HBL551 (Numerical Analysis):
Homework 1

Raymart Jay Canoy

September 9, 2020

Problem

1. Show that the IEEE Standard 754 *single precision* floating-point representation

1	1	0	0	0	0	1	0	1	1	1	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

   is equal to $(-118.625)_{10}$[*]

2. Convert $(-118.625)_{10}$ into IEEE Standard 754 *single precision* floating-point representation.

Solution

Problem 1:

a. This will be the notation that I will be using throughout my solution:

$$(-1)^s(1.dddd...d) \times 2^{e=k-127} \tag{1}$$

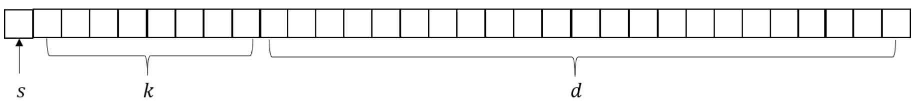

Figure 1: IEEE 754 *single precision* floating-point representation

where $s \equiv$ `sign`, $e \equiv$ `true exponent`, $k \equiv$ *biased* `exponent`, and $d \equiv$ `significand`.

b. From the IEEE Standard 754 *single precision* floating-point representation

1	1	0	0	0	0	1	0	1	1	1	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

,

$s = (1)$, $k = (10000101)_2$, and $d = (11011010100000000000000)_2$[**].

[*]I used the subscript 10 to denote the *decimal* system.

[**]I used the subscript 2 to denote the *binary* system

c. The general formula that I will be using is

$$\boxed{(-1)^s\,[1+(d)_{10}]\times 2^{e=(k)_{10}-127}}. \tag{2}$$

Note: Since there are 8 bins allocated for the exponent, the range of values of the exponent is $[0, 2^8-1] = [0, 255]$. To extend the exponent's range to negative numbers, the first bin in the exponent is allocated for the sign. Note, however, that it will be redundant to denote 10000000 as -0, so it is assigned instead to 128. Hence, the exponent's range becomes $[-(2^7-1), (2^7-1)+1] = [-127, 128]$. But the exponent, k, that is stored in the computer is the sum of the true exponent and the number $(2^7-1) = 127$. Therefore, if k is the value of the *biased* exponent that is stored in the computer, the number 127 must be deducted from k to get the true exponent, e; i.e., $\boxed{e = k - 127}$.

d. Converting k and d into decimal system gives:

$(k)_{10} = 2^7 + 2^2 + 2^0 = 133,$
$(d)_{10} = 2^{-1} + 2^{-2} + 2^{-4} + 2^{-5} + 2^{-7} + 2^{-9} = 0.853515625.$

Therefore, the IEEE Standard 754 *single precision* floating-point representation

1	1	0	0	0	0	1	0	1	1	1	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

is equal to $\boxed{(-1)^1(1+0.853515625)\times 2^{133-127} = (-118.625)_{10}}$†

Problem 2:

a. In converting $(-118.625)_{10}$ into IEEE Standard 754 *single precision* floating-point representation, I will break my solution into two parts. The first part involves the conversion of the whole number $(118)_{10}$ into binary system and the second part involves the conversion of the decimal number $(0.625)_{10}$ into binary system.

b. First part:
A positive base-10 integer I can be expressed as an expansion of powers of 2 as:

$$I = (b_n \times 2^n) + (b_{n-1}\times 2^{n-1}) + \ldots + (b_1 \times 2^1) + (b_0 \times 2^0). \tag{3}$$

Therefore, if $I = 118$, then the process of converting I into binary system involves:

$$\begin{array}{lcl} 118 = 59 \times 2 + 0 & \rightarrow & b_0 = 0, \\ 59 = 29 \times 2 + 1 & \rightarrow & b_1 = 1, \\ 29 = 14 \times 2 + 1 & \rightarrow & b_2 = 1, \\ 14 = 7 \times 2 + 0 & \rightarrow & b_3 = 0, \\ 7 = 3 \times 2 + 1 & \rightarrow & b_4 = 1, \\ 3 = 1 \times 2 + 1 & \rightarrow & b_5 = 1, \\ 1 = 0 \times 2 + 1 & \rightarrow & b_6 = 1. \end{array}$$

$$\boxed{\therefore\ (118)_{10} = (1110110)_2} \tag{4}$$

† The code that I made for converting IEEE standard 754 *single precision* floating-point representation to decimal can be found in page 4..

c. Second part

A fractional base-10 number R can be expressed as an expansion of powers of 2^{-1} as:

$$R = (b_1 \times 2^{-1}) + (b_2 \times 2^{-2}) + ... + (b_n \times 2^{-n}) + ... \tag{5}$$

such that

$$R = 0.b_1 b_2 ... b_n ... \tag{6}$$

Therefore, if $R = 0.625$, then the process of converting R into binary system involves:

$$\begin{array}{lll} 0.625 - 2^{-1} = 0.125 \geq 0 & \rightarrow & b_1 = 1, \\ 0.125 - 2^{-2} = -0.125 < 0 & \rightarrow & b_2 = 0, \\ 0.125 - 2^{-3} = 0 \geq 0 & \rightarrow & b_3 = 1. \end{array}$$

$$\boxed{\therefore\ (0.625)_{10} = (101)_2} \tag{7}$$

d. To convert $(-118.625)_{10}$ into IEEE 754 *single precision* floating-point representation, the sum of eqs. (4) and (7) is multiplied by $(-1)^1$:

$$\begin{aligned} (-1)^1 (118.625)_{10} &= (-1)^1((118)_{10} + (0.625)_{10}), \\ &= (-1)^1((11101101)_2 + (101)_2), \\ &= (-1)^1((1110110.101)_2), \\ &= (-1)^1(1.110110101 \times 2^6), \\ &= \boxed{(-1)^1(1 + 110110101) \times 2^6}. \end{aligned} \tag{8}$$

Comparing the result in eq. (8) with eq. (2), it is easy to see that $\boxed{s = 1}$, $\boxed{d = (110110101)_2}$, $e = (6)_{10}$, and $k = e + 127 = (133)_{10}$. The final step is to convert k into binary system, which follows the same steps done in part b:

$$\begin{array}{lll} 133 = 66 \times 2 + 1 & \rightarrow & b_0 = 1, \\ 66 = 33 \times 2 + 0 & \rightarrow & b_1 = 0, \\ 33 = 16 \times 2 + 1 & \rightarrow & b_2 = 1, \\ 16 = 8 \times 2 + 0 & \rightarrow & b_3 = 0, \\ 8 = 4 \times 2 + 0 & \rightarrow & b_4 = 0, \\ 4 = 2 \times 2 + 0 & \rightarrow & b_5 = 0, \\ 2 = 1 \times 2 + 0 & \rightarrow & b_6 = 0, \\ 1 = 0 \times 2 + 1 & \rightarrow & b_7 = 1. \end{array}$$

$$\boxed{\therefore k = (133)_{10} = (10000101)_2} \tag{9}$$

e. Therefore, using the notation presented in Figure (1), it is easy to see that the IEEE 754 *single precision* floating-point representation of $(118.625)_{10}$ is given by

1	1	0	0	0	0	1	0	1	1	1	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[‡]

[‡]The code that I made for converting decimal to IEEE 754 *single precision* floating-point representation can be found in page 6.

Codes

Problem 1

```
function dec = SingPrec2Dec(bin)
format long
bin = bin(~isspace(bin)); % remove spaces
bits = bin(:) - ‘0’;
len = size(bits);

% This is done because zeros(1,32) won’t give an accurate ans
zero = [‘00000000000000000000000000000000’];
zer = zero(:) - ‘0’;

%%%%%%%%%%%%%%%%%%%%%%%%%%32 bits%%%%%%%%%%%%%%%%%%%%%%
if len(1) == 32
    if bits == zer % this guarantees [‘000...0’]_2 = (0)_10
        dec = 0;

    else
        s = bits(1);
        k = bits(2:9); % 8 bins are allocated for exp
        d = bits(10:32); % 23 bins for the significand

        %%%%%%%%%%%%%%%%%%%%%%exponent%%%%%%%%%%%%%%%%%%
        for i = 1:8
            if (s == 1 | s == 0) & (k(i) == 1 | k(i) == 0)
                k_i(i) = k(i) * 2 ^ (8-i);
            else
                error(‘a bit can only be 0 or 1’)
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%significand%%%%%%%%%%%%%%%%
        for i = 1:23
            if d(i) == 1 | d(i) == 0
                d_i(i) = d(i) * 2 ^ (-i);
            else
                error(‘a bit can only be 0 or 1’)
            end
        end

        %%%%%%%%%%%%%%%%%%%%%%values in decimal%%%%%%%%%%%
        k_10 = sum(k_i);
        d_10 = sum(d_i);
```

```
            format long g
            %%%%%%%%%%%%%%%%%%%%%%%%converted value%%%%%%%%%%%%%%
            dec = ((-1)^(s))*(1 + d_10)*2^(k_10 - 127);
        end

    else
        error('Single Precision: bits = 32!')
end
```

Command Prompt:

```
>> SingPrec2Dec([‘11000010111011010100000000000000’])

ans =

                  -118.625

>> SingPrec2Dec([‘00000000000000000000000000000000’])

ans =

     0

>> SingPrec2Dec([‘10111111100000000000000000000000’])

ans =

    -1

>> SingPrec2Dec([‘00111111100000000000000000000000’])

ans =

     1

>> SingPrec2Dec([‘01000001001000000000000000000000’])

ans =

    10
```

Problem 2

```
function bin = Dec2SingPrec(dec)
if dec == 0
    bin = num2str(zeros(1, 32));
else
    %%%%%%%%%%%%%%%%%%%%%%%sign%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if dec < 0
        s = [1]; % s = 1
    else
        s = [0]; % s = 0
    end

    dec = abs(dec); % the sign is ignored

    %%%%%%%%%%%%%%%whole number%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    whole_number = floor(dec);


    w_n = whole_number;
    i = 1;
    while (1)
        w_n_2(i) = mod(w_n, 2);
        w_n = floor(w_n / 2);
        i = i + 1;
        if w_n == 0, ...
                break, end
    end
    w_n_2 = wrev(w_n_2); % whole number in binary
    l = size(w_n_2); % size of the whole num vect

    %%%%%%%%%%%%%%%%fractional number%%%%%%%%%%%%%%%%%%%%%%
    frac_number = dec - whole_number;

    f_n = frac_number;
    for i = 1:100
        if (f_n - 2^-i) >= 0
            f_n_2(i) = 1;
            f_n = f_n - 2^(-i);
        else
            f_n_2(i) = 0;
            f_n = f_n;
        end
    end
```

```
    %%%%%%%%%%%%%%%%%%%exponent%%%%%%%%%%%%%%%%%%%%%%%%%
    if whole_number ~= 0
        e = l(2) - 1; % (true exp) going to the left: pos
        k = e + 127; % biased exponent

        %%%%%%%%%%%%%%significand%%%%%%%%%%%%%%%%%%%%%%%
        index = [2: l(2)];
        d = [w_n_2(index) f_n_2];

        %%%%considers dec num whose whole num is zero%%%%%%
    else
        %go through the values of f_n_2 to look for the 1st 1
        m = 1;
        while(1)
            if f_n_2(m) == 1, break, end
            m = m + 1;
        end
        %true exp = index of 1st 1
        e = -m; % (true exp) going to the right: neg
        k = e + 127; %biased expoenent

        %%%%%%%%%%%%%%significand%%%%%%%%%%%%%%%%%%%%%%%

        len_f_n = size(f_n_2); % len of frac number in bin
        index_l_f_n = [m + 1: len_f_n(2)];%from num after 1
        f_n_2 = f_n_2(index_l_f_n);

        d = f_n_2; %significand contains bits from frac num
    end

    %%%%%%%%%%%converts exp to bin%%%%%%%%%%%%%%%%%%%%%%
    k_loop = k;
    k_2 = zeros(1, 8); % k_2 is set to contain 8 elements
    j = 1;
    while (1)
        k_2(j) = mod(k_loop, 2);
        k_loop = floor(k_loop / 2);
        j = j + 1;
        if k_loop == 0, break, end
    end

    k_2 = wrev(k_2); % biased exp in binary

    %%%%%%%%%%IEEE 754 single precision%%%%%%%%%%%%%%%%%%%
    index_sig = [1 : 23];
    bin = num2str([s k_2 d(index_sig)]);
end
end
```

Relative Percent Error Function

```
function rel_per_accu()
true_value = input('True value = ');
num_value = input('Numerical value = ');

eps = (abs(true_value - num_value)/true_value)*100;
fprintf('\n')
fprintf('Relative percent error is %.20f%%\n', eps)
fprintf('\n')
end
```

Command Prompt

```
>> a = Dec2SingPrec(-118.625)

a =

    '1  1  0  0  0  0  1  0  1  1  1  0  1  1  0  1  0  1
0  0  0  0  0  0  0  0  0  0  0  0  0  0'

>> SingPrec2Dec(a)

ans =

                  -118.625

>> b = Dec2SingPrec(0)

b =

    '0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0'

>> SingPrec2Dec(b)

ans =

     0
```

```
>> c = Dec2SingPrec(-1)

c =

    '1  0  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0'

>> SingPrec2Dec(c)

ans =

    -1

>> d = Dec2SingPrec(1)

d =

    '0  0  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0'

>> SingPrec2Dec(d)

ans =

     1

>> e = Dec2SingPrec(-10)

e =

    '1  1  0  0  0  0  0  1  0  0  1  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0'

>> SingPrec2Dec(e)

ans =

   -10

>> f = Dec2SingPrec(10)

f =

    '0  1  0  0  0  0  0  1  0  0  1  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0'
```

```
>> SingPrec2Dec(f)

ans =

    10

>> g = Dec2SingPrec(-0.5)

g =

    '1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0'

>> SingPrec2Dec(g)

ans =

                              -0.5

>> h = Dec2SingPrec(0.5)

h =

    '0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0'

>> SingPrec2Dec(h)

ans =

                               0.5

>> i = Dec2SingPrec(-0.2)

i =

    '1 0 1 1 1 1 1 0 0 1 0 0 1 1 0 0 1 1
0 0 1 1 0 0 1 1 0 0 1 1 0 0'

>> SingPrec2Dec(i)

ans =

          -0.199999988079071
```

```
>> rel_per_accu()
True value = -0.2
Numerical value = -0.199999988079071

Relative percent error is 0.0000059604645108457 5%

>> j = Dec2SingPrec(0.2)

j =

    '0  0  1  1  1  1  1  0  0  1  0  0  1  1  0  0  1  1
0  0  1  1  0  0  1  1  0  0  1  1  0  0'

>> SingPrec2Dec(j)

ans =

         0.199999988079071

>> rel_per_accu()
True value = 0.2
Numerical value = 0.199999988079071

Relative percent error is 0.0000059604645108457 5%

>> k = Dec2SingPrec(-0.0001)

k =

    '1  0  1  1  1  0  0  0  1  1  0  1  0  0  0  1  1  0
1  1  0  1  1  1  0  0  0  1  0  1  1  1'

>> SingPrec2Dec(k)

ans =

     -9.99999974737875e-05

>> rel_per_accu()
True value = -0.0001
Numerical value = -9.99999974737875e-05

Relative percent error is 0.0000025262125019891 9%
```

Reference

1. M. R. King and N. A. Mody, *Numerical and Statistical Methods for Bioengineering: Applications in MATLAB*, Cambridge University Press, Cambridge, United Kingdom (2011).