

ICS Answer Sheet #5

Sakar Gopal Gurubacharya
s.gurubacharya@jacobs-university.de

Problem 5.1: b-complement

Base $b = 5$,
Digits $n = 4$.

a. What are the smallest and the largest number that can be represented and why?

- I. The smallest number that can be represented is $b^{n-1} - 1$ different integers.

$$5^{4-1} = 125$$

$$125 = 25 * 5 + 0$$

$$25 = 5 * 5 + 0$$

$$5 = 1 * 5 + 0$$

$$1 = 0 * 5 + 1$$

$$(1000)_5$$

The absolute smallest would be 0000..., but that is applicable in any digit representation and would account for nothing to exist. We have to set some criteria, so to suffice for 4 digits, the MSB should be at the least, 1.

$(1000)_5$ will be the smallest, below that in base 5 representation would be 444, and that would be the highest for 3-digit representation.

Therefore $(1000)_5$, or $(125)_{10}$ would be the smallest number with base 5 and 4 digits.

II. The largest number that can be represented is $b^n - 1$ different integers.

$$5^4 = (624)_{10}$$

$$624 = 124 * 5 + 4$$

$$124 = 24 * 5 + 4$$

$$24 = 4 * 5 + 4$$

$$4 = 0 * 5 + 4$$

$$(4444)_5$$

624 would be the largest integer and its base 5 representation would be $(4444)_5$

b. What is the representation of -1 and -8 in b-complement notation?

I. For -1 ,

$$(1)_{10} = (0001)_5$$

Here we subtract **each bit from 4** as 4 is the highest possible representation in a base 5 format.

So, we have,

$$(4443)_5$$

For b's complement we add a 1

$$\begin{array}{r} 4443 \\ + 0001 \\ \hline \hline (4444)_5 \end{array}$$

Therefore, $(-1)_{10} \Rightarrow (4444)_5$

II. For -8,

$$(8)_{10} = (0013)_5$$

(because 5 goes into 8, **1** time and remainder is **3**, so **13**)

Now we toggle the bits,

We have,

$$(4431)_5$$

For b's complement we add a 1

$$\begin{array}{r} 4431 \\ + 0001 \\ \hline \hline (4432)_5 \end{array}$$

Therefore, $(-8)_{10} \Rightarrow (4432)_5$

- c. Add the numbers -1 and -8 in b-complement notation. What is the result in b-complement representation? What is the result converted back into the decimal number system?

I. $(-1) + (-8)$

In decimal representation, the answer would be -9

In terms of base 5 format,

We have,

$$\begin{array}{r} 4444 \\ + 4432 \\ \hline 111 \\ \hline 14431 \end{array}$$

We have $(14431)_5$ but since we are limited to 4 digits, we conclude it as $(4431)_5$

- II. Now we convert $(4431)_5$ to a decimal number system. To do that we can follow the steps above in reverse and changing the division to multiplication.

$$\begin{array}{r} 4431 \\ - 0001 \\ \hline (4430)_5 \end{array}$$

Now we toggle bits, here we subtract **4 from each bit** as 4 is the highest possible representation in a base 5 format.

We have,

$$(0014)_5$$

Since the subtraction consisted of a scenario like $0 - 4$, the above representation should have a negative side in front of it.

$$-(0014)_5$$

To convert from a lower order (base) to higher we multiply each bit with the base of its representation.

$$-(0 * 5^3 + 0 * 5^2 + 1 * 5^1 + 4 * 5^0)$$

$$-(5 + 4)$$

$$\mathbf{-9}$$

Therefore, both the decimal addition as well as the base-5 addition resulted in the same value.

Problem 5.2: IEEE 754 floating point numbers

a. $(-273.15)_{10} \rightarrow$ IEEE 754 single precision floating point number

I. First, we convert -273 into binary,

For IEEE 754, we convert -273 in the same fashion as 273.

$$\begin{aligned} 273 &= 136 * 2 + 1 \\ 136 &= 68 * 2 + 0 \\ 68 &= 34 * 2 + 0 \\ 34 &= 17 * 2 + 0 \\ 17 &= 8 * 2 + 1 \\ 8 &= 4 * 2 + 0 \\ 4 &= 2 * 2 + 0 \\ 2 &= 1 * 2 + 0 \\ 1 &= 0 * 2 + 1 \end{aligned}$$

Since it is above 8bit representation, we restrict it down to 8bit by “ignoring” the MSB (1) and hence the MSB will be our sign bit for IEEE 754 representation,

$$(273)_{10} = (100010001)_2$$

II. Secondly, we calculate the fractional part,

$$\begin{aligned} 0.15 * 2 &= 0.30 \Rightarrow 0 \\ 0.30 * 2 &= 0.60 \Rightarrow 0 \\ 0.60 * 2 &= 1.20 \Rightarrow 1 \\ 0.20 * 2 &= 0.40 \Rightarrow 0 \\ 0.40 * 2 &= 0.80 \Rightarrow 0 \\ 0.80 * 2 &= 1.60 \Rightarrow 1 \end{aligned}$$

As we can see, we have already encountered 0.60 in the process above.

That means 1001 is a repeating mantissa and hence absolute 0 can never be precisely represented in binary form/ floating point numbers.

III. Normalization step,

So far, we have the binary representation in 2's complement as

100010001.001001

Now we need to move the binary pointer to the MSB, which is 1 as said above, and have a base 2 multiplier.

Shifting the dot 8 digits ahead, we get,

$$1.0001000100\overline{1001} \times 2^8$$

This way of representing the number gives us a better idea of the IEEE 754 format of the sign bit, the mantissa and the exponent of the base. We know that the sign bit would be 1 since -273.15 is a negative number.

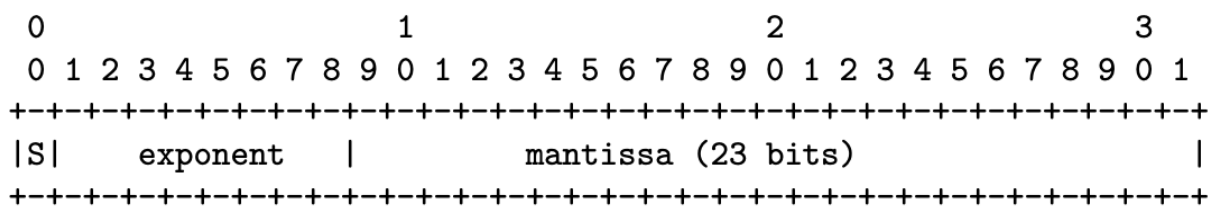
So now, we can list the components of the IEEE 754 that we know of,

$$\text{SB} = 1,$$

Mantissa = **0010001001001**,

E = 8.

IV. IEEE 754 step,



Our exponent field will be biased by the amount we shifted in for the mantissa. We have to include an exponential bias when representing in IEEE 754 format because they need to represent both negative and the positive values.

The lowest number an 8bit representation can represent is -127 and the highest is 128.

Now, we take 127 ($255 \% 2$) and map it by adding 8 to it as that was our exponent.
We get,

$$127 + 8 = 135.$$

V. Fill out the exponent,

Next, we convert 135 to binary representation.

$$(135)_{10} = (1000\ 0111)_2$$

VI. Finally, fill up.

The final step, after gathering all that we need, is to fill out the 32-bit space.

We have $S = 1$,

Exponent = 10000111

Mantissa = 00010001001001

So, we have,

1 10000111 00010001001001100110011

Therefore, we found the IEEE 754 single precision floating point number.

b. What is the decimal fraction that is actually stored in the single precision floating point number?

As mentioned in the conversions above, we encountered an infinitely repeating mantissa, 1001, and hence would require infinite storage space to represent it. In the computer world, this number cannot be represented perfectly.

So, we convert the floating-point number represented in 32-bits back to decimal fraction to observe the error due to conversion.

We have,

1 10000111 00010001001001100110011

The decimal fraction will be negative, and have exponent of 8 but with regards to the mantissa, it'll be a different story.

We have to find the mantissa in its decimal format.

To do that,

We multiply each bit with decreasing powers of 2 starting from the left-most bit and add them all up.

$$\sum_{p=-1}^{-23} (0 \text{ or } 1) * 2^p$$

We can obviously disregard the zeros, as they will remain the same.

So, we have,

$$m = 0 * 2^{-1} + \dots + 1 * 2^{-4} + \dots + 1 * 2^{-8} + \dots + 1 * 2^{-11} + \dots + 1 * 2^{-14} + 1 * 2^{-15} + \dots + 1 * 2^{-18} + 1 * 2^{-19} + \dots + 1 * 2^{-22} + 1 * 2^{-23}$$

$$m = 0.06699216365814208984375$$

Now, we have s, m and e, we can plug these into the formula below to get our decimal “equivalent”.

The formula goes $= (-1)^s \times (1 + m) \times 2^e$

$$= (-1)^1 \times (1 + 0.066992...) \times 2^8$$

$$= \mathbf{-273.149993896484375}$$

Therefore, the decimal fraction that is actually stored in the single precision floating point number is -273.149993896484375.

Also, this has an error of $273.15 - 273.14... = 0.000006103515625$

This shows that some numbers are impossible to represent perfectly in a binary form and the higher the bit representation the more precise it will be.

Problem 5.3: unicode and utf-8 encoding

UTF-8 sequence in hexadecimal notation = **f0 9f 90 84**

I. Convert to binary notation

0 = 0000, 4 = 0100, 8 = 1000, 9 = 1001, f = 1111

Therefore,

f0 = 11110000

9f = 10011111

90 = 10010000

84 = 10000100

f0 9f 90 84 = 11110000 10011111 10010000 10000100

II. The Unicode code point representation in Hexadecimal

For 4 bytes, we can have representation from U+10000 to U+10FFFF

Where,

byte 1 = 11110xxx

byte 2 = 10xxxxxx

byte 3 = 10xxxxxx

byte 4 = 10xxxxxx

We can take the binary expression from I and only take the x's

11110xxx 10xxxxxx 10xxxxxx 10xxxxxx = 11110**0000** 1001**1111** 1001**0000** 1000**0100**

00001111010000000100

Now, making groups of 4 from the back, we get,

0 0001 1111 0100 0000 0100


We can ignore the first zero, it could be the sign bit and hence there is a + in the representation.

So now, converting the above binary expression to hexadecimal, we get,

U+1F404

Therefore, the Unicode code point representation in Hexadecimal is U+1F404

III. Which character does it represent?

It represents 
(moos)

Problem 5.4: evil and odious numbers (haskell)

(evil_odious.txt present in the same .zip file)

```
evil_odious.hs — Assignment 5

evil_odious.hs x
evil_odious.hs
1  --Using recursion to count all the 1s in a binary representation.
2  onecounter :: Int -> Int
3  onecounter 0 = 0
4  onecounter 1 = 1
5  onecounter n = onecounter(div n 2) + (mod n 2)
6
7
8  --Returns TRUE if the count function has even number of 1s.
9  --zero 1s is also even so no extra conditions
10 isEvil :: Int -> Bool
11 isEvil n = even(onecounter n)
12
13
14 --Using list comprehension,
15 --and listing the numbers which are evil,
16 --user specified number of times.
17 evils :: [Int]
18 evils = [x | x <- [0..], (isEvil x) == True]
19
20
21 --Returns TRUE if the count function has odd number of 1s.
22 --but zero 1s is not odd so False
23 isOdious :: Int -> Bool
24 isOdious 0 = False
25 isOdious n = odd(onecounter n)
26
27
28 --Using list comprehension,
29 --and listing the numbers which are odious,
30 --user specified number of times.
31 odious :: [Int]
32 odious = [x | x <- [0..], (isOdious x) == True]
```