Lab1: Hello World

# 1 Objective

To become familiar with the basics: text editor, assembler, linker, and debugger. After finishing this experiment, you should be able to do the following:

1. Use a text editor to create an assembly source code (.s).
2. Understand the general procedure to develop and debug an assembly program.

# 2 Background

## 2.1 Assembler

In the following, we assume you have created aliases, e.g., “as32” for arm-linux-gnueabihf-as, and “ld32” for arm-linux-gnueabihf-ld.

To assemble a program (assuming the file name is lab1.s), one should type the command line:

$ as32 -g -o lab1 . o lab1 . s

where .s file is the source file and .o file is the output object file containing the machine code. We include debugging information by using -g.

## 2.2 Linker

The linker creates an executable file (or a library) from one or more object files:

$ ld32 −o lab1 lab1 . o

To run the program:

$ ./ lab1

The entry point of an assembly source program is usually referred to as “\_start”. If necessary, we can change the entry point to “main”:

$ ld −e main −o lab1 lab1 . o

As before, the linker will generate an executable file named “lab1”.

## 2.3 Debugger

The GNU debugger (**gdb**) allows you to execute, trace, inspect, and change variables during program execution. GNU DDD is a graphical front-end for command-line **gdb** debuggers.

# 3 Laboratory Workflow

## Part 1: “Hello World” Program

In this section you will create an assembly program that calls the “printf” function from the C runtime library.

1. Create a file named lab1p1.s by copying its content from the Appendix. You can use vim or some other text editor.
2. Assemble and link the files.

$ as32 -o lab1p1.o lab1p1.s

$ ld32 -o lab1p1 lab1p1.o

1. Execute the program by typing ./lab1p1. You should see the output“Hello World!”.

Next, run lab1p2.s, which implements the “printf” function from the C library in Assembly. You should see similar output as lab1p1.s.

## Part 3: Use the command-line tool gdb for debugging

In this section, we will use gdb to debug the **lab1p2.s** program.

In C programming, you can print out the value of each variable to make sure your program is functioning properly. In assembly, the **registers** take the position of “variables”, and you can examine their values with a debugger.

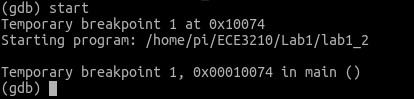
1. To start the debugger:

$ gdb lab1p2

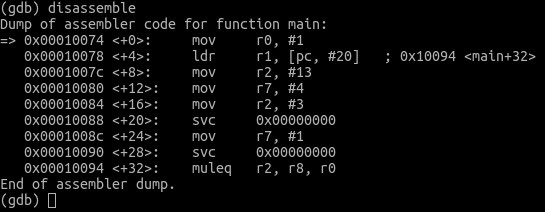
You should see the following message on your screen.



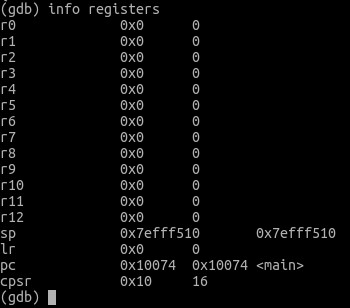
1. Start the debug procedure by typing “**start**” within gdb:



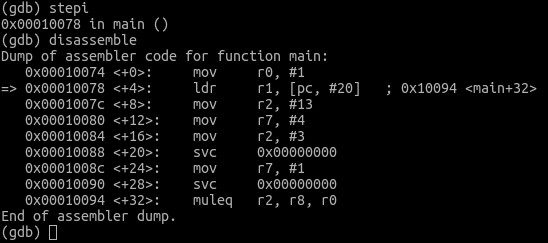
1. Next, we use the command “**disassemble**”. There is an arrow =>pointing to the next instruction to be run.



1. Before running the program, let’s inspect some register values by using the “**info**” command.



1. To step into each instruction one by one, we use the “**stepi**” command. Then we can follow with another “**disassemble**” to see what has happened.



1. Repeat Steps 3˜5 and fill the table with register values after each instruction has been run.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | R0 | R1 | R2 | R7 |
| MOV R0,#1 |  |  |  |  |
| LDR R1, =message |  |  |  |  |
| MOV R2, =length |  |  |  |  |
| MOV R7, #4 |  |  |  |  |
| MOV R2, #3 |  |  |  |  |
| SWI 0 |  |  |  |  |
| MOV R7, #1 |  |  |  |  |
| SWI 0 |  |  |  |  |

Table 1: Instruction trace table.

**Lab deliverable 1**

Include Table 1 above in your lab report.

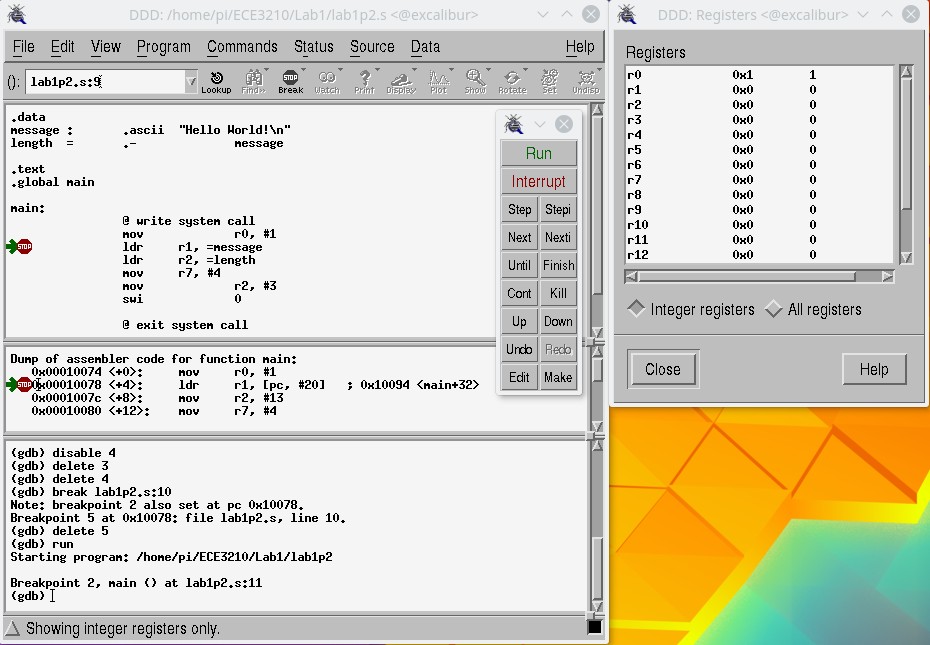


Figure 1: The DDD debugger.

## Part 3: Use the graphical interface DDD for debugging

In this section, we will use DDD to debug the **lab1p2.s** program. Enter the command:

$ ddd lab1p2

Under the “**View**” tab, open the “**Machine Code Table**”. Under the “**Status**” tab, click “**Registers**”. Now your interface should look similar to Figure 1. You can set a breakpoint by clicking in the blank area (left side as shown in Figure 1) next to each instruction.

Once the breakpoint has been set, click the “**Run**” button to start debugging and the “**stepi**” button to trace each instruction. The value of each register should be displayed in the Registers status window on the right side.

**Lab deliverable 2**

**lab1p2.s** contains a small bug. After fixing the bug, change the program to print your name before Hello World, e.g., “John Doe Hello World!”. Include the modified program in your lab report.

# 5 Report

Please use the project report template. Describe your experiences in completing the project, and make sure to include Lab deliverables 1 and 2.

# 6 Appendix

**lab1p1.s**

.data

message: .ascii "Hello World!\n"

.text

.global main

main:

push {ip, lr}

ldr r0, =message @ Load the starting address of the message

bl printf @ Call the printf function

mov r0, #0 @ Return 0.

pop {ip, pc}

**lab1p2.s**

.data

message: .ascii "Hello World!\n"

length = . – message @ Returns string length of message

.text

.global main

main:

@ write syscall

mov r0, #1 @ For stdout

ldr r1, =message @ buffer is loaded with message

ldr r2, =length @ count is the length of message

mov r7, #4 @ write is syscall 4

mov r2, #3

swi 0 @ interrupt

@ exit syscall

mov r7, #1

swi 0