SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

LABORATORY REPORT

To: Dr. Randy Hoover

From: Benjamin LeBrun

Subject: Lab Assignemnt 1: AVR Simulation

Date: February 11, 2019

Introduction

For this lab, we used Atmel studio to compile a simple counting program in order to view how to use certain debugging features for future labs as well as view C assembly to see how our compiler makes certain optimizations in code and understand how the program counter works.

Equipment

This lab only required the use of Atmel studio and a computer to compile, debug, and view the assembly.

Implementation

The lab asked us to configure Atmel studio to use a simulator in it's debug mode, which allows us to add breakpoints and debug the program easily without purchasing a third party addon or hardware debugger.

To test certain features of the simulator we wrote a simple counter program inside of Atmel studio. We then added a breakpoint inside the program when the code makes a port assignment to catch the program as it looped through it's process. When the breakpoint was activated, we then used Atmel's disassembly feature to view the assembly code that was compiled for the program. The diassembly can be viewed in Appendix A of this memo.

Discussion

Lab was certainly straightfoward. It's incredibly cool that Atmel studio allows us to move within or add breakpoints in the disassembly and would be interesting to see the optimizations with larger and more complicated programs or certain tricks to work in tandem with the assembler.

Responses

1. There are several different debugging technologies on the market, with one of the largest players being the JTAG standard. What is JTAG, and what are some benefits of its use?

   JTAG is a standard for debugging and testing printed, embedded, or other microcomputer boards without having to rely on new or proprietary standards for communication or costly hardware solutions such as a bed of nails tester or physical access altogether.

2. We saw that the compiler removed i as a variable, yet our code still worked! How did the compiler match the functionality of i in the assembly? Translate the relevant assembly back into a snippet of C code showcasing the compiler's workaround.

   The compiler in this program instead of using an address for "i" and constantly making a reassignemnt between the compiler and the port address, wrote directly to the "PORTB" register in assembly. Equivalent C code inside the progam loop would probably look like:

   ```
   for(PORTB = 0; PORTB < 16; PORTB++){
     _delay_ms(500);
   }
   ```

3. Sometimes microcontrollers don't immediately executing code at 0x0000 on reset. For example: if the BOOTRST and BOOTSZ fuses are burn when flashing code to our AVRs, on reset they will start executing code at locations other than 0x0000; this is commonly used to run a bootloader, which will then point execution back to 0x0000. What is a fuse? What is a bootloader? Do our Arduinos have a bootloader?

   Fuses in our AVR system are special settings that can be set to change certain hardware properties like different clock speeds or minimum voltages or memory allocation which are usually "burned" when the board is programmed. A bootloader is a way to bypass having to use an external programmer for a microcontroller or microcomputer to load the program or kernel of your code. However, this comes at the cost of less usable memory on the microcontroller. By default, the Arduino does not include a bootloader to allow full access to the entire memory space but can be changed to allow one at the cost of flash space.

4. For the while(1) assembly code, describe the program's flow/path starting from address 0x0000. How does this differ from the assembly code with while(1) removed?

   Instead of a relative jump to the beginning of the program, our assembly uses a "Return from Subroutine" command that loads the return address from stack.

Appendicies

(A) C code

```
/*
 * Lab1.c
 *
 * Created: 1/29/2019 08:14:49
 * Author : Ben L
 */
```

```c
#define F_CPU 16000000

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

int main(void)
{
        DDRB = 0x0F; // Enable outputs
        uint16_t i = 0; // Iterator variable
    /* Replace with your application code */
  while (1)
  {
                for( i = 0; i < 16; i++){
                        PORTB = i; // Copy i to output register
                }
  }
}
```

(B) Disassembly

```
0000002E  JMP 0x0000003E                ;Jump
00000030  JMP 0x0000003E                ;Jump
00000032  JMP 0x0000003E                ;Jump
;--- ../../../../crt1/gcrt1.S ------------------------------------------------
00000034  CLR R1                  ;Clear Register
00000035  OUT 0x3F,R1                  ;Out to I/O location
00000036  SER R28                 ;Set Register
00000037  LDI R29,0x08                  ;Load immediate
;--- ../../../../crt1/gcrt1.S ------------------------------------------------
00000038  OUT 0x3E,R29                  ;Out to I/O location
00000039  OUT 0x3D,R28                  ;Out to I/O location
0000003A  CALL 0x00000040                  ;Call subroutine
0000003C  JMP 0x0000004A                ;Jump
0000003E  JMP 0x00000000                ;Jump
;--- C:\Users\Ben L\Documents\Atmel Studio\7.0\Lab1\Lab1\Debug/.././main.c ----
;{
;        DDRB = 0x0F; // Enable outputs
00000040  LDI R24,0x0F                  ;Load immediate
00000041  OUT 0x04,R24                  ;Out to I/O location
00000042  LDI R24,0x00                  ;Load immediate
;                       PORTB = i; // Copy i to output register
00000043  OUT 0x05,R24                  ;Out to I/O location
00000044  SUBI R24,0xFF                  ;Subtract immediate
                ;for( i = 0; i < 16; i++){
00000045  CPI R24,0x10                  ;Compare with immediate
00000046  BRNE PC-0x03                  ;Branch if not equal
;}
00000047  LDI R24,0x00                  ;Load immediate
00000048  LDI R25,0x00                  ;Load immediate
```

```
00000049  RET                 ;Subroutine return
;--- No source file ------------------------------------------------------
0000004A  CLI                 ;Global Interrupt Disable
0000004B  RJMP PC-0x0000              ;Relative jump
0000004C  NOP                 ;Undefined
```