**Batch: B2 Roll No.: 1611103**

**Experiment / assignment / tutorial No. 5**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE: Implementation of IEEE-754 floating point representation** |

**AIM:** To demonstrate the single and double precision formats to represent floating point numbers.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

CO 2-Detail working of the arithmetic logic unit and its sub modules

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, TataMcGraw-Hill.
2. William Stallings, “Computer Organization and Architecture: Designing for Performance”, Eighth Edition, Pearson.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a [technical standard](https://en.wikipedia.org/wiki/Technical_standard) for [floating-point](https://en.wikipedia.org/wiki/Floating_point) computation established in 1985 by the [Institute of Electrical and Electronics Engineers](https://en.wikipedia.org/wiki/Institute_of_Electrical_and_Electronics_Engineers) (IEEE). The standard [addressed many problems](https://en.wikipedia.org/wiki/Floating_point#IEEE_754_design_rationale) found in the diverse floating point implementations that made them difficult to use reliably and [portably](https://en.wikipedia.org/wiki/Software_portability). Many hardware [floating point units](https://en.wikipedia.org/wiki/Floating_point_unit) now use the IEEE 754 standard.

The standard defines:

* *arithmetic formats:* sets of [binary](https://en.wikipedia.org/wiki/Binary_code) and [decimal](https://en.wikipedia.org/wiki/Decimal) floating-point data, which consist of finite numbers (including [signed zeros](https://en.wikipedia.org/wiki/Signed_zero) and [subnormal numbers](https://en.wikipedia.org/wiki/Subnormal_number)), [infinities](https://en.wikipedia.org/wiki/Infinity), and special "not a number" values ([NaNs](https://en.wikipedia.org/wiki/NaN))
* *interchange formats:* encodings (bit strings) that may be used to exchange floating-point data in an efficient and compact form
* *rounding rules:* properties to be satisfied when rounding numbers during arithmetic and conversions
* *operations:* arithmetic and other operations (such as [trigonometric functions](https://en.wikipedia.org/wiki/Trigonometric_functions)) on arithmetic formats
* *exception handling:* indications of exceptional conditions (such as [division by zero](https://en.wikipedia.org/wiki/Division_by_zero), overflow, *etc*

**Example (Single Precision- 32 bit representation)**

**#include <iostream>**

**#include<stdlib.h>**

**//#include<conio.h>**

**#define MAX 32**

**#define MAX64 64**

**using namespace std;**

**int bic(int a[],int x)**

**{**

**int i = 0;**

**while(x && i<MAX)**

**{**

**a[i++] = x%2;**

**x = x/2;**

**}**

**return i;**

**}**

**int floatB(int b[],float y)**

**{**

**int i = 0;**

**while(y!=0.000000 && i<MAX)**

**{**

**b[i++] = (int)(y\*2);**

**y = (y\*2) - (int)(y\*2);**

**}**

**for(int j = 0;j<i/2;j++)**

**{**

**int temp = b[i-j-1];**

**b[i-j-1] = b[j];**

**b[j] = temp;**

**}**

**return i;**

**}**

**int main()**

**{**

**//clrscr();**

**float x = 0;**

**cout<<"WELCOME TO IEEE CONVERTER:-)\nEnter your number : ";**

**cin>>x;**

**int num[MAX];**

**if(x >=0)**

**{**

**num[0] = 0;**

**}**

**else**

**{**

**num[0] = 1;**

**x = -x;**

**}**

**int exp = (int)x;**

**x = x -(int)x;**

**int EXP[MAX],MANT[MAX];**

**int pow = bic(EXP,exp) - 1;**

**int len = floatB(MANT,x) - 1;**

**/\*pow +=127;\*/int temp = 128; //use pow+=1023 and temp to 1024 for 64 bit**

**if(pow >0 || (pow ==0 && EXP[pow] ==1)) //to insert binary equivalent of exponent in array**

**{**

**pow += 127;**

**for(int i = 1;i<9;i++ ) //i<11 for 64 bit**

**{**

**num[i] = (pow&temp)?1:0;**

**temp/=2;**

**}**

**}**

**else**

**{**

**temp = len;**

**while(temp>=0)**

**{**

**if(MANT[temp--] == 1)**

**{pow = temp - len ;break;}**

**}**

**len += pow;**

**pow+=127;temp = 128;**

**for(int i = 1;i<9;i++ ) //i<11 for 64 bit**

**{**

**num[i] = (pow&temp)?1:0;**

**temp = temp>>1;**

**}**

**}**

**pow -=127;**

**//temp = pow - 1; //setting pointer to get mantissa**

**for(int i = 0;i<pow;i++)**

**{**

**MANT[++len] = EXP[i]; //copy mantissa part**

**}**

**for(int i = 9;i<MAX && len>=0;i++)**

**{**

**num[i] = MANT[len--];**

**temp = i;**

**}**

**cout<<"GIVEN NO. IN IEEE FORMAT IS : \n";**

**cout<<num[0]<<" ||| ";**

**for(int i = 1;i<9 ;i++)**

**{**

**cout<<num[i]<<" ";**

**}**

**cout<<" ||| ";**

**for(int i = 9;i<=temp;i++)**

**{**

**cout<<num[i]<<" ";**

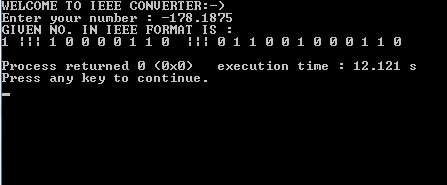
**}**

**cout<<"0\n";**

**//getch();**

**return 0;**

**}**

****

**Example (Double Precision- 64 bit representation)**

**#include <iostream>**

**#include<stdlib.h>**

**//#include<conio.h>**

**#define MAX 64**

**using namespace std;**

**int bic(int a[],int x)**

**{**

**int i = 0;**

**while(x && i<MAX)**

**{**

**a[i++] = x%2;**

**x = x/2;**

**}**

**return i;**

**}**

**int floatB(int b[],float y)**

**{**

**int i = 0;**

**while(y!=0.000000 && i<MAX)**

**{**

**b[i++] = (int)(y\*2);**

**y = (y\*2) - (int)(y\*2);**

**}**

**for(int j = 0;j<i/2;j++)**

**{**

**int temp = b[i-j-1];**

**b[i-j-1] = b[j];**

**b[j] = temp;**

**}**

**return i;**

**}**

**int main()**

**{**

**//clrscr();**

**float x = 0;**

**cout<<"WELCOME TO IEEE CONVERTER:-)\nEnter your number : ";**

**cin>>x;**

**int num[MAX];**

**if(x >=0)**

**{**

**num[0] = 0;**

**}**

**else**

**{**

**num[0] = 1;**

**x = -x;**

**}**

**int exp = (int)x;**

**x = x -(int)x;**

**int EXP[MAX],MANT[MAX];**

**int pow = bic(EXP,exp) - 1;**

**int len = floatB(MANT,x) - 1;**

**/\*pow +=127;\*/int temp = 1024; //use pow+=1023 and temp to 1024 for 64 bit**

**if(pow >0 || (pow ==0 && EXP[pow] ==1)) //to insert binary equivalent of exponent in array**

**{**

**pow += 1023;**

**for(int i = 1;i<12;i++ ) //i<11 for 64 bit**

**{**

**num[i] = (pow&temp)?1:0;**

**temp/=2;**

**}**

**}**

**else**

**{**

**temp = len;**

**while(temp>=0)**

**{**

**if(MANT[temp--] == 1)**

**{pow = temp - len ;break;}**

**}**

**len += pow;**

**pow+=1023;temp = 1024;**

**for(int i = 1;i<12;i++ ) //i<11 for 64 bit**

**{**

**num[i] = (pow&temp)?1:0;**

**temp = temp>>1;**

**}**

**}**

**pow -=1023;**

**//temp = pow - 1; //setting pointer to get mantissa**

**for(int i = 0;i<pow;i++)**

**{**

**MANT[++len] = EXP[i]; //copy mantissa part**

**}**

**for(int i = 12;i<MAX && len>=0;i++)**

**{**

**num[i] = MANT[len--];**

**temp = i;**

**}**

**cout<<"GIVEN NO. IN IEEE FORMAT IS : \n";**

**cout<<num[0]<<" ||| ";**

**for(int i = 1;i<12 ;i++)**

**{**

**cout<<num[i]<<" ";**

**}**

**cout<<" ||| ";**

**for(int i = 12;i<=temp;i++)**

**{**

**cout<<num[i]<<" ";**

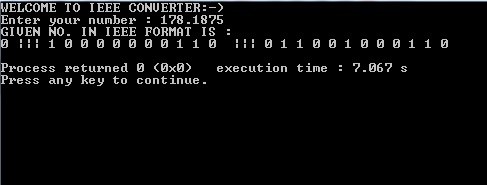
**}**

**cout<<"0\n";**

**//getch();**

**return 0;**

**}**

****

**Post Lab Descriptive Questions (Add questions from examination point view)**

1. **Give the importance of IEEE-754 representation for floating point numbers?**

**Ans. IEEE-754 provides a proper algorithm for arithmetic operations on floating point values. Its standards are as follows:-**

* **arithmetic formats: sets of**[**binary**](https://en.wikipedia.org/wiki/Binary_code)**and**[**decimal**](https://en.wikipedia.org/wiki/Decimal)**floating-point data, which consist of finite numbers (including**[**signed zeros**](https://en.wikipedia.org/wiki/Signed_zero)**and**[**subnormal numbers**](https://en.wikipedia.org/wiki/Subnormal_number)**),**[**infinities**](https://en.wikipedia.org/wiki/Infinity)**, and special "not a number" values (**[**NaNs**](https://en.wikipedia.org/wiki/NaN)**)**
* **interchange formats: encodings (bit strings) that may be used to exchange floating-point data in an efficient and compact form**
* **rounding rules: properties to be satisfied when rounding numbers during arithmetic and conversions**
* **operations: arithmetic and other operations (such as**[**trigonometric functions**](https://en.wikipedia.org/wiki/Trigonometric_functions)**) on arithmetic formats**
* **exception handling: indications of exceptional conditions (such as**[**division by zero**](https://en.wikipedia.org/wiki/Division_by_zero)**, overflow, etc.)**

**Conclusion**

**Hence we have successfully demonstrated the single and double precision formats to represent floating point numbers in C programming language.**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**