mkdir CS 250 2017 Fall Lab 07 Instructions

ARM Assembly and Calling Conventions

Download the Lab 7 tarball from Blackboard and untar it in on your Pi in your ~/cs250/lab07-src directory. Use the command line

$ tar xfv lab07-support.tar

Open and inspect the file “arm\_template.s” provided in the archive above. It provides a template that you can use for all of your assembly programs. The template executes an empty command and then exits.

**1. Using GDB**

We will begin by using the GNU Debugger (GDB) that you are familiar with from CS 240. To use GDB on a program, you need to invoke gcc with an extra flag, then run GDB:

$ gcc -g -o gdb\_example gdb\_example.s

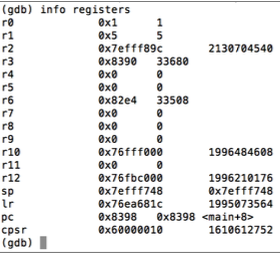
$ gdb gdb\_example

Inside GDB, run these instructions:

(gdb) break main

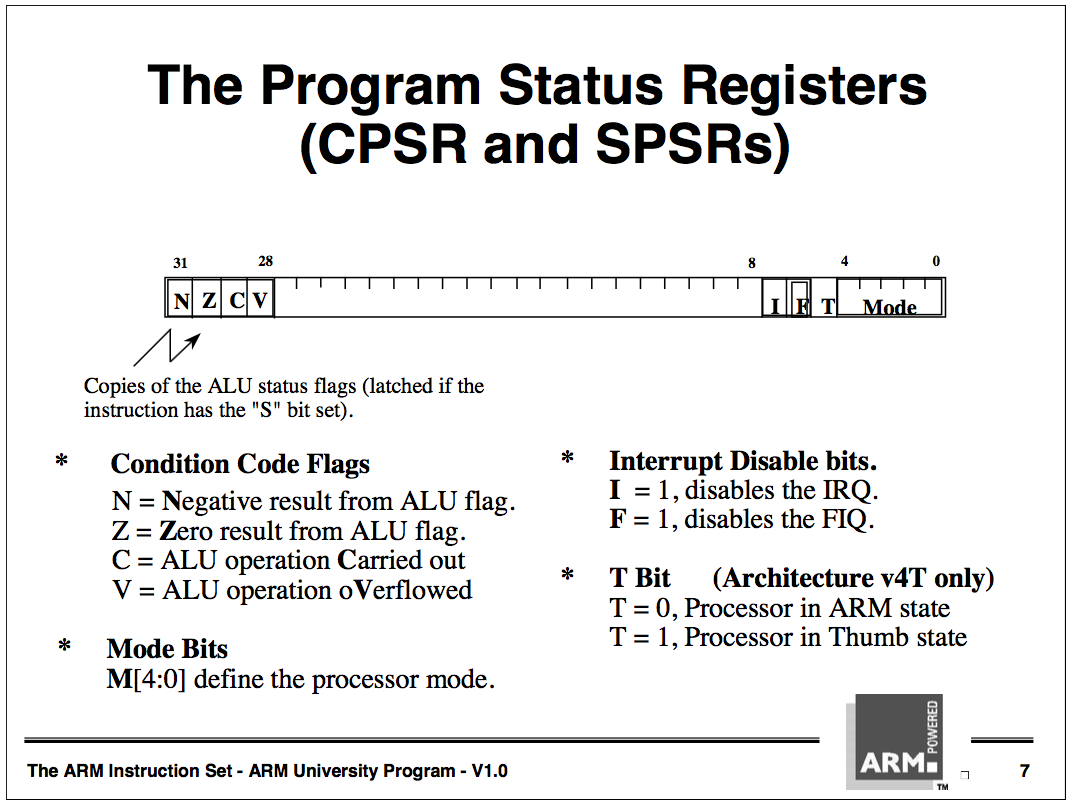
(gdb) run

These set a break point at the beginning of main and start the execution of your program inside GDB. Now type “next” and “info registers” and you should see this:



Notice how the first line of the program stored 5 in register r1.

Now, examine the Current Program Status Register (cpsr). The first two 1-bit fields in the format of the bit string stored in this register are of interest for the example program.

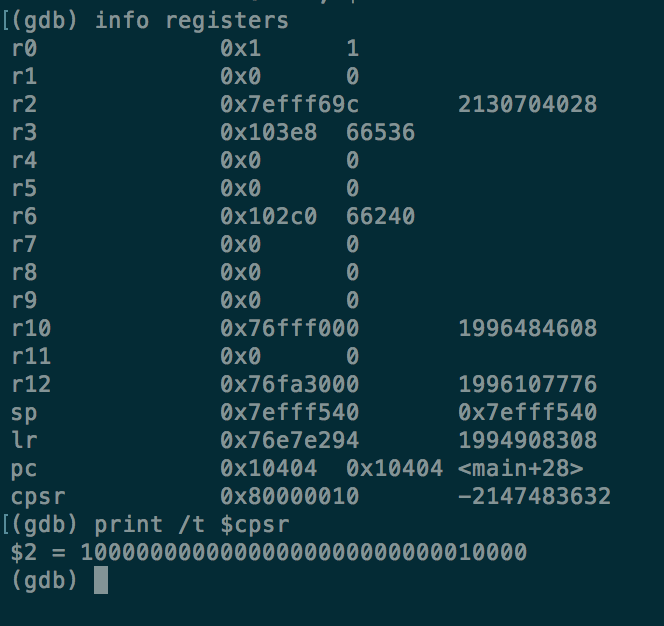


The diagram above shows that bit 31 is the “N,” or negative bit and bit 30 is the “Z,” or Zero bit. Bit N will be set to 1 if the signed value from the ALU is negative. Bit N will be cleared, made a logic 0, if the aLU result is positive or zero. The Z bit is set if the ALU result is zero, which is the case if the ALU is used to compare and the two compared bit strings were equal. If the result from the ALU is non-ero, then the Z bit is cleared.

Within your running instance of GDB, to print the contents of cpsr, type “print /t $cpsr”, where /t means print as a binary integer. Be aware, /t omits printing leading 0 bits, therefore, notice that only 31 bits are displayed by the “print /t $cpsr” command at this time, instead of 32 bits.

Continue stepping through the program in GDB. After each “cmp” instruction, each of which compares register r4 to the integer 4, print cpsr. This should give you a good idea how comparisons work in ARM.

**Problem 1 [10 pts]** Take a screenshot where you have printed the value of cpsr at any point after the first comparison in this program. Identify on the screenshot where the N and Z bits are and why the N and Z bits have the value that they do.



N bit is 1 and Z bit is 0

N bit is 1 because the signed value from the ALU is negative

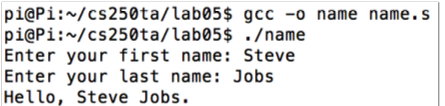
The result from the ALU is non zero that’s why the Z bit is cleared

**Submit your solution to Blackboard.**

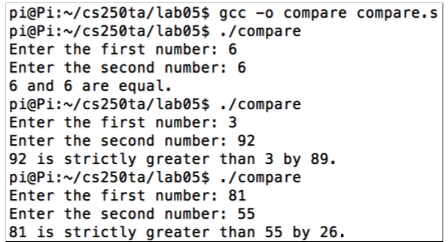
**2. ARM Assembly Programming**

In this section, we will increase our familiarity with ARM assembly.

**Problem 2 [20 pts]** Create a program, “name.s”, that reads the user’s first name from stdin, then reads their last name from stdin, and finally prints “Hello, firstName lastName.”. If you find that your output will not work with the testall program (read the grading section below for info on testall), add the commands “mov r0 $0” and “bl fflush” right at the end of your program to flush your output to stdout.



**Problem 3 [20 pts]** Create a program, “compare.s”, that reads in a number x from stdin, then reads in a second number y from stdin. If the numbers are equal, print “x and y are equal.”; otherwise, print “max{x, y} is strictly greater than min{x, y} by |x − y|.”.



You have been provided with a testall file. Type “./testall” in the terminal to check the correctness of name.s and compare.s.

**3. Calling Convention - C to Assembly and Assembly to C**

To learn more about calling conventions from C to assembly and vice versa, we will write our own C and assembly functions and call them from the other language.

For this section, you should develop two programs. Each will consist of two source files, one in C and another in assembly. The task of both programs is to extract a specific sub-string from a given string.

The difference between the two projects lies in the way that they are implemented:

* For program 1, the main() function should be written in assembly (main.s), and the subroutine to extract the sub-string should be written in C (sub\_string.c)
* For program 2, the main() function should be written in C (main.c), and the sub-routine to extract the sub-string should be written in assembly (sub\_string.s)

Use ARM assembly for this task, so you should use your Raspberry Pi to execute the programs.

**3.1 Specification**

Your main() function should:

* Read a string from stdin. Ex: “publication”
* Read the start index and end index as integers. Ex: start index = 3, end index = 7.
* Call the function, or subroutine, “sub\_string” and pass the above three inputs as arguments
* Print the returned sub-string from the function. Ex: “blica”. This is the sub-string starting from character 3 and ending at 7.

Your sub\_string() function should:

* Read the three arguments from the caller (main)
* Extract the sub-string from the given string
* Return the sub-string to the caller (main)

The output for both the programs should match that given below:

Enter a string: Publication

Enter the start index: 3

Enter the end index: 7

The substring of the given string is 'blica'

**3.2 Program 1 – Assembly to C**

What follows are some prototypes and pseudocode for your first program. You should produce two files: “main.s” and “sub\_string.c”. To compile and execute your program, execute the following:

$ gcc -o asm2c main.s sub\_string.c

$ ./asm2c

sub\_string.c

char\* sub\_string(char \*in\_string, int start\_index, int end\_index) {

char \*out\_string;

/\* code to extract the sub-string \*/

return out\_string;

}

main.s

/\* Data declarations as necessary \*/

/\* Code to receive inputs from user \*/

bl sub\_string

/\* Code to print the sub-string \*/

**3.3 Program 2 – C to Assembly**

What follows are some prototypes and pseudocode for your first program. You should produce two files: “main.c” and “sub\_string.s”. To compile and execute your program, execute the following:

$ gcc -o c2asm main.c sub\_string.s

$ ./c2asm

sub\_string.s

/\* assembly declarations and code as required \*/

sub\_string:

/\* Code to extract the sub-string \*/

/\* Suitable code to return the sub-string \*/

main.c

int main() {

int start\_index, end\_index;

char \*in\_string;

char \*out\_string; /\* Code to receive inputs from user \*/

/\* Code to print the sub-string \*/

out\_string = sub\_string(in\_string, start\_index, end\_index);

return 0;

}

**Problem 4 [50 pts]** Complete the above templates and implement the above specifications.

**You are required to use the file names that are given above.**

**NOTE:** Credit for this lab will be given based on the functionality of your program, useful comments on every line, simplicity/organization of your code, and regularity matching the filenames and output format shown above.

**4. Turnin**

Follow these instructions to turn in Lab 07:

1. Your six program files need to be named exactly as specified in the lab handout: name.s, compare.s, main.s, sub\_string.c, main.c, and sub\_string.s
2. Put all six of these files in a directory named lab07-src.
3. Copy this directory to data.cs.purdue.edu:  
   $ scp -r lab07-src username@data.cs.purdue.edu:~/
4. Submit this directory electronically (from data.cs.purdue.edu):  
   $ ssh username@data.cs.purdue.edu  
   $ turnin -c cs250 -p lab7 lab07-src   
   $ turnin -c cs250 -p lab7 -v  
     
   The second command verifies that you have successfully submitted your files.

**5. Due Date**

The due date for this lab assignment is the week of Oct. 31 – Nov. 03

Problem 1 should be submitted on Blackboard.

All other problems should be submitted using the turnin command and appropriately demonstrated in lab to your TA.