**Riga Technical University**

**Faculty of Computer Science and Information Technology**

Institute of Applied Computer Systems

Department of Applied Computer Sciences

Denys Haponov

Student Id No.: 161ADB119

Course DPI343

"Computer Organization and Assembly Language"

Report on Practice Assignment

**Integer arithmetic calculations**

# TaskS Desciption

Please use [Introduction to Microsoft C/C++ Inline Assembler](https://estudijas.rtu.lv/mod/resource/view.php?id=1009600) to read basic information about Microsoft Visual Studio IDE and features related to the Visual C/C++ Inline Assembler.

You will have to use Visual C/C++ and use its inline or full assembler to implement various arithmetic calculations with signed integers. When implementing the tasks below, try to apply the following instructions:

* arithmetic **ADD, SUB, IMUL, IDIV, INC, DEC, NEG**;
* data type extension **CBW, CWDE, CDQ**;
* stack operations **PUSH, POP**.

# Individual TASKS

There are given three arithmetic expression categories A and B:

|  |  |  |
| --- | --- | --- |
| **Variant** | **A** | **B** |
|  |  |  |
|  |  |  |
|  |  |  |
| **3** | X + Y – Z | –5X + 2YZ |

Assume that the input values for X, Y and Z are at least signed 8-bit integers and that the precedence of mathematic operations must be taken into account.

Select only one expression from each column – depending on an individual variant (0, 1, 2 or 3), that must be obtained from your RTU student ID’s digits:

* Your RTU student ID’s 2nd, 8th and 9th digits are: 6,1,9 (e.g. 123ADB456 🡪 2, 5, 6)
* Expression A variant: (2nd + 8th) mod 4 = 6+1 mod 4= 7 mod 4 = 3
* Expression B variant: (2nd + 9th) mod 4 = 6+9=15 =15 mod 4= 3

Start your solution by creating C/C++ **Win32 Console Application** in the **Microsoft Visual Studio** and entering the following project characteristics:

* Name: *Practice1*
* Solution Name: *COAL*

Continue by implementing the lowest level task as shown in the table below and complete the tasks in the given order. You can stop at any level task and assume that if implemented correctly the practice grade will match it. It will be decreased, however, if the solutions contain flaws or errors.

|  |  |
| --- | --- |
| **Grade level** | **Task to complete** |
| **4** | *Complete with the working template solution below.* |
| **5** | Compute A |
| **6** | Compute B |
| **7** | Compute B – A |
| **8** | Compute B \* A |
| **9** | Compute B / A |
| **10** | Use at least C/C++ function *assert()* to demonstrate the testing of solutions. |

Note: Expression calculations, conditions and jumps must be implemented by using Full or Inline Assembler!

Each next level task assumes that you retain all lower level solutions. However, you must not reuse lower grade solutions as function calls from higher level solutions. Instead copy reusable part(s) into higher lever task solution.

# SolutionS

**Replace this template with your solutions (in-line comments are welcome):**

// Assembly\_HW\_1.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <cstdio>

#include <cassert>

// :::

// NOTE:

// char data type is signed 8-bit integer (DB in assembler)

// int data type is signed 16-bit integer (DW in assembler)

// int data type is signed 32-bit integer (DD in assembler)

// You can use any integer data type for input values.

// Update the given template code appropriately.

int solution\_for\_grade\_5(int x, int y, int z)

{

int result;

\_\_asm

{

// Your Inline Assembler instructions for grade 5 go here

// :::

mov eax, x;

add eax, y;

sub eax, z;

mov[result], eax; save the result assuming it is in the processor register AX

}

return result;

}

int solution\_for\_grade\_6(int x, int y, int z)

{

int result2;

\_\_asm

{

mov eax, x;

imul eax, -5;

mov ebx, y;

imul ebx, z;

imul ebx, 2;

add eax, ebx;

mov[result2], eax; saves the result

}

return result2;

}

int solution\_for\_grade\_7(int x, int y, int z)

{

int result3;

\_\_asm

{

mov eax, x;

add eax, y;

sub eax, z;

mov ebx, x;

imul ebx, -5;

mov ecx, y;

imul ecx, z;

imul ecx, 2;

add ebx, ecx;

sub ebx, eax;

mov eax, ebx;

mov[result3], eax;

}

return result3;

}

int solution\_for\_grade\_8(int x, int y, int z)

{

int result4;

\_\_asm

{

mov eax, x;

add eax, y;

sub eax, z;

mov ebx, x;

imul ebx, -5;

mov ecx, y;

imul ecx, z;

imul ecx, 2;

add ebx, ecx;

imul eax, ebx;

mov[result4], eax;

}

return result4;

}

int solution\_for\_grade\_9(int result, int result2)

{

int result5;

\_\_asm

{

mov eax, result;

mov ecx, result2;

div ecx;

mov[result5], eax;

}

return result5;

}

// :::

int main()

{

int first = 320; // Note the int data type range is -32768..32767

int second = -250;

int third = 122;

int result = solution\_for\_grade\_5(first, second, third);

int result2 = solution\_for\_grade\_6(first, second, third);

int result3 = solution\_for\_grade\_7(first, second, third);

int result4 = solution\_for\_grade\_8(first, second, third);

int result5 = solution\_for\_grade\_9(result, result2);

printf("solution\_for\_grade\_5(%d, %d, %d) = %d\n", first, second, third, result);

printf("solution\_for\_grade\_6(%d, %d, %d) = %d\n", first, second, third, result2);

printf("solution\_for\_grade\_7(%d, + , %d,) = %d\n", result, result2, result3);

printf("solution\_for\_grade\_8(%d, \* , %d,) = %d\n", result, result2, result4);

printf("solution\_for\_grade\_9(%d, / , %d,) = %d\n", result, result2, result5);

// You might need to invoke solution\_for\_grade\_X() functions several times

// with different parameter values (e.g., to demonstrate branching).

// :::

getchar(); // Wait for the Enter key

return 0;

}

# Conclusions

In this homework we were introduced to simple mathematical operations using assembly.

Also, we have encountered the limitations and different data types, in my case I have changed short type to int for the whole value to fit.

Finish practical task assignment by uploading this report to the course ORTUS site!