Workshop 4

Introduction

We have just learned how to use various loops in both C and assembly. In this workshop, we will implement these loops in assembly to practice structured programming. The problems we are solving in these loops are fixed-point arithmetic operations for real numbers.

We have learned in Lecture 9 how to express real numbers using fixed-point Qm.n representation. We have also learned there how to do additions and subtractions on those numbers using ARM instructions. Basically, we just perform additions and subtractions on these numbers as if they are normal integers.

In this workshop, we will extend mathematical operations on these numbers to include multiplications and divisions, as well as modulo operations.

Here we consider signed real numbers expressed in Qm.n format. Note that the ARM processors considered are 32-bit processors. To avoid using long multiplications and long divisions, we can choose the values of m and n to satisfy m + n + 1 = 16. For the programming, we can see n as a variable which assumes values from 0 to 15, but for our calculations, we make it 8.

Arithmetic operations of Qm.n numbers

Let s_1, s_2, and s_3 be the integer numbers representing three Qm.n real numbers r_1, r_2, and r_3. Then, we have $s_1 = r_1 * 2^n$, $s_2 = r_2 * 2^n$, and $s_3 = r_3 * 2^n$. Below, we discuss the arithmetic operations of these numbers.

1. Addition

Let $r_3 = r_1 + r_2$ and $s_a = s_1 + s_2$. Then $s_a = (r_1 + r_2) * 2^n = r_3 * 2^n = s_3$ Hence, we just perform normal additions on these Qm.n numbers. This is true for subtractions as well.

2. Multiplication

Let $r_3 = r_1 * r_2$ and $s_p = s_1 * s_2$. Then $s_p = r_1 * r_2 * 2^{2n} = r_3 * 2^{2n} = s_3 * 2^n$. Hence, we need to shift the contents of s_p, the product of s_1 and s_2, n bits to the right to obtain s_3. As such, you need to use `MUL` and `ASR` together to perform the multiplication of two Qm.n numbers.

3. Division

Let $r_3 = r_1/r_2$ and $s_d = s_1/s_2$. Then we see $s_d = r_1/r_2$. Yet, we want to have $s_3 = (r_1/r_2) * 2^n$. As a result, we need to shift s_1, the dividend, to the left by n bits before the division to keep the accuracy. Note that we need to use the `SDIV` instruction for the division as the Q in Qm.n stands for signed quotient.

The modulo operation of integers can be implemented by using `SDIV` and `MLS` instructions. See p.7 of the post-class version of class notes of Lecture 18. Note that you cannot do the pre-shift of the dividend for this case. (You need to convince yourself about this. You can ask the instructor to explain this topic three days after you start the project.)

Tasks for the workshop

In the provided C code, we have two utility functions

- `int32_t float2Qmpn(float realnum, uint32_t n);`
- `float Qmpn2float(int32_t intnum, uint32_t n);`

which are used for converting a real number to its Qm.n representation and from a Qm.n representation to its represented real value. You are supposed to understand them very well.

With the above utility functions, the given real numbers in two arrays `real_array1` and `real_array2` are converted to Qm.n representations at the beginning of the C code. These two Qm.n representation arrays will be used for the following four assembly tasks.

Task 1. Implementing the addition of two arrays of Qm.n numbers and save the results into a third array

In this task, you are provided with the C implementation of the addition of two arrays of real numbers with the results saved to the third in a **for loop**. The size of the three arrays is given as variable `M`. You need to implement aTask1, the corresponding assembly implementation in a **for loop**. You can use the following code in the for loop of aTask1 to load the two operands in each pass of the loop:

```
LDR     r5, [r0], #4
LDR     r6, [r1], #4
```

Note that as we discussed before, you can just use the normal addition instructions to perform the addition of two Qm.n numbers since they have the same shifting of the **point**.

The print out in the C code should be the same for both the C and assembly versions.

Task 2. Implementing the element-wise multiplication of two arrays of Qm.n numbers and save the result into a third array

In this task, you are given the C implementation of the element-wise multiplication of two arrays of real numbers and save the results to the third in a **while loop**. You need to implement aTask2, the corresponding assembly operation in a **while loop**. You can use the same implementation as Task 1 since the implementation of the for and while loops can be the same in assembly.

See the text given above for the discussion of implementation of the multiplication. Again, the print out in the C code should be the same for both the C and assembly versions.

Task 3. Implementing the element-wise division of two arrays of Qm.n numbers and save the result into a third array

In this task, you are given the C implementation of the element-wise division of two arrays of real numbers and save the results to the third in a **do-while loop**. You need to implement aTask3, the corresponding assembly operation in a **do-while loop**. Note that you may have to refer to Task 9 of 170_conditional_execution for an efficient implementation of the do-while loop in assembly.

Again, the print out in the C code should be the same for both the C and assembly versions.

Task 4. Implementing element-wise modulo operation of two arrays of Qm.n numbers and save the result into a third array

In this task, you are given the C implementation of the element-wise modulo operation of two arrays of real numbers and save the results to the third in a **do-while loop**. Note that we cannot use the `%` operator to the real numbers. Hence, we use the `fmod` function in the C implementation. You need to implement aTask4, the corresponding assembly operation in a **do-while loop**.

Again, the print out in the C code should be the same for both the C and assembly versions.

Team working and submission of the work

You are supposed to do the assignment in teams of two members. Below is the point distribution of the assignment.

- 20 points each for correct programming of Tasks 1 to 4.
- 20 points for following the assignment/submission requirements:
    - You need to submit a renamed version of the `main.c` file of the project. Use this convention: `cec320_sec_x_ws4_lastname1_firstname1__lastname2_firstname2.c`, where `x` is your section number.
    - You need to submit a renamed version of the `asm_functions.s` file of the project. Use this convention: `cec320_sec_x_ws4_asm_lastname1_firstname1__lastname2_firstname2.s`.
    - You need also submit a pdf file which contains the screenshots of your assembly functions for the four tasks you are programming and the screenshots of the running results. Use the same naming convention as above C file but with a suffix of `pdf` for this file.