

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования

**«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)**

Факультет систем управления (ФСУ)

Кафедра автоматизированных систем управления (АСУ)

Массивы и векторные операции.

Отчет по лабораторной работе №5 по дисциплине
«Вычислительная техника»

Студент гр. 431-3
_____ Гурулёв А.В
«18» декабря 2022 г.

Руководитель
_____ Алфёров С.М.
«__» _____ 2022 г.

Томск 2022

Оглавление

ВВЕДЕНИЕ	3
1 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ	4
ВЫВОД	6
Приложение А	7

ВВЕДЕНИЕ

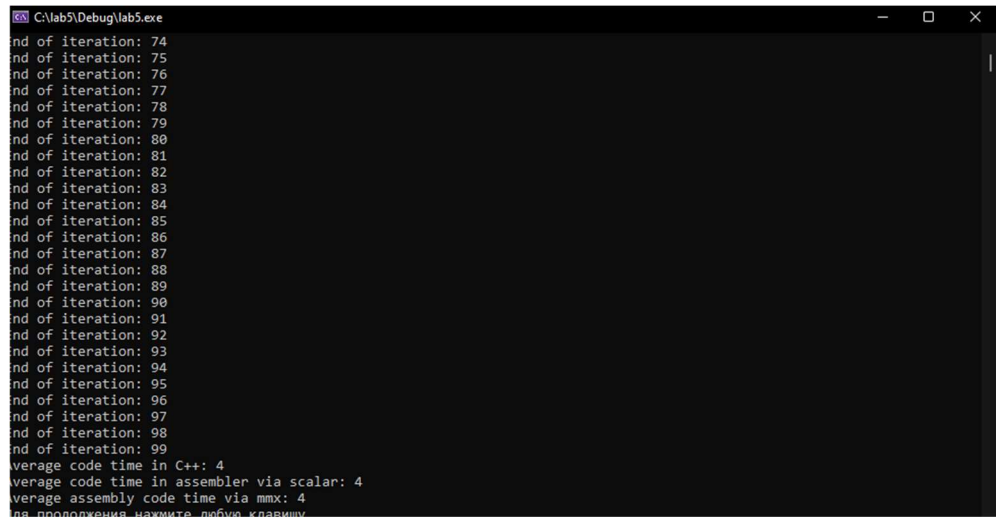
Цель работы – Научиться обрабатывать массивы данных на языке Ассемблер. Познакомиться с векторными операциями процессора.

Задание:

- Задание состоит из трёх частей. В первой части нужно выполнить задание на языке c++. Во второй части требуется написать программу обработки массива согласно заданию, используя скалярные команды обработки данных. Во второй части требуется написать такую же программу, но используя векторные операции. Для каждой программы засечь время выполнения, провести не менее 100 замеров. Вычислить среднее время выполнения для каждой программы и сделать вывод об эффективности векторных операций. На выполнение всего задания выделяется 8 часов аудиторного времени.
- 7) Увеличить яркость вертикальными полосами по 64 пикселя.

1 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

На рисунке 1.1-1.2 представлен результат работы программы на картинке 800x600.



```
C:\lab5\Debug\lab5.exe
nd of iteration: 74
nd of iteration: 75
nd of iteration: 76
nd of iteration: 77
nd of iteration: 78
nd of iteration: 79
nd of iteration: 80
nd of iteration: 81
nd of iteration: 82
nd of iteration: 83
nd of iteration: 84
nd of iteration: 85
nd of iteration: 86
nd of iteration: 87
nd of iteration: 88
nd of iteration: 89
nd of iteration: 90
nd of iteration: 91
nd of iteration: 92
nd of iteration: 93
nd of iteration: 94
nd of iteration: 95
nd of iteration: 96
nd of iteration: 97
nd of iteration: 98
nd of iteration: 99
Average code time in C++: 4
Average code time in assembler via scalar: 4
Average assembly code time via mmx: 4
на продолжение нажмите любую клавишу
```

Рисунок 1.1 - Результат работы программы

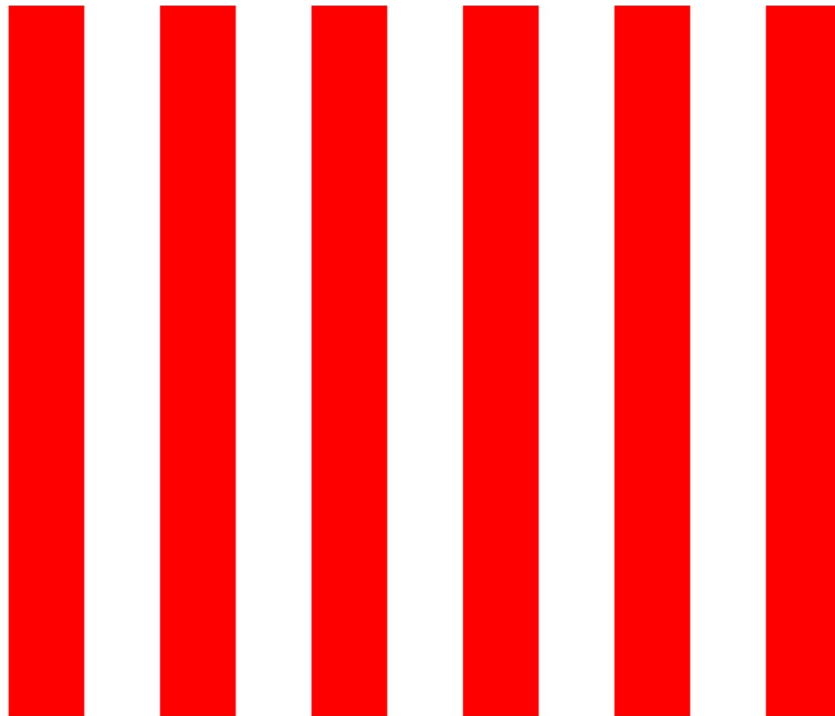


Рисунок 1.2 - Итоговые изображения

На рисунке 1.3-1.4 представлен результат работы программы на картинке 5184x3456.

```
C:\lab5\Debug\lab5.exe
End of iteration: 74
End of iteration: 75
End of iteration: 76
End of iteration: 77
End of iteration: 78
End of iteration: 79
End of iteration: 80
End of iteration: 81
End of iteration: 82
End of iteration: 83
End of iteration: 84
End of iteration: 85
End of iteration: 86
End of iteration: 87
End of iteration: 88
End of iteration: 89
End of iteration: 90
End of iteration: 91
End of iteration: 92
End of iteration: 93
End of iteration: 94
End of iteration: 95
End of iteration: 96
End of iteration: 97
End of iteration: 98
End of iteration: 99
Average code time in C++: 198
Average code time in assembler via scalar: 221
Average assembly code time via mmx: 188
Для продолжения нажмите любую клавишу . . .
```

Рисунок 1.3 - Результат работы программы

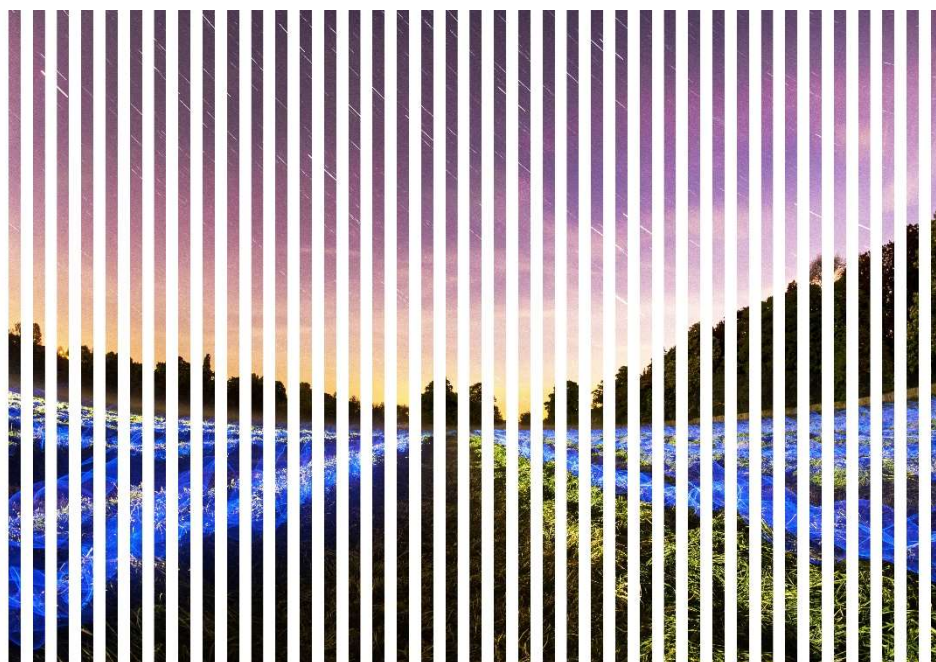


Рисунок 1.4 - Итоговые изображения

ВЫВОД

Векторные операции позволяют работать с массивами данных эффективнее за счёт команд, которые не получится использовать при работе с скалярными операциями, а так же, за счёт возможности работы сразу с несколькими элементами массива.

Приложение А

Листинг кода

```
#include <iostream>
#include <stdlib.h>
#include <conio.h>

using namespace std;

//7 Вариант. Увеличить яркость вертикальными полосами по 64 пикселя.
int main()
{
    int sumtimec = 0;
    int sumtimes = 0;
    int sumtimev = 0;
    time_t start;
    for (int i = 0; i < 100; i++)
    {
        //Начало замера
        start = clock();

        //Блок с++. Считывание и запись заголовка
        FILE* in, * out;
        unsigned __int8* buffer;
        unsigned __int32 wight, height;

        fopen_s(&in, "test.bmp", "rb");
        buffer = (unsigned __int8*)malloc(54);
        fread(buffer, 1, 54, in);

        for (int i = 0; i <= 3; i++)
        {
            wight = buffer[21 - i] | wight << 8;
            height = buffer[25 - i] | height << 8;
        }

        if (i == 0)
        {
            cout << wight << "x" << height << endl;
        }

        fopen_s(&out, "test2_cpp.bmp", "wb");
        fwrite(buffer, 1, 54, out);

        free(buffer);

        //Блок с++. Считывание, модификация и запись цвета
        buffer = (unsigned __int8*)malloc(wight * height * 4);

        fread(buffer, 1, wight * height * 4, in);
        fclose(in);

        for (int i = 0; i < wight; i += 128)
        {
            for (int j = 0; j < height; j++)
            {
                for (int k = i; k < (i + 64); k++)
                {
                    buffer[wight * 3 * j + k * 3 + 0] = 255;
                    buffer[wight * 3 * j + k * 3 + 1] = 255;
                    buffer[wight * 3 * j + k * 3 + 2] = 255;
                }
            }
        }
    }
}
```

```

    }
}

fwrite(buffer, 1, wight * height * 4, out);
fclose(out);

free(buffer);

//Конец замера
sumtimec += clock() - start;

//Начало замера
start = clock();

//Блок ассемблера. Скаляр. Считывание и запись заголовка

in = NULL;
out = NULL;
buffer = NULL;
wight = 0;
height = 0;

fopen_s(&in, "test.bmp", "rb");
buffer = (unsigned __int8*)malloc(54);
fread(buffer, 1, 54, in);

//Получим высоту и ширину
__asm
{
    mov esi, buffer;                //Указатель на массив

    mov ecx, 3;                    //Начинаем извлекать высоту
    и ширину

    mov ebx, 18;
    mov eax, [esi + ebx];
    mov wight, eax;
    mov eax, [esi + ebx + 4];
    mov height, eax;
}

fopen_s(&out, "test2_scal.bmp", "wb");
fwrite(buffer, 1, 54, out);

free(buffer);

//Блок ассемблера. Скаляр. Считывание, модификация и запись цвета
buffer = (unsigned __int8*)malloc(wight * height * 4);

fread(buffer, 1, wight * height * 4, in);
fclose(in);

__asm
{
    //Получаем адресс из буфера
    mov esi, buffer;

    //Делаем счётчик для 1 цикла
    mov ecx, wight;

l1:
    //Убираем счётчик в стек
    push ecx;

    //Делаем новый счётчик для 2 цикла

```



```

mov ecx, height;

12:
//Подготавливаем часть индекса, который используется в 3 цикле
(wight*3*j)
mov eax, 3;
mul wight;
mov ebx, height;
sub ebx, ecx;
mul ebx;
mov ebx, eax;

//Подготавливаем остальную часть индекса(+i*3). Тк там используется
значения из 1 цикла, делаем махинации с стеком
mov eax, ecx; //Сохраняем текущее значение счётчика
pop ecx;      //Достаём значения счётчика с прошлого цикла
push eax;     //Тк нам понадобится eax, сохраняем значения те-
кущего счётчика в стеке

mov eax, wight;
sub eax, ecx;
mov edx, 3;
mul edx;
add ebx, eax;
mov eax, 0;

//Возвращаем значения счётчика 1 цикла в стек и достаём значения те-
кущего

pop eax;
push ecx;
mov ecx, eax;
mov eax, 0;

push ecx;
mov ecx, 192;

13:
//Освещаем пиксель
mov[esi + ebx], 255;
inc ebx;

//Циклы
loop 13;

pop ecx;      //Забираем счётчик 2 цикла из стека
loop 12;

pop ecx;      //Забираем счётчик 3 цикла из стека
sub ecx, 127; //Смещаемся не на 1 пиксель, а на 64*2-1, тк взаимодей-
ствие с 64 нужными было в 3 цикле
cmp ecx, 0;   //Если наше смещение привело к отрицательно-
му(или равному нулю) ecx - выходим из 1 цикла
jle ex;
loop 11;
ex:
}

fwrite(buffer, 1, wight * height * 4, out);
fclose(out);

free(buffer);

//Конец замера
sumtimes += clock() - start;

```

```

//Начало замера
start = clock();

//Блок ассемблера. Вектор. Считывание и запись заголовка

in = NULL;
out = NULL;
buffer = NULL;
wight = 0;
height = 0;

fopen_s(&in, "test.bmp", "rb");
buffer = (unsigned __int8*)malloc(54);
fread(buffer, 1, 54, in);

//Получим высоту и ширину
__asm
{
    mov esi, buffer;                //Указатель на массив

    mov ecx, 3;                    //Начинаем извлекать высоту
и ширину

    mov ebx, 18;
    mov eax, [esi + ebx];
    mov wight, eax;
    mov eax, [esi + ebx + 4];
    mov height, eax;
}

fopen_s(&out, "test2_vect.bmp", "wb");
fwrite(buffer, 1, 54, out);

free(buffer);

//Блок ассемблера. Вектор. Считывание, модификация и запись цвета
buffer = (unsigned __int8*)malloc(wight * height * 4);

fread(buffer, 1, wight * height * 4, in);
fclose(in);

__asm
{
    //Получаем адресс из буфера
    mov esi, buffer;

    //Делаем счётчик для 1 цикла
    mov ecx, wight;

l11:
    //Убираем счётчик в стек
    push ecx;

    //Делаем новый счётчик для 2 цикла
    mov ecx, height;

l12:
    //Подготавливаем часть индекса, который используется в 3 цикле
(wight*3*j)
    mov eax, 3;
    mul wight;
    mov ebx, height;
    sub ebx, ecx;
    mul ebx;
    mov ebx, eax;

```

```

        //Подготавливаем остальную часть индекса(+i*3). Тк там используется
значения из 1 цикла, делаем махинации с стеком
        mov eax, ecx; //Сохраняем текущее значение счётчика
        pop ecx;      //Достаём значения счётчика с прошлого цикла
        push eax;      //Тк нам понадобится eax, сохраняем значения те-
кущего счётчика в стеке

        mov eax, wight;
        sub eax, ecx;
        mov edx, 3;
        mul edx;
        add ebx, eax;
        mov eax, 0;

        //Возвращаем значения счётчика 1 цикла в стек и достаём значения те-
кущего

        pop eax;
        push ecx;
        mov ecx, eax;
        mov eax, 0;

        push ecx;
        mov ecx, 48;

l13:
        //Освещаем пиксель
        movd mm0, [esi + ebx];
        mov eax, 0xFFFFFFFF;
        movd mm1, eax;
        paddusb mm0, mm1;
        movd[esi + ebx], mm0;
        add ebx, 4;

        //Циклы
        loop l13;

        pop ecx;      //Забираем счётчик 2 цикла из стека
        loop l12;

        pop ecx;      //Забираем счётчик 3 цикла из стека
        sub ecx, 127; //Смещаемся не на 1 пиксель, а на 64*2-1, тк взаимодей-
ствие с 64 нужными было в 3 цикле
        cmp ecx, 0;    //Если наше смещение привело к отрицательно-
му(или равному нулю) ecx - выходим из 1 цикла
        jle ex1;
        loop l11;

ex1:
}

fwrite(buffer, 1, wight * height * 4, out);
fclose(out);

free(buffer);

//Конец замера
sumtimev += clock() - start;

cout << "\nEnd of iteration: " << i;
}

cout << "\nAverage code time in C++: " << sumtimec / 100 << "\nAverage code time
in assembler via scalar: " << sumtimes / 100 << "\nAverage assembly code time via mmx: "
<< sumtimev / 100 << endl;

system("pause");

```

```
    return 0;  
}
```