МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

**«Организация связи модуля на ассемблере с модулями, созданными на языке высокого уровня»**

Отчет по лабораторной работе № 5 по дисциплине

«Технологии программирования»

Выполнил студент группы: ПИб-2301\_\_\_\_\_\_\_\_\_\_\_\_\_ /Морозов Д.А.

Проверил преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Долженкова М.Л.

**Киров 2023**

**Цели работы:**

На основе лабораторной работы 3 модифицировать приложение. Вынести  одну из вычислительных операций и переписать ее в виде модуля на ассемблере. Обеспечить вызов внешней ассемблерной функции из основной программы.  
Сделать выводы

**Задание:**

Создать ассеблерный модуль (asm), содержащий в себе поиск минимального количества цветов для раскраски графов

**Подключение ассемблерного модуля:**

Подключение ассемблерного модуля в программу на C++ с использованием технологий cdecl и MASM x86 включает несколько шагов. Вот основные этапы, которые необходимо выполнить:

1. Создание ассемблерного модуля: написать код на языке ассемблера, используя синтаксис MASM x86. Этот код будет содержать реализацию требуемой функции на ассемблере. Например, функция с именем `GC`, которая принимает аргументы и возвращает результат.

2. Экспортирование функции: В ассемблерном модуле нужно указать, что функция `GC` должна быть доступна извне. Для этого используется директива `PUBLIC` Например:

```

PUBLIC GC

```

3. Сохранение ассемблерного модуля: сохраняем ассемблерный код в отдельный файл с расширением `.asm`.

4. Объявление прототипа функции: В главном файле C++, объявляем прототип ассемблерной функции `GC`. Прототип должен соответствовать аргументам и возвращаемому значению функции на ассемблере. Например:

```

extern "C" int GC(int(\*)[6], int numVertices, int\*);

```

Здесь `extern "C"` указывает, что функция использует соглашение о вызовах

5. Использование ассемблерной функции: теперь мы можем использовать функцию `GC` в своей основной программе на C++. Вызывайте ее так, как вызываем любую другую функцию на C++.

Таким образом, подключение ассемблерного модуля в программу C++ с использованием технологий cdecl и MASM x86 включает написание ассемблерного кода, экспортирование функции, объявление прототипа функции в главном файле C++, связывание и использование ассемблерной функции в основной программе.

**Состояние стека:**

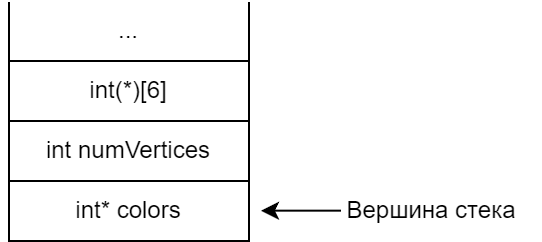


Рис.1 - состояние стека на момент выполнения первой команды

Так как было использовано соглашение о вызовах cdecl – аргументы функции записались в обратном порядке (справа налево).

**Листинг кода:**

WNDmain.h

#pragma once

#include<Windows.h>

#include<string>

//макросы для облегчения работы

#define OnButtonClicked 1

#define TextBufferSize 50

#define DlgIndexNumberA 200

#define DlgIndexNumberB 210

char Buffer[TextBufferSize];

int readChar;

//беззнаковое целое число

unsigned numA;

unsigned numB;

//extern "C" int GraphColoring(int(\*)[6]);

extern "C" int GC(int(\*)[6], int numVertices, int\*);

//инициализация оконных переменных для передачи их значений между функциями

HWND childText;

HWND hEditControl;

HWND hStaticControl;

HWND hStaticControlNums;

HWND hNumberAControl;

HWND hNumberBControl;

HWND hWnd;

//инициализация необходимых глобальных переменных

int matrix[6][6];

int colors[6] = { 1, 1, 1, 1, 1, 1 };

std::string StrMat;

std::string path;

//прототипы функций

void MainWndAddWidgets(HWND hWnd);

LRESULT CALLBACK SoftwareChildProcedure(HWND childWindow, UINT msg, WPARAM wp, LPARAM lp);

//конструктор класса главного окна

WNDCLASS NewWindowClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, HICON Icon, LPCWSTR Name, WNDPROC Procedure) {

WNDCLASS NWC = { 0 };

NWC.hCursor = Cursor;

NWC.hIcon = Icon;

NWC.hInstance = hInst;

NWC.lpszClassName = Name;

NWC.hbrBackground = BGColor;

NWC.lpfnWndProc = Procedure;

return NWC;

}

//конструктор класса дочернего окна

WNDCLASS NewWindowChildClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, HICON Icon, LPCWSTR Name, WNDPROC ProcedureChild) {

WNDCLASS NWC = { 0 };

NWC.hCursor = Cursor;

NWC.hIcon = Icon;

NWC.hInstance = hInst;

NWC.lpszClassName = Name;

NWC.hbrBackground = BGColor;

NWC.lpfnWndProc = ProcedureChild;

return NWC;

}

//процедура преобразования текстовой матрицы в матрицу типа int

void ParseMatrix()

{

int k = 0;

for (int i = 0; i < 6; i++) {

for (int j = 0; j < 6; j++) {

if (StrMat[k] == '0') { matrix[i][j] = 0; };

if (StrMat[k] == '1') { matrix[i][j] = 1; };

k++;

}

}

}

//проверка корректности введенных значений

bool ValidateMatrix() {

if (StrMat.size() != 36) {

return false;

}

for (int i = 0; i < 36; i++) {

//если символ не 1 и не 0, то проверка не пройдена

if (StrMat[i] != 48 && StrMat[i] != 49) { return false; };

}

return true;

}

int countVertices(const int matrix[6][6]) {

int numVertices = 0;

for (int i = 0; i < 6; i++) {

bool hasNonZeroElement = false;

for (int j = 0; j < 6; j++) {

if (matrix[i][j] != 0) {

hasNonZeroElement = true;

break;

}

}

if (hasNonZeroElement) {

numVertices++;

}

}

return numVertices;

}

//процедура обработки сообщений главного окна

LRESULT CALLBACK SoftwareMainProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp) {

switch (msg) {

case WM\_COMMAND:

switch (wp)

{

//при нажатии кнопки

case OnButtonClicked:

//получение значений из полей выбора вершин

readChar = GetWindowTextA(hEditControl, Buffer, TextBufferSize);

//замена переносов строк в текстовой матрице

StrMat = Buffer;

StrMat.erase(std::remove(StrMat.begin(), StrMat.end(), '\r'), StrMat.end());

StrMat.erase(std::remove(StrMat.begin(), StrMat.end(), '\n'), StrMat.end());

//запуск проверок корректности ввода

if (ValidateMatrix()) {

SetWindowTextA(hStaticControl, "Матрица задана корректно");

ParseMatrix();

}

else {

SetWindowTextA(hStaticControl, "Матрица задана некорректно!");

};

if (ValidateMatrix()) {

//выключаем главное окно

//это необходимо, так как дочернее окно должно быть модальным

EnableWindow(hWnd, FALSE);

int numVertices = countVertices(matrix);

int result;

if (numVertices == 0) {

result = 0;

}

else {

result = GC(matrix, numVertices, colors);

};

//int result = GraphColoring(matrix);

path = "\nThe minimum number of colors needed is " + std::to\_string(result) + "\n";

//создается дочернее окно

HWND childWindow = CreateWindowEx(

0, L"ChildWndClass", L"Дочернее окно", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE,

300, 300, 300, 150, hWnd, NULL, NULL, NULL

);

//проверка на случай если есть проблемы с созданием дочернего окна

if (childWindow == NULL) {

MessageBox(childWindow, L"Не удалось создать дочернее окно", L"Ошибка", MB\_OK | MB\_ICONERROR);

return -1;

}

}

break;

default: break;

}

break;

//при создании

case WM\_CREATE:

//процедура с созданием необходимых компонентов главного окна

MainWndAddWidgets(hWnd);

break;

//при уничтожении

case WM\_DESTROY:

//этот код передастся в цикл, после чего приложение закроется

PostQuitMessage(0);

break;

default: return DefWindowProc(hWnd, msg, wp, lp);

}

}

//процедура обработки сообщений дочернего окна

LRESULT CALLBACK SoftwareChildProcedure(HWND childWindow, UINT msg, WPARAM wp, LPARAM lp) {

switch (msg) {

case WM\_CREATE: {

//создание статического текста, куда будет записан разультат Дийкстры

if (!IsWindow(childText)) {

childText = CreateWindowA("STATIC", "", WS\_CHILD | WS\_VISIBLE,

0, 0, 300, 300, childWindow, NULL, NULL, NULL);

}

//вывод на дочернее окно ответа

SetWindowTextA(childText, path.c\_str());

break; }

//при закрытии

case WM\_CLOSE:

//включаем главное окно и удаляем дочернее

EnableWindow(hWnd, TRUE);

DestroyWindow(childWindow);

break;

default: return DefWindowProc(childWindow, msg, wp, lp);

}

}

void MainWndAddWidgets(HWND hWnd) {

hStaticControl = CreateWindowA("static", "Введите матрицу смежности(6 на 6)", WS\_VISIBLE | WS\_CHILD, 0, 0, 490, 20, hWnd, NULL, NULL, NULL, NULL);

hEditControl = CreateWindowA("edit", "000000\r\n000000\r\n000000\r\n000000\r\n000000\r\n000000", WS\_VISIBLE | WS\_CHILD | ES\_MULTILINE | ES\_NUMBER, 10, 40, 55, 100, hWnd, NULL, NULL, NULL, NULL);

//в таких компонентах как кнопка необходимо указывать идентификаторы для обработки нажатия

//в данном случае это макрос OnButtonClicked со значением 1

CreateWindowA("button", "Посчитать", WS\_VISIBLE | WS\_CHILD | ES\_CENTER, 200, 140, 100, 30, hWnd, (HMENU)OnButtonClicked, NULL, NULL, NULL);

}

assemblyModule.asm

.model flat, C

.data

colors DWORD 0, 0, 0, 0, 0, 0 ; Внутренний массив для хранения цветов

.code

; eax -> кол-во вершин

; ebx -> указатель на массив matrix[6][6]

GC PROC

;--------------------- INIT ------------------------------

push ebp

mov ebp, esp

mov ebx, [ebp + 8] ; указатель на первый элемент массива matrix

mov edx, [ebp + 12] ; записали в edx количество вершин

;--------------------- INIT ------------------------------

;--------------------- NULLING COLORS ------------------------------

mov eax, 0

nulling:

mov esi, dword ptr [OFFSET colors + eax \* 4]

mov dword ptr [OFFSET colors + eax \* 4], 0

inc eax

cmp eax, 6

jne nulling

;--------------------- NULLING COLORS ------------------------------

;--------------------- MAIN FUNC ------------------------------

start:

mov eax, 0 ; eax = 0 ----------------- i

for1:

cmp eax, edx

jge endfor1

mov ecx, 0 ; ecx = 0 ------------- j

for2:

cmp ecx, edx

jge endfor2

mov esi, ebx ; esi получает указатель на matrix

mov edi, OFFSET colors ; edi получает указатель на colors

lea esi, [esi + eax \* 4]

lea esi, [esi + ecx \* 4]

movzx esi, word ptr [esi] ; Загружаем значение matrix[i][j] в регистр esi

mov edi, [OFFSET colors + ecx \* 4] ; Загружаем значение colors[j] в регистр edi

cmp esi, 1 ; matrix[i][j] == 1

je step1fine

jmp hef2

step1fine:

cmp edi, [OFFSET colors + ecx \* 4] ; colors[i] == colors[j]

je colorinc

jmp hef2

colorinc:

mov esi, [OFFSET colors + ecx \* 4] ; Загрузка значения colors[j] в регистр esi

inc esi ; Увеличение значения в регистре esi на 1

mov [OFFSET colors + ecx \* 4], esi ;

cmp ecx, 0

hef2:

inc ecx

jmp for2

endfor2:

inc eax

jmp for1

endfor1:

;--------------------- MAIN FUNC ------------------------------

;--------------------- FIND MAX ------------------------------

; Найти максимальное значение в массиве colors

mov ecx, 6 ; Загружаем количество элементов в массиве colors

mov ebx, OFFSET colors ; Загружаем указатель на массив colors

mov eax, [ebx] ; Инициализируем eax значением первого элемента colors

find\_max:

cmp ecx, 1 ; Сравниваем счетчик с 1

jle endfindmax ; Если остался только один элемент, переходим к завершению

add ebx, 4 ; Увеличиваем указатель на 4 байта для доступа к следующему элементу

cmp eax, dword ptr [ebx] ; Сравниваем текущий максимум с текущим элементом

jge skip\_update ; Если текущий максимум больше или равен текущему элементу, пропускаем обновление

mov eax, dword ptr [ebx] ; Обновляем текущий максимум

skip\_update:

loop find\_max ; Продолжаем цикл для остальных элементов

endfindmax:

; Теперь максимальное значение находится в регистре eax

;--------------------- FIND MAX ------------------------------

;--------------------- RETURN ------------------------------

mov esp, ebp

pop ebp

ret

;--------------------- RETURN ------------------------------

GC ENDP

end

**Экранные формы:**

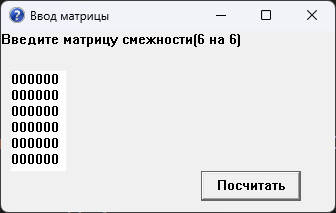


Рис.2 – главное окно при запуске

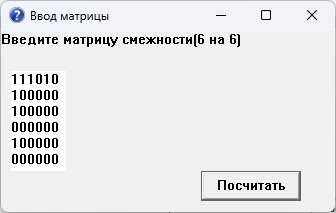


Рис. 3 – главное окно с заданными значениями

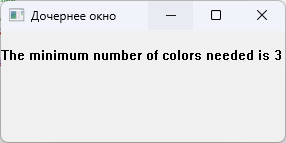


Рис.4 – дочернее окно с ответом

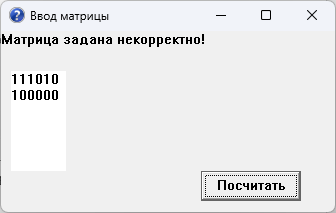


Рис. 5 – обработка некорректно ввода матрицы

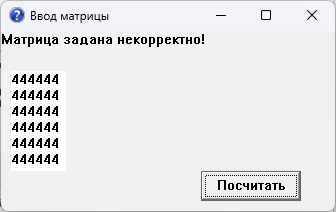


Рис. 6 – обработка некорректно ввода матрицы

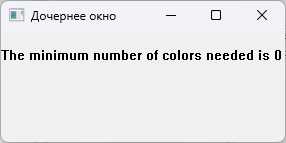


Рис. 8 – ответ в случае, когда матрица заполнена нулями (вершин нет)

**Вывод:**

Проделанная работа состояла в модификации приложения путем выноса одной из вычислительных операций и переписывания ее в виде модуля на ассемблере. Целью было обеспечить вызов внешней ассемблерной функции из основной программы.

Для достижения данной цели были выполнены следующие шаги:

1. Изолирование вычислительной операции: изначально, в исходном коде приложения, определенная операция была выделена и вынесена в отдельную функцию.
2. Написание ассемблерного модуля: Создание нового файлового модуля на языке ассемблера, в котором была реализована вычислительная операция из пункта 1. Данный модуль содержит код, написанный на ассемблере, который выполняет необходимые вычисления.
3. Сборка и связывание: Сборка основного приложения и ассемблерного модуля вместе с помощью компилятора и линкера. При связывании основной программы и ассемблерного модуля, основная программа получает доступ к функции, реализованной на ассемблере.
4. Вызов внешней ассемблерной функции: В основной программе был добавлен вызов внешней ассемблерной функции, реализованной в ассемблерном модуле. Это позволяет основной программе использовать функциональность, предоставляемую ассемблерной функцией.