**ICS 2202: COMPUTER ORGANISATION AND ARCHITECTURE**

**ASSIGNMENT 2**

**GROUP 1**

**DATE: 9TH DECEMBER 2020**

|  |  |
| --- | --- |
| **NAMES** | **ADMISSION NUMBERS** |
| Faith Sanne Odhiambo | **121058** |
| Tracy Achieng Ogutu | **112473** |
| Dennis Kipkorir Koech | **115325** |
| Tevin Musau Mutua | **118640** |

**INTRODUCTION**

* For this assignment, we created a **Java program** using the **NetBeans IDE** to implement the solutions to the assignment posted on the e learning platform.
* For both parts (2a and 2b), the **output has been snapshot** and stored in the folders “**Part 2a Snapshots”** and “**Part 2b Snapshots”,** respectively, which have also been included in the zipped folder.
* In the zipped folder, there also contains two folders namely.

1. **Part 2a Demonstration**
2. **Part 2b Demonstration**

These folders contain the **demonstration** for each part of the assignment.

* The program has also been **uploaded to GitHub**; you may access it through the following link (https://github.com/TevinMusau/COA-Group-Assignment-2).
* This was a **group effort.**

**Part 2a**

1. **Snapshots of the output (Can also be found in the “Part 2a Snapshots” Folder)**

* Kindly check in the folder “**Part 2a Snapshots”** which has been included in the zipped file

1. **Code explanation and short demonstration**

* For this part, we were to create a program to calculate the equivalent binary and hexadecimal values for the first 256 numbers, i.e., 1 to 256 and store the results in a table format.
* The created program was implemented with loops to perform the following:

1. **Print out numbers from 1 to 256**. This will represent the 256 decimal numbers which we are to convert to their binary and hexadecimal equivalents.
2. Calculate the **binary equivalent** for each of the numbers from 1 to 256 and **store the binary value**. There is an if statement, that checks if **the number is divisible by 2**. If it is divisible by 2, the value **zero (0) is stored in an array**. If the number is **not divisible by 2**, the **value 1 (one) is stored in an array**.
3. Calculate the **hexadecimal equivalent** for each number from 1 to 256 and **store t**he resulting **hexadecimal value in an array**. There is a predefined array, called **hexchars**, with all the **hexadecimal values (0 to F)**. This array will be used to match the modulus value of the division, *e.g., 14 % 16 = 14*. Therefore, the fourteenth value in the **array hexchars** is **fetched (E)** and **stored in the hexadecimal array**.
4. **Print out the array** with the **equivalent hexadecimal value**. The arrays are **printed backwards** to **correctly** form the calculated **hexadecimal value** the decimal number.
5. **Print out the array** with the equivalent **binary value**. The arrays are printed backwards to correctly form the calculated binary value the decimal number.

***N|B:***

* Provided in the folder “**Part 2a Demonstration**” is the demonstration for the working of this program
* The program consists of **comments** which seek to explain what is happening at each step

**Part 2b**

1. **Snapshots of the output (Can also be found in the “Part 2b Snapshots” Folder)**

* Kindly check in the folder “**Part 2b Snapshots”** which has been included in the zipped file

1. **Code explanation and short demonstration**

* For this part, we were to create a program which **randomly generates 30 floating point numbers** and calculate their **equivalent binary values** and provide a **remark** about the binary values (either “**Approximate” or “Exact**”) and store the results in a table format provided.
* The created program was implemented using lops which do the following:

1. **List the numbering from 1 to 30** for each of the random numbers generated. Each random number will be given an **identifying index from 1 to 30**, to identify the start and end points
2. **Generate the 30 random floating numbers**. Since we are required to calculate the binary equivalents for **30 randomly generated floating point numbers**, the random numbers will be generated until the **counter reaches 30**, then it will cease to generate anymore random floating numbers. The random numbers are generated between the **range of 0 to 1000**
3. Converting the **integer parts** of the floating numbers to their **equivalent binary**. The program **splits the number into the fraction and integer parts** and calculates the binary equivalent for the **integer part** and **stores it in an array**
4. Converting the **fraction parts** of the floating numbers to their **equivalent binary**. The program splits the number into the fraction and integer parts and calculates the binary equivalent for the **fraction** part and **stores it in an array**.
5. Checking if the binary equivalent of the random floating point has **more or less than 5 decimal places**. For a binary value **with 5 decimal places**, then the remark is set to “**Approximate**” but for a binary value with **less than 5 decimal places**, the remark is set to “**Exact**”.
6. **Printing the array for both the integer and fraction binary equivalents**. The arrays are printed backwards to **correctly form the calculated binary value** the decimal floating number.

***N|B:***

* Provided in the folder “**Part 2b Demonstration**” is the demonstration for the working of this program.
* The program consists of **comments** which seek to explain what is happening at each step.