# Arduino Programming

## Introduction

One fun aspect of using computers is that you can create programs that will run on them, which is, for example, very complicated on mobile devices. My project aims to create a way to program an Arduino board without needing any other electronic device. The goal of this project is by no means to make programming for Arduino boards simpler or more efficient but rather to have a broader reach because it's 'fun'.

## Development

The main challenge is to adapt to the Arduino environment, which means very little RAM (2k bytes) and even less permanent EEPROM memory (1k bytes). The second challenge is to determine the form the code will take and how to make the Arduino execute it.

There are generally two types of programming languages: compiled languages and interpreted languages. The first option to program the Arduino with itself would be to use a compiled language (as when programming it with a computer). However, using an existing compiler is not really feasible since the Arduino has a much more limited environment than a computer, and writing a compiler seems to be an even more complicated task. So, the remaining solution is to use an interpreted language, much simpler to implement (using an existing one is still unthinkable in such a restricted environment).

So, we need to implement an interpreted language. I'm going to create a language in the form of instructions (1 instruction = 1 action), somewhat like assembly language. I've decided to use instructions written in hexadecimal (so that 1 byte contains 2 characters), allowing us to exploit the 256 possibilities provided by 1 byte. The instructions will be 16 bits: 8 bits for the operation (2 hexadecimal digits) and 8 bits for the parameters. Since the language resembles assembly language in its organization, I've decided to use 4 'registers' of 16 bits each and have made available 256 memory spaces of 16 bits each to serve as variables.

You can see all the instructions implemented in the attached document "commands.pdf".

A program will consist of a maximum of 128 instructions (equivalent to 256 bytes in memory) to allow for storing 3 in EEPROM memory and minimizing RAM usage.

I've decided to write the code editor program in this interpreted language to demonstrate that even with only 128 instructions, you can accomplish a lot. The program contains 127 instructions, and you can see them in the file "test4.c," which was used to save it in the Arduino's memory.

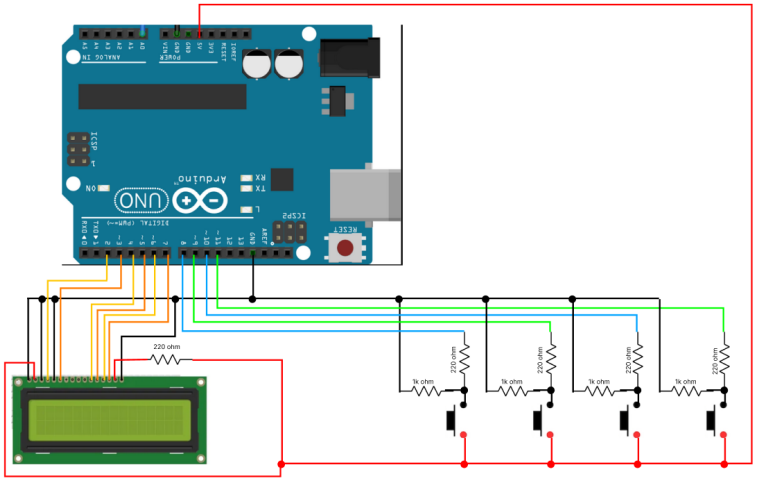
The Arduino program will simply ask which program in memory to run (1, 2, or 3) and then interpret it.

## Electronic assembly

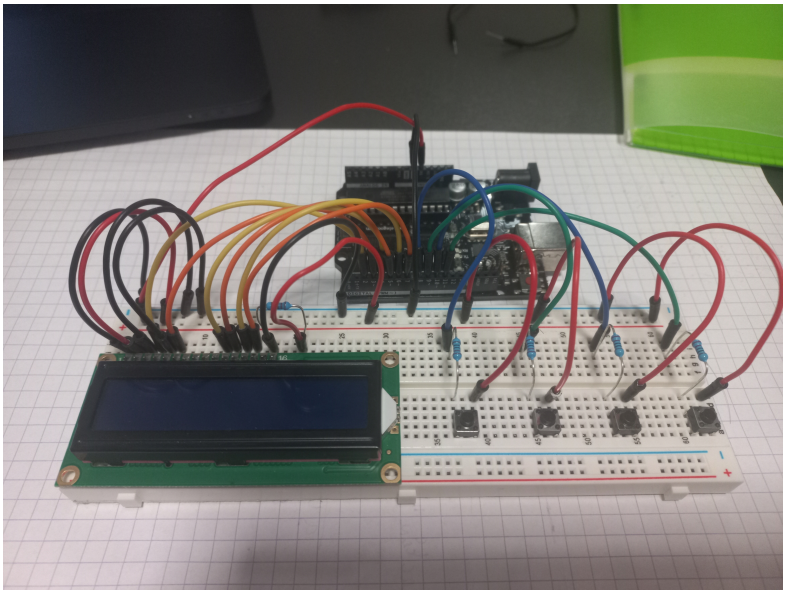
The setup is quite simple since the majority of the project is related to creating the interpreted language, which doesn't require additional hardware apart from the Arduino. So, I'm using the following hardware:

- 1 Arduino Uno
- 4 resistor 1k ohms
- 5 resistor 220 ohms
- 1 LCD1602
- 4 push buttons
- 1 bread board
- some male/male cables

I used this wiring diagram:




And there is the result:



## Code

All the code files are attached to this report:

- main.cpp

It contains the setup part (where only the LCD is initialized) and the loop like all Arduino programs. In the loop, it asks which program to launch and then executes it.

- prog.hpp and prog.cpp

These functions contain the logic to interpret programs.

The "start_prog" function loads the requested program from EEPROM into RAM and initializes the variables necessary for program execution.

The "run_prog" function simply executes the instructions one by one using the "exec_inst" function and updates the program counter.

Finally, the "exec_inst" function's mission is to decode the instruction and call the function that will execute it.

- debug.hpp

Usefull debug macro I used that format and print but placing the string in the flash memory with “PTSR” to save RAM because the string will not be modified. Else, with as much printing as in prog.cpp the Arduino will not run correctly because the RAM would be full.

- test4.hpp and test4.cpp

Parts used to save the editor in EEPROM memory. To use it: simply uncomment the test4 lines in main.cpp and comment all in the loop in main.cpp.

## Conclusion

This project was really "fun" to carry out because it's something I really wanted to try, and it's quite different from what we've learned to do, even at my school in France. Moreover, with a bit of experience, I never got stuck and quickly found solutions.

I believe I have truly succeeded in the project because it works well: I was able to write the Hello World and Blink programs from the Arduino board. However, it is still quite complicated compared to coding from a computer, even though that wasn't the goal, and I don't think it can be resolved due to the very limited RAM and EEPROM memories.