Ira A. Fulton Schools of Engineering
School of Computing, Informatics and Decision Systems Engineering

SER 450 **COMPUTER ARCHITECTURE**

Your Name: **Jacob Hreshchyshyn** Jacob Hreshchyshyn

When completing this worksheet, please use a different color text or typeface so that we can easily identify your work. Remember to show your work / give justification for your answers.

Portions of this material are derived from "Computer Organization and Design", Patterson & Hennessy.

PRACTICE PROBLEMS 3

Problem 1:

Write down the binary representation of the decimal number 63.25 assuming IEEE 754 single-precision format.

We begin by dividing the whole number portion of the number by 2 repeatedly to find the binary representation of 63.

63/2 = 31 R 1

31/2 = 15 R 1

15/2 = 7 R 1

7/2 = 3 R 1

3/2 = 1 R 1

1/2 = 0 R 1

Thus, 63 in binary is 111111_2

We can use a similar technique for finding the binary representation of 0.25 by multiplying repeatedly by 2 until reaching the integer 1.

0.25 * 2 = 0.5

0.5 * 2 = 1

We then construct the binary number as 0.01_2

Thus, the binary number now looks like the following:

111111.01_2

This can be normalized to appear as the following:

$1.1111101_2 * 2^5$

We should now consider our exponent and add the bias 127 to it since we are working with single precision.

5 + 127 = 132

We can now convert the exponent to binary through repeated divisions by 2.

132/2 = 66 R 0

66/2 = 33 R 0

33/2 = 16 R 1

16/2 = 8 R 0

8/2 = 4 R 0

4/2 = 2 R 0

2/2 = 1 R 0

1/2 = 0 R 1

Thus, the binary representation of our exponent is now 10000100_2

We can now reformat our mantissa by removing the leading bit and decimal point so that it looks like the following:

1111101_2

Finally, to represent this in the single precision floating point format with 1 sign bit, 8 exponent bits, and 23 mantissa bits, we get the following result:

$01000010011111010000000000000000_2$

Problem 2:

What number does the bit pattern 0x0C000000 represent interested as a two's compliment integer?

One way of approaching this problem is to convert each hex value to a group of binary numbers and interpret the binary result as a two's complement integer.

0x0 -> 0000

0xC -> 1100

0x0 -> 0000

0x0 -> 0000

0x0 -> 0000

0x0 -> 0000

0x0 -> 0000

0x0 -> 0000

Thus, the resulting binary number is $0000\ 1100\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2$.

We can interpret from this series of binary numbers that the result will be positive since it leads with 0's. Thus, to represent this as a decimal number, we can simply convert from hex to decimal as normal:

$(0 * 16^7) + (12 * 16^6) + (0 * 16^5) + \ldots + (0 * 16^0) = 201{,}326{,}592$

What number does it represent interpreted as an IEEE single-precision floating point?

We can use the previously found binary number to help interpret single-precision floating point value.

The binary value written in the single-precision floating point format would look like the following:

$0\ 00011000\ 00000000000000000000000_2$

From this, we know that the number will remain positive. We can now begin to parse the exponent.

$00011000_2 = (1 * 2^4) + (1 * 2^3) = 24$

This is the adjusted exponent. We can find the unadjusted exponent by subtracting 127 from 24, resulting in -103.

Lastly, our mantissa consists of a collection of zeroes, but we know that it initially led with 1. Thus, the binary representation we have so far is the following:

$1.0_2 * 2^{-103}$

Solving this in base ten results in approximately $9.86076132 * 10^{-32}$

Problem 3:

Using IEEE 754 single-precision floating point format, write down the bit pattern that would represent -1/4. Can you represent -1/4 exactly?

To represent this pattern, it helps to represent -1/4 as -0.25. We can then use similar steps as in problem 1 to find the bit pattern.

Since there is no whole number associated with this number, we can begin converting 0.25 to binary through repeated multiplication of 2.

$0.25 * 2 = 0.5$

$0.5 * 2 = 1$

We then construct the binary number as 0.01_2

Our entire number now looks something like -0.01_2. This can be normalized to appear as follows:

$-1.0_2 * 2^{-2}$

We can find our adjusted exponent by adding 127 to it.

$-2 + 127 = 125$

We can then find the binary representation of our adjusted exponent.

$125/2 = 62$ R 1

$62/2 = 31$ R 0

$31/2 = 15$ R 1

$15/2 = 7$ R 1

$7/2 = 3$ R 1

$3/2 = 1$ R 1

$1/2 = 0$ R 1

Thus, our exponent is 1111101_2.

We no longer need the leading 0 of our mantissa and, because we know that this number is negative, we know the sign bit will be 1. Our resulting bit pattern will be the following:

$10111110100000000000000000000000_2$.

Because -1/4 can be easily represented with just a few digits in decimal, this can be represented exactly in a single-precision floating point value as well.

Problem 4:

Using a 32-bit bitfield, assume two's complement encoding with a binary point between bits 23 and 24.

What is the largest possible number that can be expressed?

To clarify, my understanding of these assumptions produces a mental image of the following example of the described binary number:

011111111.11111111111111111111111

The problem does not specify the base of the number. Therefore, the above is the largest possible number that can be expressed since the leading bit is 0, making the value positive, and because the remaining bits have a value of 1.

What is the numeric precision of this format?

Since there are 23 binary values that follow the binary point between bits 23 and 24, we can say that it is approximately as precise as 2^{-23}, which, according to the slides, is approximately 6 decimal digits of precision.

Problem 5:

Encode the number 1/3 in the fixed-point format described in Problem 5. Show the binary, and decimal representation.

Problem 5 does not describe a fixed-point format. I will assume that this refers to the format specified for problem 4.

We know that 1/3 is positive and that it can be represented in decimal form as 0.3 with the 3 repeating infinitely. This means that everything leading up to the binary point is 0.

We can figure the rest out algorithmically. We know that everything that follows the binary point can be represented as $2^{bit\_position-binary\_point}$. For example, .1 would be equivalent to 2^{-1}, .01 would be equivalent to 2^{-2}, and so on. What we can do is create a summation of binary digits after the binary point such that the result does not exceed 0.3 repeating and do so until we run out of bits after the binary point, giving us an approximate value of 1/3 with the fixed-point format.

This gives us $2^{-2}+2^{-4}+2^{-6}+2^{-8}+2^{-10}+2^{-12}+2^{-14}+2^{-16}+2^{-18}+2^{-20}+2^{-21}$. This alternating pattern, except at the end, keeps us at 0.3 repeating up to 6 decimal places.
The binary value is the following:

0.01010101010101010101011

This gives us an approximate value of 0.3333334923, clearly losing precision after the 6^{th} decimal place.