Lab 2: Basic Assembly Programming and Debugging using AVR Studio

Due: December 13, 2011

1 Outcomes

- Familiarize yourself with the capabilities of the ATMEGA32 embedded microcontroller and AVR Studio.
- Develop a simple software program in assembly for the Atmega32 embedded microcontroller.
- Learn the steps involved in creating, assembling, loading, running/simulating, and debugging programs for the Atmel system.
- Practice using an Oscilloscope to make simple measurements.

2 Equipment Needed from Tech Support

1 scope probe

3 Pre-lab

1. Watch the video online on setting up and measuring with the scope.
2. Watch the video online showing an operating version of this program.
3. Watch the video online showing how to interpret the listing file.

4 Overview

The purpose of this lab is for you become more familiar with the embedded software development tools that you will be using throughout the quarter – specifically the AVR Studio IDE.

In this lab, you'll reinforce your knowledge of the steps involved in creating, assembling, loading, running/simulating, and debugging programs for the Atmel system.

Each student must work independently on this project.

5 Lab activity

Create a project (e.g. **lab2**) in AVR Studio. (Hint: If you need to refresh your memory on how to create a project, go back to last week's lab.) Make sure you select the appropriate type of project when you run the Project Wizard. Also, like last week, make certain the "Create List File" option is enabled.

You'll have an empty **lab2.asm** file to begin with. You may use the program demonstrated in lecture as a starting point. **Observe the following assembly language coding conventions when writing your program in order to avoid a mangled mess of code**:

- Directives and labels begin in the left-most column.
- Instructions begin in a second column; use the dialog under Tools/Options/Editor to set the tab width to 10 (the default is 4).
- Operands begin in a third column

- Instruction comments begin in a fourth column; assembly language programs must be heavily commented.

Using the template given in Appendix A, create a source code file (**lab2.asm**) for your program. The program should contain the following content:

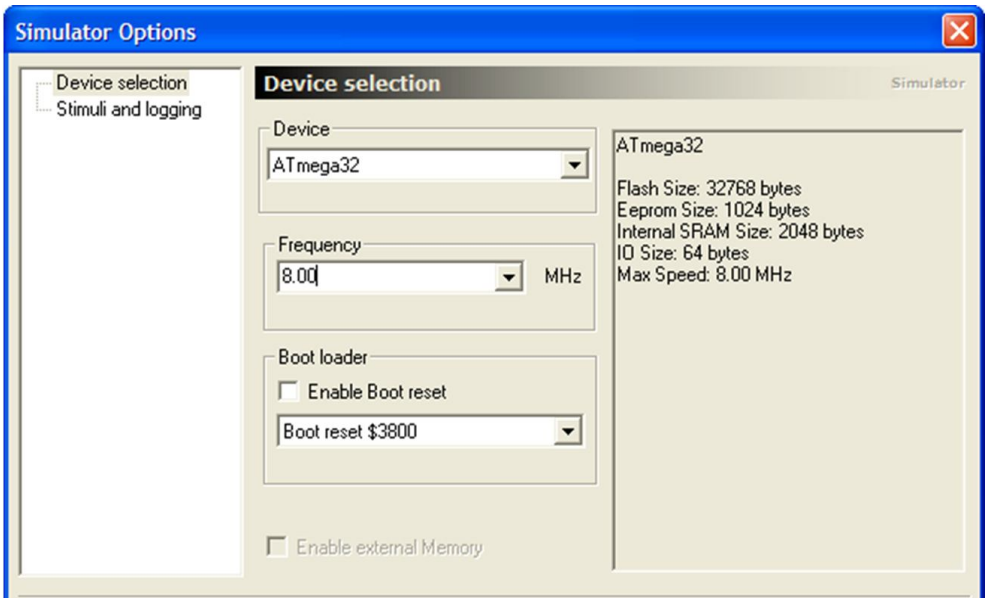
1. Definitions and declarations
    1. Place comments at the top of your **lab2.asm** file similar to those found in the example code template in Appendix A of this document.
    2. **`.include`** the file **m32def.inc** file for symbolic definitions that you will be using subsequently in your program; surround this directive with **`.nolist`** and **`.list`** to suppress it from appearing in the **lab2.lst** file
    3. Equate (**`.equ`**) the symbol **`PROGRAM_START`** to the value **0x2a**. This symbol represents where your program actually starts in Flash memory, and is located right after the vector table. (Hint: You'll use the **`.org`** directive along with this symbol later on.)
    4. Define (**`.def`**) a symbolic name for register 16 as **`TEMP`**.
    5. Define a symbolic name for register 17 as **`ONE`**.

2. Variable declarations (in the data segment)
    1. Define a variable named **`counter`** which has a size of 1 byte.
    2. Define variables **`delayc1`** and **`delayc2`** which are each 1 byte.

3. Code segment
    1. Use **`.cseg`** and **`.org`** directives along with the **RJMP** instruction to initialize the Reset Vector to cause the program to jump to **`PROGRAM_START`** upon power-on.
    2. Use **`.org`** and the **`PROGRAM_START`** symbol to cause the assembler to place the remainder of your code beginning at the **`PROGRAM_START`** location in Flash memory.
    3. Set the **DDRB** I/O register to configure **PORTB** as a digital output port. *Note that **DDRB** and **PORTB** are I/O memory locations predefined in the **m32def.inc** file.*
    4. Initialize the value of **`counter`** to 0.
    5. Initialize the value of **`ONE`** to 1.
    6. Initialize the variables **`delayc1`** and **`delayc2`** to 0xFF.
    7. Add the code of Figure 1 to initialize the stack.
    8. Write a segment of assembly language code which will take the value of **`counter`** and add a value of **`ONE`** in a loop. The value will then be stored back into the **`counter`** variable as well as output to **PORTB** in order to illuminate the LEDs.

6 Simulation and Debugging in AVR Studio

Build the project in AVR Studio. Once built successfully, set a breakpoint on the first instruction in your program. Next, select the "Start Debugging" command from the Debug menu; the program will start to execute, but will stop on the breakpoint you just set. Before continuing, set the clock frequency of the simulator by selecting "AVR Simulator Options" from the Debug menu. In the dialog that appears (below), select 8MHz as the clock frequency since the simulator does not permit you to select 16MHz.



Test-run your program in AVR Studio debugger in order to verify that it runs as expected. Demonstrate the program to your instructor, or ask for help if you're having difficulty. Port Values will show on the left, and like Eclipse, you can single step through the source code using the debug option. Be cognizant of the Port B values on the screen when the program executes.

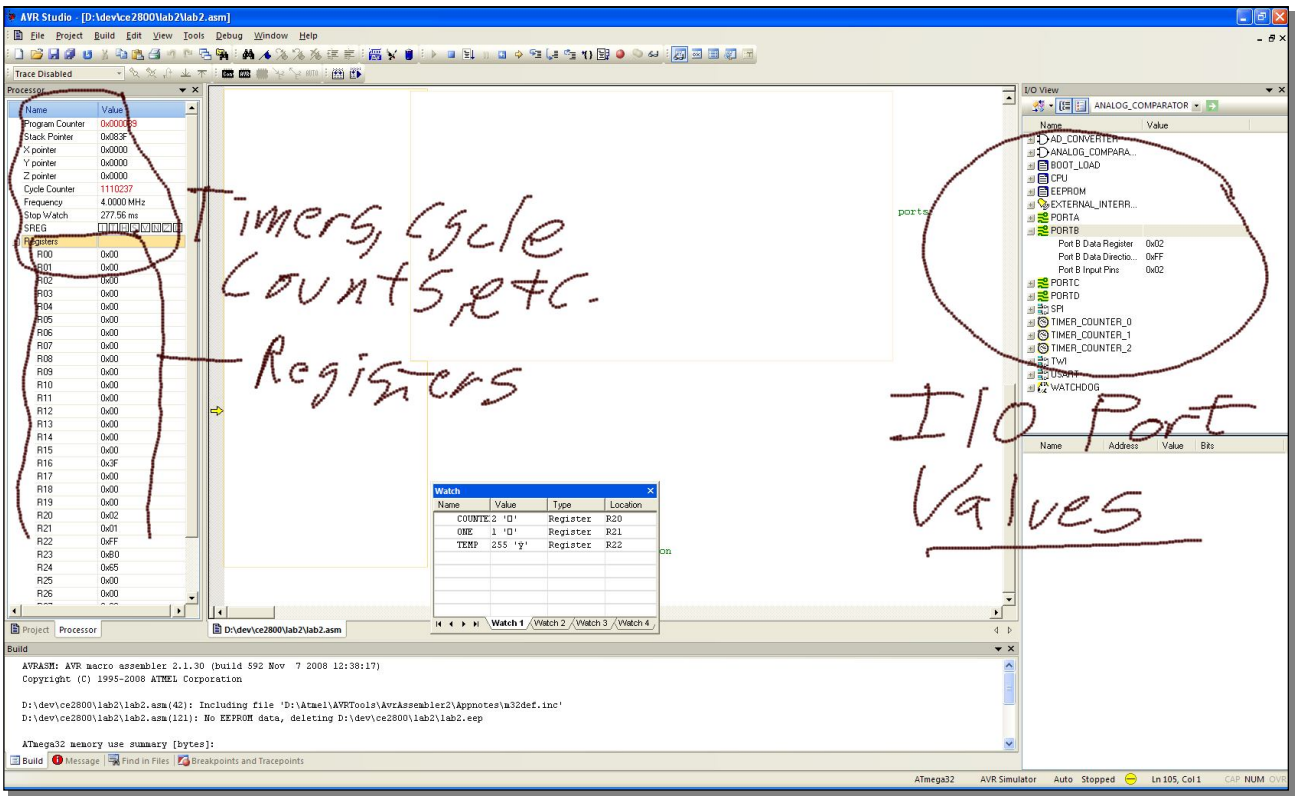


Figure 2: AVR Studio Simulator execution (Note: Source code has been "omitted" from the screen capture.)

Once your program is running correctly, examine the generated list file (**lab2.lst**) to answer the following questions for your report. (Hint: If you do not have a **lab2.lst**, click on the assembler options menu item from the project menu and check the box "Create List File". This will create a listing file for you each time the code builds which can be examined.)

Q1) How many bytes of SRAM is your program using? Does this make sense based on the variables you declared? (Hint: There is a table at the very end of the file which shows this information.)
Q2) How many bytes is the code of your program occupying? (Hint: There is a table at the very end of the file which shows this information.)
Q3) How many different instructions are you using? Which is the most common instruction?
Q4) What address has been assigned to the **`counter`** variable?

Use the built-in debugger to single-step your program. Expand the *Processor* icon to display the *Stop Watch*. Determine the answer to the following questions for your report:

Q5) How many CPU cycles does your loop take to execute?
Q6) How many microseconds does your loop take to execute?

Note: Adjust values for a 16MHz clock

Examine **SREG**. The **SREG** display can be expanded to show each of its individual bits. Single-step and or run your program in another manner to answer the following questions:

Q7) What is the value of **`counter`** when the "H" bit of **SREG** is first set?
Q8) What is the value of **`counter`** when the "Z" bit is first set?
Q9) What is the value of **`counter`** when the "C" bit is first set?

7 Execution on the board

Now that you have debugged through your program, remove all breakpoints by using the "remove all breakpoints" menu option. Change the debug platform to be "JTAG ICE" and the Device to be an ATMEGA32. Build the code and select "start debugging" from the debug menu. (Hint: You may want to watch the video about downloading code your board again.) Reset the part by selecting "Debug" and "Reset". Then click on run to start executing the program. Observe the LEDs.

Q10) Why do the LEDs all appear to be lit continuously?

Q11) Change the value of **`ONE`** to be 2 instead of 1. How does this change the behavior? Why?

should be "out", not "sts"

Place a breakpoint in the code at the line "sts PORTB, TEMP". Reset the part and run the code again. Are the LEDs changing? Now connect an oscilloscope to LED 7 on your board. Remove the breakpoint and run your program again.

Q12) What is the frequency of the waveform you are generating? in Hz (cycles/sec)

Now we'll add what is referred to as a delay loop to your code. Somewhere between your jump instruction and the label to which you are jumping, add the code shown in Figure 3.

Q13) What happens now? What is the frequency seen on LED7? What happens to the visibility of the led's?

```
outer_loop:
      ldi TEMP, 0xFF
      sts delayc2, TEMP
inner_loop:
      nop
      nop
      nop
      nop
      nop
      nop
      nop
      nop
      nop
      nop

      lds TEMP, delayc2
      dec TEMP
      sts delayc2, TEMP
      brbc 1, inner_loop

      lds TEMP, delayc1
      dec temp
      sts delayc1, TEMP

      brbc 1, outer_loop
      rjmp repeat             ; Jump to the repeat instruction
```

This assumes your program uses a label named "repeat"

Figure 3: Delay loop code.

8 Lab Report

Now that you have completed your lab assignment, submit the following lab report detailing your experiences. The lab report should be submitted electronically through Blackboard as a pdf file.

1. Introduction -> What did you accomplish with this lab?
2. Questions -> Answer each of the Questions Q1 – Q13.
3. Things gone Right -> What things went right in performing this lab?
4. Things gone Wrong -> What problems did you have? What mistakes did you make?
5. Conclusions -> This section shall discuss what has been learned from this laboratory experience.
6. Listing file -> Include the file lab1.lst, the listing file, generated by the assembler. Simply copy and paste the file to the end of your report as an appendix.
7. Source code. -> Copy and paste your source code into the back of your document before printing it to a pdf file.

should be lab2, not lab1

If you have any questions, consult your instructor.

Submit both your report (.pdf file containing your program listing) and your lab2.asm to Blackboard.

Appendix A: Assembly language source code template.

```
;###############################################################################################################
; Course: CE2800 Winter 2010-2011
; Assignment: Lab 2
; Author: schilling@msoe.edu
; Date: 12-1-2010
;
;###############################################################################################################
; <Add a generous definition of the program here.>
;
; CONFIGURATION
; - <Define the configuration for the program here…>   e.g. Atmega32
;
;###############################################################################################################
; Define and equal directives go here.
;
;###############################################################################################################
.list

; Define the input and output addresses to access the hardware registers.
.equ . . .

; Define any register assignments here.
.def . . .


;###############################################################################################################
; DATA SEGMENT DEFINITION
; Define your variables here.  For each variable, comment on what it represents and is used for.
;###############################################################################################################
.dseg ; Begin the data segment
.
.
.

;###############################################################################################################
; CODE SEGMENT DEFINITION
;###############################################################################################################
.cseg                          ; Begin code segment
.
.          one-time initialization code goes here
.

;###############################################################################################################
; Main program goes here.
; Major logic should be commented to explain what is happening within your code.
;###############################################################################################################
.org    PROGRAM_START
init:      use whatever label name you prefer
.
.
.
```