Success Criteria

Success criteria are specific, measurable targets which I will attempt to hit and will use to evaluate my project once it is complete. If not enough of these success criteria are met, my project will not have been successful.

Some of these success criteria are critical to the project, and all of these must be complete. Their numbers have been marked with asterisks to indicate their necessity. If any of these critical success criteria are not met, then the project overall will not have been successful as they are basic elements of any solution to the problem I am trying to solve. Other success criteria are also presented, and I will not consider the project sufficiently successful if less than 80% of these are included. They are not individually critical to the program, but overall their inclusion is necessary for a sufficiently useful program.

|  |  |  |
| --- | --- | --- |
| No. | Success criteria | Justification |
| 1\* | The compiler can take a subset of C which includes arithmetic, printing and accepting user input, variables, if statements and for and while loops, as well as filtering out. | This is the basic set of statements required for a Turing-complete implementation of C (though this issue gets complex). Realistically, any algorithm can be programmed with these, and without all of these, there will be serious issues in the language’s usefulness. |
| 2\* | The compiler produces somewhat human-understandable bytecode in a form specified in “3.06 Decomposing the Problem”, roughly based on Intel syntax for x86. | This language is an ideal mid-way point between realism (as it is based on actual assembly for a real architecture) and simplicity, as I will choose a smaller, more understandable instruction set. |
| 3\* | The assembler must take the assembly and produce a custom form of bytecode specified in 3.06, based on the x86 instruction set and readable by the interpreter. | This is a basic requirement for the assembler to be functional and work with the rest of the toolkit. |
| 4\* | The interpreter must run the bytecode and simulate a basic processor, including registers and memory. | If this does not happen, then the compiler and assembler have no use. This is necessary to run their results. |
| 5 | The compiler should be able to understand more complex C concepts, such as pointers, structs, functions and pre-processor directives. | This is to make the programming experience as unrestricted as possible, so that the user is not limited. It also allows the user to see how these concepts are implemented at a lower level. |
| 6 | All of the programs should be able to read directly from and to files where applicable, rather than just through text boxes. | This will speed up the process when the user does not want to go through any hassle, and also is more realistic. |
| 7\* | The compiler needs to show each stage of the compilation process and what is being done to each line in the program. | Educationally, this is the purpose of the program; without this it is just another C compiler, but this makes it an educational tool. |
| 8\* | The assembler must show how it is interpreting each line, including an English description of its meaning and showing the correct opcode. | As above this is what makes the assembler a useful teaching and learning tool. |
| 9\* | The interpreter must show a GUI, displaying the elements of the processor and memory as it runs through the program. | This is to make the program similar to the LMC, simply more advanced and realistic. It is like seeing the inner workings of the CPU. |
| 10 | All of the programs should be able to run as fast as possible, without showing a GUI at all, if the user requests it. | This is a general utility that means the user is not forced to watch the animation every time if they would rather simply perform the compilation/assembly etc. |