

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет
Московский институт электронной техники»

**ЛАБОРАТОРНАЯ РАБОТА № 5
ПО АРХИТЕКТУРАМ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ
РАБОТА С ЧИСЛАМИ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ**

ВАРИАНТ 6

Егоров Вадим, Кудреватых Павел, Саркисов Эрик
ПМ-31

Январь 2022 г.

Содержание

Задание Л5.№1..	1
Задание Л5.№2..	2
Задание Л5.№5..	3
Задание Л5.№6..	4
Задание Л5.№7..	5
Задание Л5.№8..	7

Цель работы: познакомиться с арифметикой чисел с плавающей запятой, а также с работой FPU и расширения AVX.

Если используемая платформа не поддерживает EVX, в соответствующих заданиях необходимо использовать команды SSE (SSE1-SSE4.2).

Штраф за одно пропущенное обязательное задание —2 балла.

Задание Л5.№1.

Разработайте программу на языке C/C++, выполняющую вычисления над числами с плавающей запятой одинарной точности (float). Проверьте, что программа действительно работает с операндами одинарной точности, а не приводит к типу float окончательный результат.

Для частичной суммы гармонического ряда $S(N) = \sum_{i=1}^N \frac{1}{i} \in \mathbf{R}$ найдите две её оценки: $S_d(N)$ — последовательно складывая члены, начиная от 1 и заканчивая N , и $S_a(N)$ — от N до 1. Сравните $S_d(N)$ и $S_a(N)$ для различных значений N : $10^3, 10^6, 10^9$. Объясните результат. Измените тип операндов на double. Объясните результат.

```
#include <stdio.h>
#include <math.h>

enum
{
    N1 = 1000,
    N2 = 1000000,
    N3 = 1000000000
};

void print_task1(float* Sd_f, float* Sa_f, double* Sd_d, double* Sa_d)
{
    int i = 0;
    for (i; i < 3; i++)
        printf("Sd_f = %.22f, Sa_f = %.22f\n", Sd_f[i], Sa_f[i]);
    i = 0;
    printf("-----\n");
    for (i; i < 3; i++)
        printf("Sd_d = %.48lf, Sa_d = %.48lf\n", Sd_d[i], Sa_d[i]);
    printf("-----\n");
}

void task1(float* Sd_f, float* Sa_f, double* Sd_d, double* Sa_d)
{
    int N[3] = { N1, N2, N3 };
}
```

```

int i, j = 0;
for (j; j < 3; j++)
{
    i = 1;
    for (i; i != N[j] + 1; i++)
    {
        Sd_f[j] += (float)(1) / i;
        Sd_d[j] += (double)(1) / i;
    }
    for (i; i != 0; i--)
    {
        Sa_f[j] += (float)(1) / i;
        Sa_d[j] += (double)(1) / i;
    }
}
}

```

Вызов функции:

```

//task 1
float Sd_f[3] = { 0, 0, 0 }, Sa_f[3] = { 0, 0, 0 };
double Sd_d[3] = { 0, 0, 0 }, Sa_d[3] = { 0, 0, 0 };
task1(Sd_f, Sa_f, Sd_d, Sa_d);
print_task1(Sd_f, Sa_f, Sd_d, Sa_d);

```

Вывод:

```

(kali@kali)~/Lab/Lab5
$ ./lab5_1
Task 1
Sd_f = 7.4854784011840820312500, Sa_f = 7.4864706993103027343750
Sd_f = 14.3573579788208007812500, Sa_f = 14.3926525115966796875000
Sd_f = 15.4036827087402343750000, Sa_f = 18.8079185485839843750000
-----
Sd_d = 7.485470860550343275008344789966940879821777343750, Sa_d = 7.486469861549341686668412876315414905548095703125
Sd_d = 14.39272672286498886385617661289870738983154296875, Sa_d = 14.39272722864773198807597509585320949554443359375
Sd_d = 21.300481502348549867065230500884354114532470703125, Sa_d = 21.300481503346148315358732361346483230590820312500
-----

```

Вывод: при прямом суммировании, начиная с больших слагаемых, результат подвергается округлению. А при обратном — получаемый результат более точный. Получаем, что последовательность S_a точнее, чем S_d . Так как тип double хранит больше знаков, данный эффект не столь заметен, как при float.

Задание Л5.№2.

Разработайте программу на языке C/C++, рассчитывающую с заданной точностью ϵ сумму лейбницевского ряда: Вариант 6: $S = \sum_{i=2}^{\infty} (-1)^i \frac{i+1}{i^2-1}$

```

#define VALUE(type_1, type_2) type_1 value_i(type_2 i)\
{ \
    return pow(-1, i) * ((type_1)(i + 1) / (i * i - 1));\
}

```

```
VALUE(double, int);
```

```

void print_task2(double s)
{
    printf("s = %.48lf\n", s);
}

```

```

double task2(double s)
{

```

```

double e = 1.0 / 1000, error = 0, s_1 = 0, s_2 = 0;
int i = 2;
for (i;; i++)
{
    s_1 = s;
    s_2 = value_i(i);
    s += s_2;
    error += (s_2 - (s - s_1)) + (s_1 - (s - (s - s_1)));
    if (fabs(value_i(i + 1)) <= e)
        break;
}
return s + error;
}

```

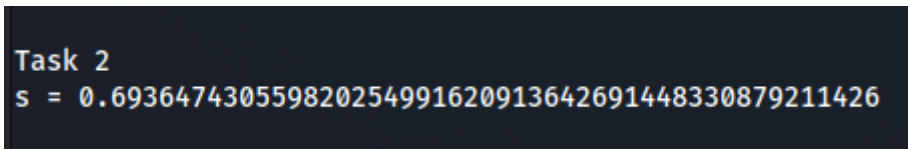
Вызов функции:

```

//task 2
double s = 0;
s = task2(s);
print_task2(s);

```

Вывод:



```

Task 2
s = 0.693647430559820254991620913642691448330879211426

```

Задание Л5.№5.

Рассчитайте (используя FPU) значение выражения от числа x с плавающей запятой (double): Вариант 6:
 $(-2, 12 + x) \cdot \log_2(x)$

Для проверки реализуйте вычисление того же выражения на C/C++ (вызывая функции libc).

Для решения задачи, используя FPU, будем применять ПОЛИЗ (польская инверсионная запись). В итоге приведенное для расчета выражение запишем в следующем виде: $-2,12 \ x + \log_2 x *$. С такой же последовательностью операций будем рассчитывать выражение в FPU.

```

#include <stdio.h>
#include <math.h>

extern double asm5(double);

double task5(double x)
{
    return (-2.12 + x)*log2(x);
}

int main()
{
    //task 5
    printf("Result C: %lf\nResult Asm: %lf\n", task5(2.5), asm5(2.5));
    return 0;
}

```

```

global asm5

```

```

section .text
value dq -2.12

```

```
asm5:    push rbp
        mov rbp, rsp

        sub rsp, 8
        movq [rsp], xmm0

        fld qword [value]      ; -2.12
        fld qword [rsp]        ; x
        faddp                    ; +

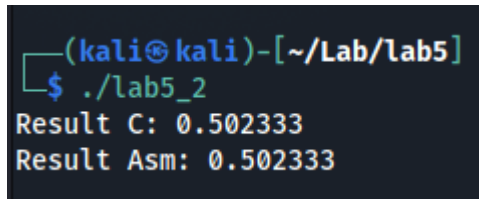
        fld1                     ; 1
        fld qword [rsp]        ; x
        fyl2x                    ; log_2(x)

        fmulp                    ; *

        fstp qword [rsp]
        movq xmm0, [rsp]

        mov rsp, rbp
        pop rbp
        ret
```

Вывод:



```
(kali@kali)-[~/Lab/lab5]
$ ./lab5_2
Result C: 0.502333
Result Asm: 0.502333
```

Данную задачу можно решить с использованием AVX при помощи команды для сложения `vaddpd`, умножения `vmulpd` и, например, используя готовую функцию для вычисления логарифма (`_mm256_log_ps()`).

Задание Л5.№6.

Рассчитайте (используя AVX) для массивов (x_0, \dots, x_3) и (y_0, \dots, y_3) из четырёх чисел с плавающей запятой (double), аналогичный массив (z_0, \dots, z_3) , где

$$z_i = (x_i + y_i) \cdot (x_i - y_i).$$

```
#include <stdio.h>
extern double* asm6(double*, double*);

int main()
{
    double x[] = {1.5, 2.5, 3.5, 4.5};
    double y[] = {4.5, 3.5, 2.5, 1.5};
    double* z = asm6(x, y);

    int i = 0;
    for(i; i < 4; i++)
        printf("%d) %lf\n", i+1, z[i]);

    return 0;
}
```

```

global asm6

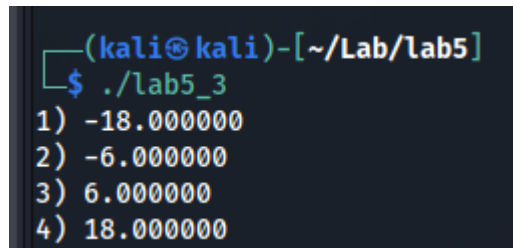
section .text
asm6:      sub rsp, 8

            vmovapd ymm0, [rdi]
            vmovapd ymm1, [rsi]
            vaddpd ymm2, ymm0, ymm1
            vsubpd ymm3, ymm0, ymm1
            vmulpd ymm0, ymm2, ymm3
            vmovapd [rdi], ymm0

            add rsp, 8
            ret

```

Вывод:



```

(kali@kali)-[~/Lab/lab5]
$ ./lab5_3
1) -18.000000
2) -6.000000
3) 6.000000
4) 18.000000

```

Данное задание можно реализовать при помощи FPU. Используя ПОЛИЗ, переведем данное выражение в следующий вид: $x_i y_i + x_i y_i - *$. Придерживаясь в FPU такой же последовательности операций и используя цикл, можно решить данную задачу.

Задание Л5.№7.

Разработайте программу, целиком написанную на ассемблере, вычисляющую (вызывая функции libc) выражение:

Вариант 6: $\sqrt[3]{x + 5.3} + y$

Для проверки реализуйте вычисление того же выражения в виде функции C/C++ и вызовите её из той же программы.

```

#include <math.h>

extern double c_7(double, double);

double c_7(double x, double y)
{
    return cbrt(x + 5.3) + y;
}



---



global main
extern printf, c_7

section .text
str_c: db "Result C: %lf", 10, 0
str_asm: db "Result Asm: %lf", 10, 0
x dq 1.5
y dq 2.5
value dq 5.3
index dq 0.3333333333333333

asm7:      push rbp

```

```

mov rbp, rsp

sub rsp, 8
movq [rsp], xmm0

fld qword [index]      ;1/3
fld qword [value]      ;5.3
fld qword [rsp]         ;x
faddp                   ;+

fyl2x
fld st0
frndint
fxch st1
fsub st0, st1
f2xm1
fld1
faddp
fscale                  ;sqrt_3(5.3+x)

movq [rsp], xmm1

fld qword [rsp]         ;y
faddp                   ;+

fstp qword [rsp]
movq xmm0, [rsp]

mov rsp, rbp
pop rbp
ret

main:                   sub rsp, 8

movq xmm0, [x]
movq xmm1, [y]
call c_7

mov rdi, str_c
mov al, 1
call printf

movq xmm0, [x]
movq xmm1, [y]
call asm7

mov rdi, str_asm
mov al, 1
call printf

add rsp, 8
xor rax, rax
ret

```

Вывод:

```

(kali㉿kali)-[~/Lab/lab5]
$ ./lab5_4
Result C: 4.394536
Result Asm: 4.394536

```

Задание Л5.№8.

Бонус +2 балла для пар. обязательное для троек. Реализуйте задание Л4.№4 для комплексных чисел, где вещественная и мнимая части хранятся как числа с плавающей запятой (double).

```

global main
extern printf

section .text
str: db "Result: %lf + i*%lf", 10, 0
re dq 1.5
im dq 1.5

asm8:
    vmulpd xmm2, xmm0, xmm0
    vmulpd xmm3, xmm1, xmm1
    vsubpd xmm2, xmm2, xmm3
    vaddpd xmm2, xmm2, xmm0    ;res re

    vaddpd xmm3, xmm0, xmm0
    vmulpd xmm3, xmm3, xmm1
    vaddpd xmm3, xmm3, xmm1    ;res im

    vmovapd xmm0, xmm2        ;res re
    vmovapd xmm1, xmm3        ;res im

    ret

main:
    sub rsp, 8

    movq xmm0, [re]
    movq xmm1, [im]
    call asm8

    mov rdi, str
    mov al, 2
    call printf

    add rsp, 8
    xor rax, rax
    ret

```

Вывод:

```

(kali㉿kali)-[~/Lab/lab5]
$ ./lab5_5
Result: 1.500000 + i*6.000000

```