****

IMPLEMENTATION OF A SIMPLE GUI-BASED CALCULATOR USING A

HIGH LEVEL PROGRAMMING LANGUAGE AND

ASSEMBLY LANGUAGE

BY

JOEL (ENG)

MASCOT (ENG)

CALEB (ENG)

SLIM SHADY (ENG)

FOR

CPE457

DEPARTMENT OF COMPUTER ENGINEERING

FEDERAL UNIVERSITY OF BENIN, UGBOWO, EDO STATE

JULY 25, 2019

Contents

[ABSTRACT 1](#_Toc15243866)

[INTRODUCTION 1](#_Toc15243867)

[OBJECTIVE 1](#_Toc15243868)

[METHOD 1](#_Toc15243869)

[Tools 1](#_Toc15243870)

[Code 2](#_Toc15243871)

[C++ 2](#_Toc15243872)

[Assembly 17](#_Toc15243873)

[RESULT 18](#_Toc15243874)

[CONCLUSION 18](#_Toc15243875)

[REFERENCES 18](#_Toc15243876)

**Table of Figures**

[Figure 1 General Concept 2](#_Toc15243855)

**Table of Listings**

[Listing 1: C++ 16](#_Toc15243860)

[Listing 2: Assembly 17](#_Toc15243861)

# ABSTRACT

Graphical User Interface (GUI) usually requires a lot of code to implement; so, sometimes, even if the programmer wishes to write his program in assembly for efficiency sake, he/she still has to write the GUI code in a high level language, for the sake of simplicity, and a reduced code-size. The program described in this paper is a simply calculator desktop application whose user interface is written in win32 C, and the calculation logic is written Intel x86 assembly.

# INTRODUCTION

Every system has a specific calling convention for calling functions. 64-bit Windows uses the *Microsoft x64 Calling Convention*. This convention uses rcx, rdx, r8 and r9 for the first four integer parameters, or xmm0 to xmm3 for the first four floating point parameters. If a function requires more parameters, they are pushed onto the stack in reverse order. For instructions that return a value, integer return values are stored in rax, while floating point return values are stored in xmm0 (Seyfarth, 2011). It should be noted that the 64-bit division uses the RDX:RAX register pair to hold the dividend, and the quotient is found in RAX and the remainder is in RDX after the division (Brey, 2008). In our program, the function is called twice for each division. One call returns the quotient immediately after the division, since the quotient is already in RAX, while the other moves the remainder into RAX before returning.

# OBJECTIVE

1. Call and assembly routine from a C++ console program.
2. Perform array summation in Intel x86\_64 assembly.

# METHOD

## Tools

1. Visual Studio 2017
2. Laptop with Windows 10 OS installed.

Figure 1 General Concept

## Code

### C++

// 475\_win32\_Desktop\_ASM.cpp : Defines the entry point for the application.

//

#include "stdafx.h"

#include "475\_win32\_Desktop\_ASM.h"

#define MAX\_LOADSTRING 100

#define ID\_BUTTON\_ADD 0X8801

#define ID\_BUTTON\_SUBTRACT 0X8802

#define ID\_BUTTON\_MULTIPLY 0X8803

#define ID\_BUTTON\_DIVIDE 0X8804

#define ID\_EDIT\_FIRST\_OPERAND 0X8805

#define ID\_EDIT\_SECOND\_OPERAND 0X8806

#define ID\_EDIT\_RESULT 0X8807

#define ID\_EDIT\_RESULT\_REMAINDER 0X8808

void add(HWND, HWND, HWND, HWND);

void subtract(HWND, HWND, HWND, HWND);

void multiply(HWND, HWND, HWND, HWND);

void divide(HWND, HWND, HWND, HWND, HWND);

extern "C" int addInts(int a, int b);

extern "C" int subtractInts(int a, int b);

extern "C" int multiplyInts(int a, int b);

extern "C" int quotientInts(int a, int b);

extern "C" int remainderInts(int a, int b);

// Global Variables:

HINSTANCE hInst; // current instance

WCHAR szTitle[MAX\_LOADSTRING]; // The title bar text

WCHAR szWindowClass[MAX\_LOADSTRING]; // the main window class name

// Forward declarations of functions included in this code module:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

HFONT hfDefault;

HWND h\_Edit\_first\_operand;

HWND h\_Edit\_second\_operand;

HWND h\_Edit\_result;

HWND h\_Edit\_remainder;

HWND h\_Button\_add;

HWND h\_Button\_subtract;

HWND h\_Button\_multiply;

HWND h\_Button\_divide;

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Place code here.

// Initialize global strings

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_MY475WIN32DESKTOPASM, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Perform application initialization:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_MY475WIN32DESKTOPASM));

MSG msg;

// Main message loop:

while (GetMessage(&msg, nullptr, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

//

// FUNCTION: MyRegisterClass()

//

// PURPOSE: Registers the window class.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_MY475WIN32DESKTOPASM));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+2); // cHANGE THE INTEGER OFFSET TO CHANGE THE COLOUR

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_MY475WIN32DESKTOPASM);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

//

// FUNCTION: InitInstance(HINSTANCE, int)

//

// PURPOSE: Saves instance handle and creates main window

//

// COMMENTS:

//

// In this function, we save the instance handle in a global variable and

// create and display the main program window.

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Store instance handle in our global variable

HWND hWnd = CreateWindowW(

szWindowClass,

szTitle,

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT,

0,

400,

300,

nullptr,

nullptr,

hInstance,

nullptr);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

//

// FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)

//

// PURPOSE: Processes messages for the main window.

//

// WM\_COMMAND - process the application menu

// WM\_PAINT - Paint the main window

// WM\_DESTROY - post a quit message and return

//

//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Parse the menu selections:

switch (wmId)

{

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

case ID\_BUTTON\_ADD:

add(h\_Edit\_first\_operand, h\_Edit\_second\_operand, h\_Edit\_result, hWnd);

break;

case ID\_BUTTON\_SUBTRACT:

subtract(h\_Edit\_first\_operand, h\_Edit\_second\_operand, h\_Edit\_result, hWnd);

break;

case ID\_BUTTON\_MULTIPLY:

multiply(h\_Edit\_first\_operand, h\_Edit\_second\_operand, h\_Edit\_result, hWnd);

break;

case ID\_BUTTON\_DIVIDE:

divide(h\_Edit\_first\_operand, h\_Edit\_second\_operand, h\_Edit\_result, h\_Edit\_remainder, hWnd);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

// TODO: Add any drawing code that uses hdc here...

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

case WM\_CREATE:

{

h\_Edit\_first\_operand = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"EDIT", L"",

WS\_CHILD | WS\_VISIBLE | ES\_NUMBER,

20, 20, 50, 50, hWnd, NULL, GetModuleHandle(NULL), NULL);

if (h\_Edit\_first\_operand == NULL)

MessageBox(hWnd, L"Could not create first edit box.", L"Error", MB\_OK | MB\_ICONERROR);

h\_Edit\_second\_operand = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"EDIT", L"",

WS\_CHILD | WS\_VISIBLE | ES\_NUMBER,

80, 20, 50, 50, hWnd, NULL, GetModuleHandle(NULL), NULL);

if (h\_Edit\_first\_operand == NULL)

MessageBox(hWnd, L"Could not create second edit box.", L"Error", MB\_OK | MB\_ICONERROR);

h\_Edit\_result = CreateWindowEx(WS\_EX\_CLIENTEDGE,

L"EDIT", // Predefined class ; Unicode assumed

L"", // Content

WS\_CHILD | WS\_VISIBLE | WS\_HSCROLL, // Styles

140, // x position

20, // y position

100, // Width

50, // Height

hWnd, // Parent Window

NULL, // Menu

GetModuleHandle(NULL),

NULL); // Pointer not needed

if (h\_Edit\_first\_operand == NULL)

MessageBox(hWnd, L"Could not create result edit box.", L"Error", MB\_OK | MB\_ICONERROR);

h\_Edit\_remainder = CreateWindowEx(WS\_EX\_CLIENTEDGE,

L"EDIT", // Predefined class ; Unicode assumed

L"REMAINDER", // Content

WS\_CHILD | WS\_VISIBLE | WS\_HSCROLL, // Styles

260, // x position

20, // y position

100, // Width

50, // Height

hWnd, // Parent Window

NULL, // Menu

GetModuleHandle(NULL),

NULL); // Pointer not needed

if (h\_Edit\_first\_operand == NULL)

MessageBox(hWnd, L"Could not create remainder edit box.", L"Error", MB\_OK | MB\_ICONERROR);

h\_Button\_add = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"BUTTON", L"+",

WS\_CHILD | WS\_VISIBLE | BS\_DEFPUSHBUTTON | WS\_TABSTOP,

20, 100, 50, 50, hWnd, (HMENU) ID\_BUTTON\_ADD, GetModuleHandle(NULL), NULL);

if (h\_Button\_add == NULL)

MessageBox(hWnd, L"Could not create addition button.", L"Error", MB\_OK | MB\_ICONERROR);

h\_Button\_subtract = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"BUTTON", L"-",

WS\_CHILD | WS\_VISIBLE | BS\_DEFPUSHBUTTON | WS\_TABSTOP,

80, 100, 50, 50, hWnd, (HMENU)ID\_BUTTON\_SUBTRACT, GetModuleHandle(NULL), NULL);

if (h\_Button\_subtract == NULL)

MessageBox(hWnd, L"Could not create subtraction button.", L"Error", MB\_OK | MB\_ICONERROR);

h\_Button\_multiply = CreateWindowEx(WS\_EX\_CLIENTEDGE,

L"BUTTON", // Predefined class ; Unicode assumed

L"x", // Content

WS\_CHILD | WS\_VISIBLE | BS\_DEFPUSHBUTTON | WS\_TABSTOP, // Styles

140, // x position

100, // y position

50, // Width

50, // Height

hWnd, // Parent Window

(HMENU)ID\_BUTTON\_MULTIPLY, // Menu

GetModuleHandle(NULL),

NULL); // Pointer not needed

if (h\_Button\_multiply == NULL)

MessageBox(hWnd, L"Could not create multiplication button.", L"Error", MB\_OK | MB\_ICONERROR);

h\_Button\_divide = CreateWindowEx(WS\_EX\_CLIENTEDGE,

L"BUTTON", // Predefined class ; Unicode assumed

L"/", // Content

WS\_CHILD | WS\_VISIBLE | BS\_DEFPUSHBUTTON | WS\_TABSTOP, // Styles

200, // x position

100, // y position

50, // Width

50, // Height

hWnd, // Parent Window

(HMENU)ID\_BUTTON\_DIVIDE, // Menu

GetModuleHandle(NULL),

NULL); // Pointer not needed

if (h\_Button\_divide == NULL)

MessageBox(hWnd, L"Could not create division button.", L"Error", MB\_OK | MB\_ICONERROR);

}

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

// Message handler for about box.

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG:

return (INT\_PTR)TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

void add(HWND hwnd\_edit1, HWND hwnd\_edit2, HWND hwnd\_edit\_res, HWND hwnd\_parent)

{

int val1 = 0, val2 = 0, sum = 0;

int len = GetWindowTextLength(hwnd\_edit1);

if (len > 0)

{

int i = 0;

LPWSTR buf;

buf = (LPWSTR)GlobalAlloc(GPTR, len + 1);

GetWindowText(hwnd\_edit1, buf, len + 1);

val1 = \_wtoi(buf);

GlobalFree((HANDLE)buf);

i = 0;

buf;

buf = (LPWSTR)GlobalAlloc(GPTR, len + 1);

GetWindowText(hwnd\_edit2, buf, len + 1);

val2 = \_wtoi(buf);

GlobalFree((HANDLE)buf);

sum = addInts(val1, val2);

TCHAR szBuffer[20];

wsprintf(szBuffer, TEXT("%d"), sum);

// MessageBox(hwnd\_parent, szBuffer, L"Error", MB\_OK | MB\_ICONERROR);

SendMessage(hwnd\_edit\_res, WM\_SETTEXT, 0, (LPARAM)szBuffer);

}

}

void subtract(HWND hwnd\_edit1, HWND hwnd\_edit2, HWND hwnd\_edit\_res, HWND hwnd\_parent)

{

int val1 = 0, val2 = 0, difference = 0;

int len = GetWindowTextLength(hwnd\_edit1);

if (len > 0)

{

int i = 0;

LPWSTR buf;

buf = (LPWSTR)GlobalAlloc(GPTR, len + 1);

GetWindowText(hwnd\_edit1, buf, len + 1);

val1 = \_wtoi(buf);

GlobalFree((HANDLE)buf);

i = 0;

buf;

buf = (LPWSTR)GlobalAlloc(GPTR, len + 1);

GetWindowText(hwnd\_edit2, buf, len + 1);

val2 = \_wtoi(buf);

GlobalFree((HANDLE)buf);

difference = subtractInts(val1, val2);

TCHAR szBuffer[20];

wsprintf(szBuffer, TEXT("%d"), difference);

// MessageBox(hwnd\_parent, szBuffer, L"Error", MB\_OK | MB\_ICONERROR);

SendMessage(hwnd\_edit\_res, WM\_SETTEXT, 0, (LPARAM)szBuffer);

}

}

void multiply(HWND hwnd\_edit1, HWND hwnd\_edit2, HWND hwnd\_edit\_res, HWND hwnd\_parent)

{

int val1 = 0, val2 = 0, product = 0;

int len = GetWindowTextLength(hwnd\_edit1);

if (len > 0)

{

int i = 0;

LPWSTR buf;

buf = (LPWSTR)GlobalAlloc(GPTR, len + 1);

GetWindowText(hwnd\_edit1, buf, len + 1);

val1 = \_wtoi(buf);

GlobalFree((HANDLE)buf);

i = 0;

buf;

buf = (LPWSTR)GlobalAlloc(GPTR, len + 1);

GetWindowText(hwnd\_edit2, buf, len + 1);

val2 = \_wtoi(buf);

GlobalFree((HANDLE)buf);

product = multiplyInts(val1, val2);

TCHAR szBuffer[20];

wsprintf(szBuffer, TEXT("%d"), product);

// MessageBox(hwnd\_parent, szBuffer, L"Error", MB\_OK | MB\_ICONERROR);

SendMessage(hwnd\_edit\_res, WM\_SETTEXT, 0, (LPARAM)szBuffer);

}

}

void divide(HWND hwnd\_edit1, HWND hwnd\_edit2, HWND hwnd\_edit\_res, HWND hwnd\_edit\_remainder, HWND hwnd\_parent)

{

int val1 = 0, val2 = 0;

int quotient = 0;

int len = GetWindowTextLength(hwnd\_edit1);

if (len > 0)

{

int i = 0;

LPWSTR buf;

buf = (LPWSTR)GlobalAlloc(GPTR, len + 1);

GetWindowText(hwnd\_edit1, buf, len + 1);

val1 = \_wtoi(buf);

GlobalFree((HANDLE)buf);

i = 0;

buf;

buf = (LPWSTR)GlobalAlloc(GPTR, len + 1);

GetWindowText(hwnd\_edit2, buf, len + 1);

val2 = \_wtoi(buf);

GlobalFree((HANDLE)buf);

quotient = quotientInts(val1, val2);

TCHAR szBuffer[20];

wsprintf(szBuffer, TEXT("%d"), quotient);

SendMessage(hwnd\_edit\_res, WM\_SETTEXT, 0, (LPARAM)szBuffer);

int remainder = remainderInts(val1, val2);

szBuffer[20];

wsprintf(szBuffer, TEXT("%d"), remainder);

SendMessage(hwnd\_edit\_remainder, WM\_SETTEXT, 0, (LPARAM)szBuffer);

}

}

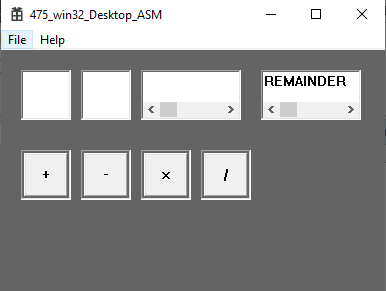
Listing 1: C++

### Assembly

****

Listing 2: Assembly

RESULT



# CONCLUSION

The above program is a functional calculator in which the calculation functions are implemented in Intel x64 assembly, and the graphical user interface is written in win32 C.

# REFERENCES

Brey, B. B. (2008). *THE INTEL MICROPROCESSORS 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4, and Core2 with 64-Bit Extensions: Architecture, Programming, and Interfacing* (8th ed.). Upper Saddle River, New Jersey, USA: Pearson Prentice Hall.

Seyfarth, R. (2011). *Introduction to 64 Bit Intel Assembly Language Programming for Linux.* CreateSpace Independent Publishing Platform.