ECE 375: Computer Organization and Assembly Language Programming

Lab 1 – Introduction to AVR Development Tools

SECTION OVERVIEW

Complete the following objectives:

For Part 1.

- Download and compile the sample AVR assembly source code provided (`BasicBumpBot.asm`).
- Understand how to connect and operate the AVR programmer.
- Upload the previously-compiled sample program to the flash memory of the TekBots AVR microcontroller board (`ATmega32U4`), and observe it running.

For Part 2.

- Download and compile the sample C program.
- Learn how to configure the I/O ports of the ATmega32U4 microcontroller.
- Write a simple C program for the ATmega32U4 microcontroller.
- Upload the code to your `ATmega32U4` board and verify its correct operation.

You are given 2 weeks to complete the first lab assignments.

PROCEDURE - PART 1

Compiling an AVR Assembly Program

1. Download the sample code (`BasicBumpBot.asm`) as well as the pre-compiler directive file (`m32U4def.inc`). This is a simple AVR assembly program that is well-commented and ready to compile. All code that you produce for this course should be as well-commented as this code. Save this code somewhere you can find it.
2. You are required to use `Windows` system to develop your AVR assembly code throughout this course in order to utilize a debugger simulator tool. You will be using it to write assembly programs for your AVR microcontroller board, which uses an ATmega32U4 microcontroller.
3. Now that you have loaded the `BasicBumpBot.asm` and `m32U4def.inc` into the same directory, take a moment to look over the sample code. Although you haven't seen any AVR assembly code in this class yet, read the comments and see if you can follow the general structure and flow of the program. For reference, the general operation of the program is described in Figure 1.
4. By following the intallation guide, learn how an AVR assembly program is compiled. The resulting output is a binary program file (called a HEX file, with a `.hex` extension). This HEX file contains the actual binary instructions that will be executed by the ATmega32 microcontroller.

- Initializes key components of the ATmega32
- Starts the TekBot moving forward
- Polls the whiskers for input
- If right whisker is hit
  - Backs up for a second
  - Turns left for a second
  - Continues Forward
- If left whisker is hit
  - Backs up for a second
  - Turns right for a second
  - Continues Forward

Figure 1: Theory of Operation for Lab 1 AVR Assembly Code

Uploading a Compiled AVR Program

1. With the HEX file uploaded to the microcontroller, you can observe the behavior of the sample program. The LEDs on the `ATmega32U4` board should indicate that the board is performing a basic BumpBot routine (remember, Figure 1 gives a description of the full routine). To receive points for Part 1 of this lab, modify the sample program so the TekBot can reverse for **twice as long** before turning away and resuming forward motion. **In your demo, explain how you have done it with reasons**.

PROCEDURE - PART 2

Looking at C Code

- 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 2: Pseudocode for Lab 2 Example Code

1. Download the sample C code. 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. Examine the source file with the same manner we've learned from Part 1, and try to understand how it is performing the behavior described in Figure 1.

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* 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 left** like it does when only the right 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.

LAB REPORT

You are **not** required to write a lab report, but you should review the study questions. Although these questions will not be graded, they **can be asked during your demo, and your grade may be affected** if you cannot answer them.

Study Questions

1. Take a look at the code you downloaded for today's lab. Notice the lines that begin with `.def` and `.equ` followed by some type of expression. These are known as **pre-compiler directives**. Define pre-compiler directive. What is the difference between the `.def` and `.equ` directives? (HINT: see Section 5.4 of the AVR Assembler User Guide).
2. Read the AVR Instruction Set Manual. Based on this manual, describe the instructions listed below.

   ADIW, BCLR, BRCC, BRGE, COM, EOR, LSL, LSR, NEG, OR, ORI, ROL, ROR, SBC, SBIW, and SUB.
3. The ATmega32U4 microcontroller has six general-purpose input-output (I/O) *ports*: Port A through Port F. 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.)

   (a) 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?

   (b) Suppose Port D's pins 4-7 have been configured as inputs. Which I/O register must be used to read the current state of Port D's pins?

   (c) Does the function of a PORTx register differ depending on the setting of its corresponding DDRx register? If so, explain any differences.

4. This lab required you to modify the sample AVR program so the TekBot can reverse for twice as long before turning away and resuming forward motion. Explain how you have done it with reasons.
5. The Part 2 of 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 `ATmega32U4` 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.
6. The C program you wrote does basically the same thing as the sample AVR program you looked at in Part 1. What is the size (in bytes) of your Part 1 & Part 2 **output `.hex` files**? Explain why there is a size difference between these two files, even though they both perform the same BumpBot behavior?