Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по лабораторной работе №4

Эмуляция АЛУ. Операция сложения и вычитания чисел с плавающей точкой

Студент: гр. 853504

Яскевич Никита Николаевич

Руководитель: старший преподаватель

Шиманский В.В.

Минск 2020

СОДЕРЖАНИЕ

1. Введение
2. Постановка задачи
3. Программная реализация
4. Выводы

Литература

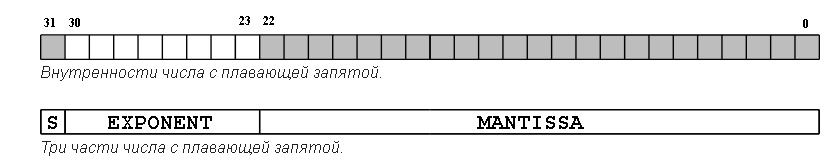
Приложение 1 – Текст программы

1.Введение

### 1.1. **Числа с плавающей запятой**

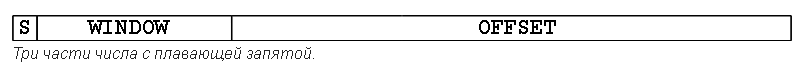
Важно понимать принципы её работы, чтобы полностью осознать её полезность при программировании 3D-движка. В языке C значения с плавающей запятой — это 32-битные контейнеры, соответствующие стандарту IEEE 754. Они предназначены для хранения и выполнения операций над аппроксимациями вещественных чисел. Пока я видел только такое их объяснение. 32 бита разделены на три части:

* S (1 бит) для хранения знака
* E (8 бит) для экспоненты
* M (23 бита) для мантиссы



### **Другой способ представления**

К счастью, их можно объяснить иначе. Воспринимайте экспоненту как окно (Window) или интервал между двумя соседними целыми степенями двойки. Мантиссу воспринимайте как смещение (Offset) в этом окне.



Окно сообщает нам, между какими двумя последовательными степенями двойки будет число: [0,1], [1,2], [2,4], [4,8] и так далее (вплоть до [2^127,2^128]. Смещение разделяет окно на 223=8388608 сегментов. С помощью окна и смещения можно аппроксимировать число. Окно — это отличный механизм защиты от выхода за границы. Достигнув максимума в окне (например, в [2,4]), можно «переплыть» вправо и представить число в пределах следующего окна (например, [4,8]). Ценой этого будет только небольшое снижение точности, потому что окно становится в два раза больше.

1.2. Нормальная и нормализованная формы

*Нормальной формой* числа с плавающей запятой называется такая форма, в которой мантисса (без учёта знака) находится на полуинтервале [0 1), то есть 0 ⩽ a < 1.

Такая форма записи имеет недостаток: некоторые числа записываются неоднозначно (например, 0,0001 можно записать как 0,000001⋅102, 0,00001⋅101, 0,0001⋅100, 0,001⋅10−1, 0,01⋅10−2 и так далее), поэтому распространена (особенно в информатике) также другая форма записи — *нормализованная*, в которой мантисса десятичного числа принимает значения от 1 (включительно) до 10 (исключительно), то есть 1 ⩽ a < 10 (аналогично мантисса двоичного числа принимает значения от 1 до 2). В такой форме любое число (кроме 0) записывается единственным образом. Недостаток заключается в том, что в таком виде невозможно представить 0, поэтому представление чисел в информатике предусматривает специальный признак (бит) для числа 0.

Старший разряд (целая часть числа) мантиссы двоичного числа (кроме 0) в нормализованном виде равен 1 (так называемая *неявная* единица), поэтому при записи мантиссы числа в ЭВМ старший разряд можно не записывать, что и используется в стандарте IEEE 754. В позиционных системах счисления с основанием большим, чем 2 (в [троичной](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F), четверичной и др.), этого свойства нет

**1.3. Алгоритм сложения / вычитания.**

1) Производится выравнивание порядков чисел:

порядок меньшего по модулю числа принимается равным порядку большего числа, а мантисса меньшего сдвигается влево на *S*-ичных разрядов, равных разности порядков.

2) Производится сложение / вычитание мантисс, и получается сумма / разность мантисс.

3) Порядок суммы / разности принимается равным порядку большего числа.

4) Полученный результат нормализуется. Арифметическое действие над порядками и мантиссами выполняется либо отдельными устройствами, либо последовательно одним устройством.

**Операция сложения / вычитания** состоит из следующих **этапов**:

1) прием операндов

2) выравнивание порядков со сдвигом мантисс

3) сложение / вычитание мантисс

4) нормализация результата

**1.4. Этап сложения \ вычитания .**

**При сравнении порядков** **возможны**пять случаев их **соотношения**:

• *Px* – *Py* > *m*, где *m* – число разрядов мантиссы, за результат в этом случае принимается слагаемое *х*, так как при сдвиге мантиссы слагаемого *y* все ее разряды примут нулевое значение.

• *Py* – *Px* > *m*, за результат принимается мантисса *y*.

• *Px* – *Py* = 0, производится суммирование / вычитание мантиссы.

• *Px* – *Py* = *K1* (*K1* < *m*), мантисса *y* сдвигается на *K1* разрядов.

• *Py* – *Px* = *K2* (*K2* < *m*), мантисса *x* сдвигается на *K2* разрядов.

Процесс сдвига мантиссы заключается в следующем: в счетчик циклов из блока обработки порядков заносится число разрядов *K1*или *K2*, на которые ее надо сдвинуть. По мере сдвига мантиссы содержимое счетчика циклов уменьшается, и когда счетчик циклов становится равным нулю, сдвиг прекращается.

Полученные модули мантисс хранятся в регистрах *RG1* и *RG3* в разрядах α*8*÷α*31*, их знаки – в триггерах знака Тзн1 и Тзн2, а принятый порядок хранится в регистре счетчика *RG*Сч1.

3) Сложение / вычитание мантисс:

а) при одинаковых знаках чисел модули мантисс передаются в *RGA* и *RGB* и складываются в сумматоре.

Если в сумматоре в разряде α*7* окажется единица, то возникло переполнение разрядной сетки, поэтому сумма сдвигается на разряд вправо, а порядок увеличивается на единицу, т.е. *RG1*:= *RG*Сч1 + 1.

Если окажется, что в сумматоре в разряде α*0* находится единица, то произошло переполнение порядка, поэтому вырабатывается сигнал прерывания вычислительного процесса.

Если переполнения порядка нет, то в регистр сумматора в разряды α*1* ÷ α*7* заносится порядок из регистра счетчика *RG*Сч1, в разряд α*0* в регистр сумматора заносится знак мантиссы, а в разряды сумматора α*8* ÷ α*31* заносится мантисса суммы.

Таким образом, в регистре сумматора полностью сформирован результат.

б) при разных знаках мантисс отрицательный из них передается на входной регистр *RGA* или *RGB*, в обратном коде производится суммирование с дальнейшим подсуммированием единицы.

Знак результата фиксируется в триггере знака. Если полученный результат нормализован, т.е. в разряде сумматора α*8* находится единица, то в регистр сумматора заносится: знак результата – в α*0*, порядок – в α*1* ÷ α*7*, модуль мантиссы – в разряды α*8* ÷ α*31*.

Если результат не нормализован, т.е. в α*8* находится ноль, и нет исчезновения мантиссы ([α*8* ÷ α*31*] ≠ 0), производится нормализация сдвигом мантиссы влево с одновременным уменьшением порядка.

При отрицательном переполнении порядка, т.е. если *RG*Сч1[0]=1, формируется признак исчезновения порядка, т.е. результат равен нулю.

Если нормализация происходит без исчезновения порядка, то формируется результат из кодов знака, порядка и мантиссы.

Примечание: в операциях с плавающей точкой сложение и вычитание выполняется приближенно, т.к. при выравнивании порядка может происходить потеря младших разрядов одного из слагаемых.

Погрешность в этом случае всегда отрицательна и может доходить до единицы младшего разряда. Поэтому применяется округление результата, для чего используется дополнительный разряд сумматора, в который после суммирования добавляется единица.

Приведем наглядный пример:

Допустим, что Наши числа с одинарной точностью:  
  
00110011001000000000000000000000  
00011101000000000000000000000000  
  
Разбиваем числа на знак, порядок, мантиссу:  
  
0 01100110 01000000000000000000000  
0 00111010 00000000000000000000000  
  
Сдвинутый порядок приводим к нормальному виду (вычитаем из него 12710=011111112, считая его 8-разрядным двоичным числом в дополнительном коде):  
  
0 11100111 01000000000000000000000  
0 10111011 00000000000000000000000  
  
Мантиссы содержат только дробную часть, дополняем их целой частью (для нормализованного двоичного числа целая часть всегда =1, фактически - приписываем слева к мантиссе 1.):  
  
0 11100111 1.01000000000000000000000  
0 10111011 1.00000000000000000000000

Переводим для удобства в десятичную форму, ну а знак числа просто напишем (старший разряд =0, знак +), только для того, чтобы взглянуть, что за числа-то? :  
  
+ -25 1.25  
+ -69 1.00  
  
Получается:  
  
+1.25\*2-25  
+1.00\*2-69  
  
Вычитаем модули порядков  
  
69-25=44  
  
Далее нужно сдвинуть мантиссу второго числа (вместе с 1.) на 44 разряда вправо, скорректировать порядок второго числа (-69+44=-25 - порядки теперь равны) и дописать 44 незначащих нуля в мантиссу первого числа, затем сложить мантиссы и **округлить получившийся результат до 24 значащих разрядов**, затем, если целая часть мантиссы результата >1.02, нормализовать её, соответственно скорректировать порядок для результата (-25 минус количество разрядов, на которые сдвинули при нормализации), записать число  
  
(знак) (порядок результата) (мантисса результата)  
  
у мантиссы 1. стереть ластиком, к порядку прибавить 127 и... и всё. записать слитно, это и будет результат в IEEE754.

**1.5. Экстремальные случаи.**

Экстремальными случаями для 32 битного контейнера являются случаи хранения чисел, выходящих за диапазон типа float:

Float : [1.175494351 E - 38; 3.402823466 E + 38]

Что означает, экстремальными случаями для программы будут являться:

1) Входные данные выходят за диапазон

2) Результат суммы выходит за диапазон

3) Результат разности выходит за диапазон

2. Постановка задачи

2.1. Текст задания

Эмуляция АЛУ. Реализовать операцию сложения, вычитания чисел

с плавающей точкой.

2.2. Примечание к заданию

Реализовать ввод двух чисел. Вычислить сумму и разность в двоичной системе счисления в 32- битном формате, соответствующем стандарту IEEE 754. Вывести результат на экран.

3. Программная реализация

3.1. С консоли вводятся два числа в десятичной системе счисления. Затем реализуется перевод их в 32-битные контейнеры, соответствующие стандарту IEEE 754. Вызывается функция для суммы и разности введенных чисел.

3.2. Примеры

3.2.1. Тест для «a = 89, b = 13»

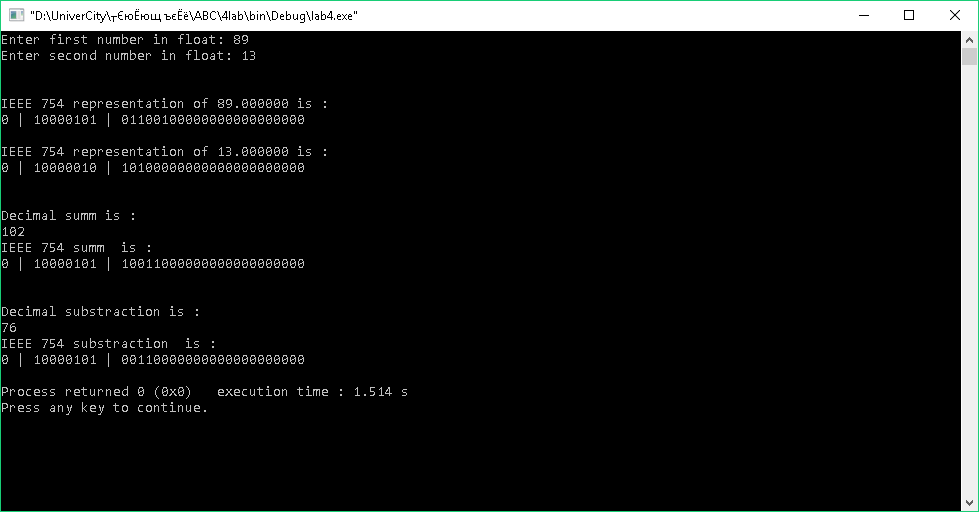


Рисунок 1. Скриншот для «a = 89, b = 13»

Ответ: Сумма 102

Разность 76

3.2.2. Тест для «a = 80.5612 , b = -6.214»

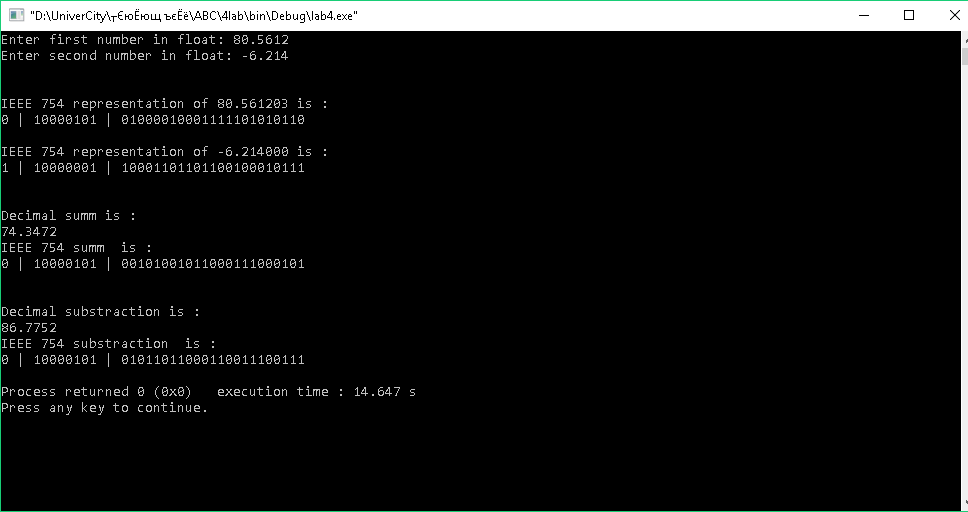


Рисунок 2. Скриншот для «a = 80.5612 , b = -6.214»

Ответ: Сумма 74.3472

Разность 86.7752

3.2.3. Тест для «a = -37.136 , b = 30.517»

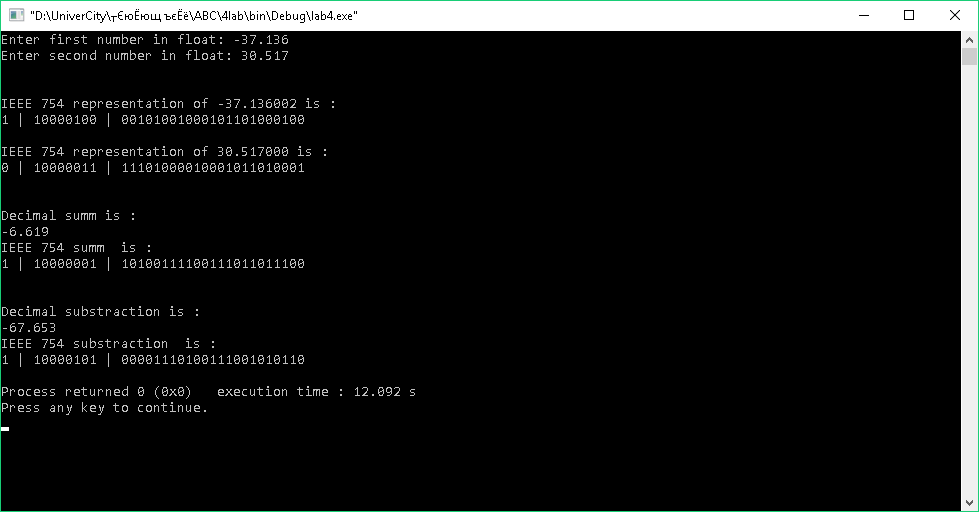


Рисунок 3. Скриншот для «a = -37.136 , b = 30.517»

Ответ: Сумма -6.619

Разность -67.653

3.2.4. Тест для «a =

555555555555555555555555555555555555555555555555,

b = 0»

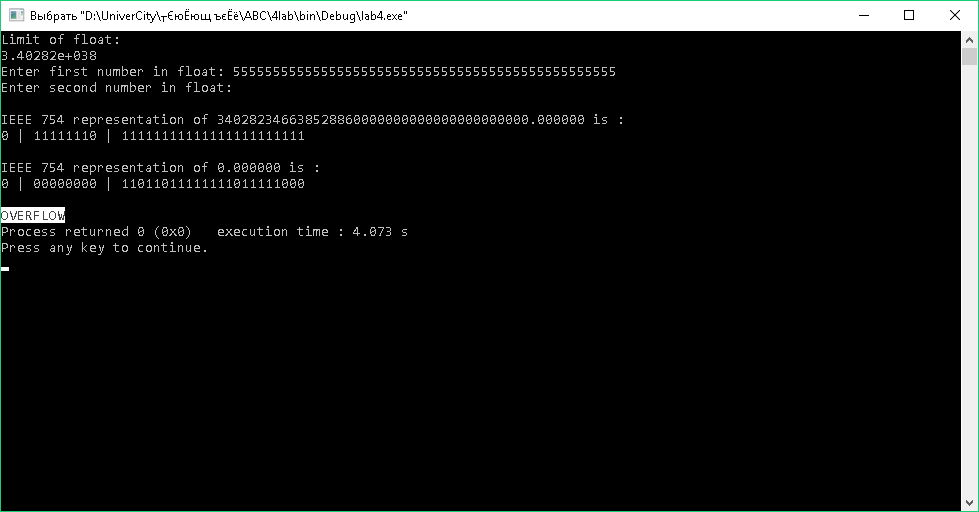


Рисунок 4. Скриншот для «a = 1000, b = 0»

Ответ: OVERFLOW

3.2.7. Резюме.

В своих примерах я старался рассмотреть максимально разнообразный набор случаев. Были сложение двух положительных чисел, отрицательного и положительного чисел.

Также были рассмотрены экстремальный случай поведения программы: переполнение результата суммы и разности.

4. Выводы

В ходе выполнения лабораторной работы ознакомился с понятиями мантиссы, экспоненты, и стандарта IEEE 754.

Так же рассмотрел работу арифметико-логического устройства (АЛУ), рассмотрел различные нюансы использования АЛУ. Для реализации использовал язык С++ и интегрированную среду разработки CodeBlocks, а так же онлайн-компилятор C++ Shell(C++14).

Литература

1. Волорова Н. А. Лабораторный практикум по курсу «Архитектура вычислительных систем» для студентов специальности «Информатика» /985-444-487-2 – Мн.: БГУИР, 2003. — 32 с.: ил.

2. Форум Habr. <https://habr.com/ru/post/337260/>

3. Форум Cyberforum. <https://www.cyberforum.ru/informatics/thread778486.html>

ПРИЛОЖЕНИЕ 1

Текст программы

Модуль 1. Source.cpp

#include <iostream>

#include <string>

#include <conio.h>

#include <stdio.h>

#include <cstdlib>

#include <math.h>

#include<limits>

using namespace std;

double n,bin,f1=0.0,f2,f3=0.1;

#include <stdio.h>

void printBinary(int n, int i)

{

// Prints the binary representation

// of a number n up to i-bits.

int k;

for (k = i - 1; k >= 0; k--) {

if ((n >> k) & 1)

printf("1");

else

printf("0");

}

}

typedef union {

float f;

struct

{

// Order is important.

// Here the members of the union data structure

// use the same memory (32 bits).

// The ordering is taken

// from the LSB to the MSB.

unsigned int mantissa : 23;

unsigned int exponent : 8;

unsigned int sign : 1;

} raw;

} myfloat;

int\* MakeArr(int\* array,myfloat var);

int\* Addition(int\* arr1, int\* arr2, int\* arr3);

// Function to convert real value

// to IEEE foating point representation

void printIEEE(myfloat var)

{

// Prints the IEEE 754 representation

// of a float value (32 bits)

printf("%d | ", var.raw.sign);

printBinary(var.raw.exponent, 8);

printf(" | ");

printBinary(var.raw.mantissa, 23);

printf("\n");

}

// Driver Code

int main()

{

// Instantiate the union

myfloat var, var2, razn, summ;

float float\_lim = fabs(numeric\_limits<float>::max());

float num, num2;

cout<<"Limit of float: "<< endl << float\_lim << endl;

cout<<"Enter first number in float: ";

cin >> num;

cout<<"Enter second number in float: ";

cin >> num2;

// Get the real value

//var.f = -2.25;

var.f = num;

var2.f = num2;

summ.f = num+num2;

razn.f = num-num2;

// Get the IEEE floating point representation

printf("\n\nIEEE 754 representation of %f is : \n",var.f);

printIEEE(var);

printf("\nIEEE 754 representation of %f is : \n",var2.f);

printIEEE(var2);

int\* arr1 = new int[32];

arr1 = MakeArr(arr1, var);

int\* arr2 = new int[32];

arr2 = MakeArr(arr2, var2);

int\* arr3 = new int[32];

arr3 = Addition(arr1, arr2, arr3);

if (fabs(num+num2) >= float\_lim)

{

cout<< "\nOVERFLOW";

return 0;

}

else

{

cout<<"\n\nDecimal summ is : \n"<< summ.f;

cout<<"\nIEEE 754 summ is : \n";

printIEEE(summ);

}

if (fabs(num-num2) >= float\_lim)

{

cout<< "\nOVERFLOW";

return 0;

}

else

{

cout<<"\n\nDecimal substraction is : \n"<< razn.f;

cout<<"\nIEEE 754 substraction is : \n";

printIEEE(razn);

}

return 0;

}

int\* MakeArr(int\* num, myfloat var){

num[0] = var.raw.sign;

for (int j=0; j<32; j++)

num[j]=0;

int k;

int n = var.raw.exponent;

int i=8;

for (k = i - 1; k >= 0; k--) {

if ((n >> k) & 1)

num[8-k] = 1;

else

num[8-k] = 0;

}

n = var.raw.mantissa;

i=23;

int z=9;

for (k = i - 1; k >= 0; k--) {

if ((n >> k) & 1)

num[z] = 1;

else

num[z] = 0;

z++;

}

return num;

}

int\* Addition(int\* arr1, int\* arr2, int\* sum){

for(int i=31; i>=0; i--)

{

sum[i] = arr1[i]+arr2[i];

}

return sum;

}