# Computing Degree Project Proposal

**Student Name:** Rhys Garman

**Course:** BSc (Hons) Software Engineering

**Project Title:** GAUNGAU (.gng) Script Interpreter for Custom 8-Bit CPU Arch

## Project Context

I intend to create a programming interpreter for my own custom designs 8-Bit CPU. I have been working on creating a CPU for several years, and I have now reached a stage where I can begin thinking about the software.

I am going to use Java to create the interpreter, this has been proven possible in an implementation by Strom and Aas (Strom & Aas, 2001). This is because it allows anyone to use the interpreter if the project was to be rolled out, as java is compiled on a VM, so it can run on any existing architecture (e.g. Windows, and iOS).

The point of an interpreter is to convert high level (human readable) code, into machine code or binary. These binaries can then be loaded onto the EEPROM (electrically erasable programmable read only memory) on the CPU and be executed. I will be creating my own language with its own syntax, structures, garbage control, etc.

Due to my prior experience with higher and lower-level programming, I will attempt to create this language with the high-level user in mind, by this, I mean I will base my syntax more off languages like Java and PHP, even though it is a direct link to the machine code like C or C++ is. By doing this, it will allow anyone to code their own program, and I will make it easy to see exactly what is going on. There will be nothing happening in the background, and everything will be visible to the user, from the interpretation to the binary output, and potentially further options to do with garbage control constraints, and manual addressing.

The issue surrounding the lack of more acute compilers are addressed in (Tanenbaum et al., 1983) which was also a motivator toward my project

## Specific Objectives

1. The interpreter must convert a custom programming language to the binaries required to be run on my 8-Bit computer
2. The interpreter should provide a deeper understanding as to the way in which programs are compiled and run computers and show the user each step of the way to execution
3. The interpreter should be able to offer corrections to the syntax if a syntax error is present. This should be displayed to the user when they attempt to compile their code.
4. The interpreter would automatically upload the binaries to the 8-Bit EEPROM, so the user does not need to handle the binaries if they didn’t feel up for the job

## Potential Ethical or Legal Issues

With any software that relies on the input from users (the source code in this case) there are many possibilities for incorrect inputs to be made. It is my job to have nothing unhandled if an incorrect input is provided. The source of the interpreter will be unavailable to anyone using the tool, so they will not be able to adjust if they find an error in my work. Therefore, intensive testing will be required to ensure nothing is left unhandled.

## Resources

As this project is purely software, the only resources that I will require are man hours, and software to write the interpreter. My preferred IDE is VS Code, this caters for my every need and also has the ability to run and debug Java as a free addon.

On an estimated hourly rate of £20 per hour. With a time estimate of 350 hours, I predict this will be £7000 worth of man hours.

While this section of the overall package is software, I have previous developed the computer for the code to be run on, and this will be a required resource as well.

## Potential Commercial Considerations - Estimated costs and benefits

The main cost of this project will be the actual 8-Bit computer. This is the only physical thing that is needed. This is to be purchased, as a kit, and it would be cruel to make the user purchase the interpreter as well. As such, the interpreter will be made available for free. I will be building the interpreter in Java, this means I can distribute the executable without sharing the source.

While the pricing of the 8-Bit computer itself has little baring on this project, it will be close to the final retail price of the package that this project contributes to.

* PCB costs £70 for 5 (£14 apiece)
* Components: £50
* Total price for one package is £64.

Most likely, this would be retailed for around £85 per unit. Meaning a profit per unit of £21.

## Proposed Approach

The way in which any script is interpreted is the same across all programming languages (Isaac Computer Science, n.d.):

1. Lexical analysis
2. Syntax analysis
3. Code generation

I will use this same approach when interpreting my programming language. As such, my first step will be writing an implementation for the lexical analysis. As this is an educational tool, I will write these in stages, so that the user will be able to see the outcome of each step and how to affects the code they are writing. Lexical analysis is all about removing code that will not be executed, such as comments and blank lines, after this, the commands will be tokenised,

Next is the Syntax analysis implementation, again, I will add this as a separate stage so the user can see the code as the lexical level, and then at the syntax level. This is the stage where the tokens can be checked that they follow the rules of the language and what the keywords actually represent.

Finally, the code generation is run, this is where the commands are converted to their respective binary codes and outputted to the user to be burned to the CPU.

As you can see, no logic has been done on the code, it has simply been converted from one form to another. While it would be incredibly easy to do all of the computation on the machine being used to interpret the code, that would void the purpose of the project. (Hadžiavdić, 2021) (‘Modern Compiler Design’, 2000)

## References

Hadžiavdić, S. (2021, June 23). *How to Approach Writing an Interpreter From Scratch*. Toptal Engineering Blog. Retrieved 1 December 2021, from https://www.toptal.com/scala/writing-an-interpreter

Isaac Computer Science. (n.d.). *Stages of compilation*. Isaaccomputerscience.Org. Retrieved 2 December 2021, from https://isaaccomputerscience.org/concepts/sys\_trans\_stages?examBoard=all&stage=all

Grune. (2000). Modern compiler design . Wiley.

Strom, O., & Aas, E. (2001). An implementation of an embedded microprocessor core with support for executing byte compiled Java code. *Proceedings Euromicro Symposium on Digital Systems Design*. Published. https://doi.org/10.1109/dsd.2001.952346

Tanenbaum, A. S., van Staveren, H., Keizer, E. G., & Stevenson, J. W. (1983). A practical tool kit for making portable compilers. *Communications of the ACM*, *26*(9), 654–660. https://doi.org/10.1145/358172.358182