

# Lab 11



## Part A

### [Test Your Platform]

1. Log into the Hydra machine or SSH into the Hydra machine.
2. Edit a new file called lab\_test.S (capital S)
3. Type the following into the file:

```
.section .rodata
hello: .asciz "Hello World\n"

.section .text
.global main
main:
    stp x29, x30, [sp, -16]!
    adr x0, hello
    bl printf
    ldp x29, x30, [sp], 16
    ret
```

4. Save the file and exit the editor
5. Assemble the file by typing: `aarch64-linux-gnu-gcc -o lab_test lab_test.S`. If you get a path not found, see your TA.
6. This will create a file called "lab\_test"
7. Execute this assembly program by typing `./lab_test`
8. You should see "Hello World". If you do not, contact your TA. This step is to ensure that your assembly environment works!

## Part B

### [Step 1]

1. Download [calc.c](#) . This will be your driver code. **DO NOT MODIFY THIS CODE!**
2. Create a file called calc.S. You will be compiling calc.c and calc.S together using the following command:  
`aarch64-linux-gnu-gcc -o calc calc.c calc.S`

## [Step 2]

1. Create the following function labels: Add, Sub, Mul, Div, ShiftLeft, ShiftRight, SetGlobal, and GetGlobal

## [Step 3]

1. Add takes two parameters (left and right) and returns an integer that is left + right.
2. Sub takes two parameters (left and right) and returns an integer that is left - right.
3. Mul takes two parameters (left and right) and returns an integer that is left \* right.
4. Div takes two parameters (left and right) and returns an integer that is left / right.
5. ShiftLeft takes two parameters (left and right) and returns an integer that is left << right.
6. ShiftRight takes two parameters (left and right) and returns an integer that is left >> right.

## [Step 4]

1. Create a global variable with 10 integers in the .data section. Initialize integer[0] to 0x11111111, integer[1] to 0x22222222, ..., integer[9] to 0xaaaaaaaa.
2. SetGlobal will take two parameters (index and value). Index will be 0-9. This function will set the integer given by the index to the given value.

## [Step 5]

1. GetGlobal will take a single parameter (index) and return an integer. Index will be 0-9. This function will return the global integer based on the index. For example, SetGlobal(1, 155) will set the 2nd integer (index 1) to 155. GetGlobal(1) will then return 155.

## [Testing Your Code]

After running the program with ./calc, type "h" for a list of commands and how to use them. **NOTE:** calc.c was modified during lab. The most recent version has the help command. If your version does not, simply download calc.c again.

## Part C


## [Overview]

You will be writing a floating point calculator for 32-bit floating point values. Therefore, you will be using Aarch64 assembly to extract the sign, exponent, and fraction portions out of a floating point number.

## [Restrictions]

1. You may not use any global variables (including .text immediates [i.e., `ldr x0, =0xffffffff`]).
2. All functions must be written in a single assembly file.

## [Download Files]

1. Download the file [float.c](#)  (Do NOT modify this file).
2. You will be writing the functions: `GetSign`, `GetExponent`, `GetFraction`, `GetNorm`, and `GetRaise2` in a new file called `float.S`

## [GetSign]

1. Write the assembly instructions for `GetSign`.
2. This function will supply you with the entire floating point number in register `w0`.
3. Store the sign bit into register `w0` ( $w0 \in [0,1]$ ) and return.

## [GetExponent]

1. Write the assembly instructions for `GetExponent`.
2. This function will supply you with the entire floating point number in register `w0`.
3. Store the actual exponent (exponent  $\in [-127..128]$ ) into register `w0` and return.

## [GetFraction]

1. Write the assembly instructions for `GetFraction`.
2. Your function will be supplied the entire floating point number in register `w0`.
3. Store the fraction portion only in register `w0` and return.

## [GetNorm]

1. Write the assembly instructions for `GetNorm`.

2. Your function will be supplied the fraction portion (from GetFraction) in register w0.
3. This function sets the integer portion of the floating point number (in w0) to 1 while leaving the fraction portion untouched.
4. The purpose of this function is to re-normalize the floating point number. What that means is that we want the mantissa's decimal portion to be 1, but we want to keep the fraction portion. The reason is because when we convert your number back into a float, the CPU will try to re-bias the number, even though we've removed the bias, since GetNorm only works with the fraction portion of the number. Basically, we want to return a floating point number whose value is 1.f, where f comes from your GetFraction function.
5. Return the result in register w0.

## [GetRaise2]

1. Write the assembly instructions for GetRaise2.
2. Your function will be supplied the exponent in register w0 and will always be supplied as  $\geq 0$  and  $< 32$ .
3. Determine  $2^{\text{exponent}}$  and return the value in w0. You may not use any external functions or repeated multiplication for this step!
4. Return the result in register w0.

## [Final product]

1. When all is finished I will calculate your number by the formula:  
$$\text{floating\_point\_value} = \text{GetSign()} [\text{GetNorm}(\text{GetFraction}()) \times \text{GetRaise2}(\text{GetExponent}())]$$
2. Compile your program as: `aarch64-linux-gnu-gcc -o float float.c float.S`
3. Test your program by: `./float -77.1234`
4. You should get the result -77.123339. Remember that there is a +/- "error" in a floating point number, that is why you don't get exactly -77.1234.
5. Do not worry about the special cases, such as infinity, -infinity, and NaN.

## [You are finished with this lab!]

Submit your calc.S file as **calc.txt** and your float.S file as **float.txt**.

Points 150

Submitting a file upload

File Types txt

<b>Due</b>	<b>For</b>	<b>Available from</b>	<b>Until</b>
Nov 7, 2017	Everyone	Nov 3, 2017 at 7:58am	Nov 7, 2017 at 11:59pm

**Lab 11 (1)**

Criteria	Ratings		Pts
PartB: Add function	<b>5.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	5.0 pts
PartB: Sub function	<b>5.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	5.0 pts
PartB: Mul function	<b>5.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	5.0 pts
PartB: Div function	<b>10.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	10.0 pts
PartB: ShiftLeft function	<b>10.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	10.0 pts
PartB: ShiftRight function	<b>10.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	10.0 pts
PartB: GetGlobal	<b>15.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	15.0 pts
PartB: SetGlobal	<b>15.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	15.0 pts
PartB: Global Integers	<b>15.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	15.0 pts
PartC: GetSign	<b>10.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	10.0 pts
PartC: GetExponent	<b>15.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	15.0 pts
PartC: GetFraction	<b>15.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	15.0 pts
PartC: GetNorm	<b>15.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	15.0 pts

<b>Criteria</b>	<b>Ratings</b>		<b>Pts</b>
PartC: GetRaise2	<b>20.0 pts</b> <b>Full Marks</b>	<b>0.0 pts</b> <b>No Marks</b>	20.0 pts
			Total Points: 165.0