**Batch: B3 Roll No.: 121**

**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: (Mention CO attained here)**

CO1 – Describe and define the structure of a computer with buses structure and 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.

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

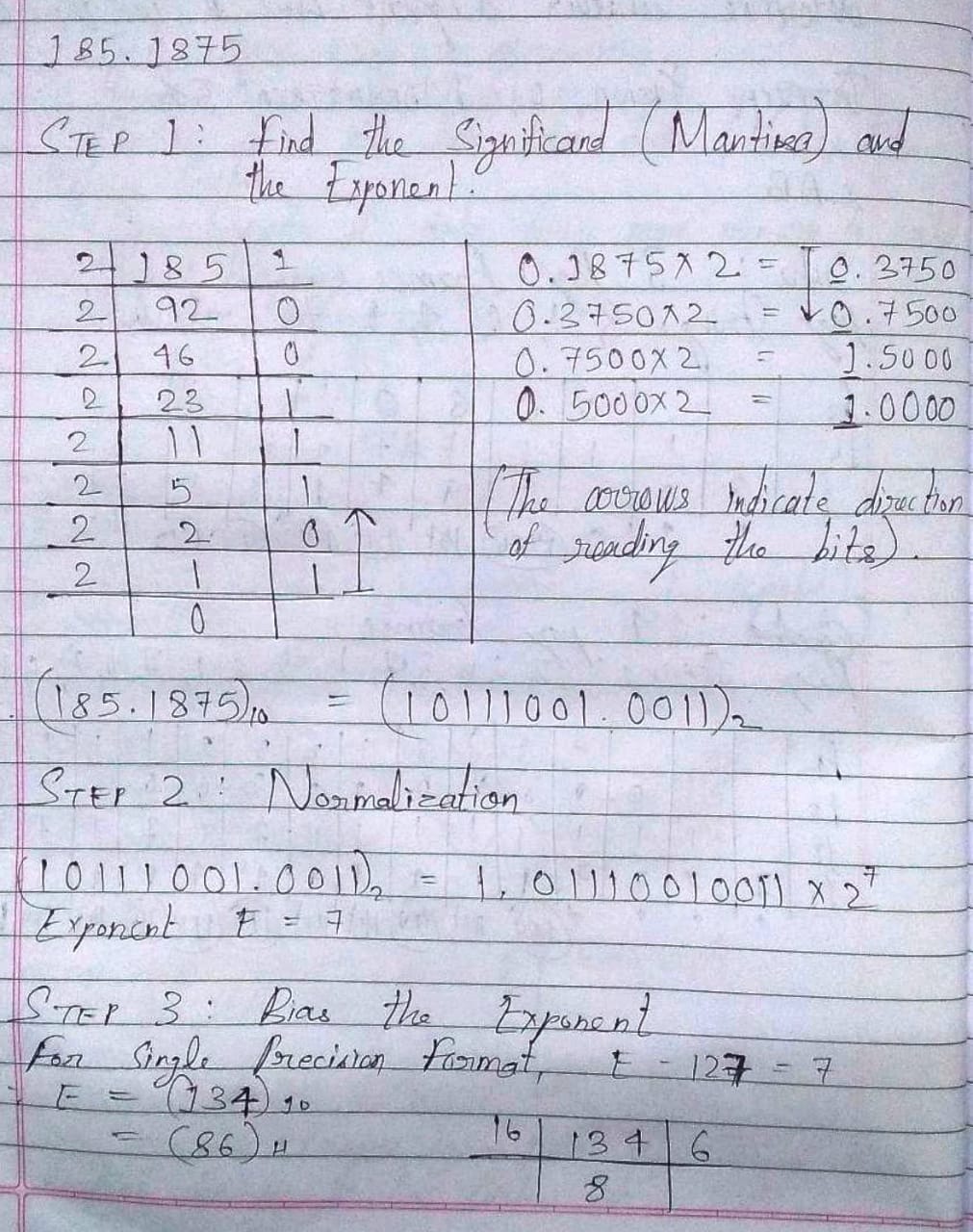
**Pre Lab/ Prior Concepts:**

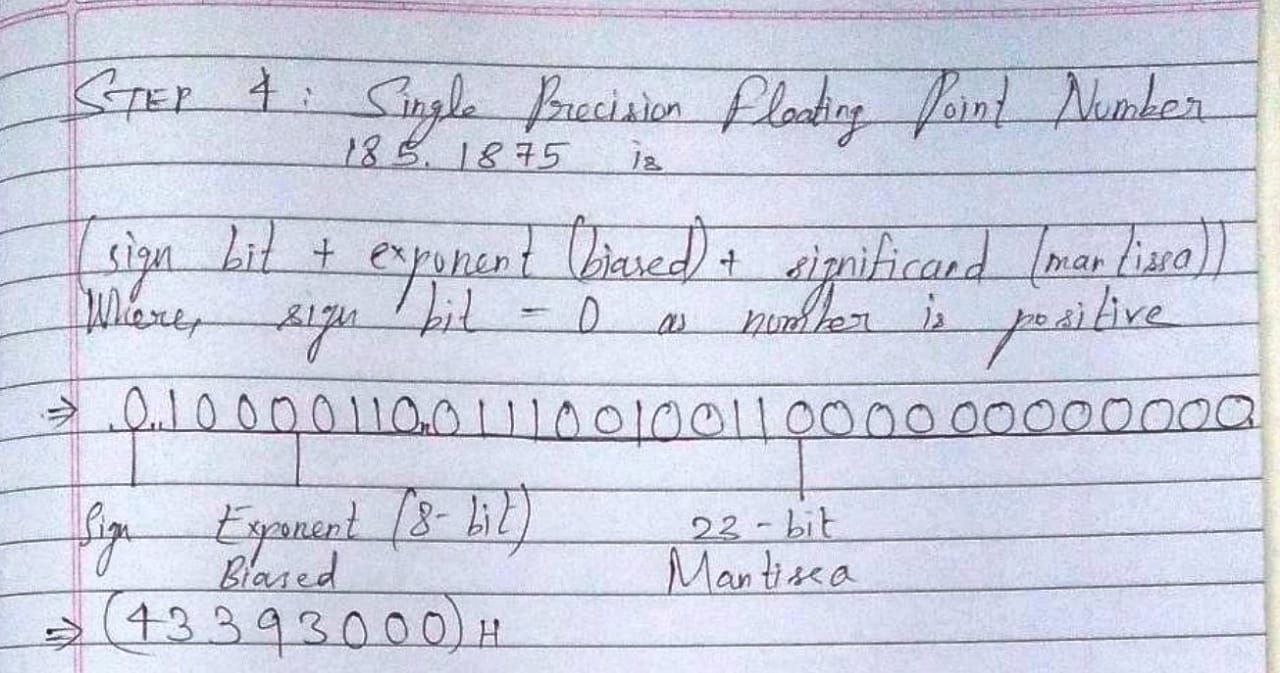
The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a [technical standard](https://en.wikipedia.org/wiki/Technical_standard) for [floating-point](https://en.wikipedia.org/wiki/Floating_point) computation established in 1985 by the [Institute of Electrical and Electronics Engineers](https://en.wikipedia.org/wiki/Institute_of_Electrical_and_Electronics_Engineers) (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*

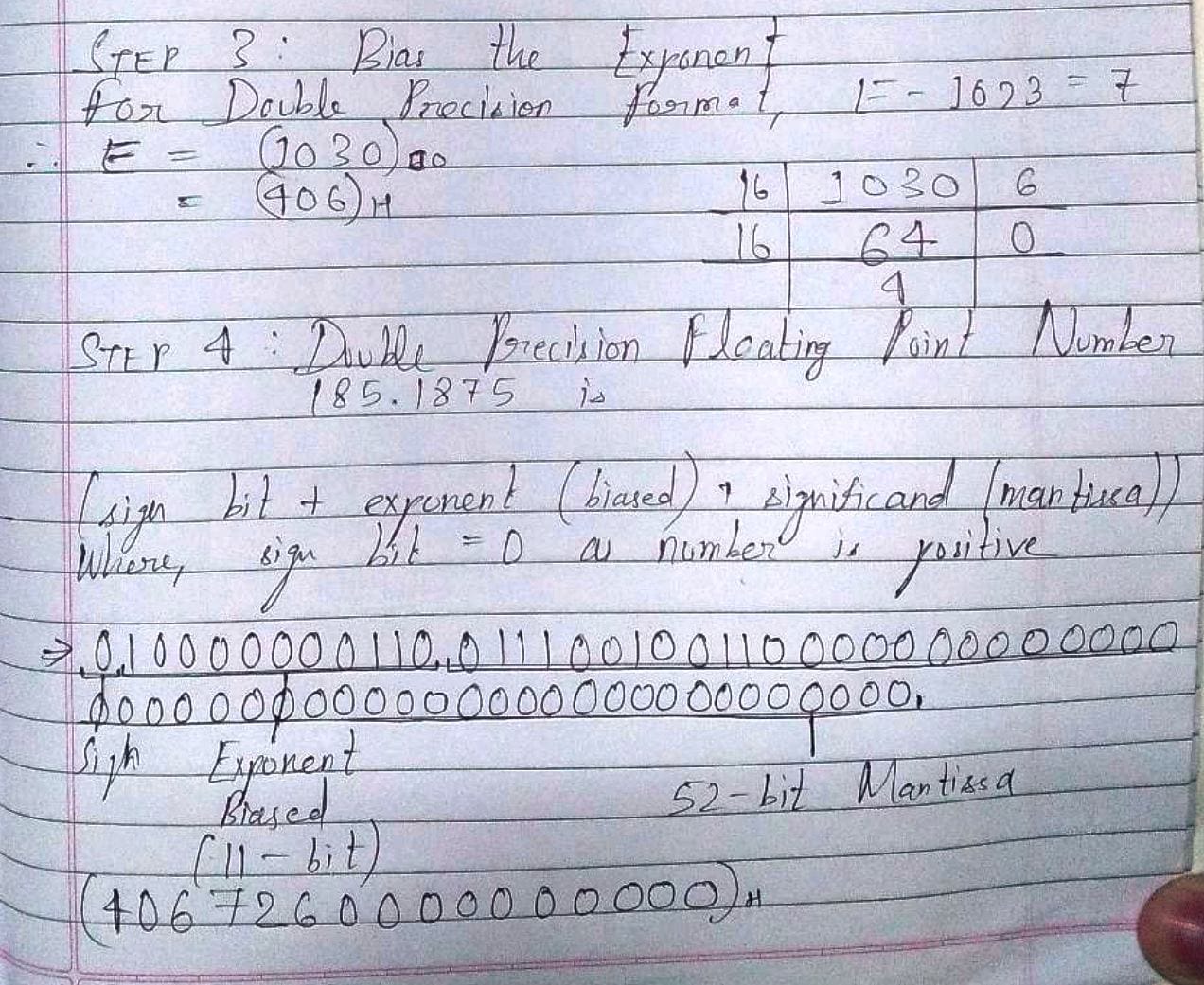
**Example (Single Precision- 32 bit representation )**





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

Steps 1 and 2 are the same as for the Single Precision example.



Code:

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include<string.h>

void binint(int \*b, int len, int max);

void binfrac(int \*b, int len);

int d\_to\_b(int num, int \*b, int bits);

void ieee();

int seperate(char val[], int \*int\_part, float \*frac\_part);

int frac\_conv(float \*frac\_binarr, int \*frac\_bin, int frac\_max);

int normal(int int\_max, int int\_len, int \*int\_bin, int frac\_max, int frac\_len, int \*frac\_bin);

void mantissa(int int\_max, int \*int\_bin, int frac\_len, int \*frac\_bin, int norexp);

int main()

{

ieee();

return 0;

}

void ieee()

{

char in\_num[20] = {'0'};

int int\_part = 0;

float frac\_part = 0;

int frac\_max = 20;

int int\_max = 10;

int int\_len = 0;

int frac\_len = 0;

int int\_bin[10] = {0}, frac\_bin[20] = {0};

float frac\_binarr[21] = {0};

int norexp = 0;

int biexp = 0, biexpsingle[9] = {0}, biexpdoub[11] = {0}, biexpsingle\_len = 0, biexpdoub\_len = 0;

int signBit = 0;

int i, j, temp = 0;

printf("\nIEEE\t754\tFLOATING\tPOINT\tREPRESENTATION");

printf("\nEnter the number whose IEEE 754 equivalent you wish to get: ");

scanf("%s", in\_num);

signBit = seperate(in\_num, &int\_part, &frac\_part);

int\_len = d\_to\_b(int\_part, &int\_bin[0], int\_max);

if(int\_len == 0)

int\_len = 1;

printf("\nThe integer in binary is %d.", int\_part);

binint(&int\_bin[0], int\_len, int\_max);

frac\_binarr[0] = frac\_part;

frac\_len = frac\_conv(&frac\_binarr[0], &frac\_bin[0], frac\_max);

printf("\nThe fraction in binary is %.5f.", frac\_part);

binfrac(&frac\_bin[0], frac\_len);

printf("\nNormalization: ");

binint(&int\_bin[0], int\_len, int\_max);

printf(".");

binfrac(&frac\_bin[0], frac\_len);

norexp = normal(int\_max, int\_len, &int\_bin[0], frac\_max, frac\_len, &frac\_bin[0]);

printf("\n1.");

mantissa(int\_max, &int\_bin[0], frac\_len, &frac\_bin[0], norexp);

printf(" × 2 ^ %d.", norexp);

printf("\nThe normalized exponent is %d.", norexp);

printf("\nBiasing: ");

biexp = 127 + norexp;

biexpsingle\_len = d\_to\_b(biexp, &biexpsingle[0], 9);

printf("\nSingle Precision: ");

printf("\n%d = E - 127", norexp);

printf("\nE = %d = ", biexp);

binint(&biexpsingle[0], biexpsingle\_len, 9);

biexp = 1023 + norexp;

biexpdoub\_len = d\_to\_b(biexp, &biexpdoub[0], 11);

printf("\nDouble Precision: ");

printf("\n%d = E - 1023", norexp);

printf("\nE = %d = ", biexp);

binint(&biexpdoub[0], biexpdoub\_len, 11);

printf("\nDisplay: ");

printf("\nSign Bit\tBiased Exponent\t\tMantissa\n");

printf("\t%d\t | \t", signBit);

binint(&biexpsingle[0], biexpsingle\_len, 9);

printf("\t | ");

mantissa(int\_max, &int\_bin[0], frac\_len, &frac\_bin[0], norexp);

printf("\n");

printf("\t%d\t | \t", signBit);

binint(&biexpdoub[0], biexpdoub\_len, 11);

printf("\t | ");

mantissa(int\_max, &int\_bin[0], frac\_len, &frac\_bin[0], norexp);

printf("\n");

}

int d\_to\_b(int num, int \*b, int bits)

{

int i = bits - 1, len = 0, rem = 0;

while(num >= 1)

{

rem = num % 2;

\*(b + (i--)) = rem;

num = num / 2;

len++;

}

return len;

}

void binint(int \*b, int len, int max)

{

int i, temp = max - len;

for(i = temp; i < max; i++)

{

printf("%d", \*(b+i));

}

}

void binfrac(int \*b, int len)

{

int i;

for(i = 0; i < len; i++)

{

printf("%d", \*(b+i));

}

}

int seperate(char val[], int \*int\_part, float \*frac\_part)

{

int i = 0, flag = 0, sign = 0, dot\_pos;

float frac\_place\_val = 0.1;

int int\_place\_val = 1, digit;

\*frac\_part = 0.0;

\*int\_part = 0;

if(val[0]=='-')

sign = 1;

else

sign = 0;

while(i < strlen(val) && flag == 0)

{

if(val[i] == '.')

{

dot\_pos = i;

flag = 1;

}

i++;

}

for(i = dot\_pos - 1; i >= sign; i--)

{

digit = val[i] - '0';

\*int\_part = (\*int\_part) + digit\*int\_place\_val;

int\_place\_val = int\_place\_val\*10;

}

for(i = dot\_pos + 1; i < strlen(val); i++)

{

digit = val[i] - '0';

\*frac\_part = (\*frac\_part) + digit\*frac\_place\_val;

frac\_place\_val = frac\_place\_val/10;

}

return sign;

}

int frac\_conv(float \*frac\_binarr, int \*frac\_bin, int frac\_max)

{

int i = 1, j, len = 0, flag = 0, temp;

char frac\_string[10] = {'0'};

while(flag == 0 && i <= frac\_max)

{

frac\_binarr[i] = frac\_binarr[i - 1]\*2.00;

if(frac\_binarr[i] > 1.00)

{

frac\_bin[i - 1] = 1;

gcvt(frac\_binarr[i], 6, frac\_string);

seperate(frac\_string, &temp, &frac\_binarr[i]);

}

else if(frac\_binarr[i] == 1.00)

{

frac\_bin[i - 1] = 1;

}

else

{

frac\_bin[i - 1] = 0;

}

len++;

if(frac\_binarr[i] == 0.00 || frac\_binarr[i] == 1.00)

{

flag = 1;

}

else

{

j = 0;

{

if(frac\_binarr[i] == frac\_binarr[j])

{

flag = 1;

}

j++;

}

}

i++;

}

return len;

}

int normal(int int\_max, int int\_len, int \*int\_bin, int frac\_max, int frac\_len, int \*frac\_bin)

{

int i, temp\_len = 0, temp = 0, flag = 0, norexp = 0;

temp\_len = int\_max - int\_len;

temp = int\_len;

flag = 0;

for(i = temp\_len; i < int\_max; i++)

{

if(int\_bin[i] == 1)

{

flag = 1;

temp--;

break;

}

else temp--;

}

if(flag == 1)

{

norexp = temp;

}

else

{

temp = 1;

for(i = 0; i < frac\_len; i++)

{

if(frac\_bin[i] == 1)

{

break;

}

else temp++;

}

norexp=-temp;

}

return norexp;

}

void mantissa(int int\_max, int \*int\_bin, int frac\_len, int \*frac\_bin, int norexp)

{

int i;

if(norexp > 0)

{

binint(&int\_bin[0], norexp, int\_max);

if((norexp+frac\_len) > 23)

frac\_len = 23 - norexp;

binfrac(&frac\_bin[0], frac\_len);

}

else

{

for(i = abs(norexp); i < frac\_len; i++)

{

printf("%d", frac\_bin[i]);

}

if(abs(norexp) == frac\_len)

{

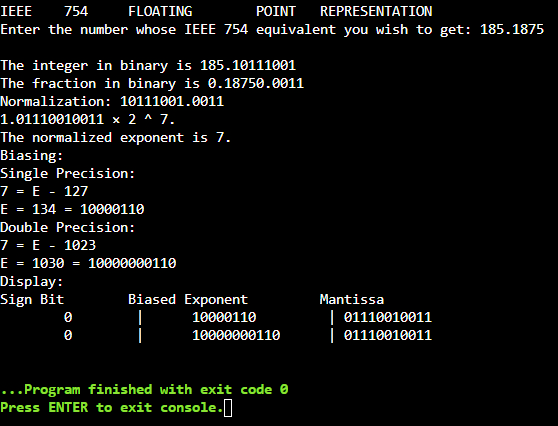
printf("0");

}

}

}

Output:



**Post Lab Descriptive Questions**

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

**Ans.** The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point computation which was established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE). The standard addressed many problems found in the diverse floating point implementations that made them difficult to use reliably and reduced their portability. IEEE Standard 754 floating point is the most common representation today for real numbers on computers, including Intel-based PC’s, Macs and most Unix platforms.

**Conclusion**

Thus, in this experiment, the concept of IEEE-754 floating point representation has been learnt and implemented in C language. It is a widely used and accurate representation for floating-point numbers.

**Date: \_\_13-11-22\_\_\_ Signature of faculty in-charge**