

Datorteknik

Lab 2: ARM Assembly Language

Todor Stoyanov

September 27, 2018

1 Objectives and Lab Materials

In this lab you will learn how to program low-level assembly code and how to interface that from C.

The materials for this lab are:

- A Windows PC with qemu installed.
- A raspbian kernel for qemu.
- A disk image with raspbian.

Complete the following tasks. Provide all source code, including appropriate inline comments. In the report explain how the code works and describe what test you performed the validity of your code.

1.1 Preliminaries

Before starting with this lab please verify you have performed these preliminary steps:

- Obtain the folder with qemu virtual machine settings from `N:\AASS\qemu_vms/` or ask the lab assistant if you have trouble finding it.
- Edit the file `start_qemu.bat` so that the kernel file and image file are taken from your local directory. Store the file in a local folder.
- Double-click on the `start_qemu.bat` file and verify that Raspbian boots up.
- In case you need them for later, the username for raspbian is `pi` and the password is `DT509G_pi`
- The rest of the lab assumes you are working within the raspbian emulator.

- The command-line text editors `nano` and `vi` are pre-installed on the raspbian image.
- The following links give some useful references for ARM assembly programming:
 - <https://azeria-labs.com/writing-arm-assembly-part-1/>
 - <https://thinkingeek.com/arm-assembler-raspberry-pi/>
 - http://www.science.smith.edu/dftwiki/index.php/Tutorial:_Assembly_Language_with_the_Raspberry_Pi

2 Task 1: Basic Setup (5 points)

Write an assembly program which loads two immediate integer values into the registers, adds them, and then calls the external function `printf` to display the result. Compile your program using the assembler `as` and then link the object file into an executable, using `gcc`. Verify that your code produces correct values and demonstrate your program to the lab instructor.

3 Task 2: Shifting Integers (10 points)

Note: this task follows Lab 7 from the textbook.

- Write a C function `int_out` which takes an integer argument and prints the value in hexadecimal notation.
- Write a separate `main` function to test that `int_out` produces the expected output.
- Compile `int_out` into an object file.
- Write an assembly program which loads an immediate value of 4 into a register and then calls the external function `int_out` to print it.
- Compile the assembly program and link the two object files (assembly and C) into an executable. Verify that the output is as expected.
- Load the integer `0xBD5B7DDE` instead of 4. Verify that the sign extension works as expected when you bit shift. Note: you should use arithmetic shift for signed numbers, and load data from memory instead of by using immediate values. Check the instructions `ldr` and `asr`).
- Modify your program to use `printf` directly instead of your custom function.



4 Task 3: Assembly in C (10 points)

Note: this is an adapted version of Lab 8 from the textbook.

- Write a C function `xor` which computes the bit-wise exclusive or of two integer arguments.
- Write a main function to test your `xor` implementation.
- Write an assembly version of the `xor` function (`axor`). Do that from scratch instead of relying on gcc to generate the assembly code for you.
- Declare `axor` as an external function in your C main source. Call `axor` and compare the results to the C code `xor` implementation to verify that the assembly code works as expected. Test your implementation over a range of random inputs.