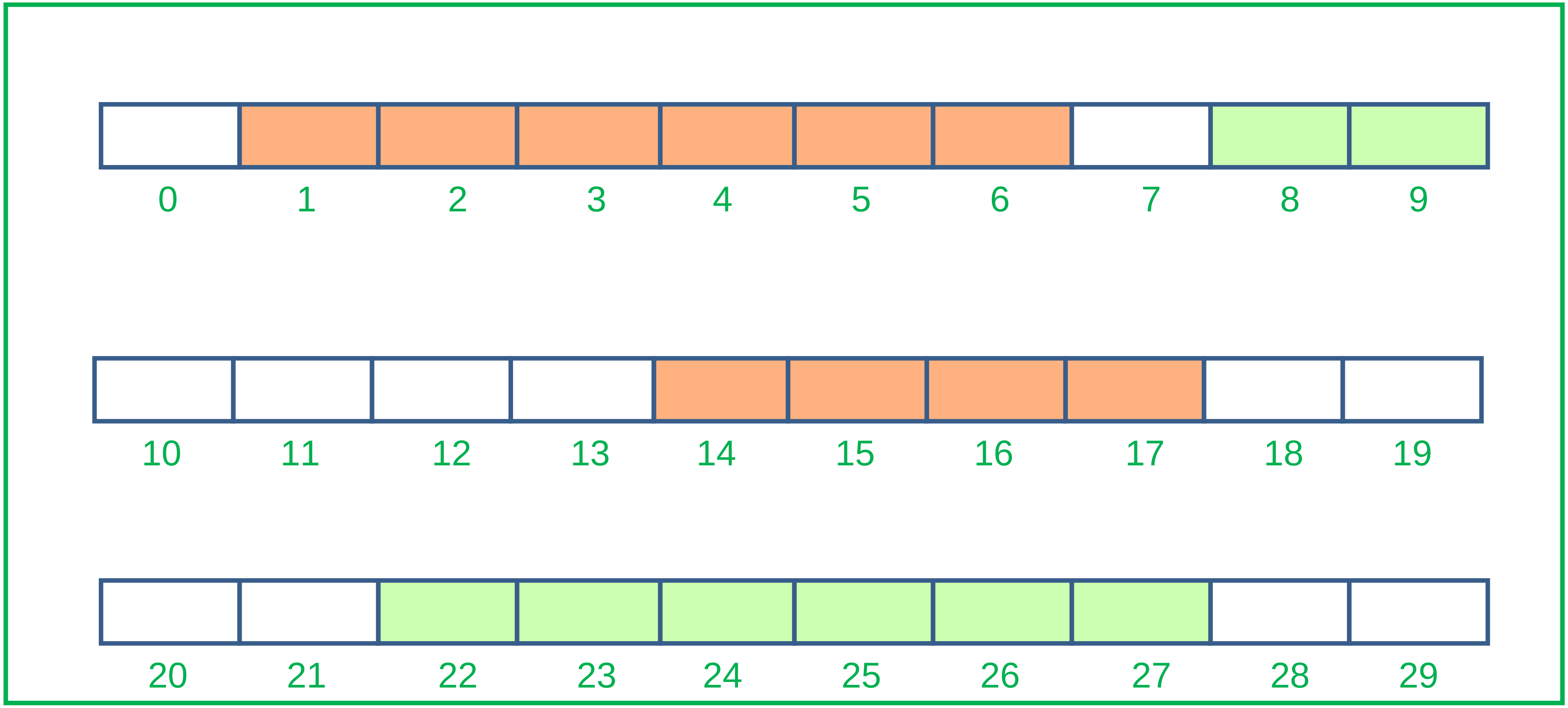




When we jump we use a special memory address (slot 10 here) to note where we came from so we can RET there

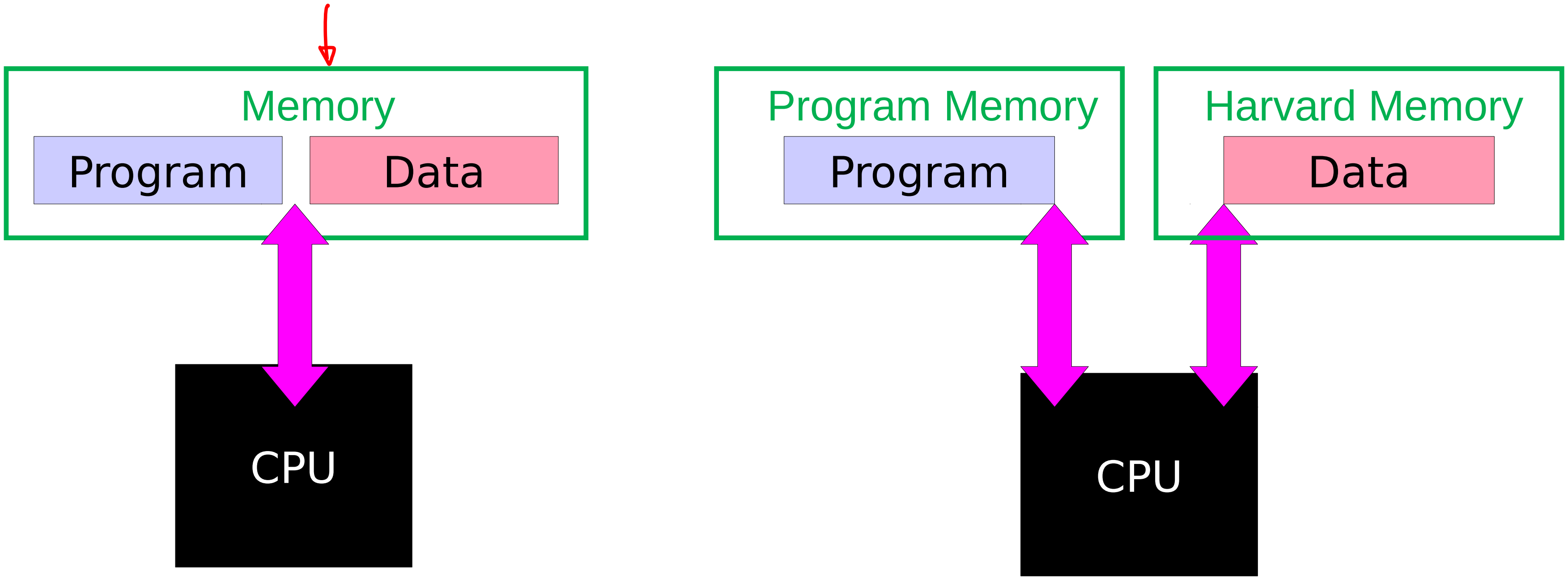
Viruses!

- The storage mixes together the program and the data... this is efficient but dangerous!



Storage Models

We use this model



Von-Neumann Architecture

Harvard Architecture

Choosing an Architecture

Von-Neumann		Harvard	
<u>Same</u> memory for programs and data		**<u>Separate</u>** memories for programs and data	
	+ Don't have to specify a partition so more efficient memory use		- Have to decide in advance how much to allocate to each
	+ Programs can modify themselves, giving great flexibility		+ Instruction memory can be declared read only to prevent viruses etc writing new instructions
	- Programs can modify themselves, leaving us open to malicious modification (viruses!)		
	- Can't get instructions and data simultaneously (therefore slower)		+ Can fetch instructions and data simultaneously

Instruction Sets

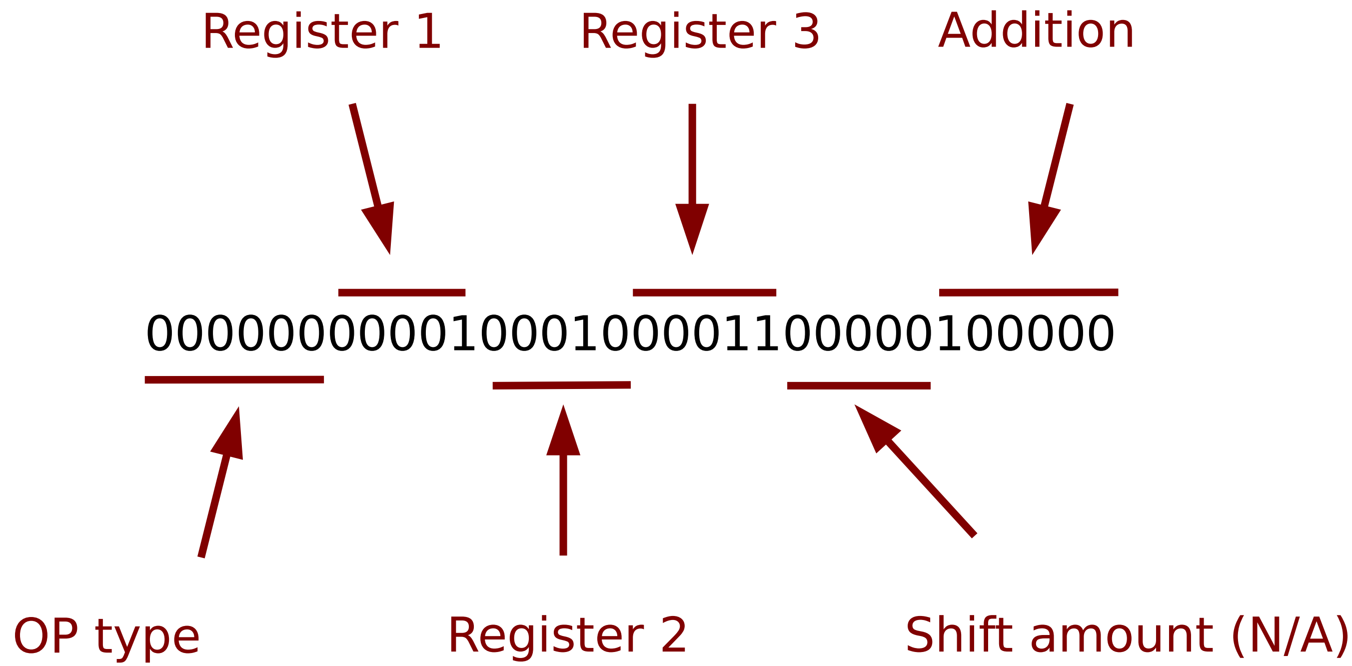
- The list of instructions a CPU supports is its Instruction Set Architecture (ISA)
  - Initially all used different instructions but there is clearly an advantage to using the same instruction sets
  - Intel's x86 set is a de-facto standard for PCs
  - ARM's v6 and v7 specifications are used for lower power applications (phones etc)

Writing Programs

- Computers don't store text instructions like L6X, but rather a binary code for each instruction
- Called machine code

Machine Code

- What the CPU 'understands': a series of instructions that it processes using the the fetch-execute technique
- E.g. to add registers 1 and 2, putting the result in register 3 using the MIPS architecture:



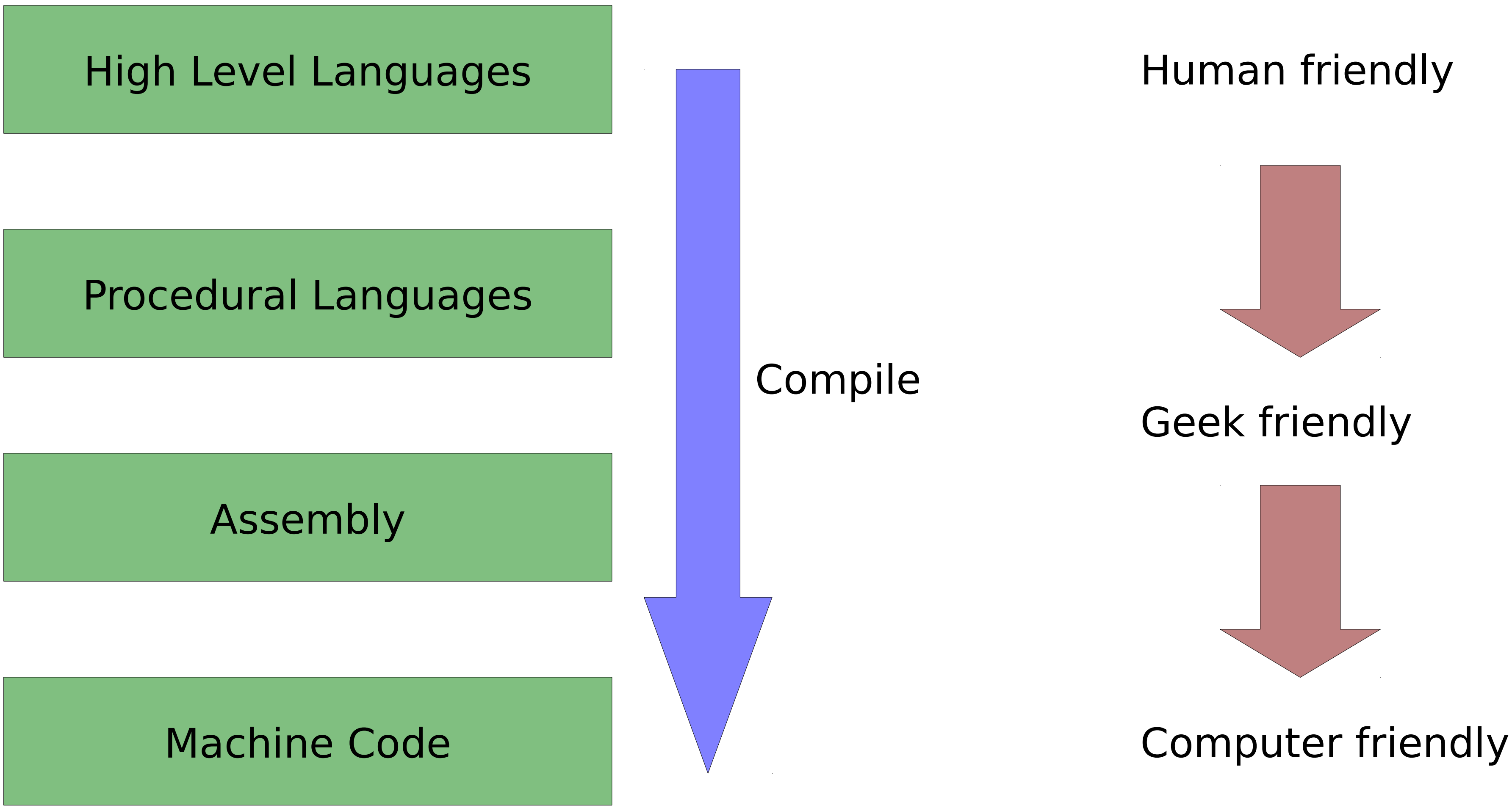
Assembly

- Essentially machine code, except we replace binary sequences with text that is easier for humans
- E.g. add registers 1 and 2, storing in 3:

```
add $s3, $s1, $s2
```

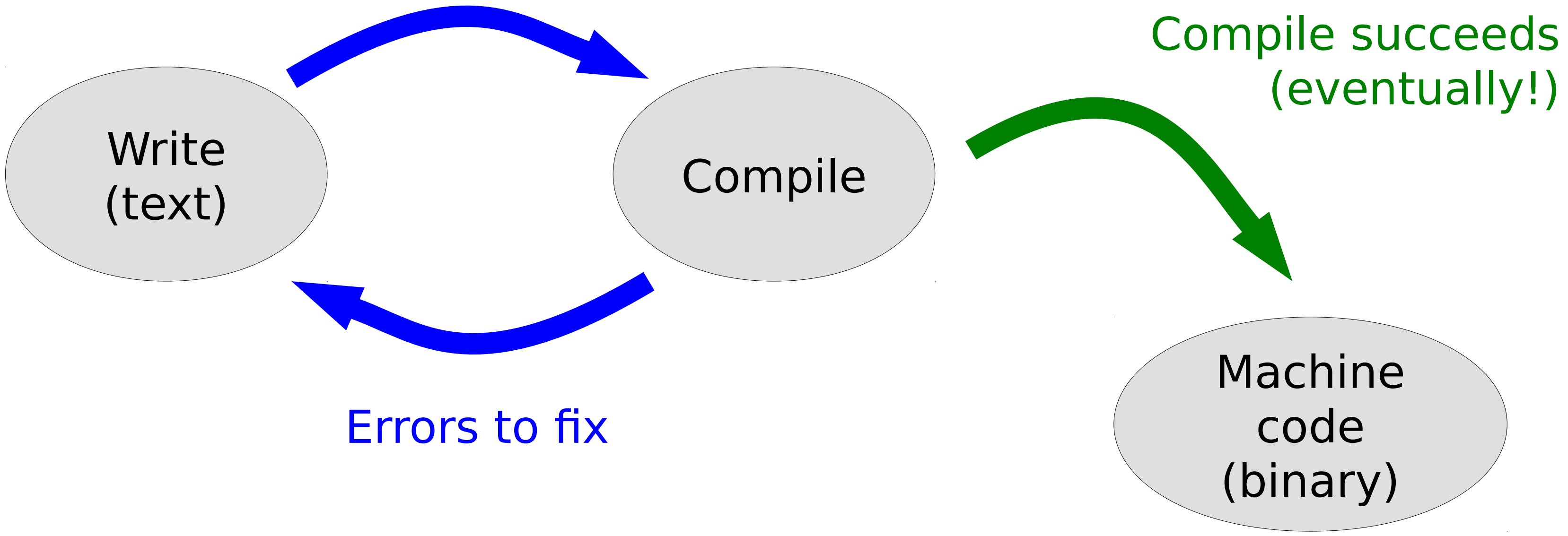
- Produces small, efficient machine code when **assembled**
- Almost as tedious to write as machine code
- Becoming a specialised skill...
- Ends up being architecture-specific if you want the most efficient results :-(

Levels of Abstraction for Programming

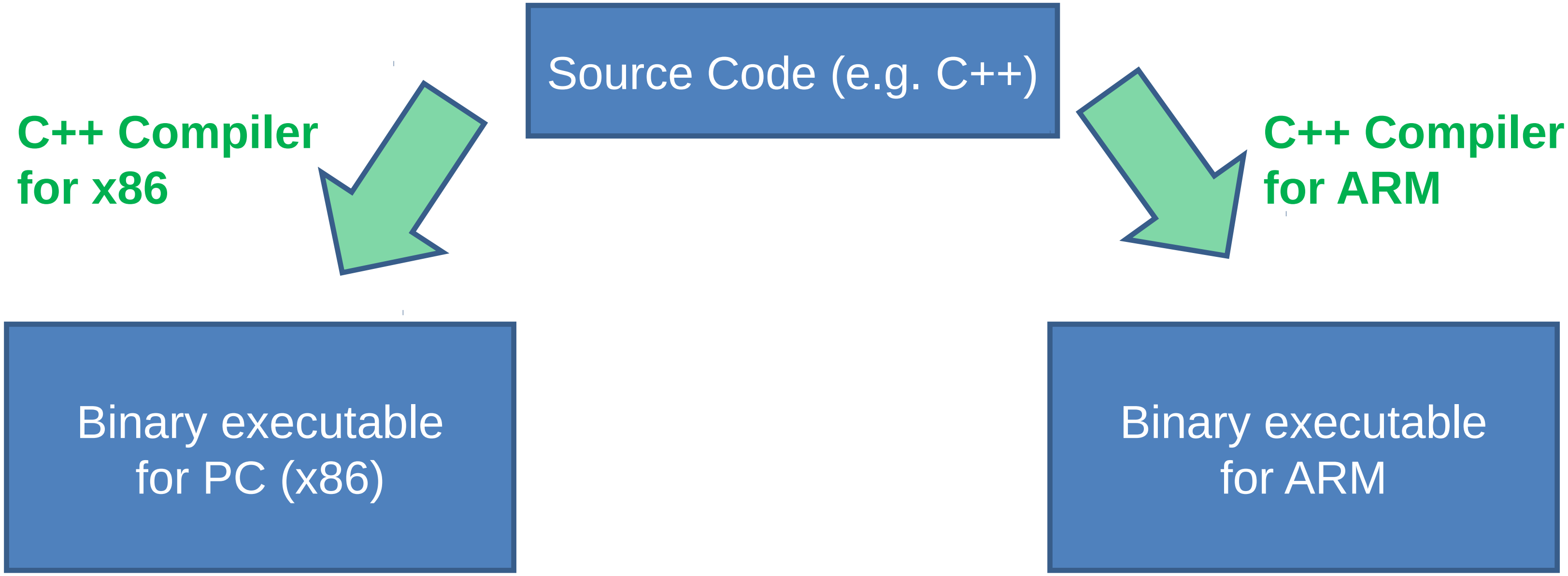


Compilers

- A compiler is just a software program that converts high-level code to machine code for a particular architecture (or some intermediary)
- Writing one is tricky and we require strict rules on the input (i.e. on the programming language). Unlike English, ambiguities cannot be tolerated!



Handling Architectures



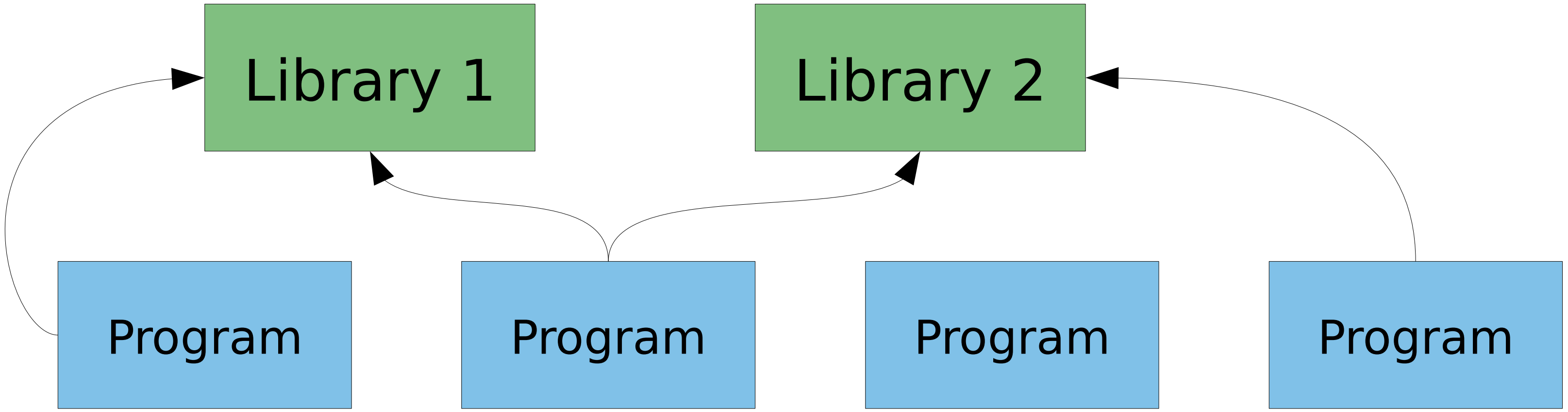
Interpreters

- The final binary is a compiled program that can be run on **one** CPU architecture.
- As computers got faster, it became apparent that we could potentially compile 'on-the-fly'. i.e. translate high-level code to machine code as we go
- Call programs that do this ***interpreters***

Architecture agnostic – distribute the code and have a dedicated interpreter on each machine	Have to distribute the code
Easier development loop	Errors only appear at runtime
	Performance hit – always compiling

Software Libraries

- Sometimes we package up useful chunks of code into libraries
  - Just a grouping of functions compiled to machine code
  - You can't 'run' a library – it's just a collection of functions
  - The intention is that each library is installed once per machine and many programs use it



Library Advantages

- Modern software makes extensive use of libraries
  - Makes the program smaller (references library functions rather than defining them itself)
  - Established libraries are well tested so fewer bugs
  - Experts in a specific area typically write the associated libraries so performance often better

Dependency Hell

- Programs are dependent on the libraries being present
  - They can be deleted
  - Or upgraded to incompatible versions
- Libraries can depend on other libraries too
- Can find yourself in a difficult state where you need multiple versions of libraries!

So what have we Learnt?

- Computers need three things: storage, processing and programs
- Computers just do a very simple fetch-execute loop very fast to run through some collection of machine code instructions one at a time
- The machine code is usually generated for us via a compilation step that takes in more human-friendly program descriptions
- We can potentially compile as we run the program: this is an interpreter
- Computers have software libraries that are collections of useful function implementations that can be used by any program

So what have we Learnt?

- Computers need three things: storage, processing and programs
- Computers just do a very simple fetch-execute loop very fast to run through some collection of machine code instructions one at a time
- The machine code is usually generated for us via a compilation step that takes in more human-friendly program descriptions
- We can potentially compile as we run the program: this is an interpreter
- Computers have software libraries that are collections of useful function implementations that can be used by any program