Justification of investigation

There are a number of reasons why the problem I am trying to solve needs a computerised solution. Initially, the existence of the problem itself is twofold: for the teachers and for the students. The stages of compilation and execution of machine code are quite complex and very abstract, and teachers can have a hard time making it understandable for the students they are trying to teach. A simple lecture format is insufficient, and there are no easy ways to make the process more understandable. Also, this problem appears on the side of the students, in that they can find it difficult to truly understand what is happening because it is a low-level process they cannot watch. They may be able to memorise enough to be able to answer exam questions, but truly understanding is much more difficult.

Any attempt at a solution to this problem without using a computer is almost certainly useless. It is possible for an exercise to be given to students where they compile some code like a compiler would by hand, but this is time consuming, boring and only works for very small pieces of code. It may aid understanding somewhat, but since the entire point of compilation is reducing large, high-level tasks to a low-level format for the CPU, there isn’t much of a point in compiling “a + b = c”, which is the limit of this method. In the same vein for execution of machine code, they could execute some very simple low level code by hand, but both the size of program and the instruction set are extremely limited.

Existing tools are unable to solve the issue of understanding. For understanding compilation, most compilers are mysterious behemoths of software. What happens in the middle is not only difficult to understand, but also impossible to see as it happens. Compilers are written for efficiency, not for educational aid. In understanding the execution of the machine code by a CPU, there is only one major existing tool that can be used as a hands-on learning system, and it is the LMC. However, it has an extremely limited instruction set and realistically cannot be used by students for any task more complex than listing the Fibonacci numbers.

My project would solve both of these problems. A student can type in a high-level language and then watch the compiler as it works through each stage of compilation, finally producing a form of specially designed bytecode output. Alternatively, the user will be able to generate bytecode from an LMC-style assembler, if they want to experiment directly with low-level workings. Finally, the user can place the bytecode into the interpreter, which will show how the CPU would work its way through the code, including showing the contents of memory and the registers.

This is a full solution to all of the problems above, and should be a significant help to any Computer Science student or teacher who has to do this topic.