CS1 Lab: First Steps with Assembly for Arduino

1 Getting Started

Make sure you have everything you need to complete this lab:

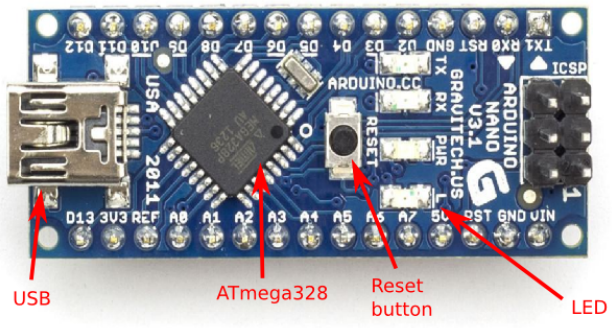
- Arduino Nano.
- USB cable.
- Atmel ATmega328 Datasheet.[1]
- Atmel 8-bit AVR Instruction Set Manual.

For this lab, you **only** need the Arduino Nano and the USB cable. In Lab 2, you will be using the breadboard and the electronic components to connect LEDs to you Arduino.

2 Arduino Components

The Arduino Nano is an open-source platform for prototyping simple electronics projects. The board itself can be considered as a small computer including **peripherals** and a **central processing unit**. Computations are done on the Atmel ATmega328 microcontroller. The microcontroller has its own instruction set architecture upon which all higher level languages are built.

If you look carefully at the chip on your Arduino Nano, you will see details of the maker (Atmel) and model (ATmega328) of the microcontroller etched onto it:



3 Pre-Lab Questions

To understand more about the specifications of this particular microcontroller, we can look at the **datasheet** and **instruction set manual** provided by the manufacturer.

Take a look at the **datasheet** and answer the following questions:

1. Is the architecture RISC or CISC? How many instructions are there?

[1] Links to PDFs are provided on KEATS.

2. What is the address of the *status register* (SREG) when using I/O specific commands (i.e. IN and OUT)?

Take a look at the **instruction set manual** and answer the following questions:

1. What is the *instruction length* for Register Direct addressing modes? For these modes, how long is the *op-code*?
2. What does the `ldi` instruction do? What operands does it take?

Take the Lab 1 Pre-Lab Quiz on KEATs. Use the answers to the above questions. Do this before the Pre-Lab 1 Quiz Deadline.

4 First Assembly Program

To get started with writing assembly code for the microcontroller, let's look at a simple example:

```
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

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

Let's take a closer look at the structure of the code.

- On line 2, we see the keyword `.org`. This is specifies the start address. The assembler uses this to tell where the program starts when translating to machine code.
- On line 1, we see the keyword `.equ`. This instructs the assembler to treat the label `SREG` as the constant `0x3f`.
- In green, are comments. These are delimited by a leading semi-colon (`;`).
- On lines 3 and 5, we see two *address labels* followed by a colon (`:`).
- On lines 3 and 4, we see that the program uses register `r16`. This is a general purpose registers used for holding data in the microprocessor's CPU.
- Highlighted in blue we see three instructions are used: `ldi`, `out` and `rjmp`.

Look again at the **instruction set manual** and make sure you understand what these instructions do, and how the program in Listing 1) works.

5 Assembling and Writing to Memory

Now that we have our first program, it's time to call the assembly process to create the machine code, and write to the arduino's memory. We will use the Unix command line to do this.

1. Open a text editor, and type out the program in Listing 1). Save the file with the extension '`.s`' (*e.g.*, `clr_sreg.s`).
2. Open a terminal shell, and navigate to the directory in which you saved the file.

3. Now, enter the following commands[2] to **call the assembler**:

```
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.

   Look in the current directory. You should find a file with the extension .hex, containing a string of HEX characters. This is the machine code for the program in Intel HEX format.

4. **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.

5. **Write the program to 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/avrdude.conf -p atmega328p -c arduino -P /dev/ttyUSB0 \
-b 57600 -D -U flash:w:clr_sreg.hex:i
```

Listing 4: Writing to memory.

   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.

The program **automatically executes** once written to memory, and 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.

6 Submit your program

Submit your program on KEATs under the Lab 1 Section Once the Lab 1 Sessions begin, you will be able to submit your clr_sreg.s file on KEATs. Submit only the clr_sreg.s file. Do not compress or zip your file before submitting it.

7 Summary

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

You learnt:

[2] Note, that, Listing 2) 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.

- How to use documentation from manufacturers to understand the specification and operation of a microprocesser, and its instruction set.
- 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) Further reading

As further reading, see if you can use the data sheet and instruction set manual to answer the following questions:

1. Is the architecture *stack*, *accumulator*, or *GPR*-based?
2. What *I/O interfaces* are available?
3. What is the maximum *clock speed*?
4. What *addressing modes* are available?