CSSE1000/7035 Prac 5

Introduction to Microcontrollers and Assembly Language Programming

Goals

- Learn and practice the basics of AVR assembly language programming
- Familiarization with programming tools and hardware for AVR microcontrollers

Preparation

Each student has a slightly different preparation task - based on your login. You must login to this page using your own stude username.

You are currently logged in as: s4209435

Read and preferably undertake the AVR Studio tutorial below. (Either will help you with the programming task). Write the required assembly language program as specified b (Programming Task) and predict the result. Document your program and expected (or actual) result in your workbook before attending the prac. (In future weeks, preparatio involve using AVR Studio before you get to the lab, so it is recommended you install/try the software now, rather than just reading the tutorial. AVR Studio is available for download from the CSSE1000 software page and is also available for use in ITEE computer labs.)

Programming Task

Write an AVR assembly program to:

- Load the value 41 (hex) into register 29
- load the value 54 (hex) into register 24
- compute the the difference of the two

You may wish the check out the following instructions in the help file: **and**, **or**, **sub** (see the tutorial below for how to access the help file and other instructions you should se help about - or consult one of the AVR instruction references on the AVR resources page).

Write your code down in your workbook, and write down the predicted result. Ideally, you should simulate your program in AVR Studio and document your simulation results

AVR Studio Tutorial

Installing AVR Studio

Install AVR Studio program at home (if needed). The installer may be downloaded from the CSSE1000 software page. AVR Studio is also available for use in the ITEE compute labs.

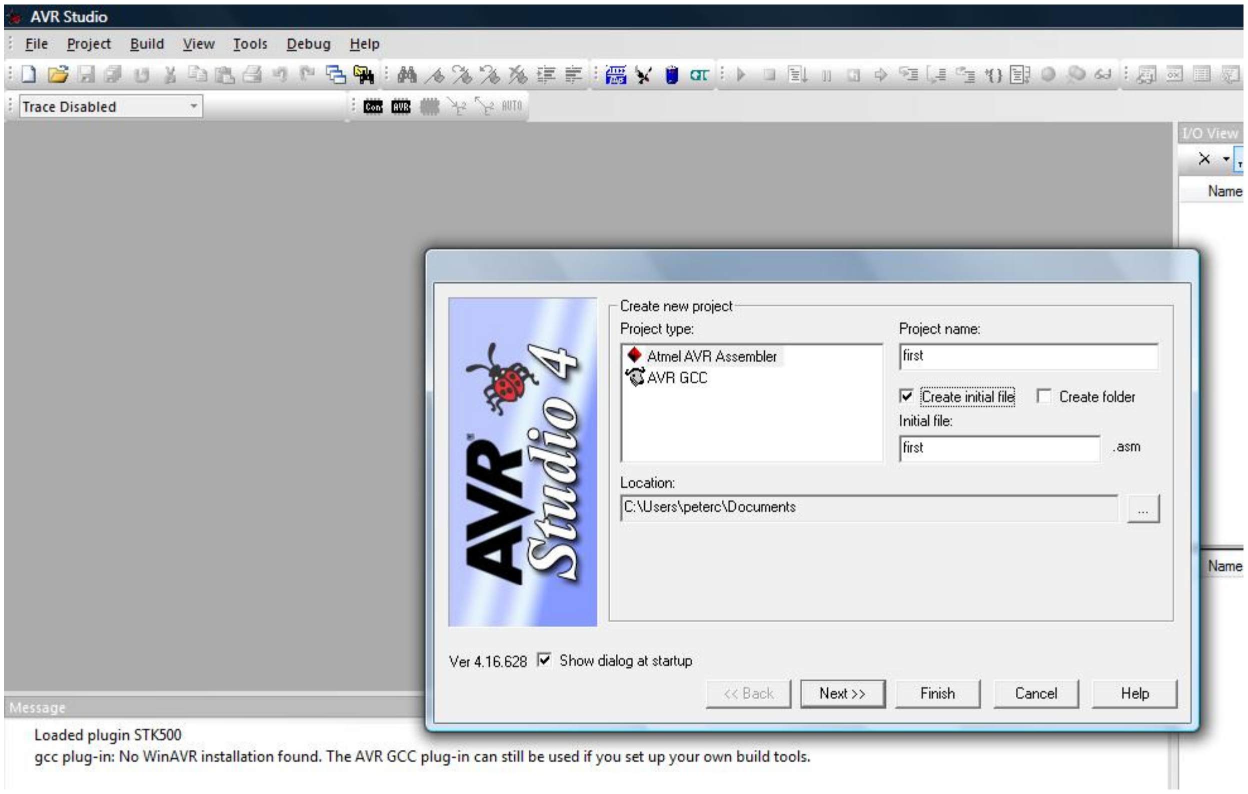
Creating a New Project

Start the AVR Studio program by clicking on:

```
Start->Programs->ATMEL AVR Tools->AVR Studio 4
```

In this tutorial we will make a simple program that writes some constant values to two general purpose registers and computes their sum.

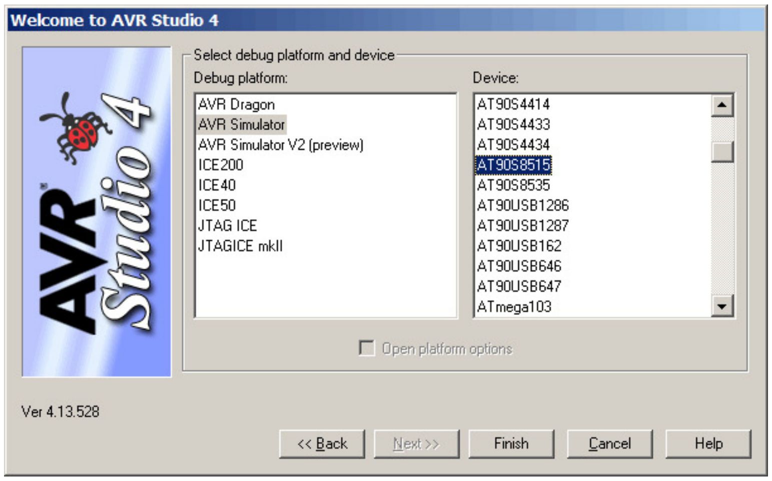
To create a new project, click on "New Project" on the Welcome Screen or go to the "Project" menu and select "New". The dialog box shown in the next figure appears.



Create new project
Project type: Atmel AVR Assembler [selected]; AVR GCC
Project name: first
[✓] Create initial file [ ] Create folder
Initial file: first .asm
Location: C:\Users\peterc\Documents
Ver 4.16.628 [✓] Show dialog at startup

Loaded plugin STK500
gcc plug-in: No WinAVR installation found. The AVR GCC plug-in can still be used if you set up your own build tools.

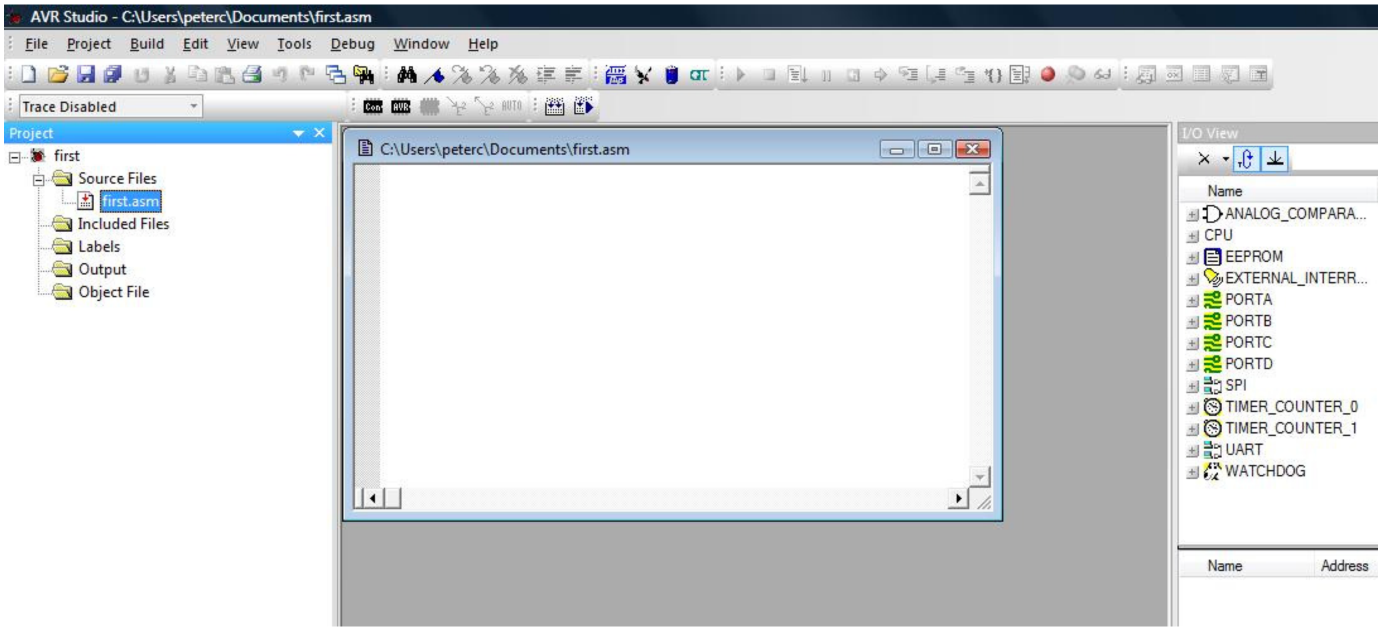
In this dialog box you should select "Atmel AVR Assembler" for the Project Type. and enter "first" as the project name. Next you'll have to select the project location. This is t location where AVR Studio will store all files associated with the project. We have used a location in C: drive but we recommend you use H:\csse1000\pracs\ as the folder. If folder does not exist, AVR Studio will ask you whether to create it or not. Click "Next >>" and you will see a window like the following:



Select debug platform and device
Debug platform: AVR Dragon; AVR Simulator [selected]; AVR Simulator V2 (preview); ICE200; ICE40; ICE50; JTAG ICE; JTAGICE mkII
Device: AT90S4414; AT90S4433; AT90S4434; AT90S8515 [selected]; AT90S8535; AT90USB1286; AT90USB1287; AT90USB162; AT90USB646; AT90USB647; ATmega103
Ver 4.13.528

Select "AVR Simulator" as the debug platform and "AT90S8515" as the device to use, then click "Finish". The AT90S8515 is the Microcontroller we will be using for the remain of CSSE1000 pracs and is available for use in the labs.

The main AVR window will now look something like this:



We can now enter the Assembly code for our program. (AVR Studio has opened an editor window called first.asm). Should the file not appear for editing you can also open th file for editing by expanding the sources directory in the Project panel and double clicking first.asm. The first.asm file is automatically marked as "Assembler Entry File" (the sr red arrow on the icon next to the filename shows this). This indicates that the code from this file will be assembled (and executed) first. Only one file can be marked as an er file.

Editing the Assembler file

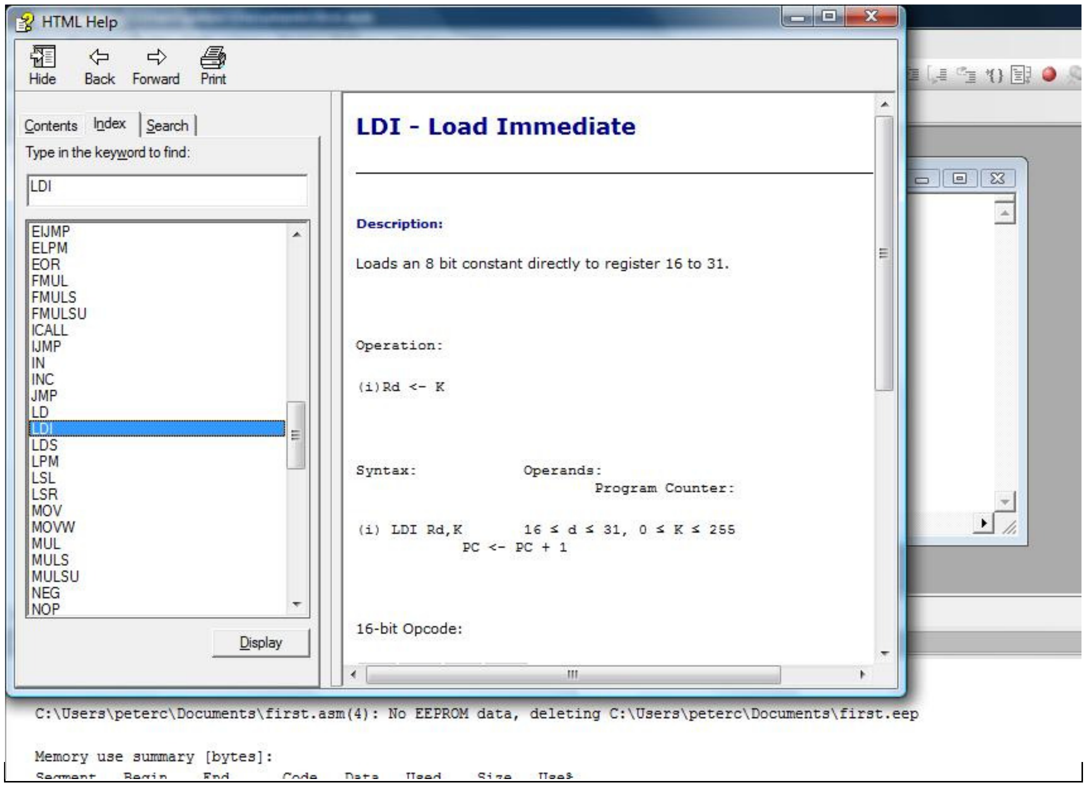
We have now added a new but empty file to our project. The next step is to fill this file with our code. The file is initially empty and you'll have to manually enter the following code: (or you may Copy and Paste the code below directly into the editor window.)

```
ldi r16, 0x56
ldi r17, 0x34
add r17, r16
```

Save the file. (Select "Save" from the "File" menu.)

Understanding the Example Source Code

The source code above consists of 3 instructions, two **ldi** (load immediate) and one **add** instruction. To understand what these instructions mean, open the AVR assembly he (Help->Assembler Help). Select and double click **ldi** from the index**,** you should see something like this:



LDI - Load Immediate

Description:

Loads an 8 bit constant directly to register 16 to 31.

```
Operation:

(i)Rd <- K

Syntax:              Operands:
                              Program Counter:

(i) LDI Rd,K       16 ≤ d ≤ 31, 0 ≤ K ≤ 255
             PC <- PC + 1
```

16-bit Opcode:

```
C:\Users\peterc\Documents\first.asm(4): No EEPROM data, deleting C:\Users\peterc\Documents\first.eep

Memory use summary [bytes]:
Segment   Begin    End       Code    Data    Used      Size    Use%
```

determine its operation.

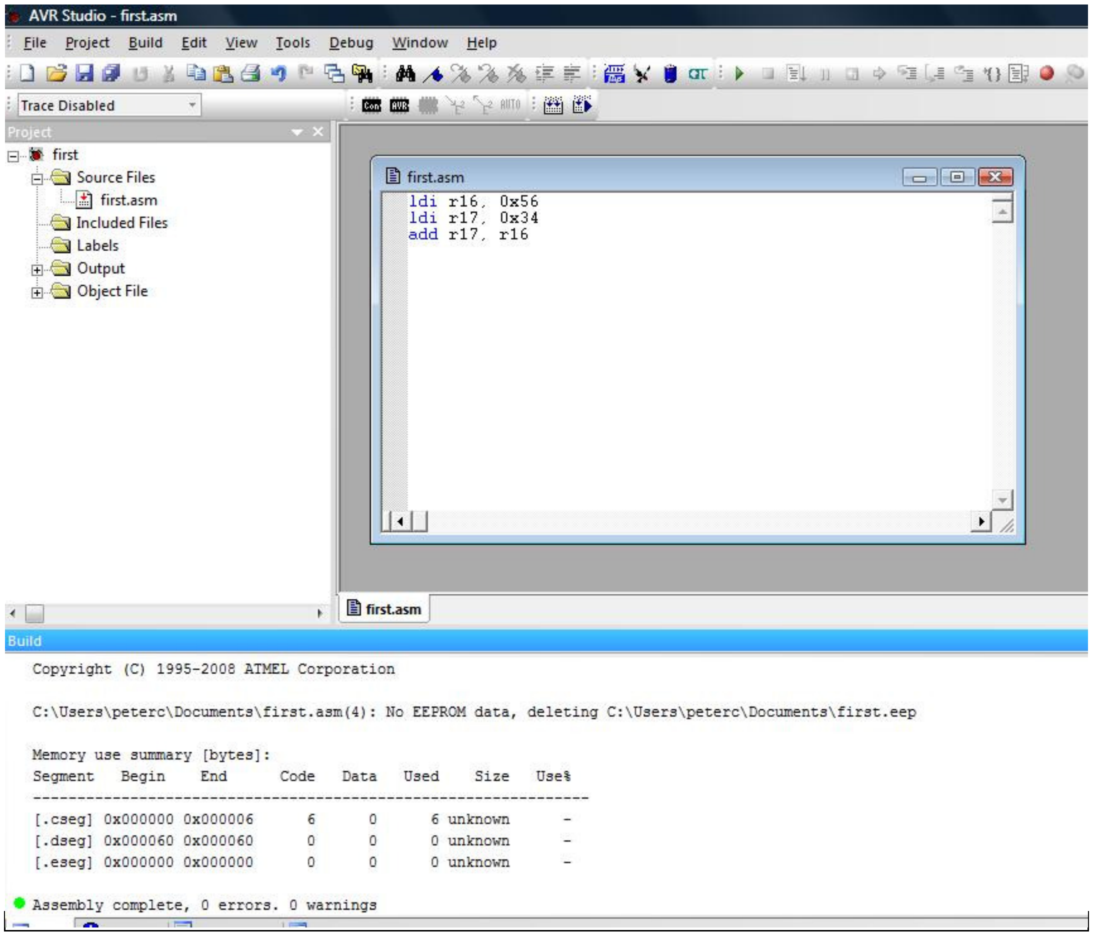
add r17, r16

In our example case the register 17 will be set to the value of r17 + r16. i.e. r17 <= r17 + r16. As we already have r16 = 56 hex, and r17 = 34 hex we expect that at the end of execution that r17 = 8A hex (56 hex+34 hex). We will
this through a simulation.

Assemble the Source Code

To assemble the code, we need to build the project. Select "Build" from the Build menu (or press <F7>):

The result of building the project will be shown in the "Build" pane and will be something like:



```
ldi r16, 0x56
ldi r17, 0x34
add r17, r16
```

```
Copyright (C) 1995-2008 ATMEL Corporation

C:\Users\peterc\Documents\first.asm(4): No EEPROM data, deleting C:\Users\peterc\Documents\first.eep

Memory use summary [bytes]:
Segment   Begin    End      Code   Data   Used    Size   Use%
---------------------------------------------------------------
[.cseg] 0x000000 0x000006      6      0      6 unknown      -
[.dseg] 0x000060 0x000060      0      0      0 unknown      -
[.eseg] 0x000000 0x000000      0      0      0 unknown      -

Assembly complete, 0 errors. 0 warnings
```

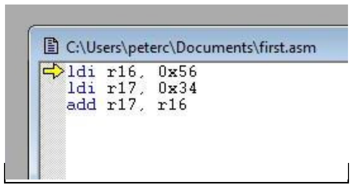
From this window we can see that the code is 6 bytes in size (most AVR instructions occupy two bytes each), and that assembly was completed with no errors.

We are now ready to advance to the next step, which is running the code in a simulator.

Simulating the Code

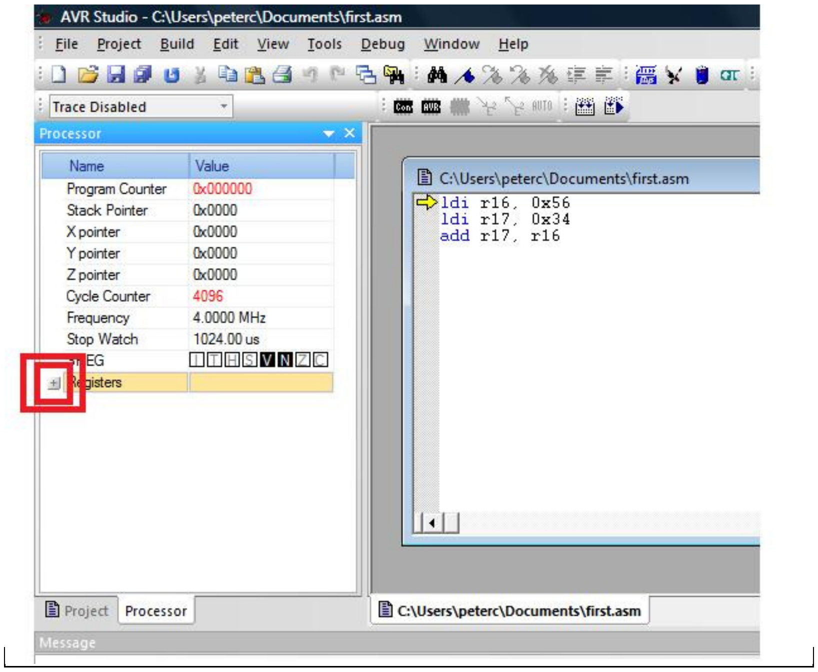
To start running the code, select "Start Debugging" from the "Debug" menu.

Take a look in the editor view, you'll see that a yellow right-arrow has appeared in the left margin of the code. This arrow indicates the position of the program counter. In o
words, it points to the next instruction to be executed (initially the first instruction in out program).



```
ldi r16, 0x56
ldi r17, 0x34
add r17, r16
```

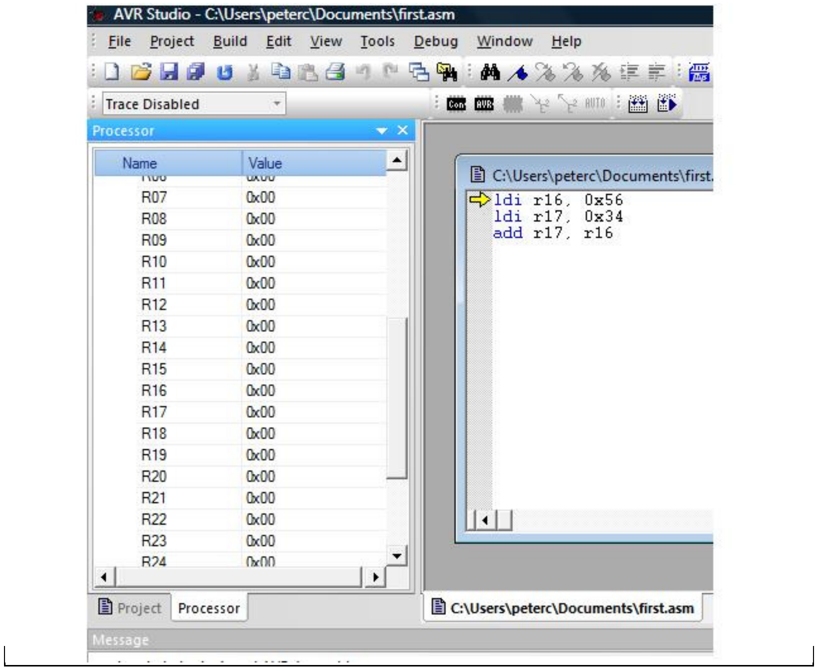
Before we start simulating we want to set the simulator view so that we can have a closer look at what is happening with registers 16 and 17 during program execution. In
Processor tab expand the Register file by click the **+** next to "registers".


AVR Studio - C:\Users\peterc\Documents\first.asm
File Project Build Edit View Tools Debug Window Help
Trace Disabled
Processor
Project Processor
C:\Users\peterc\Documents\first.asm
Message

Name	Value
Program Counter	0x000000
Stack Pointer	0x0000
X pointer	0x0000
Y pointer	0x0000
Z pointer	0x0000
Cycle Counter	4096
Frequency	4.0000 MHz
Stop Watch	1024.00 us
SREG	I T H S V N Z C
Registers	

```
ldi r16, 0x56
ldi r17, 0x34
add r17, r16
```

Scroll down so we can see registers 16 and 17. Note that both registers have a value of 0x00.


AVR Studio - C:\Users\peterc\Documents\first.asm
File Project Build Edit View Tools Debug Window Help
Trace Disabled
Processor
Project Processor
C:\Users\peterc\Documents\first.asm
Message

Name	Value
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0x00
R18	0x00
R19	0x00
R20	0x00
R21	0x00
R22	0x00
R23	0x00
R24	0x00

```
ldi r16, 0x56
ldi r17, 0x34
add r17, r16
```

Single Step Execution

There are two commands to single step through the code. These are "Step Over" <F10> and "Step Into" <F11>. The difference between these commands is that "Step Over does not trace into subroutines. Since our example does not contain any subroutines, there is no difference between the operation of these commands in this example. Pres <F11> to execute the first instruction (i.e. the one being pointed at by the arrow). Watch the value of r16 change as expected (to 0x56). Note that the arrow has moved to t next instruction. Press <F11> twice more to execute the 2nd and 3rd instructions. Are the values of registers 16 and 17 their expected value?

An alternate version of this tutorial may be found at the Atmel AVR resources page for a more complete description of the simulator's capabilities using a larger example program. Simulating processor I/O is also covered.

Procedure

Task 1:

Create a new AVR Studio Assembler project called "prac5". Write an assembly language program for the AVR8515 Project Board to:

- Input 8-bits of data from the port C pins
- Output the result to port B

Please note that this program is to run continuously - constantly monitoring port C and updating port B if necessary.

The skeleton code for this program is provided in prac5.asm. Replace the lines ";<-YOUR CODE HERE->" with your code. The tutors will give a brief introduction to using the A I/O ports with assembly programming as a group exercise.

The following I/O registers will be used:

PINC	Port C Input Pins

The bits in a data direction register for a port should be set to 1 if that bit/pin is an output or 0 if that bit/pin is an input.

The following instructions may be of use:

Instruction	Usage	Example	Meaning
clr	clr reg	clr r1	Clear the given general purpose register (i.e. make all the bits equal to 0)
in	in reg, ioreg	in r17,PINA	Load the contents of an I/O register (ioreg) into a general purpose register (reg). This is the only instruction that may be used t retrieve data from an IO register (E.G. PINA).
out	out ioreg, reg	out PORTA, r18	Store the contents of a general purpose register (reg) to an I/O register (ioreg). This is the only instruction that may be used to write data to an IO register (E.G. DDRA or PORTA).
neg	neg reg	neg r1	Perform a two's complement negation of the contents of the given general purpose register (reg)
ser	ser reg	ser r16	"Set" the given general purpose register, i.e. make all 8 bits equal to 1. (Only works on registers r16 to r31)
inc	inc reg	inc r17	Increment (i.e. add 1 to) the given general purpose register
com	com reg	com r7	Invert the contents of the given register (one's complement, i.e. flip all the bits)

Build and simulate your project using AVR Studio.

Wire port B (pins 0 to 7) on the AVR board to 8 LEDs on the logic workstation. Wire Port C (pins 0 to 7) to 8 toggle switches.

Complete the Pony Prog tutorial to familiarize yourself with how to program the AVR board. Use the hex file you generated above (prac5.hex if you named your project prac5
Test that your program is working correctly.

Task 2

Modify your code for task 1 such that the output (Port B) is the 2's complement negation of the input (port C pins). Demonstrate your program working in both simulation anc the board.

Tutor Task

The tutor task will be based on you and your partners' preparation. You will be required to write, simulate and download a program that is an extension your preparation ta

Challenge Task 1 (Complete after you have been marked off)

Write an AVR program that reads in 8 bits from the port C pins and then treats the upper and lower nibbles as two separate unsigned binary numbers which are then addec together with the result output to port B. (This process should repeat continuously.) A nibble is half a byte - i.e. 4 bit grouping. For example, if the input is 01011011, the upp nibble is 0101 (5 decimal) and the lower nibble is 1011 (11 decimal). The output in this case should be 00010000 (16 decimal). HINT: Consider the following instructions: **mov** **swap**, **andi**.

Challenge Task 2

The *parity* of a data word is considered to be odd (1) if there are an odd number of 1s in the data. Likewise, the parity is even (0) if there are an even number of 1s in the da For example 11011000 has even parity as there are 4 1s in the data. 11101100 has odd parity as there are 5 1s in the data. Write and simulate an AVR program that detern the parity of an input 8-bit data word. You will need to investigate some new instructions to achieve this. HINT: Consider the **andi** and **lsr** instructions.

Back to Contents

Assessment

This practical is marked out of 4 and worth 2% of your mark for CSSE1000:

- Preparation - AVR assembly language code written (using supplied values for your login to this page) and result predicted - ***1 Mark***
- Demonstration - Required circuits demonstrated. - ***1 Mark***
- Documentation - All testing results are documented in workbook - ***1 Mark***
- Tutor Task - Tutor task completed and documented in workbook - ***1 Mark***

Equipment

- Computer
- 12V Power Supply
- AVR8515 Project Board with STK200 Download Cable
- Logic Workstation
- Hook up Wire
- Wire Strippers

References

- Atmel AVR Resources
- AVR Studio Tutorial (or PDF version)
- PonyProg Tutorial (or PDF version)
- Tanenbaum, Andrew S., *Structured Computer Organization*, 5th Ed., Prentice/Hall, 2006. ISBN: