

Fall 2006
CS/ECE 333
Lab 1 – Programming in SRC Assembly

Lab Objectives:

1. How to assemble an assembly language program for the SRC using the SRC assembler.
2. How to simulate and debug programs using the SRC ISA level simulator.
3. How to write simple SRC programs including programs which make procedure calls.

Lab Procedure

This lab will be comprised of an in-lab exercise to be completed in lab and a post-lab exercise to be completed and electronically submitted by the start time of your lab section the following week. For instance, if you have lab on Wednesday, then your postlab is due the following Wednesday.

You will need your textbook (Appendix B1) and the slides from class on implementing procedure calls in SRC to complete this lab.

In-lab Deliverable:

- Demonstrate your working findMax() function to the TA.
- **2 points extra in-lab credit** for spotting a minor bug/typo in the lab document.

Post-lab Deliverables:

Electronically submit the following to Toolkit (One code submission per pair):

1. findMax.asm. The procedure you generated to calculate the findMax function during the in-lab exercise.
2. call_findMax.asm program. The header of your assembly file must contain a comment header that contains your name and email ID and the name and email ID of any partner you may have worked with. If you collaborated significantly with others/other groups, please also credit them in the comments and describe the nature of the help you received.

Each code submission should contain comments at the top which include the following information:

- Name and email ID of both partners (if working with a partner)
- Typed Honor Pledge
- Comments crediting any help received

Collaboration Policy

You may work with one partner on the in-lab and post-lab exercise. If you choose to work with a partner, this partner must be the same partner for both in-lab and post-lab. You must put the names and email IDs of both partners in the top of your submitted assembly code files. **If you discuss with another person/group to solve the problem, you must also credit them in the comments of your program describing the nature of the help you received.**

You may talk with others about aspects of the coding exercise, but **you may not copy code** directly from others.

In-lab Exercise

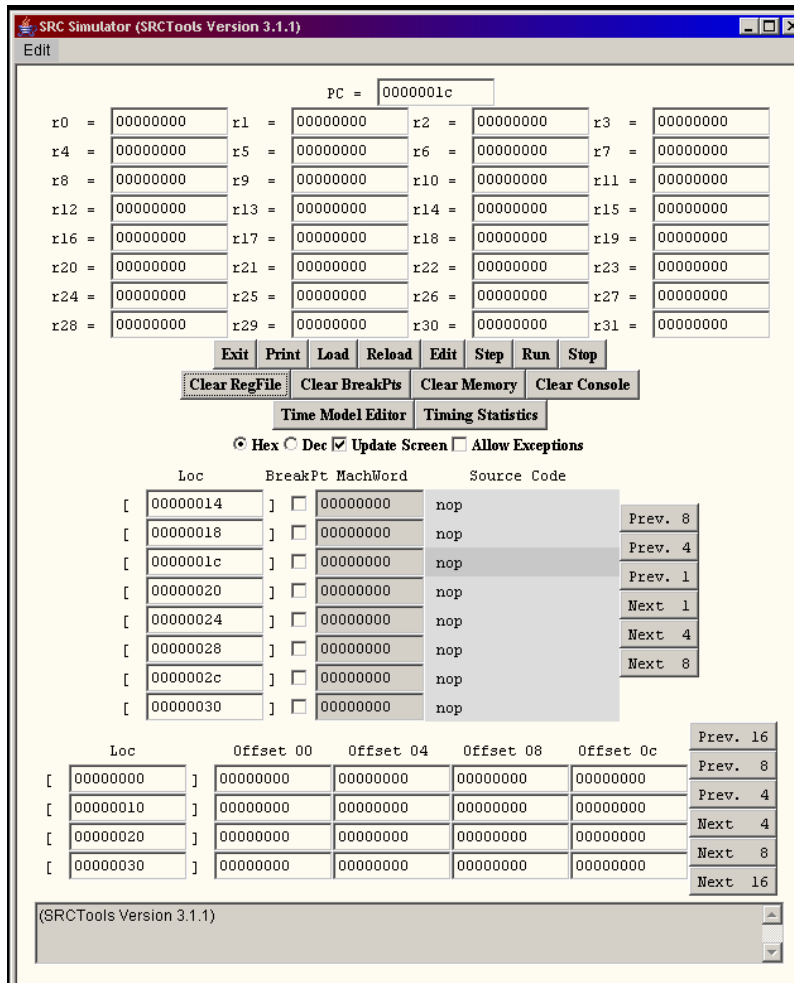
Getting started

This lab exercise is designed to show you how to use the SRC assembler and the SRC ISA level simulator. The assembler translates SRC assembly language, written according to the conventions shown in Table B.1 of Appendix B in the Computer Systems Design and Architecture text, into valid machine code for the SRC microprocessor. The SRC simulator simulates the SRC microprocessor at the ISA level. That means that it executes the SRC instruction set and shows the effects that each instruction has on the content of the programmer visible registers and the memory of an SRC-based system.

The SRC assembler and simulator have been built as a Java application which requires a minimum of Java 1.1 Runtime Environment to run.

1. If SRCTools is not already on the machine you are working on, download [SRCToolsv3.1.1.jar](http://www.cs.virginia.edu/~cs333/f06_simulators/SRCToolsv3.1.1/) (http://www.cs.virginia.edu/~cs333/f06_simulators/SRCToolsv3.1.1/) from the class website.

The simulator should be runnable in Windows by double-clicking on the application. You should see an interface that looks like this:



2. The display shows an editable text field showing the contents of the program counter (PC) register, an editable text fields displaying the 32 general-purpose registers numbered from r0 to r31, a row of control buttons, a display of 8 machine words with their addresses, complete with checkboxes for setting breakpoints, disassembled source code, and navigation buttons for displaying the program code in other memory locations (these fields are not editable) and an editable display of memory addresses and contents. [NOTE: All numbers are represented in hexadecimal.]

The functions of the control buttons are as follows:

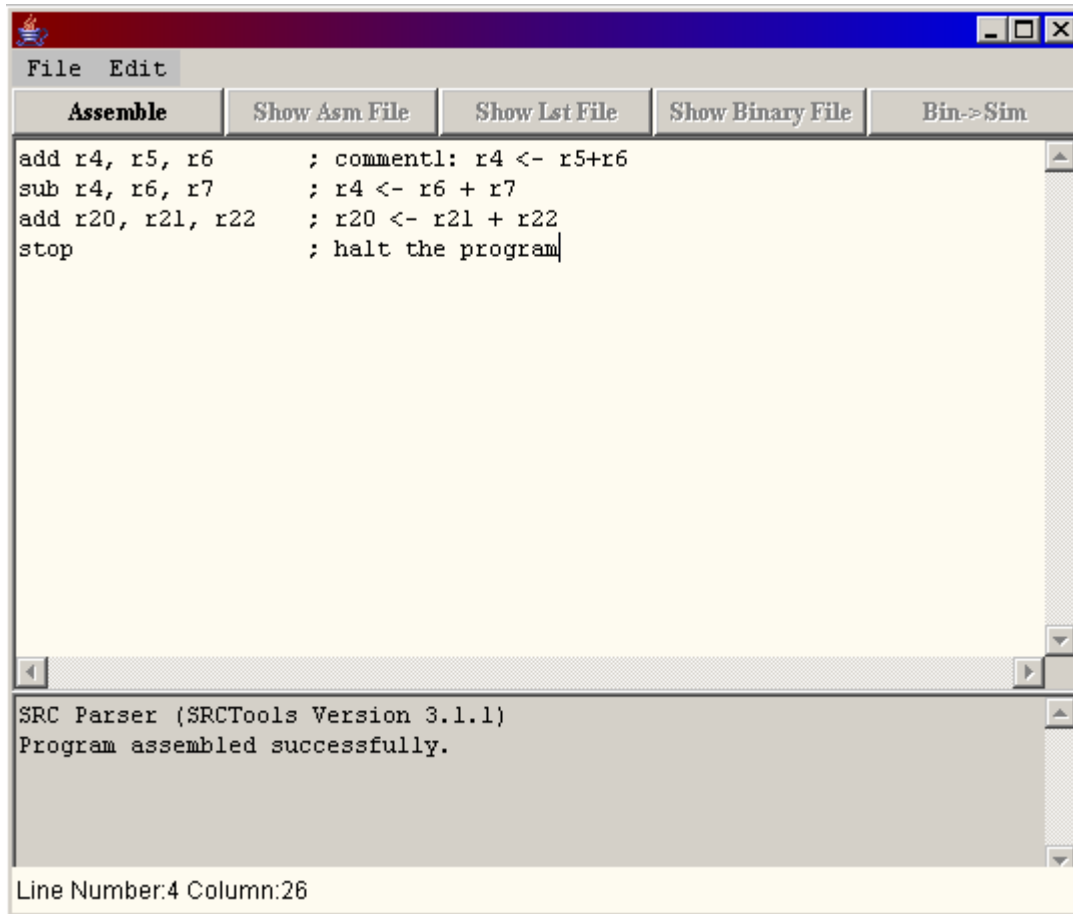
- The **Exit** button exits the simulator by terminating the application.
- The **Load** button allows the loading of a binary SRC file by bringing up the standard file opening dialog that your machine supports.
- The **Reload** button allows the reloading of a binary SRC file.
- The **Edit** button opens the SRC editor and assembler. It allows you to open an existing file or create new files.
- The **Step** button executes the single machine instruction pointed to by the PC. Note that this instruction is displayed in the center of the screen and is highlighted.

- The **Run** button runs the program beginning at the address pointed to by PC. Note that the machine will run until it encounters an SRC *stop* instruction or a breakpoint. In the event that the program does not contain a *stop* instruction or a breakpoint, it can be stopped by pressing the **Stop** button.
- The **Clear RegFile** button clears all registers. The contents of individual registers can be modified by editing the fields at the bottom of the window.
- The **Clear BreakPts** button clears all breakpoints in the program. Individual breakpoints can be set and cleared using the checkboxes in the center of the window next to the machine language display.
- The **Clear Memory** button clears all memory. The contents of individual memory locations can be modified by editing the fields at the bottom of the window.
- The **Clear Console** button clears the console at the bottom.
- The **Stop** button stops execution of the simulator.
- The **Next 4** and **Next 8** buttons and the other buttons like them next to the program and memory display are used to step through the machine code and memory display.
- The **Time Model Editor** button allows you to change various aspects of the machine such as:
 - i. Instruction parameters. The execution time (in clocks) of groups of instructions or individual instructions
 - ii. Memory I/O parameters.
 1. L1 cache size and latency
 2. L2 cache size and latency
 3. Memory read and write latency
- The **Timing Statistics** button allows you view various statistics associated with the execution of the program.

Creating and Assembling a Program

1. Click on the **Edit** button. The editor/assembler interface should open. Copy and paste the following simple program that demonstrates register adds to the editor window.

```
add r4, r5, r6      ; r4 <- r5 + r6
sub r4, r6, r7      ; r4 <- r6 - r7
add r20, r21, r22   ; r20 <- r21 + r22
stop                ; halt the program
```



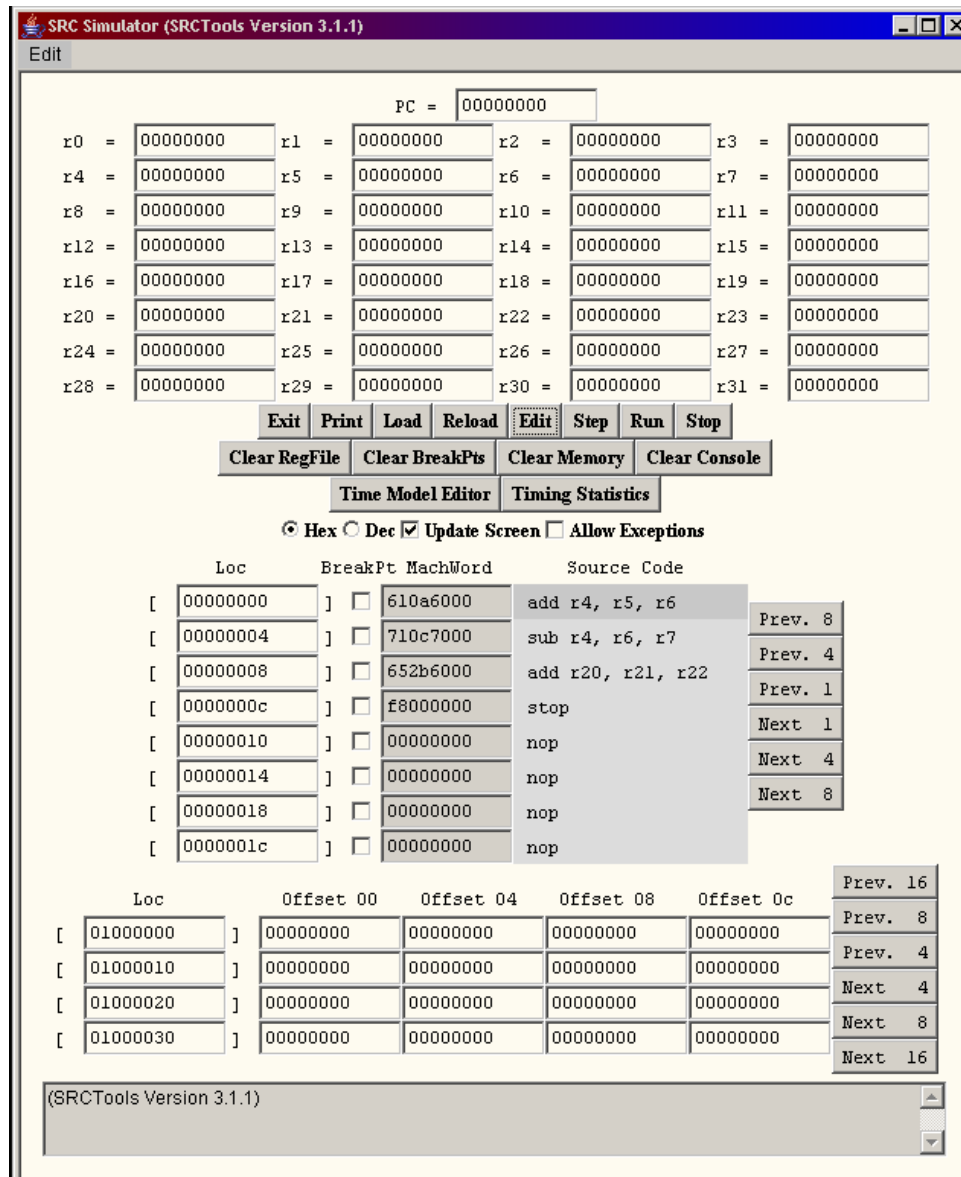
2. Click on the **Assemble** button to assemble the program. A display of **.lst** file will then be visible:

```
(SRCTools Version 3.1.1)
HexLoc    DecLoc    MachWord  Label  Instruction      Comment
00000000  000000000    610a6000          add r4, r5, r6   ; comment1: r4 <- r5+r6
00000004  000000004    710c7000          sub r4, r6, r7   ; r4 <- r6 - r7
00000008  000000008    652b6000          add r20, r21, r22 ; r20 <- r21 + r22
0000000c  000000012    f8000000          stop             ; halt the program
```

--- Symbol Table ---

If there are errors in the assembly file, messages will be displayed in the text window beneath the editor window. The file can be edited and reassembled by hitting the **Show Asm File** button and making the desired changes. Files can be saved from **File->Save** or **File->Save As**.

3. Click on **Bin->Sim** to load the program into the simulator. Alternatively, if you have already created and assembled a program, from the simulator window, click **Load** and load the **.bin** file for your assembled program.



- Notice that the PC starts at 0x00000000 which is the default program start address if no other is specified and that all the registers are initialized to 0.
- Edit the values of r4, r5, r6, r7, r20, r21, and r22 to reflect non-zero values, then **Step** through the program and watch the register contents.

Assembler Error Messages

- Download `errors.asm`, load it in the SRC editor and assemble it.
- Examine the error messages displayed in the `.lst` file output. These are some common semantic errors caught by the assembler's parser. Syntax errors are handled differently since a `.lst` file cannot be created and will be displayed in the text window beneath the editor window.

Writing a Procedure : findMax()

- Below is a C procedure which describes finding the maximum of three integers. It overwrites val1 to contain the maximum of the 3 input parameters.

```
int findMax(int val1, int val2, int val3)
{
    /* find the max of val1 and val2 */
    if(val1 < val2)
        val1=val2;    /* set val1 = the max of the comparison */

    /* find the max compared to val3 */
    if(val1 < val3)
        val1=val3;    /* set val1 = max of val1, val2, val3 */

    return val1;
}
```

- Download findMax.asm. This file contains a small amount of setup code to allow you to test your findMax() implementation. It is not an actual main() function even though there is a Main label. This file contains some code that sets up the stack with some values on it so that your function implementation can start by reading the parameters from the stack.

Using your textbook **Appendix B1** and class notes on procedure calls as reference, write an assembly code program that implements this procedure. For this exercise the convention is:

- The stack pointer (SP) will be held in r30
 - The link register will be r31
 - Arguments will be saved on the stack starting with the first parameter (val1), then second parameter (val2), then third parameter (val3)
- To implement this function in assembly code, you will have to:
 - Load the parameters passed to findMax() in the stack to r0 (param1), r1 (param2), and r2 (param3).
 - Write the body of the function
 - Hint:** SRC has no compare instruction. Consider using sub to help perform the comparison needed
 - Write code to perform the return sequence
 - Place the return value in the appropriate place on the stack
 - Load the return value
 - Restore stack pointer
 - Transfer control back to the calling function

Post-lab Assignment

The electronic submission for your post-lab is due at the ending time of your next lab section the following week.

- Write a main() function to perform the call to findMax(). This file should also contain your findMax() function.

- a. It should:
 - i. Load the parameters `val1`, `val2`, `val3` from their respective memory addresses
 - ii. Save the arguments to `findMax` on the stack, making the appropriate adjustments to the stack pointer
 - iii. Allocate space for the return value
 - iv. Call `findMax`
 - v. Terminate execution (stop)

Follow these guidelines when implementing your `main()` function:

1. Place inputs `val1`, `val2`, `val3` at address `0x0000`, `0x0004`, and `0x0008` respectively
2. Allocate a space for the return value at memory address `0x0010`
3. Start the main program at memory address `0x1000` and have the stack grow **downwards** (towards memory address `0x0000`) away from `0x1000`.