HW03 - Floating Point

Q1 Floating Point Numbers

10 Points

REMINDER: Gradescope does not have submission history for in-browser assignments such as lecture questions and homeworks. If you press "save answer" or "submit" after the deadline, the autograder *will* mark it as late and you'll get 0 points in the next TCP rerun. When you review old homework and lecture assignments, do not click any buttons otherwise you risk getting a zero on that assignment.

For the following questions, we will be referring to the IEEE 32-bit floating point representation except with a 6 bit exponent (bias of -($\frac{2^6}{2} - 1) = -31$) and a denorm implicit exponent of -30. The extra bits will go toward the significand, meaning it is now 25 bits long.

Q1.1

1 Point

Convert the floating point number `0x80000000`, which follows the 6 bit exponent representation as described above, to decimal.

Please specify infinities as +inf or -inf, and not a number as NaN.

-0

EXPLANATION

`0x80000000`

`1 000000 0000000000…0`

Sign: negative

Exponent: -30 (denormalized)

Value = $-2^{-30} * (0.000000000) = -0$

✔ Correct

Save Answer Last saved on **Sep 21 at 3:02 PM**

Q1.2

1 Point

Convert the floating point number `0x53530000`, which follows the 6 bit exponent representation as described above, to decimal.

Please specify infinities as +inf or -inf, and not a number as NaN.

1702

EXPLANATION

`0x53530000`

`0 101001 1010100110…0`

Sign: positive

Exponent: $41 - 31 = 10$

Value = $2^{10} * (1.101010011) = 1702$

✔ Correct

Save Answer Last saved on **Sep 21 at 3:02 PM**

Q1.3

1 Point

Convert the floating point number `0xFE530000`, which follows the 6 bit exponent representation as described above, to decimal.

Please specify infinities as +inf or -inf, and not a number as NaN.

NaN

EXPLANATION

`0xFE530000`

`1 111111 0010100110…0`

We have all bits in the exponent and some bits in the mantissa which means that we have defined NaN.

✔ **Correct**

Save Answer Last saved on **Sep 21 at 3:02 PM**

Q1.4

1 Point

Convert the floating point number `0xFE000000`, which follows the 6 bit exponent representation as described above, to decimal.

Please specify infinities as +inf or -inf, and not a number as NaN.

-inf

EXPLANATION

`0xFE000000`

`1 111111 0000000000…0`

Sign: negative

Exponent: $63 - 31 = 32$ (Note the exponent is normalized)

We now have all the bits in the exponent and none in the mantissa which defines this number as infinity. We must then multiply it by the sign bit which is why it is `-inf`.

✔ Correct

Save Answer Last saved on **Sep 21 at 3:02 PM**

Q1.5

1 Point

What's the smallest non-infinite positive integer (an integer has nothing to the right of the decimal) this representation CANNOT represent?

Hint: Think about how your floating point value changes when you change the mantissa by 1.

67108865

EXPLANATION

We are limited by our mantissa, which has 25 bits. This means we can represent integers up to 2^{26}, which would be $2^{26} *$ `1.0000 0000 0000 0000 0000 0000 0`. $2^{26} + 1$ would be $2^{26} +$ `1.0000 0000 0000 0000 0000 0000 01`, requiring us to put a 1 in the non-existent 26th bit of the mantissa. So the smallest positive integer we CANNOT represent is $2^{26} + 1 = 67108865$. (The next largest number we can represent after 2^{26} is 2^26 + 2, which would be $2^{26} *$ `1.0000 0000 0000 0000 0000 0000 1`).

✔ Correct

Save Answer Last saved on **Sep 21 at 3:02 PM**

Q1.6

1 Point

What's the smallest positive value it can represent that is not a denormal number? We want to find x in 2^x.

$x =$

-30

EXPLANATION

Exponent field: `000001`

The number cannot be a denorm, so the exponent must be nonzero.

So exponent is $1 - 31 = -30$

2^{-30}

✔ Correct

Save Answer Last saved on **Sep 21 at 3:03 PM**

Q1.7

2 Points

What's the smallest positive value this representation can express? Leave your answer as a power of 2. We want to find x in 2^x.

$x =$

-55

EXPLANATION

The floating point values closest to 0 are denorms. Denorms in this representation have an implicit exponent of -30. We set the mantissa as small as possible to $00...01$, meaning we multiply $2^{-30} \cdot 0.00...01_2$ to give us the smallest positive value.

$= 2^{-30} \cdot 2^{-25}$

$= 2^{-55}$

✔ Correct

Save Answer Last saved on **Sep 21 at 3:03 PM**

Q1.8

1 Point

How many NaNs can be represented with this floating point system?

67108862

EXPLANATION

There's $2^{25} - 1$ possible positive NaNs (we subtract one for infinity), and there's $2^{25} - 1$ possible negative NaNs (we subtract one for infinity again). Combined, that gives us $2^{26} - 2$ possible NaN values, which equals 67108862.

✔ Correct

Save Answer Last saved on **Sep 21 at 3:03 PM**

Q1.9

1 Point

What's the most negative possible denormalized number representable in this system? Write it in hex.

0X81FFFFFF

EXPLANATION

The most negative denormalized number must have a sign bit of 1, an exponent value equaling 0, and all 1s to fill out all of the significand bits. This thus produces 0b 1 000000 111...1, which can be written in hex as 0x81FFFFFF.

✔ Correct

Save Answer Last saved on **Sep 21 at 3:05 PM**

Q2 Miscellania

3 Points

These questions explore some quirkier aspects of floating-point numbers.

Q2.1 Non-associativity

Addition and subtraction on unsigned and two's complement numbers is both commutative and associative. (You can arbitrarily re-order a series of additions and subtractions and get the same result.) In contrast, floating point addition is *not* associative. Given the floating point encoding from Question 1, which of the following sequence of floating point additions and subtractions returns the correct result.

- ☐ $2^{-10} + 2^{22} - 2^{22} + 2^{-1}$
- ✔ $2^{-10} + (2^{22} - 2^{22}) + 2^{-1}$
- ☐ $(2^{-10} + 2^{22}) + (-2^{22} + 2^{-1})$
- ✔ $2^{-10} + (2^{22} + (-2^{22} + 2^{-1}))$

EXPLANATION

Unlike in the two's complement and unsigned representations we studied, adding a small magnitude floating-point number (e.g., 2^{-10}) to a significantly larger number (e.g., 2^{22}) will yield the larger magnitude number unchanged (e.g., $2^{-10} + 2^{22} = 2^{22}$). Since we have 25 bits of mantissa, our encoding correctly computes $-2^{22} + 2^{-1}$ and so we need not worry about doing the right-most addition first. (this would not be the case for representations with fewer than 23 bits of significand). This is because in order to make sure 2^{-1} gets properly represented with an exponent value of 22, the signficand would have have all 0s and a 1 only in the 23rd bit since $22 - (-1) = 23$. With fewer than 23 bits used, that -1 gets lost.

Q2.2 Can floats represent ints?

An IEEE-754 single-precision floating point number is 32 bits, consisting of 1 sign bit, 8 exponent bits, and 23 mantissa bits. Can this floating-point encoding represent the space of all integers a 32-bit, two's complement number can represent?

- ○ true
- ◉ false .

EXPLANATION

It cannot. Here's one intuitive proof: both this floating-point and two's complement encoding can represent at most 2^{32} unique values, since they are 32 bits long. We know that the two's complement number represents exactly 2^{32} distinct integers. Thus, if there exists any floating point number that is *not* one of these integers, it cannot possibly represent all of the same values. There are many such values: oddities like NaNs, -0; integers that are larger than $2^{31} - 1$; integers smaller than -2^{32}; and of course, non-integer, but rational numbers.

✔ **Correct**

Save Answer Last saved on **Sep 21 at 3:04 PM**

Q3 Incorrect Convergence

5 Points

For this problem, please code up a naive solution in python3. We ran the code snippet we created on this online python3 emulator though it should work locally as well.

Hint: You **will** need to code up a solution to figure out the answers to this problem.

Q3.1

Consider the following sequence:

$$v_1 = 5$$

$$v_2 = -135$$

$$v_n = 111 - \frac{1130}{v_{n-1}} + \frac{3000}{(v_{n-1} * v_{n-2})}$$

As n grows larger, this series should converge to 6. Due to small amounts of error in floating point, what **integer** does this approach? Hint: You should try coding this out in python3!

100

Q3.2

For what value of n does this converge to the number you found above?

16

EXPLANATION

Here is the solution of the code we used to figure this out:

```
v1 = 5

v2 = -135

def seq(v_n_minus_1, v_n_minus_2):
```

```
    return 111 - 1130 / v_n_minus_1 + 3000 / (v_n_minus_1 * v_n_mir
iters = 100
v_n = v2
v_n_minus_1 = v1
print(f"n = 2; v_n: {v_n}; v_{{n-1}}: {v_n_minus_1}")
for i in range(3, iters + 1):
    tmp = v_n
    v_n = seq(v_n, v_n_minus_1)
    v_n_minus_1 = tmp
    print(f"n = {i}; v_n: {v_n}; v_{{n-1}}: {v_n_minus_1}")
```

Here are the first 20 n values of output from the code snippet above.

```
n = 2; v_n: -135.0; v_{n-1}: 5
n = 3; v_n: 114.92592592592592; v_{n-1}: -135.0
n = 4; v_n: 100.97421849822753; v_{n-1}: 114.92592592592592
n = 5; v_n: 100.06754371686726; v_{n-1}: 100.97421849822753
n = 6; v_n: 100.00453229042036; v_{n-1}: 100.06754371686726
n = 7; v_n: 100.00029604415029; v_{n-1}: 100.00453229042036
n = 8; v_n: 100.00001896854535; v_{n-1}: 100.00029604415029
n = 9; v_n: 100.00000119840993; v_{n-1}: 100.00001896854535
n = 10; v_n: 100.00000007491947; v_{n-1}: 100.00000119840993
n = 11; v_n: 100.00000000464591; v_{n-1}: 100.00000007491947
n = 12; v_n: 100.00000000028629; v_{n-1}: 100.00000000464591
n = 13; v_n: 100.00000000001755; v_{n-1}: 100.00000000028629
n = 14; v_n: 100.00000000000107; v_{n-1}: 100.00000000001755
n = 15; v_n: 100.00000000000006; v_{n-1}: 100.00000000000107
n = 16; v_n: 100.0; v_{n-1}: 100.00000000000006
```

```
n = 17; v_n: 100.0; v_{n-1}: 100.0

n = 18; v_n: 100.0; v_{n-1}: 100.0

n = 19; v_n: 100.0; v_{n-1}: 100.0
```

✔ Correct

Save Answer Last saved on **Sep 21 at 3:04 PM**

Save All Answers Submit & View Submission ›