CS 382
Fall 2023

Computer Architecture and Organization

Lab 2

In this lab, we're going to write and execute assembly programs for the first time.

1 Task 1: Install QEMU Emulator

Recall that assembly programs are closely related to the hardware platform they are performed on, so an assembly program written in ARM syntax cannot be executed on any other machines. Most of the time, however, we also want to execute ARM assembly on some other machines. In this case, we can install an **emulator** that can simulate the hardware execution of ARM machines on any type of machine. This is called **cross-compilation**. The emulator we are using is called QEMU.

To install QEMU on your virtual machine, type the following in your terminal:

```
$ sudo apt-get install qemu-user
```

Once you have finished the installation, you need to write the simplest ARM assembly program (the one listed in B.1.1 in textbook), and assemble and execute it. Upon success, there should be nothing printed out — no warnings or any type of output. To learn how to assemble, link, and execute an assembly program, read B.2.1 in textbook.

To get credits for attendance, show your CA that you have installed QEMU, and can assemble/link/execute the simplest ARM program. No submission needed.

2 Task 2: Write a Simple Program

One very common thing we do in programming is to print something to the screen. Although printing numerical numbers takes a bit more work, it is relatively much easier to print a string. In this task, you will learn how to declare a string, and how to invoke system call to print the string out. The code listed in B.1.1 in textbook is your starter code. Please learn how to declare data and load address by reading B.1.3 in textbook before starting this task.

Unlike high-level languages, in assembly, if we want to print a string, we have to declare it first in the `.data` segment:

```
.data
msg: .string "Hello World!\n"
```

To print this string out, we need to prepare several registers before invoking the system call, because it needs to retrieve the information about this string from these registers:

Register	Content
`X0`	Destination (for printing, its value is `1`)
`X1`	The address of the string to be printed
`X2`	The length of the string to be printed
`X8`	System call number (for printing, its value is `64`)

Once these registers are ready, we can invoke the system call by using instruction: `SVC 0`, and the string will be printed out!

Lastly, to terminate the program successfully, we need the following instructions: [1]

```
MOV   X0, 0
MOV   X8, 93
SVC   0
```

Requirements

- **Note** your code is a **complete** assembly program (not just a sequence of instructions). It must be able to assemble, link, and execute without error and warnings. When executed, the program must finish without problems;
- If your code cannot assemble, you get no credit – this is the same to C programs that cannot be compiled;
- You must declare the length of the string as a quadword in the `.data` segment;
- You must put comments on every instruction you wrote; no need to comment on labels and directives;
- You must put your name and honor code pledge at the top of your code in comments.

3 Grading

Task 2 will be graded based on a total of 10 points. The following lists deductibles, and the lowest score is 0 – no negative scores:

- **-10:** the code does not assemble, or the program terminates abnormally/unsuccessfully;
- **-10:** the code is generated by compiler;
- **-2**: the length of the string is not declared as a quad data in the `.data` segment;
- **-2**: one or more instructions is missing comments;
- **-2**: no pledge and/or name.

Earlybird Extra Credit: 2% of extra credit will be given if the lab is finished by Wednesday 11:59PM EST (1 day before the lab deadline). For specific policy, see syllabus.

Attendance: show your CA completed task 1 and check off at the end of the lab to get attendance credit.

Deliverable

Submit a single `.s` file.

[1] For those who are curious and interested: you must have noticed some similarities between the procedure of printing string and terminating the program: both of them prepared some values to registers, and moved a number (system call number) to register `X8`, and executed `SVC`. This is because printing needs to invoke a system call called `write()`, while terminating a program called `exit()`. You will learn all about these in detail in CS-392 Systems Programming. You can see the details here: https://chromium.googlesource.com/chromiumos/docs/+/HEAD/constants/syscalls.md#arm64-64_bit.