ECE 375: Computer Organization and Assembly Language Programming

Lab 2 – C → Assembler → Machine Code → TekBot

SECTION OVERVIEW

Complete the following objectives:

- Look at a sample C program.
- Learn how to configure the I/O ports of the ATmega128 microcontroller.
- Write a simple C program for the ATmega128 microcontroller.
- Compile your program using the Atmel Studio GCC compiler.
- Upload the code to your `mega128` board and verify its correct operation.

PRELAB

To complete this prelab, you may find it useful to look ahead to Section 5.2.1 of the ECE 375 textbook, or look at the "ATmega128 I/O Ports" document linked on the lab webpage. If you consult any non-OSU online sources to help answer the prelab questions, you **must** list them as references in your prelab. **Remember – late prelabs will not be accepted.**

The ATmega128 microcontroller has seven general-purpose input-output (I/O) *ports*: Port A through Port G. An I/O port is a collection of *pins*, and these pins can be individually configured to send (output) or receive (input) a single binary bit. Each port has three I/O registers, which are used to control the behavior of its pins: PORTx, DDRx, and PINx. (The "x" is just a generic notation; for example, Port A's three I/O registers are PORTA, DDRA, and PINA.)

1. Suppose you want to configure Port B so that all 8 of its pins are configured as outputs. Which I/O register is used to make this configuration, and what 8-bit binary value must be written to configure all 8 pins as outputs?
2. Suppose all 8 of Port D's pins have been configured as inputs. Which I/O register must be used to read the current state of Port D's pins?
3. Does the function of a PORTx register differ depending on the setting of its corresponding DDRx register? If so, explain any differences.

- Initialize relevant I/O ports
- Loop forever:
  - Move forward for 500 ms
  - Move backward for 500 ms
  - Turn left for 1000 ms
  - Turn right for 2000 ms
  - Turn left for 1000 ms

Figure 1: Pseudocode for Lab 2 DanceBot Code

PROCEDURE

Looking at C Code in Atmel Studio

1. Download the sample code (`DanceBot.c`) available on the lab webpage. This simple C program is well-commented, and is ready to compile. All code that you produce should be as well-commented as this code. The behavior of this program is described in Figure 1. Save this code somewhere you can find it.
2. Open Atmel Studio. Instead of creating an Assembler project like in Lab 1, create a new C project by choosing **C/C++** under "Installed" and selecting **GCC C Executable Project** as the project type. As with Lab 1, choose **ATmega128** when prompted to make a device selection.
3. Next, include the sample source file you downloaded earlier into the project you just created. If you need a reminder of how to do this, part of Section 2.1.2 of the AVR Starter Guide explains the basic process.
4. Examine the source file you just included in your project, and try to understand how it is performing the behavior described in Figure 1.

   Even though certain parts may look unfamiliar to you at this point, ultimately it is just C code, which you should have been exposed to in CS 151, CS 261, or elsewhere. If you are having difficulties, look around online for a basic C language tutorial, or ask your fellow lab students or your TA for help understanding the C syntax.

Compiling and Uploading to the `mega128` Board

Compiling a C program in Atmel Studio is very similar to compiling an assembly program, which you did previously in Lab 1. To compile `DanceBot.c`, simply go to **Build → Build Solution**, or just press F7. The log at the bottom of the screen should tell you that there are no errors, assuming you have done everything correctly up to this point. Common mistakes include trying to compile the C program within the wrong project type (make sure you didn't accidentally create an Assembly project), or trying to build with both `DanceBot.c` and `main.c` included in the project (remove `main.c`).

You may notice that a `.hex` file was created from the build operation, just like in Lab 1. Connect your `mega128` board to your computer using the `usbasp` programmer, and then use the Universal GUI application to upload the `.hex` file to your board (refer back to the Universal GUI User Guide if needed).

Your Own Code

Next, you need to write a simple C program that will make a TekBot perform the basic BumpBot routine, as seen and described in Lab 1. The TekBot should travel forward until it encounters an object (that is, until one or both of the whiskers are bumped), back up and turn away from the object, and then resume moving forward. Here are a few tips for completing this task successfully:

- You will probably want to start from the *skeleton code* available on the lab webpage and add code to it as needed. Even if you do not use the skeleton code, you **must adhere** to the port map given in the skeleton code when writing your whisker detection & motor control logic.
- If *both whiskers are triggered at the same time*, the TekBot must back up and turn to the right like it does when only the left whisker is hit.
- Don't forget to make use of the `_delay_ms()` function.

When you have completed your C-based BumpBot program, demonstrate its correct operation to your lab TA to receive implementation credit for this lab.

STUDY QUESTIONS / REPORT

A full lab write-up is required for this lab. When writing your report, be sure to include a summary that explains **what you did and why, discusses any problems you may have encountered, and answers the study questions given below**. Your write-up and code must be submitted online by the beginning of next week's lab. Remember, NO LATE WORK IS ACCEPTED.

Study Questions

1. This lab required you to compile two C programs (one given as a sample, and another that you wrote) into a binary representation that allows them to run directly on your `mega128` board. Explain some of the benefits of writing code in a language like C that can be "cross compiled". Also, explain some of the drawbacks of writing this way.
2. The C program you just wrote does basically the same thing as the sample assembly program you looked at in Lab 1. What is the size (in bytes) of your Lab 1 & Lab 2 **output `.hex` files**? Can you explain why there is a size difference between these two files, even though they both perform the same BumpBot behavior?

CHALLENGE

Modify your C program so that the TekBot will attempt to push any objects that it encounters. One way to perform this "push" operation is to implement the following TekBot behavior:

1. TekBot moves forward indefinitely until it hits an object.
2. TekBot continues forward for a short period of time *after* hitting the object.
3. TekBot backs up slightly.
4. TekBot turns slightly toward the object.
5. TekBot returns to Step 1.

As is often the case, there are many ways to write code to solve this problem. Think about the approach you would like to take, and then revise your firmware (i.e., make changes to your C program) accordingly.

To get credit for completing this challenge, you must demonstrate the "push" operation of the TekBot to your TA, and submit a copy of your challenge code online, **in addition to** your lab write-up and regular (non-challenge) lab code.