Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 01, January-2023

Basic Implementation of Fixed-Point Arithmetic in Numerical Analysis

M.A. Sandoval-Hernández[1], G.C. Velez-López[2], H. Vázquez-Leal[3,4,*], U.A. Filobello-Nino[3], G. J. Morales-Alarcón[5], E. De-Leo-Baquero[1], A.C. Bielma-Pérez[1], C.E Sampieri-González[3], J.E. Pérez-Jácome Friscione[3], A.D. Contreras-Hernández[3], O. Álvarez-Gasca[3], J. Sánchez-Orea[3], L. Cuellar-Hernández[3].

[1] Centro de Bachillerato Tecnológico Industrial y de Servicios No. 190, Av. 15 Col. Venustiano Carranza 2da Sección, Boca del Río, 94297, Veracruz, México.

[2]Instituto Nacional de Astrofísica, Óptica y Electrónica, Luis Enrique Erro 1, Sta. María Tonantzintla, 72840, Puebla, México.

[3]Facultad de Instrumentación Electrónica, Universidad Veracruzana, Circuito Gonzalo Aguirre Beltrán S/N, Xalapa, 91000, Veracruz, México.

[4]Consejo Veracruzano de Investigación Científica y Desarrollo Tecnológico, Av. Rafael Murillo Vidal No. 1735, Cuauhtémoc, Xalapa, 91069, Veracruz, México.

[5] Instituto de Psicología y Educación, Universidad Veracruzana, Agustín Melgar 2, Col. 21 de Marzo, Xalapa, 91010 Veracruz, México.

***Abstract*— In this paper, we present a teaching aid for the implementation of fixed-point arithmetic in numerical analysis algorithms using the C/C++ programming language. Through the examination of four case studies, including the solution of a system of equations through LU factorization, determination of the definite integral through numerical integration, determination of a root through the Newton-Raphson method, and the evaluation of a polynomial of order 9, we demonstrate the advantage of fixed-point arithmetic in reducing computation times when dealing with complex numerical calculations. In particular, the case study utilizing the Newton-Raphson method illustrates the potential for a significant reduction in computation time of more than 100 times in comparison to the use of floating point arithmetic. This makes the implementation of these algorithms in embedded systems, where a math coprocessor is not present a viable option.**

Keywords*— Fixed-point, computing time, float point, numerical analysis, C language.

I. INTRODUCTION

In fixed-point arithmetic, operations such as division and multiplication are carry out with bit shifts and are treated as any normal signed or unsigned integer. This arithmetic is very important when there is no math co-processor. For example, pic microcontrollers do not have the necessary hardware to handle fixed-point arithmetic. The advantage is the speed that is obtained when handling data in this format, however accuracy is lost [1].

Fixed-point arithmetic (FP) is used in applications that do not require or cannot have a floating-point arithmetic unit [2]. The floating point arithmetic system has disadvantages because the operations require higher energy consumption but with lower speed of operation compared to the arithmetic fixed point [3]. Small processors and low cost embedded systems they do not have a unit for handling floating point [4].

For example Fixed point arithmetic it is widely used in Digital Signal Processing (DSP) [5]. Fixed-point arithmetic's can be carried out using the hardware built for integer arithmetic. In [6] a division-free algorithm was presented for exponential function using Newton-Raphson improving computational speed. In [7] the fixed point calculation is presented to evaluate logarithms, it introduces variants of the logarithm that inputs a floating point number and outputs the fixed point result. In [8] a method for the rapid division of integers in software was presented, for its implementation in processors with an integrated hardware multiplier. It is based on determining the reciprocal of the divisor with Newton-Raphson, with 16 fixed-point bits. In [9] the architecture of a fixed-point arithmetic unit based on the use of integer arithmetic operations in a field-programmable gate array (FPGA) was proposed. Likewise, [10] presents the user guide for a package for fixed-point arithmetic implemented in VHDL (Very High Speed Integrated Circuit).

This paper is organized as follows. In Section II, we introduce the basics fixed-point concept. Implementations of four case studies are presented. In section III presents four case studies where a fixed point is applied; the first one, decomposition to solve a system of 6x6 simultaneous equations; the second one, determination of a definite integral using Simpson's rule; the third one, the implementation for Newton-Raphson; and finally, the fourth one, the evaluation of a polynomial. The discussion is presented in section IV. Finally, a concluding remark is given in Section V.

II. SOME BASICS FIXED-POINT

In the implementation of fixed-point arithmetic, fixed numbers are treated as integers, but the programmer must keep in mind that root point tracking must be carry out during each operation [11,12,13]. Figure 1 shows the point radix for $N = 8$ bit with $Q3.4$ format. It has 3 bits for integer part and 4 bits for fractional part and 1 bit for sign.

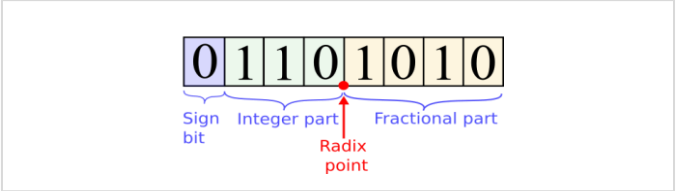

Figure 1. Fixed point representation for $Q3,4$.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 01, January-2023

In fixed point operations, the following aspects have to be considered

1) The number has a sign or unsigned.

2) The position of the radix point in signed numbers in relation to sign bit. For unsigned numbers, the position of the radix point relative to the most significant bit. The bits of numbers of the fractional part to be stored.

The Q notation is used to represent fixed point numbers that are given as $Qm.n$ where m bits for the integer part n bits for the fractional part. Total number of bits $N = m + n + 1$ for signed numbers.

To obtain the value of $N-$ bit number in $Qm.n$ format, it can be calculated by

$$\text{value} = -b_{N-1}2^m + \sum_{i=0}^{N-2} b_i 2^{i-n} \tag{1}$$

Arithmetic operations we considered i_1 and f_1 as the integer parts of the first number and i_2 and f_2 the integer parts of the second one. In addition and subtraction require that the numbers are aligned, the same numbers of bits in the integer part and in the fractional part.

In the multiplication of two numbers, one with n bits, and another with m bits, we obtain a number of $n + m$. Also the size of the integer and fractional parts are the sum of the integer and fractional parts of both numbers. i.e. $i_1 + i_2$ and $f_1 + f_2$ respectively [13].

In the case of division, the results can be reduced to following cases.

1) The quotient of dividing two unsigned fixed point number is given by a number with an integer part $i_1 + f_2$ and a fractional part $f_1 + f_2$.

2) The quotient of dividing a number with sign and other unsigned, we obtain a number with an integer part $i_1 + f_2$ and a fractional part $f_1 - f_2$.

3) The quotient of dividing two signed numbers gives a numbers with an integer part $i_1 + f_2 + 1$ and a fractional part $f_1 - f_2$.

We must be careful when obtaining $f_2 > f_1$, which leads to a negative fractional part. To avoid this, the dividend can be shifted to the left. In this way, we can have available at least as many fractional bits as the divisor. This leads to the next rule: if $f_2 > f_1$ then convert divisor to i_1, x, where $x \leq f_2$ [11, 12, 13]. To convert from floating-point to fixed point we follow these steps

- Multiply the floating point number by 2 raised to the number of desired fractional bits.
- Round the number to the nearest whole number.
- If the number is negative take twos complement of the value arrived at step 2. Store the rounded x in an integer container.

In practice fixed point is programmed in some programming languages such as C, C++ [14]. This is usually done through macros that are defined at the begin of the program [14], [15].

In the appendix is presented the code in Maple that shows the use of (1) to understand how fixed-point arithmetic is implemented.

III. APPLICATION OF FIXED POINT

A. Case study 1: LU Decomposition to solve a system of 6x6 simultaneous equations.

In this case study we will solve a system 6x6 of linear equations through the LU factorization [16,17], given for (5)

$$\begin{bmatrix} 5 & 1 & 1 & 2 & 5 & 2 \\ 1 & 4 & 6 & 2 & 4 & 3 \\ 1 & 2 & 2 & 2 & 3 & 1 \\ 1 & 1 & 2 & 1 & 4 & 3 \\ 2 & 3 & 1 & 5 & 3 & 5 \\ 4 & 5 & 3 & 5 & 3 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 9 \\ 10 \\ 2 \\ 6 \end{bmatrix} \tag{2}$$

The implementation of (5) in fixed point is given

```
//////////// Fixed point
printf("start LU at Fixed point: \n");
auto start = std::chrono::steady_clock::now();
for(h=0;h<10000;h++)
 {
  for(i=0;i<n;i++)
    for(j=0;j<n;j++)
      if(i>j)
        {
          u[i][j]=floatfix(0);
          u_f[i][j]=floatflo(u[i][j]);
        }
      else if(i==j)
        {
          l[i][j]=floatfix(1);
          l_f[i][j]=floatflo(l[i][j]);
        }
      else
        {
          l[i][j]=floatfix(0);
          l_f[i][j]=floatflo(l[i][j]);
        }
 for(i=0;i<n;i++)
    {
      for(j=0;j<n;j++)
        {
         sum=floatfix(0);
         if(i<=j)
           {
            for(k=0;k<n;k++)
              {
               if(k!=i)
                  sum=sum+FMUL(l[i][k],
                      u[k][j]);
                u[i][j]=a[i][j]-sum;
                u_f[i][j]=floatflo(u[i][j]);
              }
           }
         else
           {
            for(k=0;k<n;k++)
              {
               if(k!=j)
                   sum=sum+FMUL(l[i][k],
                   u[k][j]);
                [i][j]=FDIV((a[i][j]-sum),u[j][j]);
                l_f[i][j]=floatflo(l[i][j]);
              }
           }
        }

       y[0]=FDIV(b[0],l[0][0]);
       y_f[0]=floatflo(y[0]);
       for(i=1;i<n;i++)
         {
          sum=floatfix(0);
          for(j=0;j<i;j++)
            {
             sum=sum+FMUL(y[j],l[i][j]);
            }
          y[i]=b[i]-sum;
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 01, January-2023

```
          y_f[i]=floatflo(y[i]);
         }
         x[n-1] = (y[n-1]);
         x[n-1] = FDIV((x[n-1]), (u[n-1][n-1]));
         x_f[n-1]=floatflo(x[n-1]);
         for(i=n-2;i>=0;i--)
           {
            sum=floatfix(0);
            for(j=n-1;j>i;j--)
              sum = sum + FMUL(x[j],u[i][j]);
            x[i] = (y[i]-sum);
            x[i] = FDIV((x[i]),u[i][i]);
            x_f[i]=floatflo(x[i]);
           }
    }
     auto end = chrono::steady_clock::now();

     printf("The solution are:\n ");
     printf("Y            X \n");
     for(i=0;i<n;i++)
     {
      printf("%5.6f\t  %5.6f\t\n",y_f[i],x_f[i]);
     }

auto elapsed =
chrono::duration_cast<chrono::microseconds>(end - start);
cout << "Chrono time " << elapsed.count() << "us." <<endl;
```

B. *Case study 2: Determine the definite integral with Simpson's Rule 1/3.*

In this case study we will calculate the numerical integral by means of Simpson's Rule 1/3 [16,17] for the function given by

$$f(x) = 6.0333x^2 + 2.0333. \tag{3}$$

The fixed-point implementation for (3) is given for

```
//Coeficients
const1_fix=floatfix(6.0333);
const2_fix=floatfix(2.0333);

// function (3)
f_fix = FMUL(FMUL(X, X),const1_fix)+  const2_fix
```

C. *Case study 3: Determine the numerical root with Newton-Raphson*

In the first case we will implemented fixed point to determine the roots of the function using Newton-Raphson [16,17].

$$f(x) = 0.3222x^5 - 1.678662x^3 - 1.559448x. \tag{4}$$

The fixed-point implementation for (2) and its derivative are given by

```
//constants
const1=floatfix(0.3222);
const2=floatfix(1.678662);
const3=floatfix(1.559448);
const4=floatfix(1.6110);
const5=floatfix(5.035986);

/* Defining equation (2)
f0_fix=    FMUL(FMUL(FMUL(FMUL(FMUL(X,X), X),X),X),const1) -
         FMUL(FMUL(FMUL(X,X), X),const2)    +FMUL(X,const3);

/* Derivative of (2) fd = 1.6110x^4 + 5.035986x^2 - 1.559448 */
fd_fix= FMUL(FMUL(FMUL(FMUL(X,X), X),X),const4) -
FMUL(FMUL(X,X),const5) +const3;
```

D. *Case study 4: Polynomial evaluation.*

In this case study we are going to evaluate the function given by (4) in the interval $0 \leq x \leq 1$ which was obtained in [18].

$$\begin{aligned} f(x) = {} & 3.173281398136x - 1.70x^2 - 1.798192792278x^3 \\ & - 0.944876267833x^4 + 0.317406667211x^5 \\ & + 1.081238617931x^6 + 0.847127841554x^7 \\ & - 0.083371054338x^8 - 7.85x^9. \end{aligned} \tag{5}$$

The implementation in fixed point for (4) is

```
// Constants
const1_fix=floatfix(3.173281398136);
const2_fix=floatfix(1.70);
const3_fix=floatfix(1.798192792278);
const4_fix=floatfix(0.944876267833);
const5_fix=floatfix(0.317406667211);
const6_fix=floatfix(1.081238617931);
const7_fix=floatfix(0.847127841554);
const8_fix=floatfix(0.083371054338);
const9_fix=floatfix(7.85);

// Equation (4)
y_fix=FMUL(const1_fix,x_fix)-
FMUL(FMUL(const2_fix,x_fix),x_fix)                              -
FMUL(FMUL(FMUL(const2_fix,x_fix),x_fix),x_fix)                  -
FMUL(FMUL(FMUL(FMUL(const4_fix,x_fix),x_fix),x_fix),x_fix) +
FMUL(
FMUL(FMUL(FMUL(FMUL(const5_fix,x_fix),x_fix),x_fix),x_fix),
x_fix)                          +                   FMUL(FMUL(
FMUL(FMUL(FMUL(FMUL(const6_fix,x_fix),x_fix),x_fix),x_fix),
x_fix),x_fix)+                                  FMUL(FMUL(FMUL(
FMUL(FMUL(FMUL(FMUL(const7_fix,x_fix),x_fix),x_fix),x_fix),
x_fix),x_fix),x_fix)          -          FMUL(FMUL(FMUL(FMUL(
FMUL(FMUL(FMUL(FMUL(const8_fix,x_fix),x_fix),x_fix),x_fix),
x_fix),x_fix),x_fix),x_fix)-            FMUL(FMUL(FMUL(FMUL(
FMUL(FMUL(FMUL(FMUL(const9_fix,x_fix),x_fix),x_fix),x_fix),
x_fix),x_fix),x_fix),x_fix);
```

IV. DISCUSION

The computer used for the simulations was an Intel Core Pentium(R) Intel® Core™ i7-7700 CPU @ running at 3.60GHz × 8 under Linux Ubuntu 18.04.6 LTS, using as compiler the gcc 7.5.0 with level 3 of optimization. In all case studies, the chrono library of the C++ language was used to compare computing times in floating point and fixed point implementations. The macros used in the executed routines of all the study cases [15] are

```
#define FMUL(a,b)  (((a)*(b))>>(FIXED_FRACBITS))
#define FDIV(a,b)  (((a)<<(FIXED_FRACBITS))/(b))
```

In the appendix, the complete code in C++ for case study 3 is presented. In this code, the macros that are used for different operations with fixed-point arithmetic have been implemented.

In the first case study, we propose to solve a system of six linear equations with six unknowns using the LU factorization method. Table I presents a comparison of the solution obtained from this system using floating point and fixed point representations. It is observed that the computing time for the floating point representation is shorter when compared to the fixed point representation.

TABLE I. COMPARATIVE ANALYSIS OF COMPUTATION TIMES FOR CASE STUDY 1

	Exact	FP
x_1	-4.131182	-4.261719
x_2	19.395699	19.867188

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 01, January-2023

x_3	-13.305374	-13.660156
x_4	-11.965590	-12.312500
x_5	8.266665	8.496094
x_6	0.081719	0.125000
Time	0.99μs	72.227μs

In the second case study, the definite integral of a quadratic function of the second degree (3) with float point coefficients was calculated using the Simpson 1/3 method. The function was implemented in a floating point format and a total of 1,000,000 iterations were conducted in the study.

Table II displays the computation times obtained in the second case study. It can be observed that the computation time using the fixed-point format is shorter than the floating-point format. However, it should be noted that an absolute error of 0.1924 was present in the fixed-point implementation.

TABLE II. COMPARATIVE ANALYSIS OF COMPUTATION TIMES FOR CASE STUDY 2.

	Area	Area in FP	Time ms
Numerical	4.044	-	87
Fixed point	3.851562	986	54

In the third case study, the Newton-Raphson algorithm was utilized to obtain the root of the function (4), see that the function used is of degree 5. The root was calculated by implementing the algorithm in both floating point and fixed point formats. The number of iterations required to obtain the root was found to be 9 in both implementations. An initial value of $x = 6$ was utilized, with an acceptable error tolerance of 0.1. A total of 100,000 iterations were conducted in the study.

Table III presents the results obtained from the root calculations using both implementations of the Newton-Raphson algorithm. It can be observed that an error tolerance of approximately 0.01 was achieved for both implementations. Additionally, it is evident that the computation time for the fixed-point implementation was significantly less than that of the floating-point implementation.

TABLE III. COMPARATIVE ANALYSIS OF COMPUTATION TIMES FOR CASE STUDY 3.

	Steps	Root	Root in FP	Time μs
Numerical	9	2.003563	-	17192
Fixed point	9	2.013672	1031	136

The fourth case study evaluated a function of degree nine, as previously published in [18], within the interval $0 \leq x \leq 1$ with increments of 0.01. The procedure was conducted 1000 times and the results indicate that the computation time in floating-point format was 41 milliseconds, while it was significantly shorter at 1 millisecond using the fixed-point format. Table IV presents a selection of y values obtained with increments of $x = 0.05$ to demonstrate the accuracy achieved with the float-point implementation.

TABLE IV. COMPARATIVE ANALYSIS OF COMPUTATION TIMES FOR CASE STUDY 4.

Value x	Value y	Value y obtained with FP
0	0	0
0.05	0.153556	0.15332
0.1	0.297191	0.296387
0.15	0.429377	0.429199
0.2	0.548501	0.547852
0.25	0.652919	0.652344
0.3	0.741019	0.742188
0.35	0.811315	0.813477
0.4	0.86254	0.867676
0.45	0.8937565	0.900391
0.5	0.904447	0.914062
0.55	0.894607	0.908203
0.6	0.864772	0.881348
0.65	0.815991	0.835938
0.7	0.749705	0.77002
0.75	0.667469	0.690918
0.8	0.570507	0.597168
0.85	0.459009	0.487305
0.9	0.331136	0.369629
0.95	0.181636	0.241699
1	-4.84E-07	0.103027

Figure 2 illustrates the absolute error present when utilizing the Fixed Point Implementation within the interval $0 \leq x \leq 1$. It is observed that the absolute error increases as the value of x approaches 1. This phenomenon is attributed to the increase in numerical values of x, which in turn leads to an increase in the numerical values generated in each of the terms in (9) as a result of the multiplications of each coefficient with the corresponding power of x. Furthermore, the computation accuracy diminishes as the error is a function of the number of bits used to represent the information. To mitigate numerical errors, an increase in the number of bits in the fractional part would be necessary.

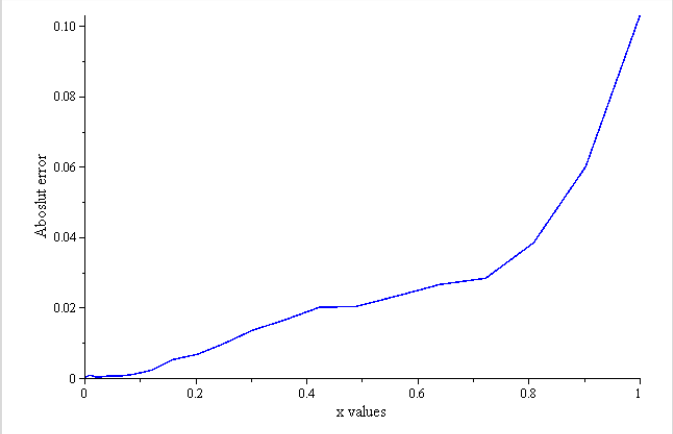

Figure 2. Absolut error for case study IV.

In the present study, the performance of fixed point and floating point implementations were compared across multiple case studies. It is worth noting that the computer utilized in these experiments was equipped with a mathematical coprocessor capable of performing floating point operations. The results indicate that in situations where the number of float-point arithmetic operations is relatively low, the float-point implementation tends to exhibit superior performance as demonstrated in Table I of the first case study, where operations such as multiplication, division, addition, and subtraction were performed. However, as the complexity of the arithmetic calculations increases, the computing times in float-point operations also increase. This phenomenon is observed in case study 2, as presented in Table II. Specifically, in this case the expression being evaluated was a second degree function with

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 01, January-2023

constant coefficients. It is important to note that the use of fixed-point arithmetic in this scenario resulted in a reduction of computation time, as well as an acceptable error level.

In the third case study, a fifth-degree function was evaluated using the Newton-Raphson method to determine one of its roots. The results indicate that the computation time using a fixed point was significantly lower compared to the use of floating point. This trend continues in the fourth case study, where a polynomial of a different degree was evaluated, with a notable reduction in computation time when using a fixed point approach. However, it is important to note that in all of the case studies, an error was observed in the implementation of fixed point algorithms. This highlights the need for careful consideration and implementation in order to effectively use fixed point in numerical analysis. Additionally, it is important to note that the use of fixed point may not always be the most appropriate method and that other techniques, such as floating point, should also be considered in the analysis.

However, it is important to note that in all the case studies presented in this article, an error occurred when implementing a fixed point. Despite this, the results of the case studies show that the use of fixed point can provide significant performance gains in certain situations. As such, it is crucial to carefully evaluate the trade-offs between computation time and numerical accuracy when deciding to implement fixed point in numerical analysis algorithms. The results obtained show a reduction in computation times, for which fixed-point arithmetic is useful and applicable to embedded systems, especially those that lack a mathematical processor. Furthermore, the article presented a teaching aid for the implementation of fixed point in numerical analysis algorithms such as the Newton-Raphson method, where a speed increase of more than 100 times was achieved compared to that obtained with float-point. In this work the computing times were measured using the C language, however it is possible to measure computing time using Fortran [19-22].

V. CONCLUDING REMARKS

In this study, the implementation of fixed point arithmetic in four numerical analysis algorithms was presented: the solution of a system of equations through LU factorization, the determination of a definite integral through numerical integration, the determination of a root through the Newton-Raphson method, and the evaluation of a ninth-order polynomial. The results of these case studies demonstrate the advantages of fixed point implementation in terms of computation time when dealing with complex numerical calculations. The aim of this article is to provide readers with a clear understanding of fixed point implementation through simple and reproducible examples. As a future research direction, it would be beneficial to implement these algorithms on embedded systems, such as PIC microcontrollers, to further evaluate the advantages of fixed point implementation in real-world applications.

DECLARATION OF INTERESTS STATEMENT

The authors declare that there are no conflicts of interest regarding the publication of this paper.

ACKNOWLEDGMENTS

Authors would like to thank Roberto Ruiz Gomez for his contribution to this project. The authors are grateful to the anonymous referee for a careful checking of details and helpful comments that improved this paper.

APPENDIX

A. *Code 1. Program in Maple*

```
#Code written by the authors
#Code Maple for understand Fixed Point
#The follow code presents a method for understanding fixed-
point numbers in Qm.n
restart;
#Let Q(1, 6);
Digits:=9;

#implementation of formula 1, see text. Q1,6;
V := -b[N-1]*2^m+sum(b[i]*2^(i-n), i = 0 .. N-2);
#NN := m+n+1;
NN := 1+6+1;


# Example 1
value := expand(subs(N = 8, m = 1, n = 6, V));
b[7] := 0;
b[6] := 1;
b[5] := 1;
b[4] := 0;
b[3] := 1;
b[2] := 0;
b[1] := 1;
b[0] := 1;
evalf(value);


##################################
#Let Q(1, 6)
#Example 2
value := expand(subs(N = 8, m = 3, n = 4, V));
b[7] := 0;
b[6] := 1;
b[5] := 1;
b[4] := 0;
b[3] := 1;
b[2] := 0;
b[1] := 1;
b[0] := 1;
evalf(value);
```

B. *Code 2. Program in C++*

```
/*Complete code for the implementation in fixed-point for
the study case number 3*/

//Libraries
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<iostream>
#include<chrono>
#include<unistd.h>
using namespace std;


//******* Macros FIXED-POINT ***************//
//////// Fractional bits
#define   FIXED_FRACBITS 11
typedef int fixedp;

// Convert to fixed point
#define shortfix(x)    ((fixedp)((x) << (FIXED_FRACBITS)))
#define intfix(x)      ((fixedp)((x) << (FIXED_FRACBITS)))
#define longfix(x)     ((fixedp)((x) << (FIXED_FRACBITS)))
#define floatfix(x)    ((fixedp)((x) * (1 <<
(FIXED_FRACBITS))))
#define doublefix(x)   ((fixedp)((x) * (1 <<
(FIXED_FRACBITS))))

//// Convert to floating point  //////////////
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 01, January-2023

```
#define shortflo(x)     ((short)((x) >> (FIXED_FRACBITS)))
#define intflo(x)       ((int)((x) >> (FIXED_FRACBITS)))
#define longflo(x)      ((long)((x) >> (FIXED_FRACBITS)))
#define floatflo(x)     ((float)(x) / (1 <<
(FIXED_FRACBITS)))
#define doubleflo(x)    ((double)(x) / (1 <<
(FIXED_FRACBITS)))

////////// Another option ////////////////////////
/* Basic operations between two numbers a and b at a fixed
point in q format returning in q format q */
#define FADD(a,b) ((a)+(b))
#define FSUB(a,b) ((a)-(b))
#define FMUL(a,b) (((a)*(b))>>(FIXED_FRACBITS))
#define FDIV(a,b) (((a)<<(FIXED_FRACBITS))/(b))

// Variables into fixed-point
fixedp x_fix, y_fix, const1_fix, const2_fix, const3_fix,
const4_fix,const5_fix,const6_fix,const7_fix,const8_fix,cons
t9_fix,const10_fix,const11_fix;


int main(void)
 {
   float x=0,y=0;
   int contador;
   x_fix=floatfix(x);
   y_fix=floatfix(y);

   const1_fix=floatfix(3.173281398136);
   const2_fix=floatfix(1.70);
   const3_fix=floatfix(1.798192792278);
   const4_fix=floatfix(0.944876267833);
   const5_fix=floatfix(0.317406667211);
   const6_fix=floatfix(1.081238617931);
   const7_fix=floatfix(0.847127841554);
   const8_fix=floatfix(0.083371054338);
   const9_fix=floatfix(7.85);
   const10_fix=floatfix(1.0);
   const11_fix=floatfix(0.01);

   // Clock
   auto start = chrono::steady_clock::now();

   for(contador=1;contador<1000;contador++)
    {
     for(x_fix=0; x_fix<=const10_fix;
                                 x_fix=x_fix+const11_fix)
       {
        y_fix= FMUL(const1_fix,x_fix)-
FMUL(FMUL(const2_fix,x_fix),x_fix) -
FMUL(FMUL(FMUL(const2_fix,x_fix),x_fix),x_fix) -
FMUL(FMUL(FMUL(FMUL(const4_fix,x_fix),x_fix),x_fix),x_fix)
+ FMUL(
FMUL(FMUL(FMUL(FMUL(const5_fix,x_fix),x_fix),x_fix),x_fix),
x_fix) + FMUL(FMUL(
FMUL(FMUL(FMUL(FMUL(const6_fix,x_fix),x_fix),x_fix),x_fix),
x_fix),x_fix)+ FMUL(FMUL(FMUL(
FMUL(FMUL(FMUL(FMUL(const7_fix,x_fix),x_fix),x_fix),x_fix),
x_fix),x_fix),x_fix) - FMUL(FMUL(FMUL(FMUL(
FMUL(FMUL(FMUL(FMUL(const8_fix,x_fix),x_fix),x_fix),x_fix),
x_fix),x_fix),x_fix),x_fix)- FMUL(FMUL(FMUL(FMUL(
FMUL(FMUL(FMUL(FMUL(const9_fix,x_fix),x_fix),x_fix),x_fix),
x_fix),x_fix),x_fix),x_fix);
       }
     }

   auto end = chrono::steady_clock::now();

    cout << "Elapsed time in ms: "<<
   chrono::duration_cast<chrono::milliseconds>(end -
   start).count() << " ms"<< endl;
   cout << "el valor  usando PFIJO es"<<floatflo(y_fix)
   <<endl;
  return 0;
}
```

REFERENCES

[1] Kraeling, Mark B. "Fixed-point math in time-critical C applications." *Wescon/96*. IEEE, 587-593,1996.

[2] Anton Cervin, “Fix Point Implementation of Control Algorithms”, Lund University. A Graduate Course on Embedded Control Systems – Pisa 8-12 June 2009.

[3] O. Schlösser, "Implementing a C++ Fixed-Point Class for Embedded Systems." 3, 2013.

[4] Ramanathan, S., et al. "Design and implementation of fixed point arithmetic unit." *Int. J. Eng. Res. Appl.* 6.6 (2016): 11-13.

[5] Roman, Kuc. "Introduction to Digital Signal Processing." (1982).

[6] Chang, Chung-Hsien, et al. "A division-free algorithm for fixed-point power exponential function in embedded system." *2013 1st International Conference on Orange Technologies (ICOT)*. IEEE, 223-226, 2013.

[7] J. Le Maire, N. Brunie, F. de Dinechin, J. M. Muller, "Computing floating-point logarithms with fixed-point operations." *2016 IEEE 23nd Symposium on Computer Arithmetic (ARITH)*. IEEE, 2016.

[8] Nikola M. Nenadic, and Svetlana B. Mladenovic. "Fast division on fixed-point DSP processors using Newton-Raphson method." *EUROCON 2005-The International Conference on" Computer as a Tool"*. Vol. 1. IEEE, 2005.

[9] Przybył, Andrzej. "Fixed-point arithmetic unit with a scaling mechanism for FPGA-based embedded systems." *Electronics* 10.10 1164.(2021):

[10] Bishop, David. "Fixed point package user’s guide." *Packages and bodies for the IEEE* (2006): 1076-2008.

[11] Oberstar, Erick L, Fixed-point representation & fractional math, “Oberstar Consulting,” vol. 9 pp. 19, 2007.

[12] Yates, Randy, Fixed-point arithmetic: An introduction, “Digital Signal Labs,” vol. 81, no. 83, pp. 15, 2009.

[13] Pyeatt, Larry and Ughetta, William, ARM 64-Bit Assembly Language, Newnes, 2019.

[14] H. Shildt, Turbo c/c++ 3.1 Manual de referencia 1ed Mcgraw Hill, 1994.

[15] Application Note 33, Fixed Point Arithmetic on the ARM, document number: ARM DAI 0033A, September 1996, https://developer.arm.com/documentation/dai0033/a/

[16] Burden, Richard L., J. Douglas Faires, and Annette M. Burden. *Numerical analysis*. Cengage learning, 2015.

[17] Chapra, Steven C., et al. *Métodos numéricos para ingenieros*. McGraw-Hill, 2011.

[18] H. Vazquez-Leal, M.A. Sandoval-Hernandez, U. A. Filobello-Nino, J. Huerta-Chua. "The novel Leal-polynomials for the multi-expansive approximation of nonlinear differential equations." *Heliyon*, 2020.

[19] M. Sandoval-Hernandez, H. Vazquez-Leal, U. Filobello-Nino, Elisa De-Leo-Baquero, Alexis C. Bielma-Perez, J.C. Vichi-Mendoza, O. Alvarez-Gasca, A.D. Contreras-Hernandez, N. Bagatella-Flores, B.E. Palma-Grayeb, J. Sanchez-Orea, L. Cuellar-Hernandez. The Quadratic Equation and its Numerical Roots." *International Journal of Engineering Research y Technology* 10.6 (2021), 301-305. 2021.

[20] M. A. Sandoval-Hernández, H. Vázquez-Leal, J. Huerta-Chua, F. J. Castro-González, U. A. Filobello-Nino. "Didáctica del graficado de funciones: el caso de las funciones piecewise." *RIDE. Revista Iberoamericana para la Investigación y el Desarrollo Educativo* 12.24 (2022).

[21] H. Vazquez-Leal, Hector, M. A. Sandoval-Hernandez, and U. Filobello-Nino. "The novel family of transcendental Leal-functions with applications to science and engineering." Heliyon 6.11 (2020).

[22] T. Fukushima, "Precise and fast computation of Lambert W-functions without transcendental function evaluations." *Journal of Computational and Applied Mathematics* 244 (2013), 77-89, 2013.