MONTCLAIR STATE UNIVERSITY

Department of CSIT

Homework # 1

Dos Santos Ferreira, Joao Paulo

CSIT 230_02 – Computer Systems

Instructor: Dr. G.E. Antoniou

September 15, 2018

Problem 1:

Write short JAVA program to convert a 32-bit FPN(IEEE 754, single precision, with bias 127) into its decimal equivalent. As test example use:

1 10000010 00100000000000000000000

Solution:
A 32-bit FPN number is divided into 3 parts, the first bit stores the sign (s), the next 8 bits stores the binary representation of the biased exponent, the decimal value of the biased exponent is represented by the formula:

$$Biased\ Exponent\ =\ 127 + Unbiased\ Exponent$$

The unbiased exponent is found by normalizing the binary representation of the number.

The last 23 bits is the mantissa composed by significant of the normalized number from left to right with the rest of the bits filled with 0's.

The program bellow reads 32-bit FPN representation of a given number. First it finds the value for s by reading the first digit (if it is 1 it means the number is negative, else it is positive). Then it reads the next 8 bits and converts them into the decimal form of the biased exponent; using the formula above it also finds the unbiased exponent.
Next, it reads the mantissa and it converts it to binary normalized value using the unbiased exponent value, and lastly it finds the number's decimal value and prints the result.

Code:

```
// Homework #1
// This program converts a 32-bit FPN into its decimal equivalent

public class FPN32ToDec {
	public static void main(String args[]) {

		// 32-bit FPN representation of the number
		String number = "1 10000010 00100000000000000000000";

		// reads the first bit to determine
		// whether it is a positive or negative number
		int s = Integer.parseInt(number.substring(0, 1));

		// reads the 8bit representation of the biased exponent
		String biasExpBin = number.substring(2, 10);

		// finds the decimal representation of the biased exponent
		int biasExpDec = 0;
		for (int i = 0; i < 8; i++) {
```

```
                  biasExpDec += (int)
Integer.parseInt(biasExpBin.substring(i, i+1))*Math.pow(2, (7-i));
            }

            // finds the decimal representation of the unbiased exponent
            int unbiasExpDec = biasExpDec - 127;


            // reads the mantissa
            String mantissaBin = number.substring(11,34);

            // Finds the decimal value of the normalized number
            // Method to find it changes based on the value of the
            // unbiased exponent
            double decNumber = 0;
            if (unbiasExpDec > 0) {

                  // Gets the binary representation of the number
                  String binNumber = "1"+
mantissaBin.substring(0,unbiasExpDec);

                  // Finds the decimal representation of the number
                  for (int i = 0; i < binNumber.length(); i++) {
                        decNumber += Integer.parseInt(binNumber.substring(i,
i+1))*Math.pow(2, binNumber.length()-1-i);
                        //System.out.println(decNumber);
                  }

            } else if (unbiasExpDec == 0) {
                  mantissaBin = "1"+ mantissaBin;
                  for (int i = 0; i < 23; i++) {
                        decNumber +=
Double.parseDouble(mantissaBin.substring(i, i+1))/Math.pow(2, i);
                  }
            } else {
                  mantissaBin = "1"+ mantissaBin;
                  for (int i = 0; i < 23; i++) {
                        decNumber +=
Double.parseDouble(mantissaBin.substring(i, i+1))/Math.pow(2, i+1);
                  }
            }

            // prints the result
            if(s == 1) {
                  decNumber = decNumber*(-1);
                  System.out.println(number + " = " + decNumber);
            } else {
                  System.out.println(number + " = "  + decNumber);
            }
      }
}
```

Sample Run:

```
----jGRASP exec: java FPN32ToDec
1 10000010 00100000000000000000000 = -9.0

----jGRASP: operation complete.
```

Result:

The decimal equivalent of *1 10000010 00100000000000000000000* is *-9*.

Brief Comments:

The program runs correctly, according to the specifications. It works perfectly for any number including integers and real numbers that have an unbiased exponent no higher than 23;

Problem 2:

Write short JAVA program to convert decimal numbers to 32-bit FPN's (IEEE 754, single precision, with bias 127). As test use the decimal Number 12.

Solution:

To be able to convert a decimal number into 32-bit FPN we must first determine whether it is positive (s=1) or negative (s=1) and assign the correct value to the first bit. Next, we need to find the normalized binary version of the number, which should give us the unbiased exponent and the significant. With the unbiased exponent we can find the decimal for of the biased exponent by using the same formula from Problem 1. After finding the decimal form of the biased exponent we convert it to its 8bit binary representation which will be added after the sign bit. Lastly, we add the significant to the mantissa from left to right and fill the rest of it with 0s;

There are many ways that can be done in JAVA. I have presented two different ways to solve this problem using JAVA.

The first program follows the steps of the algorithm in the paragraph above. It reads the decimal representation of the number and determines its sign bit. Then it finds the binary representation of the biased exponent by finding the unbiased exponent first. Next, it converts the number into its normalized binary form and creates the mantissa. Both the biased exponent and the mantissa are store in arrays for printing purposes. Lastly, the program prints the result accordingly.

The second program uses a built-in method in java that converts the number into a string that represents it as 32-bit FPN.

Code:

```
// Homework #1 - Problem 2 (Main Program)
// The following program converts a decimal number
// into its 32-bit FPN representation

public class DecToFPN32 {
	public static void main(String args[]) {

		// reads the decimal representation of the number
		int decNumInt = 12;

		// determines the bit value of the sign
		int s = 0;
		if(decNumInt < 0) {
			s = 1;
		}

		// gets the absolute value of the number
		// the program does not need to care about the sign anymore
		int positiveNum = Math.abs(decNumInt);


		int i = 0;
		int unbiasExpo = 0;
		int indexHolder = 0;

		// finds the unbiased exponent
		while (Math.pow(2, i) <= positiveNum) {
			indexHolder = i;
			i++;
		}
		unbiasExpo = indexHolder;

		// finds and converts the biased exponent into
		// its 8-bit binary form and stores it in as array
		int biasedExpo = 127 + unbiasExpo;
		int[] biasExponentBin = new int[8];
		for(i = 7; i >= 0; i--) {
			biasExponentBin[i] = biasedExpo%2;
			biasedExpo /= 2;
		}

		// converts the number into its binary form
		// and stores it in an array
		int[] binNumber = new int[unbiasExpo+1];
		while (positiveNum > 0) {
			binNumber[indexHolder] = positiveNum%2;
			positiveNum = positiveNum/2;
			indexHolder--;
		}

		// creates and populates the mantissa
		// in an array
```

```
		int[] mantissa = new int[23];
		for(i = 0; i < 23; i++) {
			if (i < unbiasExpo) {
			mantissa[i] = binNumber[i+1];
			}
		}

		// prints the result
		System.out.print(decNumInt + " = " + s + " ");
		for(i = 0; i < 8; i++) {
			System.out.print(biasExponentBin[i]);
		}
		System.out.print(" ");
		for(i = 0; i < 23; i++) {
				System.out.print(mantissa[i]);
		}
	}
}
```

Second program (alternative):

```
// Homework #1 - Problem 2 (alternative program)
// This program uses the built-in method in java
// to convert a number into its 32-bit FPN representation

public class DecToFPN32x2 {
	public static void main(String args[]) {
		// reads the number
		float num = (float) 12;

		// uses built in method to convert it into
		// 32-bit FPN
		int intBits = Float.floatToIntBits(num);
		String binary = Integer.toBinaryString(intBits);

		// prints the result
		System.out.print(num + " = ");
		if(binary.length() < 32) {
			for (int i =0; i < (32 - binary.length()); i++)
				System.out.print(0);
			System.out.print(binary);
		} else {
			System.out.println(binary);
		}
	}
}
```

Sample Run:

Main program:

```
----jGRASP exec: java DecToFPN32
12 = 0 10000010 10000000000000000000000
----jGRASP: operation complete.
```

Second program (alternative):

```
 ----jGRASP exec: java DecToFPN32x2
12.0 = 01000001010000000000000000000000
 ----jGRASP: operation complete.
```

Result:

The 32-bit FPN representation of *12* is *0 10000010 10000000000000000000000*

Brief Comments:

Both programs run correctly, according to the specifications. The first program details the algorithm that is followed to convert an integer number into its 32-bit FPN representation, it only works for integer values. The second program on the other hand, although it uses a built-in method in JAVA it works for any float number in the range of the 32-bit FPN representation.

Problem 3:

Write short JAVA program to convert Binary Numbers into Hexadecimal Numbers and back.

Solution:
There are multiple ways to convert a binary number into hexadecimal. One of the is by first converting the binary number into decimal and then converting the decimal into hexadecimal.

This program reads the input from the user that should be a binary number. It then finds the decimal representation of the number, and next it finds the hexadecimal representation of the number.
This program also converts the hexadecimal number back into binary by finding the decimal representation of the hexadecimal number found and the converting the decimal into binary. Printing both results on the screen.

Code:

```
// HW #1 - Problem 3
// This program converts Binary Numbers
// into Hexadecimal Numbers and back.

import java.util.Scanner;
```

```
public class BinaryHexa {
	public static void main(String args[]) {

		// Asks the user for a binary number and reads the input
		Scanner scan = new Scanner(System.in);
		System.out.println("Enter a Binary Number to be converted into
Hexadecimal: ");
		String binNumber = scan.next();

		// Finds the decimal representation of the binary number
		int i = 0;
		int decNumber = 0;
		for (i = 0; i < binNumber.length(); i++) {
			decNumber += Integer.parseInt(binNumber.substring(i,
i+1))*Math.pow(2, binNumber.length()-1-i);
		}

		// Finds how many elements the hexadecimal number have
		i = 0;
		int indexHolder = 0;
		while (Math.pow(16, i) <= decNumber) {
			indexHolder = i;
			i++;
		}
		i = indexHolder;

		// Finds the Hexadecimal representation of the number
		// and stores it in an array;
		String[] hex = new String[i+1];
		hex[0] = "0";
		while(decNumber > 0) {
			switch (decNumber%16) {
			case 10: {
				hex[i] = "A";
				break;
			}
			case 11: {
				hex[i] = "B";
				break;
			}
			case 12: {
				hex[i] = "C";
				break;
			}
			case 13: {
				hex[i] = "D";
				break;
			}
			case 14: {
				hex[i] = "E";
				break;
			}
			case 15: {
				hex[i] = "F";
				break;
```

```
                        }
                        default: {
                                hex[i] = Integer.toString(decNumber%16);
                        }
                        }
                        decNumber = decNumber/16;
                        i--;
                }

                // Prints the results
                System.out.print(binNumber + " in hexadecimal is: ");
                for(i = 0; i <= indexHolder; i++) {
                        System.out.print(hex[i]);
                }

                // Converts the hexadecimal number into its decimal form
                decNumber = 0;
                for (i = 0; i <= indexHolder; i++) {
                        switch (hex[indexHolder-i]) {
                        case "A": {
                                decNumber += (int) (10*Math.pow(16, i));
                                break;
                        }
                        case "B": {
                                decNumber += (int) (11*Math.pow(16, i));
                                break;
                        }
                        case "C": {
                                decNumber += (int) (12*Math.pow(16, i));
                                break;
                        }
                        case "D": {
                                decNumber += (int) (13*Math.pow(16, i));
                                break;
                        }
                        case "E": {
                                decNumber += (int) (14*Math.pow(16, i));
                                break;
                        }
                        case "F": {
                                decNumber += (int) (15*Math.pow(16, i));
                                break;
                        }
                        default: {
                                decNumber += (int)
(Integer.parseInt(hex[indexHolder-i])*Math.pow(16, i));
                        }
                        }
                }

                // finds how many elements the binary number has
                int indexHolder2 = 0;
                while (Math.pow(2, i) <= decNumber) {
                        indexHolder2 = i;
                        i++;
```

```
                }
                i = indexHolder2;



                // converts the number into binary form
                // and stores it in an array
                int[] binNum = new int[i+1];
                while (decNumber > 0) {
                        binNum[i] = decNumber%2;
                        decNumber = decNumber/2;
                        i--;
                }

                // prints the result
                System.out.println();
                for(i = 0; i <= indexHolder; i++) {
                        System.out.print(hex[i]);
                }
                System.out.print(" in binary is: ");
                for(i = 0; i <= indexHolder2; i++) {
                        System.out.print(binNum[i]);
                }
        }
}
```

Sample Run:

```
 ----jGRASP exec: java BinaryHexa
Enter a Binary Number to be converted into Hexadecimal:
1110011010101
1110011010101 in hexadecimal is: 1CD5
1CD5 in binary is: 1110011010101
 ----jGRASP: operation complete.
```

Result:

From the example the binary number *1110011010101* in hexadecimal is *1CD5*.

Brief Comments:

The program runs correctly according to the specifications.

Problem 4:

Write Perform the following operation using BCD representation:

(a) $(199+12)_{10}$

Solution:

The BCD representation of a number is done by representing each number individually as 4-bit binary. For example. 199 in BCD representation is found by representing 1, 9 and 9 as 4-bit binary, that is:

199 = 0001 1001 1001

12 = 0001 0010

Addition with BCD is similar to regular addition, however if any of the terms of the binary addition is greater than 1001 (9 in decimal) we must add 0110 (6 in decimal) to find its correct value in BCD notation. Repeating this process until none of the terms are greater than 1001.

```
                      1
199          0001 1001 1001
+12        +      0001 0010
           ------------------
           0001 1010 1011
```

Since the second and third terms of the result are both greater than 1001 we must add 0110 to each one of them, we have:

```
     1  11   1  11
0001 1010 1011
+      0110 0110
-----------------
0010 0001 0001
```

The result contains no terms that are greater than 1001, that is the BCD representation of the number $(199+12)_{10}$. Converting that number into decimal we have $(211)_{10}$.

Result:

$(199+12)_{10}$ using BCD notation is $(0010\ 0001\ 0001)_{BCD}$ which is the same as $(211)_{10}$.

Problem 5:

Write Perform the following operation using signed binary 8-bit 2's complement representation:

(a) $(4 - 12)_{10}$ using signed binary 8-bit 2's complement.

Solution:

To subtract using 2's complement we need to convert the first number (positive) into binary and the second number (the one that is being subtracted from the first, therefore with the negative sign) into its 2's complement representation. Since we are using 8-bit 2's complement we have the binary representation of the numbers:

$(4)_{10} = (0000\ 0100)_2$

$(12)_{10} = (0000\ 1100)_2$

To find the 2's complement of 12 we need to flip all of the bits (which gives us the 1's complement of 12) and then add 1 to the result.

$(12)_{10} = (0000\ 1100)_2 = (1111\ 0011)_{1\text{'s complement}} = (1111\ 0111)_{2\text{'s complement}}$

```
          1 1
1111 0011
+         1
-----------
1111 0100
```

To find the result of $(4 - 12)_{10}$ we just need to add the binary representation of 4 to the 2's complement of 12.

```
  4                  1
-12        0000 0100
         + 1111 0100
           ---------
           1111 1000
```

We have $(4 - 12)_{10}$ equals to 1111 1000 in 2's complement, since the first bit is 1 we know this number is negative, to convert it back into decimal we need to flip all the bits (1's complement) and add 1 to the result.

```
                  1 1 1
1111 1000   0000 0111
          +         1
            ---------
            0000 1000
```

$(0000\ 1000)_2$ is 8, therefore $(1111\ 0100)_{2\text{'s complement}}$ is -8;

Result:

$(4 - 12)_{10}$ is equals to $(1111\ 0100)_{2\text{'s complement}}$ which is the representation of -8.