System Program

Professor Honguk Woo

Name : Jihwan Kim

Student Number : 2015312911

Major : Biomechatronics / Electrical and Electronic Engineering

Before Explaining the Actual Implementation, first thing to note is that data type sfp is unsigned short, which is 16 bits (2 bytes)

Expressing NaN and Infinity

In this assessment, NaN for 16 digit half precision floating point is expressed as follows
NaN : Chose Quiet NaN
NaN = 1 11111 1111111111

Likewise,
Positive Infinity : 0 11111 0000000000
Negative Infinity : 1 11111 0000000000

This convention will be unified throughout the code written for the assignment.

Also, the range of SFP should be discussed in order to use those values to notify invalid input.

Denormalized : $(-1)^s M 2^{1-bias}$: Representing numbers close to zero
exp = 00000
(0 00000 0000000001)sfp = (2^-10) * 2^(1-15) = (0.000000059607)dec : Closest to zero

Normalized : $(-1)^s M 2^{\exp - bias}$: Representing the rest of the expressible numbers
exp = 00001 ~ 11110

Special :
exp = 11111

Hence, maximum for SFP is 0 11110 1111111111 = $+\sum_{i=1}^{10} 2^{-i} \times 2^{30-15} = +32736$
and minimum is 1 11110 1111111111 = $-\sum_{i=1}^{10} 2^{-i} \times 2^{30-15} = -32736$

sfp2bits

- First thing to implement for this task will be converting sfp to bit-stream of string type. Because sfp is 16 bits long, loops will need to run 16 times. High and Low byte must also be considered. Since x86 stores multi byte objects in a Little-Endian fashion, bit shifting by 8 bits(=byte) then refreshing the data after calculating all the bits was required.

```
char* sfp2bits(sfp result)
{
    sfp tmp = result;
    char *bitStream = malloc(sizeof(char) * 16);
    // push binary to sfp and count the number of index
    tmp = tmp >> 8;
    for(int i = 7; i >= 0; i--)
    {
        if ((tmp & 0x01) == 1)
            bitStream[i] = '1';
        else
            bitStream[i] = '0';
        tmp = tmp >> 1;
    }
    tmp = result;
    for(int i = 15; i >= 8; i--)
    {
        if ((tmp & 0x01) == 1)
            bitStream[i] = '1';
        else
            bitStream[i] = '0';
        tmp = tmp >> 1;
    }
    printf("%s\n", bitStream);
    return bitStream;
}
```

The range of i was specified as 7 to 0 and 15 to 8 to notify that the code above tries to interpret little endian architecture. It does not have a functional meaning.

int2sfp

First, setting all the exception was done at the start of the code. Then, since integer is signed by default, checking sign of the input value was required. Then comes the processing of exponent and mantissa. All bits of the integer except for the sign bit is used to show mantissa. Even for integer, the number is expressed in bits. For example, 17.00 = 10001.00. Therefore, just knowing the bit sequence and index of the highest bit will help determining floating point bit sequence.

43 (in decimal) = 101011 (in binary)
Bias = 2^(5-1)-1 = 15
Integer Representation is 00000000000000000000000000101011
Need to be represented as 1.01011 * 2 ^ 5 for normalization
When implied 1 is removed and bias is applied, it will be 0 10100 0101100000
Please refer to the code snippet below.

```
// return highest index from number
    for (int i = 0; i < 31; i++)
    {
        if (tmp % 2 == 1)
        {
            exponent = i;
        }
        tmp /= 2;
    }
    // remove implied 1
    mantissa = mantissa & ~(0x01 << exponent);
    index--;
    exp = 15 + exponent;
    // shift left
    exp = exp << 10;
```

To illustrate the use of arithmetic operation instead of bit shifting, the for loop uses modulus and divide operation. Variable 'tmp' refers to the mantissa part. The for loop is used to find the index of the highest bit of mantissa. This index number is then used to remove the highest bit of mantissa.

sfp2int

Using the expression for counting

$$v = (-1)^s \, M * 2^{Exponent-Bias}$$

Where Bias = 15

When calculating M, it is important to note that the largest number of denominator is 1024. (a) accumulates the nominators that has common denominator of 1024.

Then, the variable 'powtwo' sets the sign of the result. This is then multiplied as much as exponent.

```
// mantissa
   for (int i = 0; i < 10; i++)
   {
       mul = mul * 2;
       if ((mantissa & 0x0200) == 0x0200)
(a)    {
           // equalize divisor
           m = m + 1024 / mul;
       }
       mantissa = mantissa << 1;
   }

   int powtwo;
   if (sign == 0x8000)
       powtwo = -1;
   else
       powtwo = 1;

   // use mathemetical expression v=m*2^e
   for (int i = 0; i < e; i++)
   {
       powtwo = powtwo * 2;
   }
   result = powtwo + (powtwo * m) / 1024;
```

The 'result' variable shows the following

$$result = 2^e + \frac{2^e m}{1024}$$

Which is

$$result = 2^e \, (1 + \frac{m}{1024})$$

The added 1 means that mantissa is normalized and is supposed to have a hidden 1.

sfp_add

Addition of sfp numbers

Series of steps taken:

1. Most times, the two numbers in the parameters a and b will be different and this means that one will be greater than another. Make sure smaller number's exponent is aligned with the larger number. When doing so, the smaller number's mantissa will shift right by 1 and exponent will increase by 1. In the code, if the variable 'shiftby' is greater than 1, then it means exponent aligning will have to happen.

* When dealing with hexadecimal numbers, it was crucial to know that '--' and "++' should not be used. Instead, for a sfp value 'v' containing binary number, v++ or v—should be used.

When dealing with negative numbers, pre-calculation helped.

Decimal	3/8 * 2 ^ (13 – 15) = 3/32	1/8 * 2 ^ (16 – 15) = -1/8
SFP	0 01101 0110000000	1 10000 0010000000
Align Exponent (hidden bit!)	0 10000 0010110000	1 10000 0010000000
Find greater mantissa and subtract	1.0010000000 – 0.0010110000 = 0.1111010000	
Normalize	M = 1.1110100000 and E = 01111	
Take Out Implied 1 and integrate into expression	? 01111 1110100000	
SFP that had greater mantissa has greater influence over the sign bit	S = 1	
Result	1 01111 1110100000	

After normalizing, round to even method should be considered, which, in this case was done by counting the number of mantissa digits and capturing the bits that must be rounded. For example, For example, if mantissa is 111001010111 (11 bits) it has to be rounded so that we only have 10 bits. The top 10 bits are 1110010101 and bits to be rounded include 11. We first need to capture all the bits that need to be rounded.

If there are any 0s in the bit following the leftmost bit, then it is greater than half. If it is exactly half check the LSB of the retained bit and if 1, add 1 to mantissa. Shift right to eliminate bits after all these processes.

sfp_mul

As done in the previous question, technicality in calculation is crucial. It determines the algorithm for calculation, hence each steps in calculation is listed in the table below.
Calculating 0101000101100000 * 0100101110000000

Decimal		
SFP	0 10100 0101100000	0 10010 1110000000
Exp A + Exp B – Bias	10100 + 10010 = (1)00110 (1)00110 – 01111 = 10111	
Multiply Mantissa (Hidden Bit)	1.0101100000 * 1.1110000000 = 10.10000101000000000000 Note that this is an 11 * 11 multiplication resulting in 22 bits	
Normalize Adjust Exponent	If there are bits greater than 20th bit, normalize 1.0100001010 E += 1	
Truncate Mantissa By 11 bits (shift right)	10.100001010	
	11th bit (hidden bit) will have to represent MSB	
Take Out Implied 1	0100001010	
Integrated Result	0 11000 0100001010	

Regarding multiplied mantissa value, because the number of digits after binary point is 10 + 10 = 20, mR & 0xFFF00000 is done to extract the bits above binary point. Then, exponent is added or subtracted in for loop in order to set the number above binary point to 1. This is done to normalize the mantissa.

After normalizing, rounding must be considered and some bits need to be dropped. Round to Even method requires that :
Int result = mA * mB;
If (last 10 bit of Result > 1000000000)
{ roundup }
Else if (last 10 bit of result == 1000000000)
{ if(11th bit == 1) roundup }
Else
{ }

Finally, Shifting of the 10 bits is done to make sure mantissa of the resultant sfp is only 10 bits. This Automatically rounds down the number