4CCS1CS1 Computer Systems

Lab Sheet 1

Getting to know your Arduino

Welcome to your first CS1 Lab. In the lab sessions throughout the semester you will learn how to use your Arduino Nano Microprocessor to carry out basic computations using assembly code. In this lab sheet you will find the exercises for your first lab class. You are always expected to finish all exercises before the next lab session. So, if you don't have enough time in class to finish them all, you'll need to complete them in your own time.

---

1 Getting organised

- Log in on the lab computer using your k-number.
- Create a folder in Documents that will contain all the resources you need for your CS1 labs. On Ubuntu, the operating system on the lab computers, this can be done by going to the toolbar and selecting Places, and then selecting Documents from the drop down menu; this will open the file manager, right click on the white space and select 'New Folder', and name it CS1. Then, download and save the following files to the new folder, all of which can be found on the CS1 KEATS page. An internet browser can be found under Applications on the toolbar.
  The files you should download are the following.
  - Arduino Schematic Circuit Diagram
  - Atmel ATmega328 Datasheet
  - Atmel 8-bit AVR Instruction Set Manual
- During your first lab, you will be given an Arduino Kit, make sure to bring it to all your CS1 labs. Check that you have all the following components in your kit. Let your lab TA know if there's something missing from your kit.
  - The Arduino Nano
  - USB to USB mini cable
  - Wire jumpers (10)
  - Breadboard
  - Resistors (10)
  - LEDs (at least 7)

2 First Assembly Program

To get started with writing assembly code for the microcontroller, let's look at the following example program.

```
1 .equ SREG,0x3f          ; define SREG label
2 .org 0
3 main:     ldi r16,0       ; set register r16 to zero
4           out SREG,r16    ; copy contents of r16 to SREG
5 mainloop: rjmp mainloop   ; jump to mainloop address
```

Listing 1: Clear Status Register

As you can see, assembly code is often not very readable for humans, especially compared to high-level programming languages (such as Java and Python). Assembly code is low-level, meaning it deals more directly with the hardware components of the computer rather than abstract concepts humans normally think about when programming. So, don't worry if you don't understand any of the code straight away, it looks more complex than it really is! Let's examine what the code is doing.

This program does little more than clear the status register referred to by `SREG` (on lines 3-4), and then loop endlessly (line 5).

- On line 1, we see the keyword `.equ`. This command instructs the assembler to treat the label `SREG` as the constant number `0x3f`. The `0x` part specifies that the number is in hexadecimal format, the `3f` is the value itself.
  By assigning the numeric value to the label, it means that for the rest of the program, whenever SREG is used it will be substituted with the value `0x3f`.
  If you've already covered the first binary lectures, what is the decimal value of `SREG`?
- On line 2, we see the keyword `.org`. This specifies the start address. The assembler uses this to establish where the program starts when translating to machine code. Here, we're setting the address to 0, which is the first address.
- In green, are comments. These are delimited by a leading semi-colon (`;`). These don't affect the running of the code, but are there to explain to a human reader what the code is doing.
- On lines 3 and 5, we see two *address labels* (`main` and `mainloop`) followed by a colon (`:`). These are signposts in the code that we can use to refer to specific parts of the code.
- On lines 3 and 4, we see that the program uses register `r16`. Registers are small but very fast memory units. The Arduino Nano has a range of registers that we have access to as programmers.
  Register `r16` is a general purpose register, used for holding data in the microprocessor's CPU.
- Highlighted in blue we see three instructions are used: `ldi`, `out` and `rjmp`. The comments on lines 3–5 explain what these instructions do.

As assembly programmers, we can make use of the instructions described in the **instruction set manual**, which you've downloaded. The manual describes all of the instructions we have available on this specific device. Take a look at the manual, and try to locate the instructions we've used in this program. For each instruction, you'll see a short description of what the instruction does written in English, and then a technical specification. We'll be using some of these instructions in future labs.

Discuss with the person next to you to make sure you both agree on what this program does. If you're both not sure what the program does, ask a TA for help!

3 Assembling and Writing to Memory

The example program in Listing 1 will be the first program that we will run on the Aruduino. However, it can't be run by the Arduino in the current format — it first needs to be translated to machine code that the Arduino can understand. We will use the a series of commands on the command line to translate the assembly program to machine code.

1. Open a text editor. You can do this in Ubuntu by going to the Applications menu on the toolbar. There is a text editor helpfully called "Text Editor", found under the *Accessories* submenu.
2. Now type out the program in Listing 1 — don't copy and paste the code as you may copy additional or incorrect characters. Save the file with the extension '`.s`' (e.g. `clr_sreg.s`). You should save the program to your CS1 folder.
3. Open the terminal (you can find this in Applications, under *System tools*), and navigate to the folder where you saved the file.
   - If you are new to the terminal in Linux you can navigate to a folder as follows.
   - Start by entering the `ls` command. This will list all of the files and folders in the current directory. One of these should be your Documents folder.
   - You can enter the Documents folder by entering the command `cd Documents`.
   - Once in the Documents folder you can again use the `ls` command to see the files and folders in Documents; one of these should be your new folder called CS1. Use the `cd CS1` command to enter your CS1 folder.
4. Once you have entered the folder where you have saved your program, enter the following commands[1] to **call the assembler**. Again, make sure you type out the commands yourself, rather than copy and pasting them!

```
avr-as -g -mmcu=atmega328p -o clr_sreg.o clr_sreg.s
avr-ld -o clr_sreg.elf clr_sreg.o
avr-objcopy -O ihex -R .eeprom clr_sreg.elf clr_sreg.hex
```

Listing 2: Calling the assembler.

   Remember to replace the text '`clr_sreg`' with whatever you called your file.

   Each of these lines is a separate command, do hit enter after each one. These three commands create one new file each. You can check the command has run succesfully by checking a new file has been created after running each command. Find the new file with the extension `.hex`, containing a string of HEX characters. This is the machine code for the program in Intel HEX format, which can be run on the Arduino.

5. **Connect your Arduino to your PC with the USB cable** Once connected you should check which device file the Arduino is using in the Linux OS, by running this command.

```
ls /dev/ttyUSB*
```

Listing 3: Listing the USB devices.

   What device file name is printed to the screen? You will use this file name in the next command that you run.

[1] Note, that, Listing 1 actually includes some additional stages that ensure that the program is properly *linked*, e.g. to any external libraries used. These are not important to our discussions here.

6. **Write the program to the Arduino's memory** with the following command. This command is for the Informatics *Linux* Lab environments. You will have to modify the command arguments if you are using your own computer (i.e. what follows the -C and the -P settings).

```
avrdude -C /etc/avrdude.conf -p atmega328p -c arduino -P /dev/ttyUSB0 \
-b 57600 -D -U flash:w:clr_sreg.hex:i
```

Listing 4: Writing to memory.

This is a single command but it's written across two lines. The backslash at the end of the first line is a line break, with the rest of the command on the second line. It's written this way because the command is too wide to fit on the page! However, it's probably easier for you to write this command on a single line in the terminal — just remove the back slash so that `/dev/ttyUSB0` is immediately followed by `-b`.

Make sure the Arduino is **connected to your PC** through the USB cable!

If all goes well, the LEDs on the Arduino board will flash for a few moments, indicating that the program is being communicated to the chip through the serial USB connection.

If you get a "not in sync" error in the terminal, you should run the modified command below instead.

```
avrdude -C /etc/avrdude.conf -p atmega328p -c arduino -P /dev/ttyUSB0 \
-D -U flash:w:clr_sreg.hex:i
```

Listing 5: Modified writing to memory.

Notice that in the modified command we have removed the `-b` flag and it's parameter. This flag is used to override the default baudrate, which is not required for all the Arduino boards.

The program **automatically executes** once written to memory, and will execute again every time the **reset** button is pressed on the Arduino board.

Unfortunately, there is not much to see with this program while it's running, since it only resets `SREG`, and then does nothing. But congratulations on writing and running your first assembly program! Next week we will create a circuit with the processor to do something more interesting.

4 Summary

Today, you took your first steps in assembly programming for a microprocessor.

You learned:

- Some basic syntax of assembly programming language for Atmel microprocessors.
- How to call the assembler to convert a program to machine readable code, and how to write a program to the microprocessor memory.

(Optional) Install the toolchain on your own computer

On KEATS you will find a guide for installing the toolchain on your own computer. There is information for Linux, Windows, and Mac. We recommend installing the toolchain on your own computer so that you can more easily work on lab exercises in your own time.

Note, that as long as there is not a timetabled class in the lab room you are welcome to come and use these computers in your own time. There are labs in Bush House north side floors 6 and 7 that are specifically for informatics students.