IDS laboratory 1

Name: Yusupov Yuldashbek ID: 317304

i3 = 304 //last 3 digits of ID

im = 304 % 64 = (64\*4 + 48) % 64 = 48 //since 304 = 64\*4 + 48, the remainder is 48

x = im + 50 = 48 + 50 = 98

y = 0.03125 \* x = 0.03125 \* 98 = 3.0625

z = -0.1177 \* x = -0.01177 \* 98 = -11.5346

Task 1

Represent x in binary, octal and hexadecimal

1. Binary

98 | 0 ↑ so x in binary is 01100010b,

49 | 1 ↑ We divide number by 2 (until we get 0) and take remainder. At the end

24 | 0 ↑ read from bottom to top

12 | 0 ↑

6 | 0 ↑

3 | 1 ↑

1 | 1 ↑  
0 | 0 ↑

1. Octal

98 | 2 so x in octal is 142o,

12 | 4 We divide number by 8 (until we get 0) and take remainder. At the end

1 | 1 read from bottom to top

0 |

1. Hexadecimal

98 | 2 so x in hexadecimal is 62x,

6 | 6 We divide number by 16 (until we get 0) and take remainder. At the end

0 | read from bottom to top

Task 2

Convert –x to signed integer in 8 bits two’s complement representation

Since we have our x in binary representation, we simply take it, toggle all bits and add 1b

Our x = 01100010b = 98

When we toggle it we receive ~x = 10011101b = -99

After we add 1b to ~x: 10011101b + 1b = 10011110b

So answer is: 10011110b = -98

Task 3

Add, using column method, db = ~xb + xb (db is 8 bits)

111111

10011110b +

01100010b

100000000b

Using column method of addition we got our db = 100000000b, but it needs 9 bits to be stored, and since task asks to interpret db using 8 bits we have to skip 1 at the beginning, so db = 00000000b

Interpreting db in decimal:

00000000b = (-0)\*27 + 0\*26 + 0\*25 + 0\*24 + 0\*23 + 0\*22 + 0\*21 + 0\*20 = 0d

We got 0 in decimal

Task 4

5 LSB of ~x are fractional part. Convert it to real

So, basically, our ~x will look like: 100.11110b (fixed point here is used for showing how the number will look like. In computers there is no points used in binary representation and is fixed)

Using binary to decimal two’s complement conversion operations, we got:

100.11110b = (-1)\*22 + 0\*21 + 0\*20 + 1\*2-1 + 1\*2-2 + 1\*2-3 + 1\*2-4 + 0\*2-5 = -4 + 0.5 + 0.25 + 0.125 + 0.0625 = -3.0625 = -49/16 = -49/24

Finally, we got our binary number as real: -3.0625

Task 5

Convert y to 8 bit two’s complement signed fixed point binary number, where F=5

Since our y = 3.0625, we will use decimal two’s complement to binary conversion operations to represent y in binary, where our fractional is equal to 5 LSB and y uses 8 bits.

Our integer part is equal to 3, so let’s start from converting it first:

3|1 So, we got our decimal 3 to be equal to 11b and since integer part in our example

1|1 uses 3 bits to represent itself, we add 0 at the beginning. Finally we got our decimal

0| 3 to be equal to 011b

Let’s now convert our fractional part:

0.0625 \* 2 = 0.125 -> 0 So, we got our fractional part (0.0625) to equal 0001b. However, we are  
0.125 \* 2 = 0.25 -> 0 using 5 bits to represent fractional part, so we have to add 0 at the end.

0.25 \* 2 = 0.5 -> 0 Finally, we got our fractional part to equal 00010b in binary

0.5 \* 2 = 1 -> 1

Finally, our y will look like 011.00010b

Task 6

Convert z to 8 bit two’s complement signed fixed point binary number, where F=3

Our z = -11.5346. For simplification let’s take positive z = 11.5346, apply operations for converting and then make it negative. Let’s proceed same steps as in task 5

First, let’s convert our integer part in to binary:

11 | 1 We now have our integer part as binary representation (1011b). However, since we need all

5 | 1 5 bits for integer part to be fulfilled, we have to add 0 at the beginning. Finally, we got our

2 | 0 integer part to be equal to 01011b

1 | 1

0 |

Second, let’s convert our fractional part in to binary:

0.5346\*2 = 1.0692 -> 1 We now have our fractional part, but it will use a lot of bits to be represented

0.0692\*2 = 0.1384 -> 0 precisely and as we only have 3 bits, we will use first 3 numbers and try to

0.1384\*2 = 0.2768 -> 0 round it for more precise answer (<2-4 = 0.0625)

0.2798\*2 = 0.5536 -> 0 Finally, we got our binary number to be equal 100b

0.5536\*2 = 1.1072 -> 1

…

Now let’s convert our number to negative:

First, we should toggle all bits:

~(01011.100b) = 10100.011b  
Now we should add 1b to least significant bit (in our case it is 0.01b)

10100.011b + 0.001b = 10100.100b

Let’s see if our error is not greater than 0.0625:

Z1 = 10100.100b = (-1)\*24 + 1\*22 + 1\*2-1 = -11.5

|error| = ||-11.5| – 11.5346| = 0.0346 < 0.0625 (acceptable)

Finally, our binary representation of a number –11.5346 will be 10100.100b

Task 7

Convert y to IEEE 754 float

So we have our y = 3.0625. Let’s proceed steps for converting decimal number in to binary (IEEE 754 float 32-bit)

1. Let’s set sign bit:

Since our number is positive our sign bit will equal to 0b

1. Let’s rewrite integer part and fractional part in binary:

3 | 1 ↑ 0.0625\*2 = 0.125 -> 0 ↓

1 | 1 ↑ 0.125\*2 = 0.25 -> 0 ↓

0 | 0.25\*2 = 0.5 -> 0 ↓

0.5\*2 = 1 -> 1 ↓

So we have our decimal part represented in binary. Now we should connect them. Finally we got our number to be equal 11.0001b

1. Now let’s normalize our number:

11.0001b = 1.10001b \* 21 (1 is our exponent)

1. Now let’s extract exponent:

e – 127 = 1 -> e = 128

128 | 0

64 | 0 So our e in binary will look like 10000000b

32 | 0

16 | 0

8 | 0

4 | 0

2 | 0

1. | 1
2. |
3. Since e is not equal to 0 and 256, we choose formula:

Val = (-1)s \* 2e-127 \* (1 + m)

1. Now we have to find m without MSB:

1b + m = 1.10001b -> m = 0.10001b

1. Since we have our m to be represented in 23 bits, we have to replace missing bits, that were not covered with zeros starting from LSB:

m = 0.1000100000000000000000b

Since our exponent covers all 8 bits, we don’t have to replace missing bits with zeros

1. Now final step is to rewrite everything in proper way:

0 10000000 1000100000000000000000b

s e m (without leading 0)

Finally our number looks like 0100000001000100000000000000000b

Task 8

Convert result from task 7 back to decimal

In task 7 we got 0100000001000100000000000000000b

1. Let’s define sign bit:

It is clearly seen that sign bit is equal to 0b

1. Now let’s define exponent bits:

Since exponent bit covers 8 bit, we take fist 8 bits after sign bit:

10000000b

1. Now we should define mantissa:

Mantissa covers 23 LSB, so we have

1000100000000000000000b

1. Now let’s proceed algorithms to convert binary floating number back to decimal:

e = 10000000b = 1\*27 = 128 (we don’t have rewrite every bit, because rest are 0s so they will not affect the final result)

m = 1000100000000000000000b = 0.1000100000000000000000b = 1\*2-1 + 1\*2-5 = 0.53125 (we don’t have rewrite every bit, because rest are 0s so they will not affect the final result)

1. Finally let’s use proper formula and convert our binary number to decimal:

val = (-1)0 \* 2128-127 \* (1 + 0.53125) = 2 \* 1.53125 = 3.0625  
It worked! We got our number y