ECE 375 Lab 2

C -> Assember -> Machine Code

Lab Time: Wednesday 10a-12n

Robert Detjens

David Headrick

TA Signature

Introduction

This lab is to teach students how to program the microcontroller board and with controlling input and output in C. The register-based I/O requires forethought on what hardware is connected in order to setup the microcontroller ports to read inputs and control outputs correctly.

Program Overview

This program performs object-avoidance detection by sensing if the TekBot has run into an object and avoiding it if so. This is done by constantly checking the antenna sensors for hits, and turning away from the side that was hit, if there was one.

The antenna are connected to pins 1 and 2 of port D, and motor controls are on pins 4-7 of port B. When a hit is detected (i.e. one of the antenna inputs is pulled low), the following procedure is followed:

- The Bot backs up slightly
- A bitwise calculation is performed to calculate the direction in which to turn
- The turn is performed
- The Bot continues forward in the new direction and resumes checking for hits

Additional Questions

- 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.

  C provides traditional control flow structures like conditional statements and loops. While these can obviously be done ins assembly, these higher-level features are much easier to read.

  C can also be compiled to multiple different architectures as it is ISA-independent in sourcecode form. However, this does come with drawbacks as the compiler may generate sub-optimal code as compared to hand-tuned purpose-written specific assembly.

- 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?

  - Lab 1 hex: 485b
  - Lab 2 hex: 369b

  This size discrepancy is likely due to the optimized bitmath used in our C program instead of the separate procedures and conditionals for each hit side in the assembly code. If the same algorithm was used for both, the C code would likely be larger due to the compiler adding extra stuff.

Difficulties

As this lab was done outside of AVR Studio with the `avr-gcc` toolchain, there was more setup work needed to get the compiler to correctly compile for the right hardware. This would have been automated by AVR Studio, but had to be done manually instead.

There were also some troubles with compiler optimizations being needed but also giving some issues when testing the partially-complete program, but once the program was complete these issues were resolved.

We needed to consult the datasheet to re-familiarize with the correct configuration of the IO registers after trying to remember what the correct settings were off-hand.

Conclusion

This lab was successful in teaching the correct way to use the I/O registers to control hardware in a C program. Our board correctly demonstrated both the lab program and the challenge program with no issues and we feel confident in our ability to write future microcontroller programs in C.

Source Code

Program code

```
/*
This code will cause a TekBot connected to the AVR board to move forward and
when it touches an obstacle, it will reverse and turn away from the obstacle and
resume forward motion.

AUTHORS:
- Robert Detjens
- David Headrick

PORT MAP
Port B, Pin 4 -> Output -> Right Motor Enable
Port B, Pin 5 -> Output -> Right Motor Direction
Port B, Pin 7 -> Output -> Left Motor Enable
Port B, Pin 6 -> Output -> Left Motor Direction
Port D, Pin 1 -> Input -> Left Whisker
Port D, Pin 0 -> Input -> Right Whisker
*/

#define F_CPU 16000000

#include <avr/io.h>
```

```c
#include <stdio.h>
#include <util/delay.h>

int main(void) {

  int hit_state = 0;

  DDRB = 0b11110000;  // set pins 4-7 as outputs (high)
  PORTB = 0b10010000; // initially disable motors

  DDRD = 0b00000000;  // set pins 1-2 as inputs (low)
  PORTD = 0b11111111; // enable pull-up resistors for input pins

  while (1) { // loop forever

    // read input whiskers (only the first 2 pins)
    int reading = PIND & 0b00000011;

    // if hit? (i.e. one of the input pins is low)
    if (reading != 0b11) {
      // back up
      PORTB = 0b00000000; // move backward
      _delay_ms(1000);

      // turn away from the hit:
      //   left hit:   0b1111_1101
      //                    __    ^^ these are the same bits inverted!
      //   turn right: 0b0100_0000
      //   right hit:  0b1111_1110
      //                    __    ^^ these are the same bits inverted!
      //   turn left:  0b0010_0000

      // if both are hit, turn left
      reading = reading ? reading : 0b10;

      // since it is the same bits just inverted,
      // do some coolguy bitmath to not need a conditional B)
      PORTB = (~reading & 0b00000011) << 5;
      _delay_ms(1000);
    }

    // continue forward
    PORTB = 0b01100000;
    _delay_ms(20);
```

```
  }
}
```

Challenge code

```
/*
This code will cause a TekBot connected to the AVR board to move forward and
when it touches an obstacle, it continue forward slightly, reverse, then turn
away from the obstacle and resume forward motion.

CHALLENGE CODE:
- Follow the hits instead of avoiding them.

AUTHORS:
- Robert Detjens
- David Headrick

PORT MAP
Port B, Pin 4 -> Output -> Right Motor Enable
Port B, Pin 5 -> Output -> Right Motor Direction
Port B, Pin 7 -> Output -> Left Motor Enable
Port B, Pin 6 -> Output -> Left Motor Direction
Port D, Pin 1 -> Input -> Left Whisker
Port D, Pin 0 -> Input -> Right Whisker
*/

#define F_CPU 16000000

#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>

int main(void) {

  int hit_state = 0;

  DDRB = 0b11110000;  // set pins 4-7 as outputs (high)
  PORTB = 0b10010000; // initially disable motors

  DDRD = 0b00000000;  // set pins 1-2 as inputs (low)
  PORTD = 0b11111111; // enable pull-up resistors for input pins

  while (1) { // loop forever
```

```
    // read input whiskers (only the first 2 pins)
    int reading = PIND & 0b00000011;

    // if hit? (i.e. one of the input pins is low)
    if (reading != 0b11) {
      // CHALLENGE:
      // continue forward for a bit longer
      _delay_ms(500);

      // back up
      PORTB = 0b00000000; // move backward
      _delay_ms(500);

      // CHALLENGE: turn towards the hit
      // do some coolguy bitmath to not need a conditional B)
      PORTB = (reading & 0b00000011) << 5;
      _delay_ms(500);
    }

    // continue forward
    PORTB = 0b01100000;
    _delay_ms(20);
  }
}
```