Do the following exercises of the book David A. Patterson and John L. Hennessy, "Computer organization and design ARM edition: the hardware software interface:"

3.20, 3.22, 3.23, 3.24, 3.27, 3.41, 3.42, 3.43, 3.47

3.20 [5] <§3.5> What decimal number does the bit pattern 0 × 0C000000 represent if it is a two's complement integer? An unsigned integer?

3.22 [10] <§3.5> What decimal number does the bit pattern 0 × 0C000000 represent if it is a floating point number? Use the IEEE 754 standard.

3.23 [10] <§3.5> Write down the binary representation of the decimal number 63.25 assuming the IEEE 754 single precision format.

3.24 [10] <§3.5> Write down the binary representation of the decimal number 63.25 assuming the IEEE 754 double precision format.

3.27 [20] <§3.5> IEEE 754-2008 contains a half precision that is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the mantissa is 10 bits long. A hidden 1 is assumed. Write down the bit pattern to represent -1.5625×10^{-1} assuming a version of this format, which uses an excess-16 format to store the exponent. Comment on how the range and accuracy of this 16-bit floating point format compares to the single precision IEEE 754 standard.

3.41 [10] <§3.5> Using the IEEE 754 floating point format, write down the bit pattern that would represent −1/4. Can you represent −1/4 exactly?

3.42 [10] <§3.5> What do you get if you add −1/4 to itself four times? What is −1/4 × 4? Are they the same? What should they be?

3.43 [10] <§3.5> Write down the bit pattern in the fraction of value 1/3 assuming a floating point format that uses binary numbers in the fraction. Assume there are 24 bits, and you do not need to normalize. Is this representation exact?

3.47 [45] <§§3.6, 3.7> The following C code implements a four-tap FIR filter on input array sig_in. Assume that all arrays are 16-bit fixed-point values.

```
for (i = 3;i< 128;i+ +)
sig_out[i] = sig_in[i - 3] * f[0] + sig_in[i - 2] * f[1]
  + sig_in[i - 1] * f[2] + sig_in[i] * f[3];
```

Assume you are to write an optimized implementation of this code in assembly language on a processor that has SIMD instructions and 128-bit registers. Without knowing the details of the instruction set, briefly describe how you would implement this code, maximizing the use of sub-word operations and minimizing the amount of data that is transferred between registers and memory. State all your assumptions about the instructions you use.

Implementare il codice C dell'ultimo esercizio usando le istruzioni del LEG V8.

Assumere che X0 abbia l'indirizzo di sig_in, X1 l'indirizzo di sig_out, X2 l'indirizzo di f.