Лабораторная работа 8 (1000 = 8)

Вычисления с плавающей запятой. Векторные команды AVX/SSE

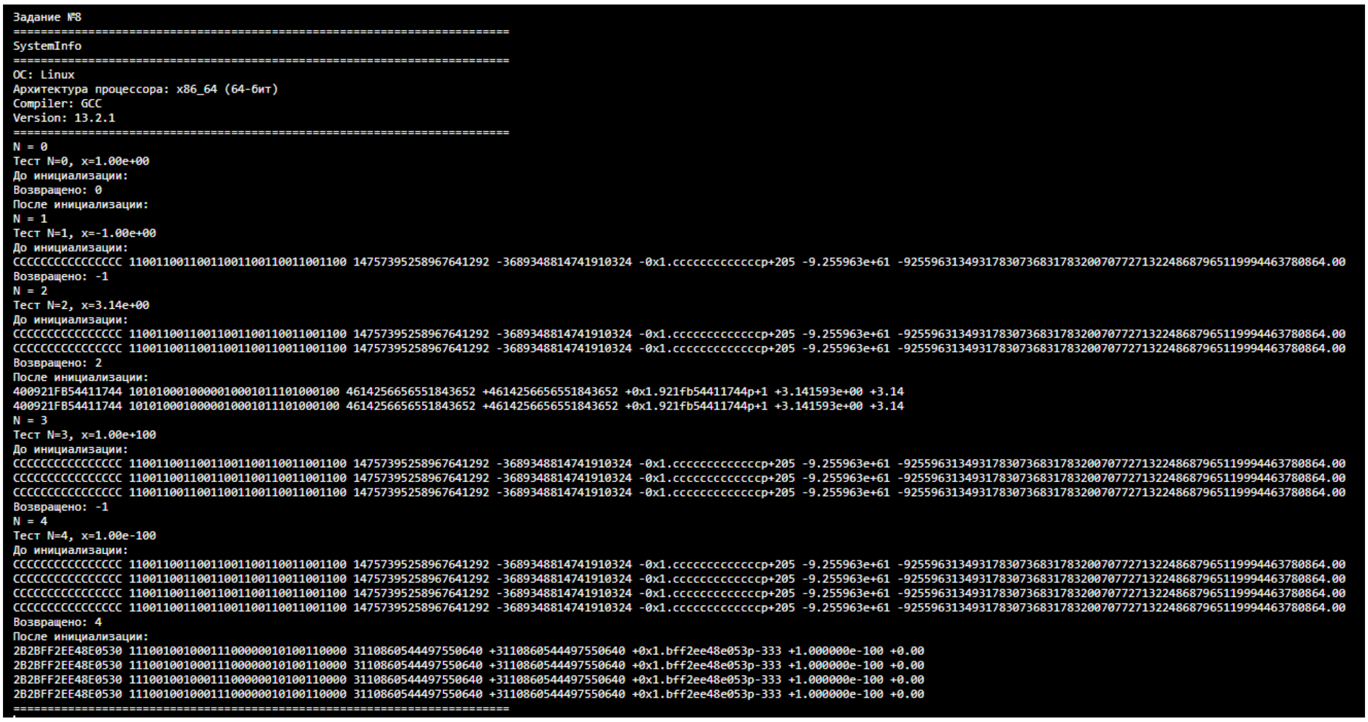
Цель работы: научиться использовать векторные команды расширений AVX/SSE.

Задание Л8.№1

Разработайте ассемблерную функцию size_t $init\_pd$(void * p,size_t N, double x), которая, если длина массива N кратна четырём, инициализирует массив из $double$ по адресу p из N элементов одинаковыми значениями x и возвращает N.

Используя векторные команды AVX $vmovupd$, $vpbroadcastd$ и ymm-регистры.

При некратном четырём N вернуть −1.



```
Задание №8
======================================================================
SystemInfo
======================================================================
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
======================================================================
N = 0
Тест N=0, x=1.00e+00
До инициализации:
Возвращено: 0
После инициализации:
N = 1
Тест N=1, x=-1.00e+00
До инициализации:
CCCCCCCCCCCCCCCC 11001100110011001100110011001100 14757395258967641292 -3689348814741910324 -0x1.ccccccccccccp+205 -9.255963e+61 -92559631349317830736831783200707727132248687965119994463780864.00
Возвращено: -1
N = 2
Тест N=2, x=3.14e+00
До инициализации:
CCCCCCCCCCCCCCCC 11001100110011001100110011001100 14757395258967641292 -3689348814741910324 -0x1.ccccccccccccp+205 -9.255963e+61 -92559631349317830736831783200707727132248687965119994463780864.00
CCCCCCCCCCCCCCCC 11001100110011001100110011001100 14757395258967641292 -3689348814741910324 -0x1.ccccccccccccp+205 -9.255963e+61 -92559631349317830736831783200707727132248687965119994463780864.00
Возвращено: 2
После инициализации:
400921FB54411744 10101000100000100010111101000100 4614256656551843652 +4614256656551843652 +0x1.921fb54411744p+1 +3.141593e+00 +3.14
400921FB54411744 10101000100000100010111101000100 4614256656551843652 +4614256656551843652 +0x1.921fb54411744p+1 +3.141593e+00 +3.14
N = 3
Тест N=3, x=1.00e+100
До инициализации:
CCCCCCCCCCCCCCCC 11001100110011001100110011001100 14757395258967641292 -3689348814741910324 -0x1.ccccccccccccp+205 -9.255963e+61 -92559631349317830736831783200707727132248687965119994463780864.00
CCCCCCCCCCCCCCCC 11001100110011001100110011001100 14757395258967641292 -3689348814741910324 -0x1.ccccccccccccp+205 -9.255963e+61 -92559631349317830736831783200707727132248687965119994463780864.00
CCCCCCCCCCCCCCCC 11001100110011001100110011001100 14757395258967641292 -3689348814741910324 -0x1.ccccccccccccp+205 -9.255963e+61 -92559631349317830736831783200707727132248687965119994463780864.00
Возвращено: -1
N = 4
Тест N=4, x=1.00e-100
До инициализации:
CCCCCCCCCCCCCCCC 11001100110011001100110011001100 14757395258967641292 -3689348814741910324 -0x1.ccccccccccccp+205 -9.255963e+61 -92559631349317830736831783200707727132248687965119994463780864.00
CCCCCCCCCCCCCCCC 11001100110011001100110011001100 14757395258967641292 -3689348814741910324 -0x1.ccccccccccccp+205 -9.255963e+61 -92559631349317830736831783200707727132248687965119994463780864.00
CCCCCCCCCCCCCCCC 11001100110011001100110011001100 14757395258967641292 -3689348814741910324 -0x1.ccccccccccccp+205 -9.255963e+61 -92559631349317830736831783200707727132248687965119994463780864.00
CCCCCCCCCCCCCCCC 11001100110011001100110011001100 14757395258967641292 -3689348814741910324 -0x1.ccccccccccccp+205 -9.255963e+61 -92559631349317830736831783200707727132248687965119994463780864.00
Возвращено: 4
После инициализации:
2B2BFF2EE48E0530 11100100100011100000010100110000 3110860544497550640 +3110860544497550640 +0x1.bff2ee48e053p-333 +1.000000e-100 +0.00
2B2BFF2EE48E0530 11100100100011100000010100110000 3110860544497550640 +3110860544497550640 +0x1.bff2ee48e053p-333 +1.000000e-100 +0.00
2B2BFF2EE48E0530 11100100100011100000010100110000 3110860544497550640 +3110860544497550640 +0x1.bff2ee48e053p-333 +1.000000e-100 +0.00
2B2BFF2EE48E0530 11100100100011100000010100110000 3110860544497550640 +3110860544497550640 +0x1.bff2ee48e053p-333 +1.000000e-100 +0.00
======================================================================
```

Рис. 1: Результат выполнения init_pd_asm с различными данными

Листинг:

Файл task8_1.c:

```c
size_t init_pd_C(void *p, size_t N, double x);
size_t init_pd_asm(void *p, size_t N, double x);
void test_init_pd(size_t N, double x);

void run_task8_1()
{
    double test_values[] = {1.0, -1.0, 3.1415926535, 1e100, 1e-100};
    int count = sizeof(test_values)/sizeof(test_values[0]);

    for (size_t i = 0; i < count; i++) {
        test_init_pd(i, test_values[i]);
    }
}

//__attribute__((target("avx")))
size_t init_pd_asm(void *p, size_t N, double x){
    if (N % 2 != 0) {               // Сокращено для лучшей наглядности
        return (size_t)-1;
    }
    asm volatile (
        "vbroadcastsd %2, %%ymm0\n\t"             // vpbroadcastd xmm0 = x в ymm0
        "xor %%rcx, %%rcx\n\t"                     // счетчик i = 0
        "1:\n\t"
        "cmp %1, %%rcx\n\t"
        "jge 2f\n\t"
        "vmovupd %%ymm0, (%0, %%rcx, 8)\n\t"      // записать 4 double (32 байта)
        "add $4, %%rcx\n\t"
        "jmp 1b\n\t"
        "2:\n\t"
        : // нет выходных
        : "r"(p), "r"(N), "m"(x)
        : "rcx", "ymm0", "memory"
    );
    return N;
}

__attribute__((target("avx")))
size_t init_pd_C(void *p, size_t N, double x) {
    if (N % 2 != 0) {               // Сокращено для лучшей наглядности
        return (size_t)-1;
    }
    double *arr = (double *)p;
    __m256d val = _mm256_set1_pd(x); // vpbroadcastd

    for (size_t i = 0; i < N; i += 4) {
        _mm256_storeu_pd(&arr[i], val); // vmovupd
    }
    return N;
}

void test_init_pd(size_t N, double x) {
    printf("N = %u\n", N);

    // Выделяем память с выравниванием по 32 байтам для AVX
    double *arr = aligned_alloc(32, N * sizeof(double));
    if (!arr) {
        fprintf(stderr, "Ошибка выделения памяти\n"); return;
    }
    // Заполняем массив "мусором" для демонстрации
    memset(arr, 0xCC, N * sizeof(double));

    printf("Тест N=%zu, x=%.2e\n", N, x);
    printf("До инициализации: \n");
    PRINT_ARRAY(arr, N, print64);

    // Вызываем ассемблерную функцию
    size_t res = init_pd_asm(arr, N, x);
    printf("Возвращено: %d\n", res);
    if(res != -1){
        printf("После инициализации: \n");
        PRINT_ARRAY(arr, N, print64);
    }
    free(arr);
}
```

Задание Л8.№2

Вычислите для массивов ($x0$, …$x3$) и ($y0$, …$y3$) из четырёх чисел с плавающей запятой двойной точности значения ($z0$, ...$z3$) согласно таблице Л8.1, используя векторные команды AVX $vmovupd$, $vaddpd$, $vsubpd$, $vmulpd$, $vdivpd$, $vpbroadcastd$ и ymm-регистры.

Выделение памяти под x, y, z и заполнение массивов x, y может быть выполнено на C/C++. Проверьте расчёт, реализовав то же самое на C/C++.

Вариант 2: $zi = xi/yi + 2$

```
Задание №2
================================================================
SystemInfo
================================================================
OC: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
================================================================

Исходные данные:
( x 0, ... x 3):
3FF0000000000000 1111111111000000000000000000000000000000000000000000000000000000 4607182418800017408 +4607182418800017408 +0x1p+0 +1.000000e+00 +1.00
BFF0000000000000 1011111111110000000000000000000000000000000000000000000000000000 1383055455654793216 -4616189618054758400 -0x1p+0 -1.000000e+00 -1.00
4001999999999999A 1000000000000110011001100110011001100110011001100110011001100110 4612136378390124954 +4612136378390124954 +0x1.199999999999ap+1 +2.200000e+00 +2.20
400A666666666666 1000000000010100110011001100110011001100110011001100110011001100110 4614613358185178726 +4614613358185178726 +0x1.a666666666666p+1 +3.300000e+00 +3.30

( y 0, ... y 3):
401199999999999A 1000000000100011001100110011001100110011001100110011001100110011010 4616639978017495450 +4616639978017495450 +0x1.199999999999ap+2 +4.400000e+00 +4.40
4016000000000000 1000000000101100000000000000000000000000000000000000000000000000000 4617878467915022336 +4617878467915022336 +0x1.6p+2 +5.500000e+00 +5.50
401A666666666666 1000000000110100110011001100110011001100110011001100110011001100110 4619116957812549222 +4619116957812549222 +0x1.a666666666666p+2 +6.600000e+00 +6.60
401ECCCCCCCCCCCD 1000000000111101100110011001100110011001100110011001100110011001101 4620355447710076109 +4620355447710076109 +0x1.eccccccccccdp+2 +7.700000e+00 +7.70

Результат ассемблерной ф-ии:
( z 0, ... z 3):
4001D1745D1745D1 1000000000000011101000101110100010111010001011101000101110100010111010001 4612197791112316369 +4612197791112316369 +0x1.1d1745d1745d1p+1 +2.227273e+00 +2.23
3FFD1745D1745D17 1111111111101000101110100010111010001011101000101110100010111010000111 4610867182131502359 +4610867182131502359 +0x1.d1745d1745d17p+0 +1.818182e+00 +1.82
4002AAAAAAAAAAAB 1000000000000101010101010101010101010101010101010101010101010101011 4612436618365282987 +4612436618365282987 +0x1.2aaaaaaaaaaabp+1 +2.333333e+00 +2.33
40036DB6DB6DB6DB 1000000000000110110110110110110110110110110110110110110110110110011 4612651075490395867 +4612651075490395867 +0x1.36db6db6db6dbp+1 +2.428571e+00 +2.43

Результат Си ф-ии:
( z 0, ... z 3):
4001D1745D1745D1 1000000000000011101000101110100010111010001011101000101110100010111010001 4612197791112316369 +4612197791112316369 +0x1.1d1745d1745d1p+1 +2.227273e+00 +2.23
3FFD1745D1745D17 1111111111101000101110100010111010001011101000101110100010111010000111 4610867182131502359 +4610867182131502359 +0x1.d1745d1745d17p+0 +1.818182e+00 +1.82
4002AAAAAAAAAAAB 1000000000000101010101010101010101010101010101010101010101010101011 4612436618365282987 +4612436618365282987 +0x1.2aaaaaaaaaaabp+1 +2.333333e+00 +2.33
40036DB6DB6DB6DB 1000000000000110110110110110110110110110110110110110110110110110011 4612651075490395867 +4612651075490395867 +0x1.36db6db6db6dbp+1 +2.428571e+00 +2.43

================================================================
```

Рис. 2: результат выполнения compute_z_avx_asm и compute_z_avx_c

Листинг:

Файл task8_2.c:

```c
void compute_z_avx_asm(const double *x, const double *y, double *z);
void compute_z_avx_c(const double *x, const double *y, double *z);

void run_task8_2()
{
    printf("\nЗадание №2\n");
    printf("================================================================================");
    printSystemInfo();
    printf("================================================================================\n");

    double arrX[] = {1.0, -1.0, 2.2, 3.3};
    double arrY[] = {4.4, 5.5, 6.6, 7.7};
    double arrZ[] = {0,0,0,0};

    printf("\nИсходные данные: ");
    printf("\n(x0, ...x3): \n"); PRINT_ARRAY(arrX, print64);
    printf("\n(y0, ...y3): \n"); PRINT_ARRAY(arrY, print64);

    compute_z_avx_asm(arrX, arrY, arrZ);

    printf("\nРезультат ассемблерной ф-ии: ");
    printf("\n(z0, ...z3): \n"); PRINT_ARRAY(arrZ, print64);

    memset(arrZ, 0, sizeof(arrZ)); // Обнуление

    compute_z_avx_c(arrX, arrY, arrZ);

    printf("\nРезультат Си ф-ии: ");
    printf("\n(z0, ...z3): \n"); PRINT_ARRAY(arrZ, print64);

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

void compute_z_avx_asm(const double *x, const double *y, double *z) {
    asm volatile (
        "vmovupd (%0), %%ymm0\n\t"              // загрузить x0..x3
        "vmovupd (%1), %%ymm1\n\t"              // загрузить y0..y3
        "vdivpd %%ymm1, %%ymm0, %%ymm2\n\t"     // ymm2 = ymm0 / ymm1
        "vbroadcastsd %3, %%ymm3\n\t"           // ymm3 = 2.0
        "vaddpd %%ymm3, %%ymm2, %%ymm2\n\t"     // ymm2 += 2.0
        "vmovupd %%ymm2, (%2)\n\t"              // сохранить z0..z3
        :
        : "r"(x), "r"(y), "r"(z), "x"(2.0)
        : "ymm0", "ymm1", "ymm2", "ymm3", "memory"
    );
}

__attribute__((target("avx")))
void compute_z_avx_c(const double *x, const double *y, double *z) {
    __m256d vx = _mm256_loadu_pd(x);        // vmovupd
    __m256d vy = _mm256_loadu_pd(y);        // vmovupd
    __m256d v2 = _mm256_set1_pd(2.0);       // vpbroadcastd

    __m256d vdiv = _mm256_div_pd(vx, vy);   // vdivpd
    __m256d vres = _mm256_add_pd(vdiv, v2); // vaddpd

    _mm256_storeu_pd(z, vres);              // vmovupd
}
```

Задание Л8.№3

Разработайте ассемблерную функцию int $v4$(void *px, void * py, void * pz, size_t N), рассчитывающую z согласно таблице Л8.1 для длины N, кратной четырём. Возвращаемое значение должно быть равно −1 при N, не кратном 4, и количеству успешно рассчитанных элементов z при корректном N.

Вариант 2: $zi = xi/yi + 2$

```
Задание №3
====================================================================
SystemInfo
====================================================================
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
====================================================================

Исходные данные: N=3
( x 0, ... x 3):
3FF0000000000000 11111111110000000000000000000000000000000000000000000000000000 4607182418800017408 +4607182418800017408 +0x1p+0 +1.000000e+00 +1.00
4000000000000000 100000000000000000000000000000000000000000000000000000000000000 4611686018427387904 +4611686018427387904 +0x1p+1 +2.000000e+00 +2.00
4008000000000000 100000000001000000000000000000000000000000000000000000000000000 4613937818241073152 +4613937818241073152 +0x1.8p+1 +3.000000e+00 +3.00

( y 0, ... y 3):
4000000000000000 100000000000000000000000000000000000000000000000000000000000000 4611686018427387904 +4611686018427387904 +0x1p+1 +2.000000e+00 +2.00
4010000000000000 100000000010000000000000000000000000000000000000000000000000000 4616189618054758400 +4616189618054758400 +0x1p+2 +4.000000e+00 +4.00
4018000000000000 100000000011000000000000000000000000000000000000000000000000000 4618441417868443648 +4618441417868443648 +0x1.8p+2 +6.000000e+00 +6.00

( z 0, ... z 3):
0000000000000000 0 0 +0 +0x0p+0 +0.000000e+00 +0.00
0000000000000000 0 0 +0 +0x0p+0 +0.000000e+00 +0.00
0000000000000000 0 0 +0 +0x0p+0 +0.000000e+00 +0.00
Ошибка: N не кратно 4
```

Рис. 3: результат выполнения v4 для $\mathrm{N} = 3$

```
Задание №3
====================================================================
SystemInfo
====================================================================
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
====================================================================

Исходные данные: N=8
( x 0, ... x 3):
3FF0000000000000 11111111110000000000000000000000000000000000000000000000000000 4607182418800017408 +4607182418800017408 +0x1p+0 +1.000000e+00 +1.00
4000000000000000 100000000000000000000000000000000000000000000000000000000000000 4611686018427387904 +4611686018427387904 +0x1p+1 +2.000000e+00 +2.00
4008000000000000 100000000001000000000000000000000000000000000000000000000000000 4613937818241073152 +4613937818241073152 +0x1.8p+1 +3.000000e+00 +3.00
4010000000000000 100000000010000000000000000000000000000000000000000000000000000 4616189618054758400 +4616189618054758400 +0x1p+2 +4.000000e+00 +4.00
4014000000000000 100000000010100000000000000000000000000000000000000000000000000 4617315517961601024 +4617315517961601024 +0x1.4p+2 +5.000000e+00 +5.00
4018000000000000 100000000011000000000000000000000000000000000000000000000000000 4618441417868443648 +4618441417868443648 +0x1.8p+2 +6.000000e+00 +6.00
401C000000000000 100000000011100000000000000000000000000000000000000000000000000 4619567317775286272 +4619567317775286272 +0x1.cp+2 +7.000000e+00 +7.00
4020000000000000 100000000100000000000000000000000000000000000000000000000000000 4620693217682128896 +4620693217682128896 +0x1p+3 +8.000000e+00 +8.00

( y 0, ... y 3):
4000000000000000 100000000000000000000000000000000000000000000000000000000000000 4611686018427387904 +4611686018427387904 +0x1p+1 +2.000000e+00 +2.00
4010000000000000 100000000010000000000000000000000000000000000000000000000000000 4616189618054758400 +4616189618054758400 +0x1p+2 +4.000000e+00 +4.00
4018000000000000 100000000011000000000000000000000000000000000000000000000000000 4618441417868443648 +4618441417868443648 +0x1.8p+2 +6.000000e+00 +6.00
4020000000000000 100000000100000000000000000000000000000000000000000000000000000 4620693217682128896 +4620693217682128896 +0x1p+3 +8.000000e+00 +8.00
4024000000000000 100000000100100000000000000000000000000000000000000000000000000 4621819117588971520 +4621819117588971520 +0x1.4p+3 +1.000000e+01 +10.00
4028000000000000 100000000101000000000000000000000000000000000000000000000000000 4622945017495814144 +4622945017495814144 +0x1.8p+3 +1.200000e+01 +12.00
402C000000000000 100000000101100000000000000000000000000000000000000000000000000 4624070917402656768 +4624070917402656768 +0x1.cp+3 +1.400000e+01 +14.00
4030000000000000 100000000110000000000000000000000000000000000000000000000000000 4625196817309499392 +4625196817309499392 +0x1p+4 +1.600000e+01 +16.00

( z 0, ... z 3):
0000000000000000 0 0 +0 +0x0p+0 +0.000000e+00 +0.00
0000000000000000 0 0 +0 +0x0p+0 +0.000000e+00 +0.00
0000000000000000 0 0 +0 +0x0p+0 +0.000000e+00 +0.00
0000000000000000 0 0 +0 +0x0p+0 +0.000000e+00 +0.00
0000000000000000 0 0 +0 +0x0p+0 +0.000000e+00 +0.00
0000000000000000 0 0 +0 +0x0p+0 +0.000000e+00 +0.00
0000000000000000 0 0 +0 +0x0p+0 +0.000000e+00 +0.00
0000000000000000 0 0 +0 +0x0p+0 +0.000000e+00 +0.00

Результат v4:
4004000000000000 100000000000100000000000000000000000000000000000000000000000000 4612811918334230528 +4612811918334230528 +0x1.4p+1 +2.500000e+00 +2.50
4004000000000000 100000000000100000000000000000000000000000000000000000000000000 4612811918334230528 +4612811918334230528 +0x1.4p+1 +2.500000e+00 +2.50
4004000000000000 100000000000100000000000000000000000000000000000000000000000000 4612811918334230528 +4612811918334230528 +0x1.4p+1 +2.500000e+00 +2.50
4004000000000000 100000000000100000000000000000000000000000000000000000000000000 4612811918334230528 +4612811918334230528 +0x1.4p+1 +2.500000e+00 +2.50
4004000000000000 100000000000100000000000000000000000000000000000000000000000000 4612811918334230528 +4612811918334230528 +0x1.4p+1 +2.500000e+00 +2.50
4004000000000000 100000000000100000000000000000000000000000000000000000000000000 4612811918334230528 +4612811918334230528 +0x1.4p+1 +2.500000e+00 +2.50
4004000000000000 100000000000100000000000000000000000000000000000000000000000000 4612811918334230528 +4612811918334230528 +0x1.4p+1 +2.500000e+00 +2.50
4004000000000000 100000000000100000000000000000000000000000000000000000000000000 4612811918334230528 +4612811918334230528 +0x1.4p+1 +2.500000e+00 +2.50

====================================================================
```

Рис. 4: результат выполнения v4 для $\mathrm{N} = 8$

Листинг:

Файл task8_3.c:

```
 1. int v4(void *px, void *py, void *pz, size_t N);
 2.
 3. void run_task8_3()
 4. {
 5.     printf("\nЗадание №3\n");
 6.     printf("================================================================================");
 7.     printSystemInfo();
 8.     printf("================================================================================\n");
 9.
10.     size_t N = 8;
11.     double *x = (double*)aligned_alloc(32, N * sizeof(double));
12.     double *y = (double*)aligned_alloc(32, N * sizeof(double));
13.     double *z = (double*)aligned_alloc(32, N * sizeof(double));
14.
15.     if (!x || !y || !z) {
16.         printf("Ошибка выделения памяти\n");
17.         return;
18.     }
19.
20.     // Инициализируем массивы
21.     for (size_t i = 0; i < N; i++) {
22.         x[i] = i + 1.0;      // 1.0, 2.0, ..., 8.0
23.         y[i] = (i + 1) * 2;  // 2.0, 4.0, ..., 16.0
24.         z[i] = 0.0;
25.     }
26.
27.     printf("\nИсходные данные: N=%u", N);
28.     printf("\n(x0, ...x3): \n"); PRINT_ARRAY(x, N, print64);
29.     printf("\n(y0, ...y3): \n"); PRINT_ARRAY(y, N, print64);
30.     printf("\n(z0, ...z3): \n"); PRINT_ARRAY(z, N, print64);
31.
32.     int res = v4(x, y, z, N);
33.     if (res < 0) {
34.         printf("Ошибка: N не кратно 4\n");
35.         free(x); free(y); free(z);
36.         return;
37.     }
38.
39.     printf("\nРезультат v4:\n");
40.     PRINT_ARRAY(z,N, print64);
41.
42.     free(x);
43.     free(y);
44.     free(z);
45.
46.     printf("\n================================================================================\n");
47. }
48.
49. int v4(void *px, void *py, void *pz, size_t N) {
50.     if (N % 2 != 0) return -1;
51.
52.     asm volatile (
53.         "xor %%rcx, %%rcx\n\t"          // i = 0
54.         "vbroadcastsd %4, %%ymm3\n\t"   // ymm3 = 2.0
55.         "1:\n\t"
56.         "cmp %3, %%rcx\n\t"
57.         "jge 2f\n\t"
58.         "vmovupd (%0, %%rcx, 8), %%ymm0\n\t"
59.         "vmovupd (%1, %%rcx, 8), %%ymm1\n\t"
60.         "vdivpd %%ymm1, %%ymm0, %%ymm2\n\t"
61.         "vaddpd %%ymm3, %%ymm2, %%ymm2\n\t"
62.         "vmovupd %%ymm2, (%2, %%rcx, 8)\n\t"
63.         "add $4, %%rcx\n\t"
64.         "jmp 1b\n\t"
65.         "2:\n\t"
66.         :
67.         : "r"(px), "r"(py), "r"(pz), "r"(N), "x"(2.0)
68.         : "rcx", "ymm0", "ymm1", "ymm2", "ymm3", "memory"
69.     );
70.
71.     return (int)N;
72. }
73.
```

Задание Л8.№4.

Разработайте ассемблерную функциюint $v1$(void *px, void * py, void * pz, size_t N), аналогичную Л8.№3 для произвольной длины N.

Проверьте, что массив z корректно заполняется (то есть ячейки от $pz[0]$ до $pz[N-1]$ перезаписываются верными значениями, а $pz[N]$ и далее не изменяются) при $N \in \{4k, 4k+1, 4k+2, 4k+3\}$ для выбранного k.

```
Задание №4
================================================================================
SystemInfo
================================================================================
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
================================================================================

--------Тест N=4, Limit=4, ControlValue=-1.111--------
                    Тест пройден.

--------Тест N=5, Limit=4, ControlValue=-1.111--------
                    Тест пройден.

--------Тест N=6, Limit=4, ControlValue=-1.111--------
                    Тест пройден.

--------Тест N=7, Limit=4, ControlValue=-1.111--------
                    Тест пройден.

--------Тест N=8, Limit=8, ControlValue=-1.111--------
                    Тест пройден.

--------Тест N=9, Limit=8, ControlValue=-1.111--------
                    Тест пройден.

--------Тест N=10, Limit=8, ControlValue=-1.111--------
                    Тест пройден.

================================================================================
```

Рис. 5: результат тестирования v1

Листинг:

Файл task8_4.c:

```c
int Equals(double a, double b);

void test_v1(size_t N);
int v1(void *px, void *py, void *pz, size_t N);

void run_task8_4()
{
    printf("\nЗадание №4\n");
    printf("================================================================================");
    printSystemInfo();
    printf("================================================================================\n");

    const size_t k = 1;

    for (int test = 0; test < 7; test++) {
        size_t N = 4*k + test;
        test_v1(N);
    }

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

int v1(void *px, void *py, void *pz, size_t N) {
    size_t limit = N - (N % 4);

    asm volatile (
        "xor %%rcx, %%rcx\n\t"          // i = 0
        "vbroadcastsd %5, %%ymm3\n\t"   // ymm3 = 2.0
        "1:\n\t"
        "cmp %4, %%rcx\n\t"
        "jge 2f\n\t"
        "vmovupd (%0, %%rcx, 8), %%ymm0\n\t"
        "vmovupd (%1, %%rcx, 8), %%ymm1\n\t"
        "vdivpd %%ymm1, %%ymm0, %%ymm2\n\t"
        "vaddpd %%ymm3, %%ymm2, %%ymm2\n\t"
        "vmovupd %%ymm2, (%2, %%rcx, 8)\n\t"
        "add $4, %%rcx\n\t"
        "jmp 1b\n\t"
        "2:\n\t"
        :
        : "r"(px), "r"(py), "r"(pz), "r"(N), "r"(limit), "x"(2.0)
        : "rcx", "ymm0", "ymm1", "ymm2", "ymm3", "memory"
    );

    return (int)N;
}

void test_v1(size_t N) {

    size_t limit = N - (N % 4);
    const double controlValue = -1.1111;

    printf("\n--------Тест N=%zu, Limit=%zu, ControlValue=%.3f--------\n", N, limit,
controlValue);

    double *x = (double*)aligned_alloc(32, N * sizeof(double));
    double *y = (double*)aligned_alloc(32, N * sizeof(double));
    double *z = (double*)aligned_alloc(32, N * sizeof(double));

    if (!x || !y || !z) {
        printf("Ошибка выделения памяти\n");
        return;
    }

    // Инициализация x и y для первых N элементов
    for (size_t i = 0; i < limit; i++) {
        x[i] = i + 1.0;
        y[i] = (i + 1) * 2.0;
    }
    // Инициализация защитных элементов (после N) для x, y и z
    for (size_t i = limit; i < N; i++) {
        x[i] = 12345.6789;
        y[i] = 98765.4321;
        z[i] = controlValue; // контрольное значение
```

```
 74.     }
 75.
 76.     // Запускаем функцию
 77.     int res = v1(x, y, z, N);
 78.
 79.     // Проверяем корректность вычислений для первых N%4 элементов
 80.     int error = 0;
 81.     for (size_t i = 0; i < limit; i++) {
 82.         if (Equals(z[i], controlValue)) {
 83.             printf("Ошибка вычисления z[%zu]: получили %f\n", i, z[i]);
 84.             error = 1;
 85.         }
 86.     }
 87.     // Проверяем, что N%4  элементы не изменились
 88.     for (size_t i = limit; i < N; i++) {
 89.         if (z[i] != controlValue) {
 90.             printf("Ошибка: z[%zu] изменён, значение %f\n", i, z[i]);
 91.             error = 1;
 92.         }
 93.     }
 94.
 95.     if (!error) {
 96.         printf("\t\t\t\t\tТест пройден.\n");
 97.     }
 98.     else{
 99.         printf("\nТест завален.\n Массив расчитанных Z\n");
100.         PRINT_ARRAY(z, N, print64);
101.     }
102.
103.     free(x); free(y); free(z);
104. }
105.
106. int Equals(double a, double b) {
107.     return memcmp(&a, &b, sizeof(double)) == 0;
108. }
109.
```