The Soul of a New Machine: A Robust Toy Virtual Machine – the core

This question was inspired by another question I answered here on code review An attempt at a toy Vm (<https://codereview.stackexchange.com/questions/244566/an-attempt-at-a-toy-vm>). The code in that question was fairly well written, and the problem intrigued me, and there was a working model to build off of. If the author of that question is offended, consider this, one of the highest forms of flattery is copying ones invention, your original was good.

Two items that I didn’t mention in the review was that the program wasn’t modularized enough (the code was tightly coupled between modules), and that the vm struct allowed too much access to the internals of the virtual machine. It may be that the author of that question has practiced too much object oriented programming and may not realize that a struct is not a C++ class, and doesn’t protect the internal implementation of the class.

Another thing that I didn’t mention in the review was that there should be a user friendly method to create programs to run on that virtual machine, programming in hex may be easy for some, but not so easy for others such as myself. My version of the Virtual Machine question is presented in 2 parts, this part that contains my updates to the original portion of the question (vm.h, vm.c and main.c) and this second question that presents a user interface for the program. This files included in the current question are:

* main.c
* virtual\_machine.h
* virtual\_machine.c
* opcode.h
* opcode.c
* common\_program\_logic.h
* common\_program\_logic.c
* arg\_flags.h (but not arg\_flags.c which is presented in the other question).

The other question contains the major extensions to the program which is basically user interface and command line parsing. The files presented in that question are

* arg\_flags.h
* arg\_flags.c
* human\_readable\_program\_format.h
* human\_readable\_program\_format.c
* console\_text\_human\_readable\_scriptor.h
* console\_text\_human\_readable\_scriptor.c

If you don’t find the definition of a function in this question it will be in the other question, you can also get the whole project from my GitHub repository.

This question also includes implementation of suggestions from the other answer in the virtual machine module (opcode, array driven functions and passing the buffer directly to the vm). One of the suggestions that I didn’t implement from the other answer was processor flags, because I didn’t know what was needed.

I’ve done my best to modularize the code as well as optimize performance, my questions are;

* Do you see any indication of tight coupling between modules?
* Are there any performance improvements I can make?
* Are there too many static global variables?
* How can I get the correct array size for opcode related arrays from the opcode enum?
* My concepts for C programming are originally from 1984, are there better ways to do what I’ve done in either C90 or C11?