Assignment 6
Floating Point Numbers

Deadline: 1 April 2022 at 23:59 on Submitty

Working individually, complete the assignment below. Submit your solution to Submitty (`https://submit.scss.tcd.ie`). By submitting your solution, you are confirming that you have familiarised yourself with College's policy on plagiarism (`https://libguides.tcd.ie/plagiarism`).

Your mark will be the auto-graded mark assigned by Submitty (10 marks).

You are allowed to submit five attempts for the assignment without penalty. Subsequent attempts will attract a 1 mark penalty each, up to a maximum penalty of 5 marks.

Submitty will allow you eight "late days" over the full semester. This means, for example, you can submit one assignment late by eight days or eight assignments late by one day (or part thereof) each, without penalty. Once your "late days" are used up, you will receive zero marks for any late submissions.

Instructions

Design and write three ARM Assembly Language subroutines that implement the interfaces described below. Develop your solution using the `asmt-6-fp` template program in the CSU11022 repository. Submit your solution to Submitty (`https://submit.scss.tcd.ie`).

You should view Lecture 4.3 before attempting this assignment. You should also review Bit Manipulation from CSU11021 in Semester 1.

fp_exp subroutine

```
1 @ fp_exp subroutine
2 @ Obtain the exponent of an IEEE-754 (single precision) number as a signed
3 @   integer (2's complement)
4 @
5 @ Parameters:
6 @   R0: IEEE-754 number
7 @
8 @ Return:
9 @   R0: exponent (signed integer using 2's complement)
```

This subroutine should decode the exponent of an IEEE-754 floating point number. The exponent should be returned as a signed word-size value in 2's complement form.

In other words, after isolating the eight-bit exponent from the IEEE-754 value (bits 23 to 30 inclusive), you should subtract the constant bias of 127 from the encoded exponent to return a value in the range $-127 \ldots 0 \ldots +128$.

fp_frac subroutine

```
@ fp_frac subroutine
@ Obtain the fraction of an IEEE-754 (single precision) number.
@
@ The returned fraction will include the 'hidden' bit to the left
@   of the radix point (at bit 23). The radix point should be considered to be
@   between bits 22 and 23.
@
@ The returned fraction will be in 2's complement form, reflecting the sign
@   (sign bit) of the original IEEE-754 number.
@
@ Parameters:
@   R0: IEEE-754 number
@
@ Return:
@   R0: fraction (signed fraction, including the 'hidden' bit, in 2's
@         complement form)
```

This subroutine should decode the fraction component of an IEEE-754 value. Remember that IEEE-754 does not store the hidden bit to the left of the radix point, so you will need to re-introduce this hidden 1 at bit number 23 (the 24th bit counting from the least-significant (right) end of the value.

Remember, IEEE-754 stores the 23 bits to the right of the radix point. Your subroutine should re-introduce a single 1 (the hidden bit) to the left of the radix point. So, your subroutine should return a value representing the fraction in the form:

$$000000001.fffffffffffffffffffffff$$

Where *fff ... fff* are the fraction bits that were encoded in the original IEEE-754 number. The radix point (or binary point) is imagined to be between bits 22 and 23 (where bit 0 is the rightmost bit).

Finally, if the original IEEE-754 number was negative (S=1), your subroutine should negate (`NEG` instruction) the fraction before returning it, treating it as a 2's complement number.

fp_enc subroutine

```
@ fp_enc subroutine
@ Encode an IEEE-754 (single precision) floating point number given the
@   fraction (in 2's complement form) and the exponent (also in 2's
@   complement form).
@
@ Fractions that are not normalised will be normalised by the subroutine,
@   with a corresponding adjustment made to the exponent.
@
@ Parameters:
@   R0: fraction (in 2's complement form)
@   R1: exponent (in 2's complement form)
@
@ Return:
@   R0: IEEE-754 single precision floating point number
```

This subroutine should take two parameters – a fraction and an exponent – in the form that they were returned by the fp_exp and fp_frac subroutines – and encode the fraction and exponent as an IEEE-754 value.

Note that, if the fraction passed to the fp_enc subroutine is not normalised, your subroutine should normalise it first, making a corresponding adjustment to the exponent, before encoding it as an IEEE-754 number.

Hints

Test your subroutines by first decoding an IEEE-754 value (using fp_exp and fp_frac) and re-encoding the same value (using fp_enc). Verify that you get the original test value.

Use the online IEEE-754 calculator at `https://www.h-schmidt.net/FloatConverter/IEEE754.html` to create test values. Start with simple values (e.g. 1.5, 10.5 and -1.5)