Department of Electrical & Computer Engineering
University of California, Davis
EEC 170 – Computer Architecture
Spring Quarter 2025

Laboratory Exercise 4: Multiplication in Half-precision

Due Date: [May 28 11:59PM]
Full Points 100

Objectives of Lab 2

1. Learn RISC-V instruction set and assembly language programming.
2. Learn bit manipulation instructions.
3. Learn how floating point representation and arithmetic works.

Specification

Write a RISC-V assembly language function called **hfmult** that takes two 16 bit numbers X and Y in half-precision floating point format (see below) and returns 16 bit product of X and Y also in half-precision representation. Use the algorithm shown in Figure 3.5.4 (COD Figure 3.16) in your textbook. The example in Participation Activity 3.5.12 will be your first test case (the activity itself may also help with understanding).

What is half-precision floating point format?

There are a few commonly encountered 16-bit floating point formats. Of interest to us are the following two:

1. IEEE 754 Half Precision: a variant of the standard 32-bit IEEE 754 floating point format that uses 10 bits of mantissa and 5 bits of exponent. A description of the format is available here: https://en.wikipedia.org/wiki/Half-precision_floating-point_format
2. Google's "brain floating point" (also called bfloat16) which uses 7 bits of mantissa and 8 bits of exponent. A description of the format, along with a comparison with IEEE 754 half-precision, is available here: en.wikipedia.org/wiki/Bfloat16_floating-point_format

Why the difference? Google's bfloat16 is intended for use in training neural networks. A peculiarity of neural networks is that the algorithms used to fit them to data require relatively low precision, but high dynamic range. That is, the algorithm needs to be able to process very large, or very small, numbers, but the exact value of the number doesn't matter all that much. The extra bits of exponent serve this purpose. There is a large and active body of work on reduced-precision training algorithms for neural networks. There's a nice blog post from Google about this topic here: https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus

The older IEEE floating point standards were developed with scientific computation in mind (e.g. physics simulations, engineering calculations). In these applications, precision typically *is* important and so the format allocates relatively more bits towards the mantissa/significand.

The bits are laid out as follows:

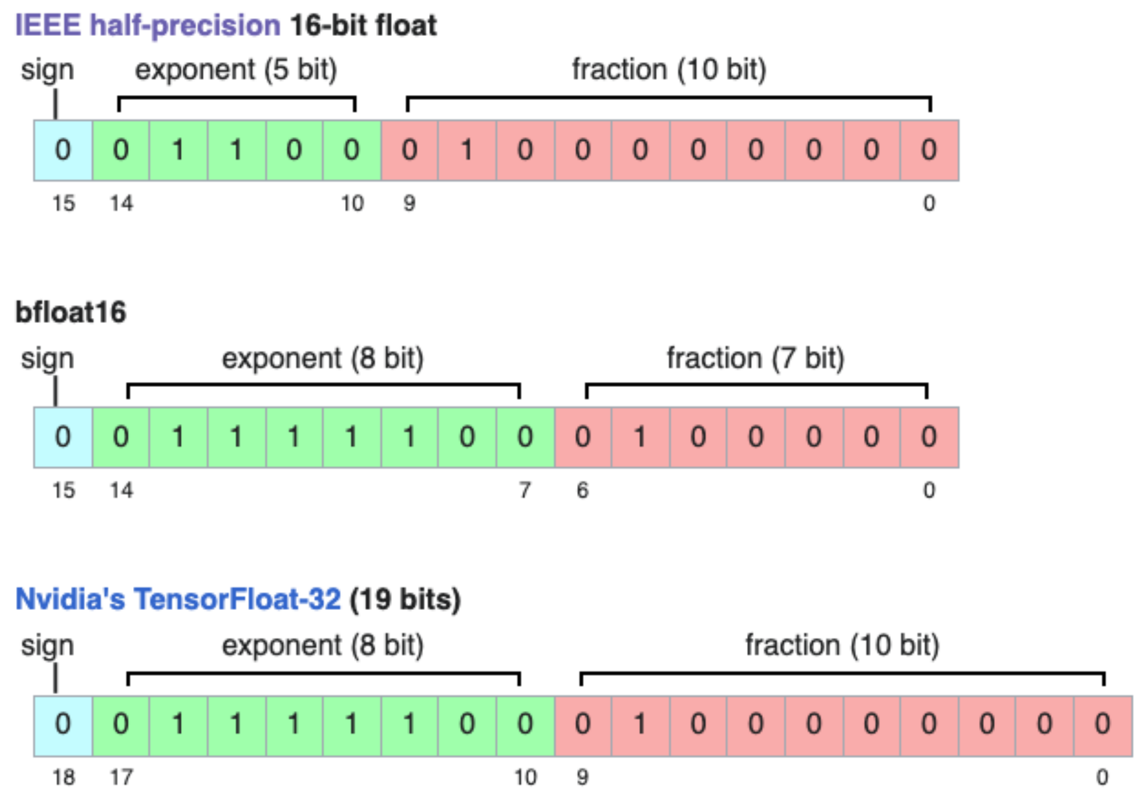

Figure 1 Source: https://en.wikipedia.org/wiki/Bfloat16_floating-point_format

What do you need to do?

a) Write a function **hfmult** that takes two 16 bit numbers X and Y in **in the half-precision floating point format of your choice** and returns 16 bit product of X and Y in the same representation.
b) X and Y are passed to the function registers a1 and a2 and the result is returned in register a3.
c) Code to call the function and print the inputs and results are provided. Don't modify that. Just add your code to the function labeled **hfmult**
d) Here is the expected answer after you run the program with the given values in the skeleton file, for X = 2.5 and Y = 5.25, the result should be 13.125.
e) Your program should work for positive and negative numbers including 0

```
X =                     0x00004020
Y =                     0x000040A8
Result =                0x00004152
Exited with error code 0
Stop program execution!
-
```

What to submit?

Submit your code with results for different combinations of X and Y given below. Use this website (https://float.exposed/) to create half float representations and convert from hexadecimal to decimal and vice-versa.

Please state at the top of your code which half-precision format you are targeting.

X	Y	Your Result (in hex)	Your Result in (decimal)	Exact	Error (percentage)
2.5	5.25			13.125	0
1.0	-5.0			-5.0	
-5.0	3.3			-16.5	
-1.5	-25.75			37.65	
0.01	0.02			0.00002	