Computer Architecture – Assignment 10

Claudio Maggioni Tommaso Rodolfo Masera

1 Exercise 1

1.1 Exercise 1.1

Cache line size = 32 bytes

Cache memory size = 16384 bytes

Number of cache lines in cache memory = (16384/32) = 512

1.2 Exercise 1.2

1.2.1 Byte:

Bytes per word = $32bits/8bits = 4$

Number of *Byte* bits = $log_2(4) = 2bits$

1.2.2 Word:

Words per cache line = $32bytes/4bytes = 8$

Number of *Word* bits = $log_2(8) = 3bits$

1.2.3 Line:

Number of *Line* bits = $log_2(512) = 9bits$

1.2.4 Tag:

Number of *Tag* bits = $32bits - 9bits - 3bits - 2bits = 18bits$

1.3 Exercise 1.3

1.3.1 Byte and Word:

The number of *Byte* and *Word* bits do not change.

1.3.2 Line:

Number of sets = 4

Number of *Set* bits = $log_2(4) = 2bits$

1.3.3 Tag:

Number of *Tag* bits = $32bits - 2bits - 3bits - 2bits = 25bits$

2 Exercise 2

2.1 Exercise 2.1

The floating point standard normalizes every binary number as 1.XXXX, storing only the fraction so that it can save 1 bit of memory by simply assuming, from the normalized format, that the first bit before the comma is always a 1. It also ranges from 1.0_2 (1_{10}) *to* 1.1111...(*almost* 2_{10}).
The reason for having three different formats for precision is, indeed, to have more precise numbers to work with. This is especially useful with big positive numbers as, sometimes, more bits can be used to represent them since one more digit in a big number makes much more difference than in a small number.

2.2 Exercise 2.2

Underflow would usually result in a value of 0 or NaN.
But, with denormalization, instead of a “flush to 0”, you get a gradual loss of precision for values that go to 0. On the other hand, denormalized numbers are not a valid solution to overflow as the gradual loss of precision towards infinity doesn’t apply.

2.3 Exercise 2.3

A) Conversion of 01000001010110000000000000000000 to decimal:

Sign bit: 0

Exponent bits: 10000010

Mantissa bits: 1.10110000000000000000000

The exponent bits map to $+3$ and the mantissa bits evaluate to 2^{-1}, 2^{-3} and 2^{-4}.
We then multiply the mantissa by the exponent and we get $1.10110000000000000000000 * 2^3 = 1101.10000000000000000000$.
We then convert the binary number to decimal:
$1101.10000000000000000000 = 2^3 + 2^2 + 2^0 + 2^{-1} = 8 + 4 + 1 + 0.5 = 13.5$.

B) Conversion of 01000100001001101001000000000000 to decimal:

Sign bit: 0

Exponent bits: 10001000

Mantissa bits: 1.01001101001000000000000

The exponent bits map to $+9$ and the mantissa bits evaluate to 2^{-2}, 2^{-5}, 2^{-6}, 2^{-8} and 2^{-11}.
We then multiply the mantissa by the exponent and we get $1.01001101001000000000000 * 2^9 = 1010011010.01000000000000$.
We then convert the binary number to decimal:
$1010011010.01000000000000 = 2^9 + 2^7 + 2^4 + 2^3 + 2^1 + 2^{-2} = 512 + 128 + 16 + 8 + 2 + 0.25 = 666.25$

2.4 Exercise 2.4

In order to compute the number of IEEE 754 single-precision floating point numbers between 0 and 1 (both included), we first consider a constant positive (= 1) sign bit (and therefore we do not count it in our calculation of possible permutations).

Then we count the number of denormalized numbers (including 0), which is: $2^{23} = 8388608$.

After that, we count the number of valid from 2^{126} to 2^1 included, which is 126. All these exponents will generate a number < 1 even with the highest possible mantissa. Then, we compute the number of numbers with these exponents, equal to: $126 * 2^{23} = 1056964608$.

Finally, we consider the number 1 itself (0x3f800000) and we sum all the combinations: $8388608 + 1056964608 + 1 = 1065353217$

Therefore, there are 1065353217 IEEE 754 single-precision floating point numbers between 0 and 1 (both included).