Jona Bako
jbako@jacobs.university.de

Homework - CS 2020 Problem Sheet #5

Problem 5.1

We plan to use a fixed size b-complement number system with the base b = 5 and n = 4 digits.

a) What are the smallest and the largest number that can be represented and why?
If the number set is unsigned, we have 624 + 1 elements in total:

- largest number = 4444_5 to base 10 -> 624_{10}
- smallest number = 0000_5 to base 10 -> 0_{10}

If the number set is signed, we have 625 elements separated in two spaces:

- largest number = 624_{10} / 2 = 312_{10}
- smallest number = -312_{10}

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

- $|(-1)_{10}| = |(1)_{10}| = 0001_5$ $|(-8)_{10}| = |(8)_{10}| = 0013_5$

```
1 = 0 * 5 + 1          1    8 = 1 * 5 + 3          3
0 = 0 * 5 + 0         01    1 = 0 * 5 + 1         13
0 = 0 * 5 + 0        001    0 = 0 * 5 + 0        013
0 = 0 * 5 + 0       0001    0 = 0 * 5 + 0       0013
```

Find the complement of our numbers in base b using $a'_i=(b-1)-a_i$:

```
0001: a_4 = 0 -> a'4 = (5-1) – 0 = 4
      a_3 = 0 -> a'3 = (5-1) – 0 = 4
      a_2 = 0 -> a'2 = (5-1) – 0 = 4
      a_1 = 1 -> a'1 = (5-1) – 1 = 3    4443₅

0013: a_4 = 0 -> a'4 = (5-1) – 0 = 4
      a_3 = 0 -> a'3 = (5-1) – 0 = 4
      a_2 = 1 -> a'2 = (5-1) – 1 = 3
      a_1 = 3 -> a'1 = (5-1) – 3 = 1    4431₅
```

Add 1 to -1 & -8 representation in this base as the last step:

$4443_5 + 1_5 = 4444_5$ $4431_5 + 1_5 = 4432_5$

```
     4443                  4431
+    0001             +    0001
----------            ----------
     4444                  4432
```

So, b-complement notation of -1 and –8 in base 5 with 4-digit representation are respectively 4444_5 and 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?

- Addition in b-complement notation of -1 and -8:

```
       4444
+      4432
-----------
      14431 (We drop the first 1 as it is an overflow // n = 4)
```

In order to convert back to decimal, we invert the bits using $a_i = b - 1 - a'_i$ (the inverse of $a'_i=(b-1)-a_i$), then we add 1 to the result and convert to decimal.

4431: a_4 = 4 -> $a_4 = 5 - 1 - 4 = 0$

a_3 = 4 -> $a_3 = 5 - 1 - 4 = 0$

a_2 = 3 -> $a_2 = 5 - 1 - 3 = 1$

a_1 = 1 -> $a_1 = 5 - 1 - 1 = 3$ $\quad 0013_4$

$0013_5 + 1_5 = 0014_5$

```
     0013             0014 -> |1 × 5 + 4 × 1| = |9|
+    0001             (value of sum is -9 however, since we took
----------            the absolute value of -1 and -8 as well the
     0014             carry-over overflow)
```

Problem 5.2

IEEE 754 floating point numbers (single precision) use the following format (the numbers on the top of the box indicate bit positions, the fields in the box indicate what the various bits mean).

```
 0                   1                   2                   3
 0 1 2 3 4 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|       exponent      |              mantissa (23 bits)           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The encoding starts with a sign bit, followed by the biased exponent (8 bits), followed by the mantissa (23 bits). For single-precision floating-point numbers, the exponents in the range of -126 to +127 are biased by adding 127 to get a value in the range 1 to 254 (0 and 255 have special meanings).

a) The absolute zero, 0 Kelvin, is at −273.15 degree Celsius. Explain step by step (and in your own words) how the decimal fraction -273.15_{10} is converted into a single precision floating point number.

Decimal fraction is negative, therefore sign bit S for -273.15_{10} is 1.

Converting integral part:

```
273 = 136 * 2 + 1        1    273₁₀ = 100010001₂
136 = 68 * 2 + 0        01
68 = 34 * 2 + 0        001
34 = 17 * 2 + 0       0001
17 = 8 * 2 + 1       10001
8 = 4 * 2 + 0       010001
4 = 2 * 2 + 0      0010001
2 = 1 * 2 + 0     00010001
1 = 0 * 2 + 1    100010001
0 = 0 * 2 + 0
```

$273_{10} = 100010001_2$

Converting fraction part:

0.15 * 2 = 0.3   0 $0.15_{10} = 00\overline{1001}_2$
0.3 * 2 = 0.6   0
0.6 * 2 = 1.2   1
0.2 * 2 = 0.4   0
0.4 * 2 = 0.8   0
0.8 * 2 = 1.6   1
Keeps repeating
0.6 * 2 = 1.2   1
0.2 * 2 = 0.4   0
0.4 * 2 = 0.8   0
0.8 * 2 = 1.6   1
...

Combine the two parts. $(100010001.00\overline{1001})_2$

Normalize number -> find the exponent: $(1.\mathbf{00010001}00\overline{1001})_2 \times 2^8$

Add the exponent bias $(127)_{10}$ to the exponent $(8)_{10}$ -> 127 + 8 = $(135)_{10}$

Convert the biased exponent $(135)_{10}$ to binary.

135 = 67 * 2 + 1   1 $135_{10} = 10000111_2$
67 = 33 * 2 + 1   1
33 = 16 * 2 + 1   1
16 = 8 * 2 + 0   0
8 = 4 * 2 + 0   0
4 = 2 * 2 + 0   0
2 = 1 * 2 + 0   0
1 = 0 * 2 + 1   1
0 = 0 * 2 + 0

Drop the leading 1 from the binary representation of the number $(1.00010001001001100110011)_2$.
So mantissa = 00010001001001100110011

Single precision floating point number representation of −273.15 is:

$$\underbrace{1}_{\text{S}}\ \underbrace{10000111}_{\text{exponent}}\ \underbrace{00010001001001100110011}_{\text{mantissa}}$$

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

To find out the decimal fraction stored in the single precision floating point number we use the formula that represent the reverse of the above steps: $(-1)^S \times (1 + m) \times 2^e$

$S = 1$, $m = (00010001001001100110011)_2$, $e = 135-127=8$

Calculation:

$(-1)^1 \times (1 + (2^{-4}+2^{-8}+2^{-11}+2^{-14}+2^{-15}+2^{-18}+2^{-19}+2^{-22}+2^{-23}) \times 2^8$

= -273.149994 -> decimal fraction actually stored in the single precision floating point number

Problem 5.3

You are given the following UTF-8 sequence in hexadecimal notation (4 bytes):

`f0 9f 90 84`

Write each byte in binary notation and identify the bits representing the unicode code point. What is the Unicode code point (in hexadecimal) and which character does it represent?

UTF-8 hexadecimal to binary

f 0 9 f 9 0 8 4

1111 00**00** 10**01** **1111** 10**01** **0000** 10**00** **0100**

1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx

Unicode point in binary

000111110100000000100 =>

0001 1111 0100 0000 0100

1 f 4 0 4 => U + 1f404

The character for the unicode code point is a cow. 🐄