Assignment 5

Due: Wed Nov 18th, 2020 @11:59PM

Please read over the entire assignment before starting to get a sense of what you will need to get done in the next week.

You MUST run the assignment on the pi cluster. You HAVE to ssh: You will not be able to compile or run the assignment otherwise.

Table of Contents

Learning Goals

- Programming in ARM assembly
- Working with floating point numbers
- Coordinating a program with both ARM assembly and C code (you aren't writing in C)

FP

Your goal for this assignment is to read the file of binary 8-bit FP numbers and convert them to the IEEE Single Precision (32 bit) format.

FP Format

The FP format is similar to the one we studied in class. It has a sign bit, 3 bits of biased exponent, and 4 bits of mantissa.

The following figure illustrates the bit assignments:

FP Format		
sign (1)	exponent (3)	mantissa (4)

There is an implied "1." in front of the mantissa. FP does not support denormal numbers (numbers less than 1.0 x 2^{-3}).

Number	Encoding in 8-bits
+0.0	8'b0000_0000 (8 bits of 0 in binary)
-0.0	8'b1000_0000

Number	8-Bit Representation	Binary Representation	Base 10 Representation
Largest Positive #	8'b0111_1111	2^4 * 5'b1.1111	5'b11111 = 31
Smallest Positive #	8'b0000_0001	2^-3 * 5'b1.0001	⅛ + 1/128 = 0.1328125
Largest Negative #	8'b1111_1111	-2^4 * 5'b1.1111	-5'b11111 = -31
Smallest Negative #	8'b1000_0001	-2^-3 * 5'b1.0001	-(⅛ + 1/128) = -0.1328125

IEEE-754 Single Precision Format		
sign (1)	exponent (8)	mantissa (23)

The bias for the IEEE Format is 127 (base 10) and the format uses an implied 1.
The smallest representable exponent is -126 which is represented as 8'b0000_0001. The exponent code 8'b0000_0000 is not used by our program.[1] However, +0 is 32 bits of 0 and -0 is 1 followed by 31 0's.

[1] the exponent code 8'b0000_0000 represents denormal numbers where there is no implied 1. In this assignment you will not need to use denormal representations.

Summary of select Conversions (0's, min, max values)

FP	8-bit FP	IEEE-754 Single
+0	0000_0000	0x00000000
-0	1000_0000	0x80000000
0.1328125	0000_0001	0x3e080000
-0.1328125	1000_0001	0xbe080000
31	0111_1111	0x41f80000
-31	1111_1111	0xc1f80000

Your Mission

Your program will take a binary file name as an argument. Read in the file that is the argument passed to your program. This file is a raw binary file containing just bytes of coordinate data. For each byte in the file, decode that byte and convert it into IEEE FP format and print it.

For example,

```
$ ./convertfp topsecret.bin
```

would read in the file `topsecret.bin`, convert each byte to a floating point number, and print those numbers as decimal values to stdout. You can assume that a valid binary file is used and there won't be other arguments given.

Arguments

Arguments	Description
`convertfp`	This is the executable
`<filename>.bin`	This .bin file contains binary information in the form of bytes of FP data. You will read this file one byte at a time, convert each number to IEEE FP format, and print it to the screen, one number per line.

Files and Methods

You are provided 4 files:

1. `Makefile` (DO NOT EDIT)
2. `main.c` (DO NOT EDIT)
3. `fp2float.s`
4. `makebin.c` (DO NOT EDIT: Described below under Testing)

1. `main.c`

This file is provided for you and already complete. DO NOT EDIT!

`int main(int argc, char **argv)`
Description
This file reads in the contents of the binary file, calls the below `FP2float` routine, and prints the decoded bytes to standard out. For each decoded byte from the binary input file, it will print the following three values on one line: the FP value, the IEEE FP value, and the decimal value.

2. `fp2float.s`

This file contains two routines in ARM assembly that you will complete.

`float FP2float(unsigned int Num);`
Description
Write a routine called `FP2float` implemented in CSE 30 ARMv6 assembler (your routine is only allowed to use the cse30 subset of ARMv6). `FP2float` is called in the main method of `main.c`. The routine `FP2float` should convert all the non-zero input cases - converting `Num` to an IEEE single precision float. If `Num` is a +0 or a -0 as described above, it should call the routine `zeroFP2float` (which is in the **same** file, described below).
Parameters

`unsigned int Num`	32-bit integer representation of an 8-bit FP number read from the input file (The value of the parameter is stored in R0 and you should store the return value in R0).

`zeroFP2float`	
Description	
This routine should be called from within the assembly routine `FP2float` if the `Num` passed in to it is a +0 or a -0 (The value of the parameter is stored in R0 and you should store the return value in R0).	
Parameters	
`unsigned int Num`	32-bit integer representation of an 8-bit FP number read from the input file (This parameter should still be in register R0. You should store the return value in R0)

Getting Started

Instructions to Run Your Program

In order to run your program you MUST ssh into the pi-cluster on ieng6. Both your ieng6 and pi-cluster accounts share the same files so anything you copy onto the lab machines (scp to ieng6) will ALSO be on the pi-cluster when you ssh in.

From the lab machines

1. Open a Terminal
2. Download the starter code
3. ssh into the pi-cluster using the following command:
   `$ ssh <your cse30 username>@pi-cluster.ucsd.edu`
   ex: `$ ssh cs30fa20xx@pi-cluster.ucsd.edu`
4. You are ready to start!

From Your Personal Computer

1. Open a Terminal
2. ssh into your ieng6 account
3. Download the starter code
4. ssh into the pi-cluster using the following command:
   `$ ssh <your cse30 username>@pi-cluster.ucsd.edu`

ex: `$ ssh cs30fa20xx@pi-cluster.ucsd.edu`

5. You are ready to start! (REMEMBER: when you exit, you need to do so twice if you want to get back to your local machine!)

Running Your Program

We've provided you with a `Makefile` and starter code.

Makefile: The `Makefile` provided will create a `convertfp` executable from the source files provided. Compile it by typing `make` into the terminal. Run `make clean` to remove files generated by `make`.

gdb: To run with gdb, the command is `gdb [executable]` to start gdb with your executable. `r [arguments]` will run your executable with `[arguments]`

Allowed Instructions

You are only allowed to use the instructions provided in the ARM ISA Green Card. Failure to comply will result in a score of 0 on the assignment.

Testing

1. The file `makebin.c` has been provided so that you can easily create examples to test your code on.

Example contents of an input file (`somehexnums`) given as an argument to `makebin`:

```
0x0
0x80
0x01
0x81
0x7f
0xff
```

Example on how to use `makebin`:

```
$ ./makebin somehexnums somehexnums.bin
```

Note: if you try to edit `somehexnums.bin` you will see a bunch of gibberish, as it's a binary file.

Style Requirements

Points WILL be given for style, and teaching staff won't be able to provide assistance or regrades unless code is readable. Please follow these Style Guide:lines for ARM assembly.

Submission and Grading

Submitting

1. Submit your files to Gradescope under the assignment titled “Assignment 5”. You will submit the following files :

    - `fp2float.s`
    - `README`

    To upload multiple files to gradescope, zip all of the files and upload the zip to the assignment, or you can multi-select all files and drag and drop. Ensure that the files you submit are not in a nested folder.

2. After submitting, the autograder will run a few tests:
    a. Checks that all required files were submitted
    b. Checks that your source code compiles
    c. Runs test input on your program and compares your output to ours

Grading Breakdown

Make sure to check the autograder output after submitting! We will be running additional tests after the deadline passes to determine your final grade.

You will receive 1 point for each test that you pass on Gradescope, and points for style. Make sure your assignment compiles correctly through the provided `Makefile` on the pi-cluster. **Any assignment that does not compile will receive 0 credit.** This assignment will also have 3 points dedicated to style (see above ARM assembly style requirements). We will be comparing the standard out of your program to ours, so please ensure nothing else is printed than what is printed by the provided main.