Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по лабораторной работе №5

Эмуляция АЛУ. Операция умножения

чисел с плавающей точкой

Студент: гр. 853504

Вечеринский Максим Сергеевич

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 Представление целых чисел.**

В двоичной системе счисления числа представляются с помощью комбинации единиц и нулей, знака "минус" и знака разделяющей точки между целой и дробной частью числа. Например, десятичное число -1.312510 в двоичном виде будет выглядеть как -1001.01012. Но в компьютере мы не можем хранить и обрабатывать символы знака и разделяющей точки — для "машинного" представления чисел могут использоваться только двоичные цифры (0 и 1). Если операции выполняются только с неотрицательными числами, то формат представления очевиден. В машинном слове из 8 бит можно представить числа в интервале от 0 до 255. Пример:

|  |  |
| --- | --- |
| *00000000  =* | *0* |
| *00000000  =* | *0* |
| *00000001  =* | *1* |
| *00101001  =* | *41* |
| *10000000  =* | *128* |
| *11111111* | *255* |

В общем случае n-битовая последовательность двоичных цифр an-1an-2…a1a0 интерпретирована как целое число А, значение которого равно

**1.2. Прямой код.**

Существует несколько соглашений о едином формате представления как положительных, так и отрицательных чисел. Всех их объединяет то, что старший бит слова (с точки зрения европейца — самый левый, или бит, которому при представлении числа без знака должен быть приписан самый большой вес) является битом хранения знака или знаковым разрядом. Все последующие биты слова представляют значащие разряды числа, которые в каждом формате интерпретируются по-своему. Значение 1 в знаковом разряде интерпретируется как представление всем словом отрицательного числа.

00010010 = +18

10010010 = -18

Общее правило математически формулируется следующим образом:

Формат представления чисел в прямом коде неудобен для использования в вычислениях. Во-первых, сложение и вычитание положительных и отрицательных чисел выполняется по-разному, а потому требуется анализировать знаковые разряды операндов. Во-вторых, в прямом коде числу 0 соответствуют две кодовых комбинации:

00000000 = +010

10000000 = -010

Это также неудобно, поскольку усложняется анализ результата на равенство нулю, а такая операция в программах встречается очень часто.

Из-за этих недостатков прямой код практически не применяется при реализации в АЛУ арифметических операций над целыми числами. Вместо этого более широкое применение находит другой формат, получивший наименование дополнительного кода.

**1.3. Алгоритм умножения.**

Если заданы A = a \* 2m и B = b \* 2m в нормальной форме, то их произведение составит :

AB = ab \* 2m1+m2

**Операция умножения** состоит из следующих **этапов**:

1) Определение знака произведения путем сложения по mod 2 знаковых цифр мантисс сомножителей.

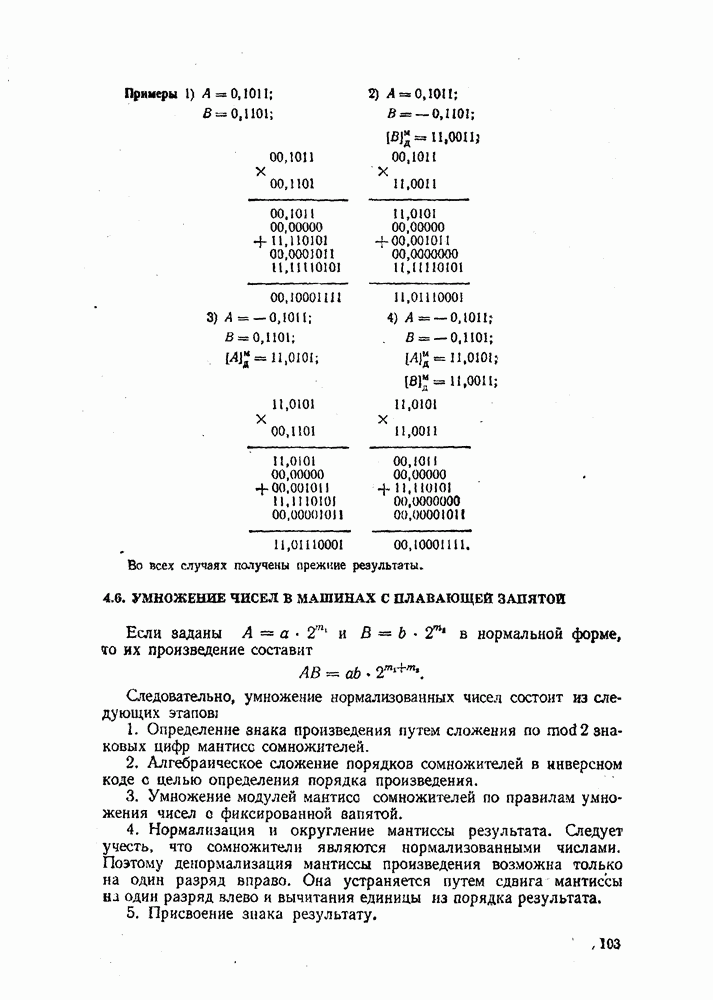
2) Алгебраическое сложение порядков сомножителей в инверсном коде с целью определения порядка произведения.

3) Умножение модулей мантисс сомножителей по правилам умножения чисел с фиксированной запятой

4) Нормализация и округление мантиссы результата. Следует учесть, что сомножители являются нормализированными числами. Поэтому денормализация мантиссы произведения возможна только на один разряд вправо. Она устраняется путем сдвига мантиссы на один разряд влево и вычитания единицы из порядка результата.

5) Присвоение знака результату.

**Примеры:**



### **1.4.** Умножение чисел с плавающей запятой

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

Умножение сводится к двум простейшим действиям, рассмотренным ранее: сложению порядков сомножителей как целых чисел и перемножению мантисс как операндов с фиксированной запятой. При необходимости к ним добавляется еще нормализация мантиссы результата:

Пz = Пx + Пy

Mz = Mx \* My

Интерес здесь представляют особые случаи, которые могут возникнуть на различных этапах обработки числа.

Так при сложении чисел с фиксированной точкой (речь идёт о порядках) может возникнуть переполнение, которое сделает невозможным дальнейшую работу с этим числом, либо при перемножении мантисс сомножителей мы получим денормализованную мантиссу результата, что потребует ее коррекции с одновременной коррекцией полученного ранее порядка произведения. Все эти и некоторые другие обстоятельства необходимо учитывать при проведении данной операции.

Ряд этих случаев был рассмотрен нами при обсуждении ГОСТа IEEE 754 на представление чисел с плавающей запятой [ в лабораторной номер 4]. В то же время иногда подобные ситуации возникают в процессе выполнения самой операции, а окончательный результат оказывается допустимым числом. Рассмотрим все эти случаи.

**1.5. Экстремальные случаи.**

При выполнении операции умножения в машине с плавающей запятой может получиться переполнение отрицательного порядка, которое будет интерпретировано как машинный нуль, если программой пользователя игнорируется признак исчезновения порядка.

Может так же возникнуть и переполнение положительного порядка. В этом случае надо нормализовать мантиссу результата. Если не помогло, то формируется признак переполнения порядка.

Так как мы исходим из того, что мантиссы обоих операндов нормализованы, то есть удовлетворяют условию

1 > |Mx,y| ≥ 2-1,

то 1 > |Mz| = |Mx| \* |My| ≥ 2-2.

Таким образом, нормализация мантиссы если и потребуется, то только путем сдвига на один разряд влево. При этом порядок, естественно, следует уменьшить на 1. Отсюда вытекает следующая последовательность действий.

1. Пz = Пx + Пy
   1. Если Пz = - ∞, то Z=0 .
   2. Если Пz = + ∞, то продолжить умножение, так как последующая операция над мантиссами может привести к коррекции порядка результата в сторону его уменьшения и, тем самым, обеспечит нормальное представление числа в целом.
2. Mz = Mx \* My

Т.к. |Mx| ≥ 2-1, |My| ≥ 2-1, то |Mz| ≥ 2-2

Возможная область ненормализованной мантиссы:

2-1 > |Mz| ≥ 2-2

Если |Mz| < 2-1, то выполнить нормализацию мантиссы с одновременной коррекцией порядка:

|Mz| = |Mz| \* 2+1

Пz = Пz -1

Если в результате получим Пz = - ∞, то Z=0.

Если в ходе перемножения мантисс получим |Mz| ≥ 2-1 , но ранее при обработке порядков получили Пz = + ∞ (см п.1.2), то Z = ∞

При Z=0 выполнение программы в ЭВМ продолжается.

При Z = ∞ устанавливается флаг прерывания, и ЭВМ приостанавливает обработку данных чисел.

2. Постановка задачи

2.1. Текст задания

Эмуляция АЛУ. Реализовать операцию умножения чисел с плавающей точкой.

2.2. Примечание к заданию

Реализовать ввод двух чисел. Вычислить произведение 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

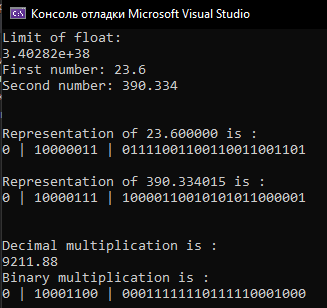
 3.2.1. Тест для «a = 23.6, b = 390.334»

Рисунок 1. Скриншот для «a = 23.6, b = 390.334»

Ответ: Произведение 9211.88

3.2.2. Тест для «a = 213.55, b = 0»

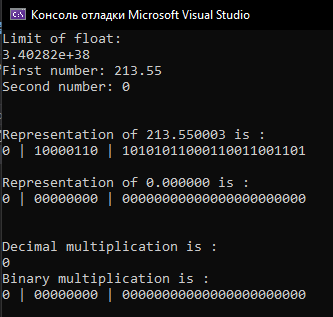


Рисунок 2. Скриншот для «a = 213.55, b = 0»

Ответ: Произведение 0

3.2.3. Тест для «a = 45.15926, b = -34»

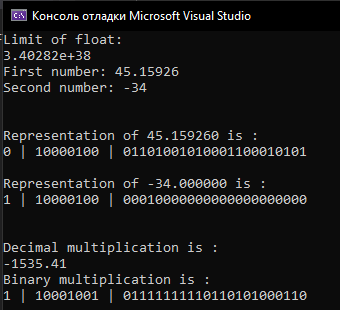


Рисунок 3. Скриншот для «a = 45.15926, b = -34»

Ответ: Произведение -1535.41

3.2.4. Тест для «a = -111.222, b = -456.001»

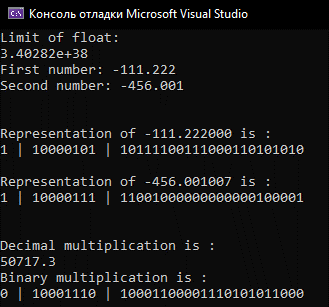


Рисунок 4. Скриншот для «a = -111.222, b = -456.001»

Ответ: Произведение 50.717.3

3.2.5. Тест для «a = 523434232.444444555,

b = 123330000044433323234334431555555543343.6»

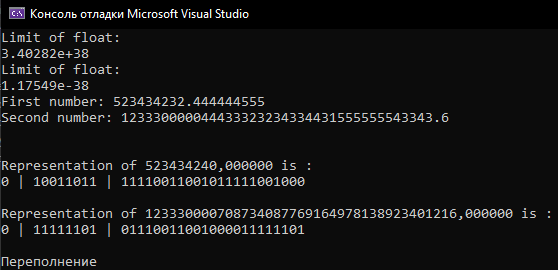


Рисунок 5. Скриншот для «a = 523434232.444444555,

b = 123330000044433323234334431555555543343.6»

Ответ: Переполнение

3.2.6. Тест для «a = 0.0000000000000000000000000000000000000001,

b = 1»

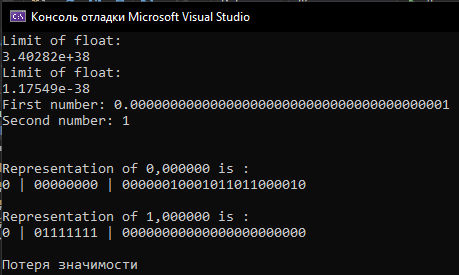


Рисунок 6. Скриншот для

«a = 0.0000000000000000000000000000000000000001,

b = 1»

Ответ: Потеря значимости

3.2.7. Резюме.

В своих примерах я старался рассмотреть максимально разнообразный набор случаев. Были приведены примеры:

* Произведение двух положительных чисел
* Произведение положительного и отрицательного чисел
* Произведение двух отрицательных чисел
* Произведение, выходящее за допустимые границы.

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++ , а так же компилятор Visual Studio 2019.

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. Tanenbaum A. S. «Structured computer organization» /Vrije Universiteit Amsterdam, The Netherlands, 2013. — 810 c.:ил.

ПРИЛОЖЕНИЕ 1

Текст программы

#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)

{

int k;

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

if ((n >> k) & 1)

printf("1");

else

printf("0");

}

}

typedef union {

float f;

struct

{

unsigned int mantissa : 23;

unsigned int exponent : 8;

unsigned int sign : 1;

} raw;

} myfloat;

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

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

void printIEEE(myfloat var)

{

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

printBinary(var.raw.exponent, 8);

printf(" | ");

printBinary(var.raw.mantissa, 23);

printf("\n");

}

int main()

{

setlocale(LC\_ALL, "Russian");

myfloat var, var2, mult;

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

float float\_min = fabs(numeric\_limits<float>::min());

float num, num2;

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

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

cout << "First number: ";

cin >> num;

cout << "Second number: ";

cin >> num2;

var.f = num;

var2.f = num2;

mult.f = num \* num2;

printf("\n\nRepresentation of %f is : \n", var.f);

printIEEE(var);

printf("\nRepresentation 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 = Multiplacation(arr1, arr2, arr3);

if (fabs(num \* num2) >= float\_max)

{

cout << "\nПереполнение";

return 0;

}

else if ((num \* num2) <= float\_min)

{

cout << "\nПотеря значимости";

return 0;

}

else

{

cout << "\n\nDecimal multiplication is : \n" << mult.f;

cout << "\nBinary multiplication is : \n";

printIEEE(mult);

}

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

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

{

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

}

return sum;

}