Лабораторная работа 7 (0111 = 7)

Вычисления с плавающей запятой. Скалярные команды AVX/SSE

Цель работы: исследовать особенности арифметики с плавающей запятой. Научиться использовать скалярные команды расширений AVX/SSE.

Задание Л7.№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 $S(N) = \sum_{i=1}^{N} \frac{1}{i} \in R$ найдите две её оценки:

$S_d(N)$ — последовательно складывая члены, начиная от $i = 1$ и заканчивая $i = N$ («наивный» порядок),

$S_a(N)$ — от $i = N$ к $i = 1$.

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.

Все N печатаются в экспоненциальной форме, а не с хвостом из нулей.

Все $Sd(N)$ и $Sa(N)$ печатаются $print32()$ и $print64()$ из Л2

```
Задание №1
========================================================================
SystemInfo
========================================================================
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
========================================================================
N = 1.000000e+03
float sum forward:  40EF890A 10000001110111110001001000001010 1089440010 +1089440010 +0x1.df1214p+2 +7.485478e+00 +7.49
float sum backward: 40EF88FC 10000001110111110001000111111100 1089439996 +1089439996 +0x1.df11f8p+2 +7.485472e+00 +7.49
double sum forward:  401DF11F45F4E618 1000101111101001110011000011000 4620113909371954712 +4620113909371954712 +0x1.df11f45f4e618p+2 +7.485471e+00 +7.49
double sum backward: 401DF11F45F4E615 1000101111101001110011000010101 4620113909371954709 +4620113909371954709 +0x1.df11f45f4e615p+2 +7.485471e+00 +7.49
N = 1.000000e+06
float sum forward:  4165B7BD 10000010110010110111011110111101 1097185213 +1097185213 +0x1.cb6f7ap+3 +1.435736e+01 +14.36
float sum backward: 4166484D 10000010110011001001000001001101 1097222221 +1097222221 +0x1.cc909ap+3 +1.439265e+01 +14.39
double sum forward:  402CC9137A1DF0D6 1111010000111011111000011010110 4624292002893000918 +4624292002893000918 +0x1.cc9137a1df0d6p+3 +1.439273e+01 +14.39
double sum backward: 402CC9137A1DF28F 1111010000111011111001010001111 4624292002893001359 +4624292002893001359 +0x1.cc9137a1df28fp+3 +1.439273e+01 +14.39
N = 1.000000e+09
float sum forward:  4176757C 10000010111011001110101011111100 1098282364 +1098282364 +0x1.eceaf8p+3 +1.540368e+01 +15.40
float sum backward: 4196769E 10000011001011001110111010011110 1100379806 +1100379806 +0x1.2ced3cp+4 +1.880792e+01 +18.81
double sum forward:  40354CEC5B11A3D5 1011011000100011010001111010101 4626688770216928213 +4626688770216928213 +0x1.54cec5b11a3d5p+4 +2.130048e+01 +21.30
double sum backward: 40354CEC5B11A131 1011011000100011010000100110001 4626688770216927537 +4626688770216927537 +0x1.54cec5b11a131p+4 +2.130048e+01 +21.30
========================================================================
```

Рис. 1: Результаты вычисления гармонического ряда

Объяснение результатов

- Для float при больших N наивный порядок даёт меньшую точность, так как маленькие слагаемые теряются при сложении с большими.
- Обратный порядок уменьшает ошибку, т.к. сначала суммируются маленькие слагаемые.
- Для double точность намного выше, расхождение между двумя порядками меньше.
- При очень больших N (например, 10^9) вычисления на float сильно теряют точность.

Листинг:

Файл task7_1.c:

```c
void run_task7_1()
{
    printf("\nЗадание №1\n");
    printf("================================================================================");
    printSystemInfo();
    printf("================================================================================\n");

    int Ns[] = {1000, 1000000, 1000000000};
    int count = sizeof(Ns)/sizeof(Ns[0]);

    for (int idx = 0; idx < count; idx++) {
        int N = Ns[idx];
        printf("N = %e\n", (double)N);

        float Sf = harmonic_sum_float_forward(N);
        float Sb = harmonic_sum_float_backward(N);

        double Sd = harmonic_sum_double_forward(N);
        double Sb_d = harmonic_sum_double_backward(N);

        printf("float sum forward:  ");
        print32(&Sf);
        printf("float sum backward: ");
        print32(&Sb);

        printf("double sum forward:  ");
        print64(&Sd);
        printf("double sum backward: ");
        print64(&Sb_d);
    }

    printf("================================================================================\n");
}

// Сумма float от 1 до N (наивный порядок)
float harmonic_sum_float_forward(int N) {
    float sum = 0.0f;
    for (int i = 1; i <= N; i++) {
        float term = 1.0f / (float)i;
        sum += term;
    }
    return sum;
}

// Сумма float от N до 1 (обратный порядок)
float harmonic_sum_float_backward(int N) {
    float sum = 0.0f;
    for (int i = N; i >= 1; i--) {
        float term = 1.0f / (float)i;
        sum += term;
    }
    return sum;
}

// Сумма double от 1 до N (наивный порядок)
double harmonic_sum_double_forward(int N) {
    double sum = 0.0;
    for (int i = 1; i <= N; i++) {
        double term = 1.0 / (double)i;
        sum += term;
    }
    return sum;
}

// Сумма double от N до 1 (обратный порядок)
double harmonic_sum_double_backward(int N) {
    double sum = 0.0;
    for (int i = N; i >= 1; i--) {
        double term = 1.0 / (double)i;
        sum += term;
    }
    return sum;
}
```

Задание Л7.№2

Рассчитайте на языке C/C++ для заданного 32-битного значения с плавающей запятой x его модуль $|x|$, используя только битовые (целочисленные) операции и преобразование указателей.

Исходное значение и результат печатаются $print32()$ из Л2.

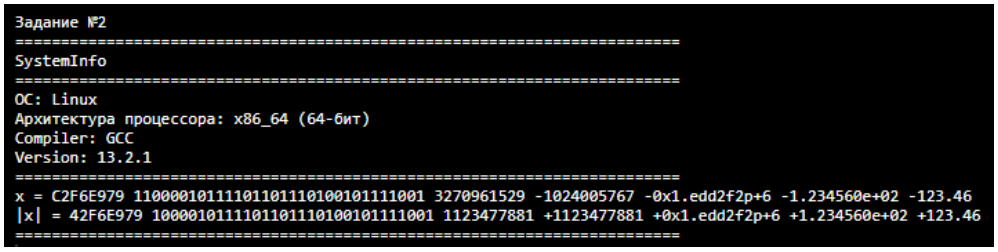

Рис. 2: результат выполнения `float_abs`

Листинг:

Файл task7_2.c:

```c
float float_abs(float x);

void run_task7_2()
{
    printf("\nЗадание №2\n");
    printf("================================================================================");
    printSystemInfo();
    printf("================================================================================\n");

    float x = -123.456f;
    printf("x = ");
    print32(&x);

    float abs_x = float_abs(x);
    printf("|x| = ");
    print32(&abs_x);

    printf("================================================================================\n");
}

float float_abs(float x) {
    uint32_t *p = (uint32_t *)&x;
    uint32_t bits = *p;

    // Сброс знакового бита (старший бит)
    bits &= 0x7FFFFFFF;

    float *res = (float *)&bits;
    return *res;
}

```

Задание Л7.№3

Разработайте на ассемблере функцию $inc32\_asm$(void *p), которая принимает адрес переменной p и выполняет целочисленный инкремент (команда inc) 32-битного (суффикс l) значения по этому адресу.

Разработайте на C/C++ программу, которая применяет $inc32\_asm$(void * p) к 32-битным переменным с плавающей запятой типа $float$. Значения переменных a, b, c, d соответствуют варианту (таблица Л2.2 из Л2). Каждое исходное значение и каждый результат печатаются $print32$() из Л2.

Как можно реализовать аналогичную функцию $inc32\_c$(void * p) на C/C++?

```
Задание №6
========================================================================
SystemInfo
========================================================================
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
========================================================================
Исходные значения:
x = 00000005 00000101 5 +5 +0x1.4p-147 +7.006492e-45 +0.00
y = FFFFFFFB 11111111111111111111111111111011 4294967291 -5 -nan -nan -nan
a = 3F800000 11111110000000000000000000000000 1065353216 +1065353216 +0x1p+0 +1.000000e+00 +1.00
b = 40000000 10000000000000000000000000000000 1073741824 +1073741824 +0x1p+1 +2.000000e+00 +2.00
c = 4B3C6159 10010110011110001100000101011001 1262248281 +1262248281 +0x1.78c2b2p+23 +1.234569e+07 +12345689.00
d = 4CEB79AF 10011001110101101111100110101111 1290500527 +1290500527 +0x1.d6f35ep+26 +1.234569e+08 +123456888.00

После inc32_asm:
a = 3F800001 11111110000000000000000000000001 1065353217 +1065353217 +0x1.000002p+0 +1.000000e+00 +1.00
b = 40000001 10000000000000000000000000000001 1073741825 +1073741825 +0x1.000002p+1 +2.000000e+00 +2.00
c = 4B3C615A 10010110011110001100000101011010 1262248282 +1262248282 +0x1.78c2b4p+23 +1.234569e+07 +12345690.00
d = 4CEB79B0 10011001110101101111100110110000 1290500528 +1290500528 +0x1.d6f36p+26 +1.234569e+08 +123456896.00

После inc32_c:
x = 00000006 00000110 6 +6 +0x1.8p-147 +8.407791e-45 +0.00
y = FFFFFFFC 11111111111111111111111111111100 4294967292 -4 -nan -nan -nan
========================================================================
```

Рис. 3: результат выполнения inc32_c и inc32_asm

Листинг:

Файл task7_3.c:

```c
#define showABCD printf("a = "); print32(&a); \
    printf("b = "); print32(&b); \
    printf("c = "); print32(&c); \
    printf("d = "); print32(&d);

#define showXY printf("x = "); \
    print32(&x); \
    printf("y = "); \
    print32(&y);

void inc32_asm(void *p);
void inc32_c(void *p);

void run_task7_3()
{
    printf("\nЗадание №6\n");
    printf("================================================================================");
    printSystemInfo();
    printf("================================================================================\n");

    // Инициализация переменных
    float a = 1.0f;
    float b = 2.0f;
    float c = 12345689.0f;
    float d = 123456891.0f;

    // Значения x=5, y=-5, a,b,c,d
    int x = 5;
    int y = -5;

    // Печать исходных значений
    printf("Исходные значения:\n");
    showXY
    showABCD

    // Применяем inc32_asm к float-переменным
    inc32_asm(&a);
    inc32_asm(&b);
    inc32_asm(&c);
    inc32_asm(&d);

    printf("\nПосле inc32_asm:\n");
    showABCD

    // Для сравнения применяем inc32_c к x и y
    inc32_c(&x);
    inc32_c(&y);

    printf("\nПосле inc32_c:\n");
    showXY

    printf("================================================================================\n");
}

void inc32_c(void *p) {
    int *ip = (int *)p;
    (*ip)++;
}

void inc32_asm(void *p) {
    __asm__ volatile (
        "incl (%0)"
        :
        : "r"(p)
        : "memory"
    );
}
```

Задание Л7.№4.

Вычислите для заданных x и y с плавающей запятой двойной точности выражение из таблицы Л7.1, используя скалярные AVX-команды либо их SSE-аналоги

Вариант 2: $z = 1 - 5/x - y^2/7$

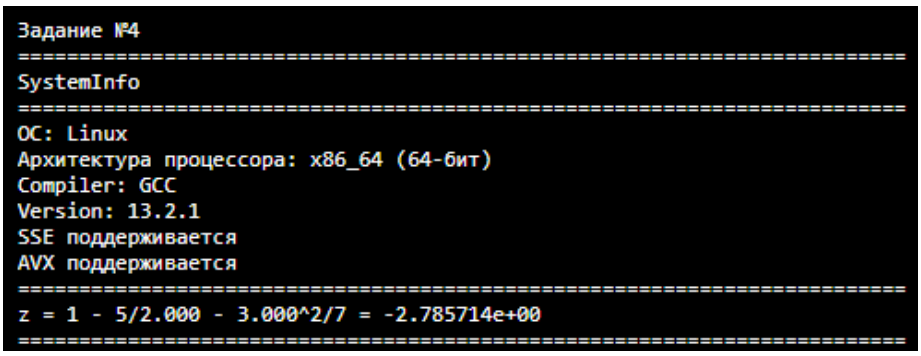

Рис. 4: выполнение avx_compute

Листинг:

Файл task7_4.c:

```c
double avx_compute(double x, double y);

void run_task7_4()
{
    printf("\nЗадание №4\n");
    printf("================================================================================");
    printSystemInfo();
    checkAVXorSSE();
    printf("================================================================================\n");

    double x = 2.0;
    double y = 3.0;

    double z = avx_compute(x, y);
    printf("z = 1 - 5/%.3f - %.3f^2/7 = %e\n", x, y, z);

    printf("================================================================================\n");
}

double avx_compute(double x, double y) {
    __m128d vx = _mm_set_sd(x);
    __m128d vy = _mm_set_sd(y);

    __m128d five = _mm_set_sd(5.0);
    __m128d one = _mm_set_sd(1.0);
    __m128d seven = _mm_set_sd(7.0);

    __m128d div1 = _mm_div_sd(five, vx);
    __m128d y_sq = _mm_mul_sd(vy, vy);
    __m128d div2 = _mm_div_sd(y_sq, seven);
    __m128d res = _mm_sub_sd(one, div1);
    res = _mm_sub_sd(res, div2);

    double result;
    _mm_store_sd(&result, res);
    return result;
}
```

Задание Л7.№5.

Разработайте программу, вычисляющую по введённым значениям x и y с плавающей запятой двойной точности значение z (таблица Л7.2), вызывая функции libm $pow()/atan2()$.

Если программа не собирается из-за отсутствия ссылок на $pow()/atan2()$, добавьте к команде сборки ключ -lm (указание компоновщику использовать libm).

Вариант 2: z = atan2(x, y), угол между вектором (x, y) и осью абсцисс.

```
Задание №5
================================================================================
Введите x y: 3 9
atan2(3.000, 9.000) = 3.217506e-01
================================================================================
```

Рис. 5: выполнение atan2 из libm

```
Задание №5
================================================================================
SystemInfo
================================================================================
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
================================================================================
Введите x y: 3 9
atan2(3.000000, 9.000000) = 3.217506e-01
================================================================================
```

Рис. 6: выполнение my_atan2 на ассемблерной вставке

Листинг:

Файл lr7_5.c:

```c
int main()
{
    printf("\nЗадание №5\n");
    printf("================================================================================\n");
    double x, y;
    printf("Введите x y: ");
    if (scanf("%lf %lf", &x, &y) != 2) {
        printf("Ошибка ввода\n");
        return 1;
    }

    double z = atan2(x, y);
    printf("atan2(%.3f, %.3f) = %e\n", x, y, z);
    printf("================================================================================\n");
}
```

Файл task7_6.c:

```c
double my_atan2(double y, double x);
void run_task7_5()
{
    double x, y;
    printf("Введите x y: ");
    if (scanf("%lf %lf", &x, &y) != 2) {
        printf("Ошибка ввода\n");
        return;
    }

    double z = my_atan2(x, y);
    printf("atan2(%.3f, %.3f) = %e\n", x, y, z);
}

double my_atan2(double y, double x) {
    double result;
    __asm__ (
        "fpatan"                    // Инструкция процессора для atan2
        : "=t" (result)             // Результат в ST(0)
        : "0" (x), "u" (y)          // Входные значения в ST(0) и ST(1)
        : "st(1)"                   // Регистр FPU, который будет изменён
    );
    return result;
}
```

Задание Л7.№6.

Разработайте функцию double mce_sd(void * p,size_t N), которая аналогично Л6.№6 обрабатывает массив из $double$ (таблица вариантов Л6.5).

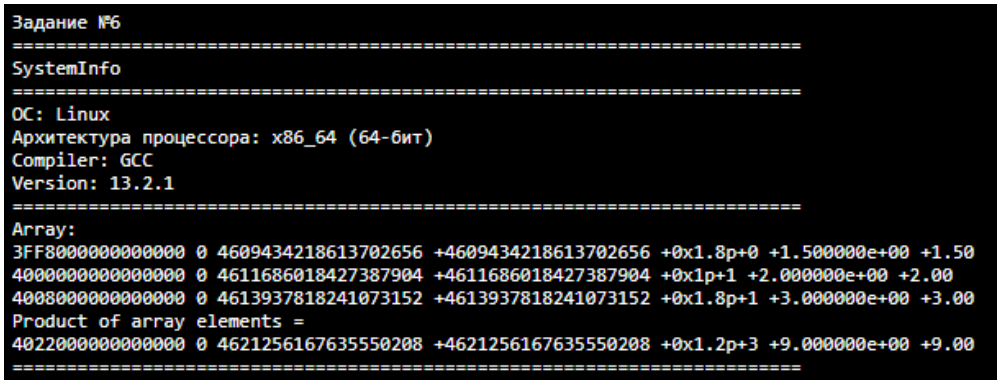

Рис. 7: выполнение mce_sd

Листинг:

Файл task7_6.c:

```c
void run_task7_6()
{
    printf("\nЗадание №6\n");
    printf("================================================================================");
    printSystemInfo();
    printf("================================================================================\n");

    double arr[] = {1.5, 2.0, 3.0};
    size_t N = sizeof(arr) / sizeof(arr[0]);

    double prod = mce_sd(arr, N);
    printf("Array:\n");
    PRINT_ARRAY(arr, print64);
    printf("Product of array elements = \n");
    print64(&prod);

    printf("================================================================================\n");
}

double mce_sd(void *p, size_t N) {
    double *arr = (double *)p;
    double result = 1.0;
    for (size_t i = 0; i < N; i++) {
        result *= arr[i];
    }
    return result;
}
```