NEW YORK CITY COLLEGE OF TECHNOLOGY

THE CITY UNIVERSITY OF NEW YORK

300 JAY STREET, BROOKLYN, NY 11201-1909

Department of Computer Engineering Technology

CET 3510 - OL25 - Microcomputer Systems Technology lab

LABORATORY REPORT No_01

Lab_No_01 Getting into x86 Microprocessor for Assembly Language

Professor: Dr. Rubén Velásquez, Ph.D.

Spring 2021

Student's Name: Ashahi Shafin

Student's ID: 23607352

Preparation date: *February 10, 2021*

Due date: *February 17, 2021*

1. Table of content

2. OBJECTIVE

Laboratory No 1. To interface C with assembly language. The step-by-step tutorial shows how to use assembly language with a Visual C/C++ for 32-bit applications. A 32-bit application is written by using any 32-bit registers, and the memory space is essentially limited to 2Gbytes for Windows. In the Visual interface, all I/O is handled by Windows operating system framework.

3. LABORATORY TOOLS TO PERFORM THE TASK

Computer NZXT
Microsoft visual studio 2019 Software
Microsoft Word 2019 Software
Microsoft Notepad Software

4. SOURCE CODE

Laboratory No_01 Example 1.2

Source Code

```
#include "stdafx.h"
#include "stdio.h"
#include<iostream>

int main(void)
{

	printf(" Lab_No_01_Getting_Stated_into_x86_Assembly_from_a_C++_program\n");
	printf(" Module B: Embedding an in-line asssembly language module in a C progrmming \n");
	printf(" Ashahi Shafin, ID#23607352\n");
	printf(" CET3510-OLXX\n");
	printf(" Presentation date: February 10, 2021\n");
	printf(" Due date: February 17, 2021\n");
	printf(" Example 1.2 SimpleMathASM.cpp\n");
	printf(" file name: SimpleMathASM.cpp\n");
	printf("--------------------------------------------------------------------------------
\n");

	int x1, y1, sum1, diff1;
	short int x2, y2, sum2, diff2;
	char x3, y3, sum3, diff3;

	printf("********************************************************************************\n")
;
```

```
	//A. Find the sum of two integers (sum = x1 + y1)

	printf("Hello\n");

	printf("Using EAX, EBX, and ECX 32-bit Registers\n");

	printf("A. Find the sum of two integers (sum = x1 + y1) using 32-bit registers \n");

	printf("Enter two integers from a keynoard to add\n");
	scanf_s("%d%d", &x1, &y1);
	_asm
	{
		MOV EAX, 0;
		MOV EBX, 0;
		MOV ECX, 0;

		MOV EAX, x1;
		MOV EBX, y1;
		MOV ECX, EAX;
		ADD ECX, EBX;
		MOV sum1, ECX;

		// sum = x1 + y1 this operation is substituted using 32-bit registers;

	}
	printf("Addition result = %d\n", sum1);
	getchar();

	printf("***********************************************\n");

	//B. Find the difference of two integers (sub = x1 - y1)

	printf("Hello\n");

	printf("Using EAX, EBX, and ECX 32-bit Registers\n");

	printf("B. Find the difference of two integers (sub = x1 - y1) using 32-bits registers
\n");

	printf("Enter two integers from a keyboard to minus\n");
	scanf_s("%d%d", &x1, &y1);
	_asm
	{
		MOV EAX, 0;
		MOV EBX, 0;
		MOV ECX, 0;

		MOV EAX, x1;
		MOV EBX, y1;
		MOV ECX, EAX;
		SUB ECX, EBX;
		MOV diff1, ECX;

		// sub = x1 - y1; this operation is substituted using registers;

	}

	printf("Subtraction result = %d\n", diff1);
```

```
	printf("*********************************************\n");
	printf("*********************************************\n");

	system("pause");


	//A. Find the sum of two integers (sum = x2 + y2)

	printf("Hello\n");

	printf("Using AX, BX, and CX 16-bit Registers\n");

	printf("A. Find the sum of two integers (sum = x1 + y1) using 16-bit registers \n");

	printf("Using AX, BX, and CX 16-bits Registers\n");

	printf("Enter two integers from a keyboard to add\n");
	scanf_s("%d%d", &x2, &y2);
	_asm
	{
		MOV EAX, 0;
		MOV EBX, 0;
		MOV ECX, 0;

		MOV AX, x2;
		MOV BX, y2;
		MOV CX, AX;
		ADD CX, BX;
		MOV sum2, CX;

		// sum = x2 + y2 this operation is substituted using 32-bit registers;

	}
	printf("Addition result = %d\n", sum2);
	getchar();

	printf("*********************************************\n");

	//B. Find the difference of two integers (sub = x2 - y2)

	printf("Hello\n");

	printf("Using AX, BX, and CX 16-bit Registers\n");

	printf("B. Find the difference of two integers (sub = x2 - y2) using 16-bit registers
\n");

	printf("Enter two integers from a keyboard to minus\n");
	scanf_s("%d%d", &x2, &y2);
	_asm
	{
		MOV EAX, 0;
		MOV EBX, 0;
		MOV ECX, 0;

		MOV AX, x2;
		MOV BX, y2;
		MOV CX, AX;
		SUB CX, BX;
```

```
		MOV diff2, CX;

		// sub = x2 - y2; this operation is substituted using registers;
	}

	printf("Subtraction result = %d\n", diff2);

	printf("***********************************************\n");
	printf("***********************************************\n");

	system("pause");


	//A. Find the sum of two integers (sum = x3 + y3)

	printf("Hello\n");

	printf("Using AL, BL, and CL 8-bit Registers\n");

	printf("A. Find the sum of two integers (sum = x3 + y3) using 8-bit registers \n");

	printf("Using AL, BL, and CL 8-bits Registers\n");

	printf("Enter two integers from a keyboard to add\n");
	scanf_s("%d%d", &x3, &y3);
	_asm
	{
		MOV EAX, 0
		MOV EBX, 0
		MOV ECX, 0;

		MOV AL, x3
		MOV BL, y3
		MOV CL, AL;
		ADD CL, BL;
		MOV sum3, CL;

		// sum = x3 + y3 this operation is substituted using 32-bit registers;

	}

	printf("Addition result = %d\n", sum3);

	getchar(); // Hold the consult for result

	printf("***********************************************\n");

	//B. Find the difference of two integers (sub = x3 - y3)

	printf("Hello\n");

	printf("Using AL, BL, and CL 8-bit Registers\n");

	printf("B. Find the difference of two integers (sub = x3 - y3) using 8-bit registers
\n");

	printf("Enter two integers from a keyboard to minus\n");
	scanf_s("%d%d", &x3, &y3);
	_asm
```

```
	{
		MOV EAX, 0;
		MOV EBX, 0;
		MOV ECX, 0;

		MOV AL, x3;
		MOV BL, y3;
		MOV CL, AL;
		SUB CL, BL;
		MOV diff3, CL;

		// sub = x3 - y3; this operation is substituted using registers;
	}

	printf("Subtraction result = %d\n", diff3);
	printf("*********************************************\n");
	printf("*********************************************\n");
	system("pause");
	return(0);
}
```

5. *SOURCE CODE LINE DESCRIPTION*

Laboratory No_01 Example 1.2

Line	***Source Code Description***
01	*#include* "stdafx.h"
Precompiled Header **stdafx**. **h** is basically used in Microsoft Visual Studio to let the compiler know the files that are once compiled and no need to compile it from scratch. ... **h**" before this **includes** then the compiler will find the compiled header files from **stdafx**. **h** and does not compiled it from scratch.	
02	*#include "stdio.h"*
The stdio.h (standard library header) is a file with ".h" extension that contains the prototypes of standard input-output functions used in c.	
03	#include<iostream>
h, **iostream** provides basic input and output services for C++ programs. **iostream uses** the objects cin , cout , cerr , and clog for sending data to and from the standard streams input, output, error (unbuffered), and log (buffered) respectively.	

04	int main(void)
	{
	MAIN PROGRAM
	}

In C and C++ **int main(void) means** that the function takes NO arguments. ... **Int main(void)** is used in C to restrict the function to take any arguments, if you do not put **void** in those brackets, the function will take ANY number of arguments you supply at call.

05	int x1, y1, sum1, diff1; short int x2, y2, sum2, diff2; char x3, y3, sum3, diff3;

Memory variables x1, y1, sum1, and diff1 are for 32-bit registers declared as "int"
Memory variables x2, y2, sum2, and diff2 are for 16-bit registers declared as "short"
Memory variables x3, y3, sum3, and diff3 are for 8-bit registers declared as "char"

06	printf("**********\n")

Output console to indicate separation marked with asterisks within parentheses.

07	printf("Hello\n");
	printf("Using EAX, EBX, and ECX 32-bit Registers\n");
	printf("A. Find the sum of two integers (sum = x1 + y1) using 32-bit registers \n");
	printf("Enter two integers from a keyboard to add\n");

Output console display information within parentheses.

08	scanf_s("%d%d", &x1, &y1);

The **scanf_s**() function reads data from the standard input stream stdin and writes the data into the location given by argument. Each argument must be a pointer to a variable of a type that corresponds to a type specifier in format. If copying takes place between strings that overlap, the behavior is undefined.

09	_asm
	{
	For mnemonic assembly instructions declaration
	explained in lines 10, 11, 12, 13, and 14.
	}

_asm-declaration gives the ability to embed assembly language source code within a **C++** program. This declaration is conditionally-supported and implementation defined, meaning that it may not be present and, even when provided by the implementation; it does not have a fixed meaning.

10	MOV EAX, 0;
	MOV EBX, 0;
	MOV ECX, 0;

Mnemonic MOV instructions load an operand source memory hexadecimal value 0 to an operand destination 32-bit EAX, EBX, ECX registers to be initialized to zero.

11	MOV EAX, x1; MOV EBX, y1;

Mnemonic MOV instructions load an operand source memory variables x1 or y1 to an operand destination 32-bit EAX or EBX registers.

12	MOV ECX, EAX;
Mnemonic MOV instruction loads an operand source 32-bit EAX register to an operand destination 32-bit ECX register.	
13	ADD ECX, EBX;
Mnemonic ADD instruction adds an operand source 32-bit EBX register to an operand destination 32-bit ECX register where the data is stored.	
14	MOV sum1, ECX;
Mnemonic MOV instruction loads an operand source 32-bit ECX register to an operand destination MEMORY variable sum1.	
15	**printf("Addition result = %d\n", sum1);**
Displays out of the assembly language the result to C++ with a decimal number stored in memory sum1	
16	getchar();
getchar is a function in **C** programming language that reads a single character from the standard input stream **stdin**, regardless of what it is, and returns it to the program. It is specified in ANSI-**C** and is the most basic input function in **C**. It is included in the **stdio**.	

17	printf("Hello\n");
	printf("Using EAX, EBX, and ECX 32-bit Registers\n");
	printf("B. Find the difference of two integers (sub = x1 + y1) using 32-bit registers \n");
	printf("Enter two integers from a keyboard to minus\n");

Output console display information within parenthesis.	
18	scanf_s("%d%d", &x1, &y1);
The **scanf_s**() function reads data from the standard input stream stdin and writes the data into the location given by argument. Each argument must be a pointer to a variable of a type that corresponds to a type specifier in format. If copying takes place between strings that overlap, the behavior is undefined.	

19	_asm
	{
	For mnemonic assembly instructions declaration
	explained in lines 20, 21, 22, 23, and 24.
	}

_asm-declaration gives the ability to embed assembly language source code within a **C++** program. This declaration is conditionally-supported and implementation defined, meaning that it may not be present and, even when provided by the implementation, it does not have a fixed meaning.	

20	MOV EAX, 0;
	MOV EBX, 0;
	MOV ECX, 0;

Mnemonic MOV instructions load an operand source memory hexadecimal value 0 to an operand destination 32-bit EAX, EBX, ECX registers to be initialized to zero.	
21	MOV EAX, x1; MOV EBX, y1;
Mnemonic MOV instructions load an operand source memory variables x1 or y1 to an operand destination 32-bit EAX or EBX registers.	
22	MOV ECX, EAX;
Mnemonic MOV instruction loads an operand source 32-bit EAX register to an operand destination 32-bit ECX register.	
23	SUB ECX, EBX;
Mnemonic SUB instruction adds an operand source 32-bit EBX register to an operand destination 32-bit ECX register where the data is stored.	
24	MOV diff1, ECX;
Mnemonic MOV instruction loads an operand source 32-bit ECX register to an operand destination MEMORY variable diff1.	
25	**printf("Subtraction result = %d\n", sum1);**
Displays out of the assembly language the result to C++ with a decimal number stored in memory sum1	
27	sytem(“pause”);
System("**pause**") runs the Windows command-line "**pause**" program and waits for that to terminate before it continues execution of the program , the console window stays open so you can read the output.	

28	printf("Hello\n");
	printf("Using EAX, EBX, and ECX 16-bit Registers\n");
	printf("B. Find the difference of two integers (sub = x1 - y1) using 16-bits registers \n");
	printf("Enter two integers from a keyboard to minus\n");

Output console display information within parenthesis.	
29	scanf_s("%d%d", &x2, &y2);
The **scanf_s**() function reads data from the standard input stream stdin and writes the data into the location given by argument. Each argument must be a pointer to a variable of a type that corresponds to a type specifier in format. If copying takes place between strings that overlap, the behavior is undefined.	

30	_asm
	{
	For mnemonic assembly instructions declaration
	explained in lines 10, 11, 12, 13, and 14.
	}

_asm-declaration gives the ability to embed assembly language source code within a **C++** program. This declaration is conditionally-supported and implementation defined, meaning that it may not be present and, even when provided by the implementation; it does not have a fixed meaning.

31	MOV EAX, 0;
	MOV EBX, 0;
	MOV ECX, 0;

Mnemonic MOV instructions load an operand source memory hexadecimal value 0 to an operand destination 32-bit EAX, EBX, ECX registers to be initialized to zero.

32	MOV EAX, x2; MOV EBX, y1;

Mnemonic MOV instructions load an operand source memory variables x1 or y1 to an operand destination 32-bit EAX or EBX registers.

33	MOV ECX, EAX;

Mnemonic MOV instruction loads an operand source 32-bit EAX register to an operand destination 32-bit ECX register.

34	ADD ECX, EBX;

Mnemonic ADD instruction adds an operand source 32-bit EBX register to an operand destination 32-bit ECX register where the data is stored.

35	MOV sum1, ECX;

Mnemonic MOV instruction loads an operand source 32-bit ECX register to an operand destination MEMORY variable sum1.

36	**printf("Addition result = %d\n", sum1);**

Displays out of the assembly language the result to C++ with a decimal number stored in memory sum1

37	_asm
	{
	For mnemonic assembly instructions declaration
	explained in lines 10, 11, 12, 13, and 14.
	}

_asm-declaration gives the ability to embed assembly language source code within a **C++** program. This declaration is conditionally-supported and implementation defined, meaning that it may not be present and, even when provided by the implementation; it does not have a fixed meaning.

38	MOV EAX, 0;
	MOV EBX, 0;
	MOV ECX, 0;

Mnemonic MOV instructions load an operand source memory hexadecimal value 0 to an operand

destination 32-bit EAX, EBX, ECX registers to be initialized to zero.	
39	MOV EAX, x1; MOV EBX, y1;
Mnemonic MOV instructions load an operand source memory variables x1 or y1 to an operand destination 32-bit EAX or EBX registers.	
40	MOV ECX, EAX;
Mnemonic MOV instruction loads an operand source 32-bit EAX register to an operand destination 32-bit ECX register.	
41	ADD ECX, EBX;
Mnemonic ADD instruction adds an operand source 32-bit EBX register to an operand destination 32-bit ECX register where the data is stored.	
42	MOV diff1, ECX;
Mnemonic MOV instruction loads an operand source 32-bit ECX register to an operand destination MEMORY variable diff1.	
43	**printf("Addition result = %d\n", diff1);**
Displays out of the assembly language the result to C++ with a decimal number stored in memory diff1	
44	sytem("pause");
System("**pause**") runs the Windows command-line "**pause**" program and waits for that to terminate before it continues execution of the program , the console window stays open so you can read the output.	

45	printf("Hello\n");
	printf("Using AX, BX, and CX 16-bit Registers\n")
	printf("A. Find the sum of two integers (sum = x1 + y1) using 16-bit registers \n");
	printf("Using AX, BX, and CX 16-bits Registers\n");
	printf("Enter two integers from a keyboard to add\n");

Output console display information within parenthesis.	
46	scanf_s("%d%d", &x2, &y2);
The **scanf_s**() function reads data from the standard input stream stdin and writes the data into the location given by argument. Each argument must be a pointer to a variable of a type that corresponds to a type specifier in format. If copying takes place between strings that overlap, the behavior is undefined.	
47	_asm { For mnemonic assembly instructions declaration explained in lines 10, 11, 12, 13, and 14.

	}

_asm-declaration gives the ability to embed assembly language source code within a **C++** program. This declaration is conditionally supported and implementation defined, meaning that it may not be present and, even when provided by the implementation; it does not have a fixed meaning.

48	MOV EAX, 0;
	MOV EBX, 0;
	MOV ECX, 0;

Mnemonic MOV instructions load an operand source memory hexadecimal value 0 to an operand destination 16-bit EAX, EBX, ECX registers to be initialized to zero.

49	MOV AX, x2; MOV BX, y2;

Mnemonic MOV instructions load an operand source memory variables x2 or y2 to an operand destination 16-bit AX or BX registers.

50	MOV CX, AX;

Mnemonic MOV instruction loads an operand source 16-bit EAX register to an operand destination 16-bit ECX register.

51	ADD CX, BX;

Mnemonic ADD instruction adds an operand source 16-bit EBX register to an operand destination 16-bit ECX register where the data is stored.

52	MOV sum2, CX;

Mnemonic MOV instruction loads an operand source 16-bit ECX register to an operand destination MEMORY variable sum2.

53	printf("Addition result = %d\n", sum2);

Displays out of the assembly language the result to C++ with a decimal number stored in memory sum2

54	printf("Hello\n");
	printf("Using AX, BX, and CX 16-bit Registers\n");
	printf("B. Find the difference of two integers (sub = x2 - y2) using 16-bit registers \n");
	printf("Enter two integers from a keyboard to minus\n");

Output console display information within parenthesis.

55	scanf_s("%d%d", &x2, &y2);

The **scanf_s**() function reads data from the standard input stream stdin and writes the data into the location given by argument. Each argument must be a pointer to a variable of a type that corresponds to a type specifier in format. If copying takes place between strings that overlap, the behavior is undefined.

56	**_asm** **{** **For mnemonic assembly instructions declaration explained in lines 10, 11, 12, 13, and 14.** **}**

_asm-declaration gives the ability to embed assembly language source code within a **C++** program. This declaration is conditionally-supported and implementation defined, meaning that it may not be present and, even when provided by the implementation; it does not have a fixed meaning.

57	**MOV EAX, 0;** **MOV EBX, 0;** **MOV ECX, 0;**

Mnemonic MOV instructions load an operand source memory hexadecimal value 0 to an operand destination 16-bit EAX, EBX, ECX registers to be initialized to zero.

58	MOV EAX, x2; MOV EBX, y2;

Mnemonic MOV instructions load an operand source memory variables x2 or y2 to an operand destination 16-bit EAX or EBX registers.

59	MOV CX, AX;

Mnemonic MOV instruction loads an operand source 16-bit EAX register to an operand destination 16-bit CX register.

60	SUB CX, BX;

Mnemonic ADD instruction adds an operand source 16-bit EBX register to an operand destination 16-bit CX register where the data is stored.

61	MOV diff2, CX;

Mnemonic MOV instruction loads an operand source 16-bit ECX register to an operand destination MEMORY variable diff2.

62	**printf("Subtraction = %d\n", diff2);**

Displays out of the assembly language the result to C++ with a decimal number stored in memory sum1

63	printf("Hello\n");
	printf("Using AL, BL, and CL 8-bit Registers\n");
	printf("A. Find the sum of two integers (sum = x3 + y3) using 8-bit registers \n");
	printf("Using AL, BL, and CL 8-bits Registers\n");
	printf("Enter two integers from a keyboard to add\n");

Output console display information within parenthesis.

64	scanf_s("%d%d", &x3, &y3);

The **scanf_s**() function reads data from the standard input stream stdin and writes the data into the location given by argument. Each argument must be a pointer to a variable of a type that corresponds to a type specifier in format. If copying takes place between strings that overlap, the behavior is undefined.

65	_asm
	{
	For mnemonic assembly instructions declaration
	explained in lines 20, 21, 22, 23, and 24.
	}

_asm-declaration gives the ability to embed assembly language source code within a **C++** program. This declaration is conditionally-supported and implementation defined, meaning that it may not be present and, even when provided by the implementation, it does not have a fixed meaning.

66	MOV EAX, 0;
	MOV EBX, 0;
	MOV ECX, 0;

Mnemonic MOV instructions load an operand source memory hexadecimal value 0 to an operand destination 8-bit EAX, EBX, ECX registers to be initialized to zero.

67	MOV AL, x3; MOV BL, y3;

Mnemonic MOV instructions load an operand source memory variables x1 or y1 to an operand destination 16-bit AL or BL registers.

68	MOV AL, BL;

Mnemonic MOV instruction loads an operand source 16-bit EAX register to an operand destination 16-bit CL register.

69	ADD AL, BL;

Mnemonic ADD instruction adds an operand source 16-bit EBX register to an operand destination 16-bit CL register where the data is stored.

70	MOV ADD3, CL;

Mnemonic MOV instruction loads an operand source 16-bit ECX register to an operand destination MEMORY variable sum3.

71	**printf("Addition result = %d\n", sum3);**

Displays out of the assembly language the result to C++ with a decimal number stored in memory sum3

72	printf("Hello\n");
	printf("Using AL, BL, and CL 8-bit Registers\n");
	printf("B. Find the difference of two integers (sub = x3 - y3) using 8-bit registers \n");
	printf("Enter two integers from a keyboard to minus\n");
	scanf_s("%d%d", &x3, &y3);

Output console display information within parentheses.	
73	scanf_s("%d%d", &x3, &y3);
The **scanf_s**() function reads data from the standard input stream stdin and writes the data into the location given by argument. Each argument must be a pointer to a variable of a type that corresponds to a type specifier in format. If copying takes place between strings that overlap, the behavior is undefined.	

74	_asm
	{
	For mnemonic assembly instructions declaration
	explained in lines 10, 11, 12, 13, and 14.
	}

_asm-declaration gives the ability to embed assembly language source code within a **C++** program. This declaration is conditionally-supported and implementation defined, meaning that it may not be present and, even when provided by the implementation; it does not have a fixed meaning.	

75	MOV EAX, 0;
	MOV EBX, 0;
	MOV ECX, 0;

Mnemonic MOV instructions load an operand source memory hexadecimal value 0 to an operand destination 8-bit EAX, EBX, ECX registers to be initialized to zero.	
76	MOV AL, x3; MOV BL, y3;
	Mnemonic MOV instructions load an operand source memory variables x3 or y3 to an operand destination 8-bit AL or BL registers.
77	MOV CL, AL;
Mnemonic MOV instruction loads an operand source 8-bit ECX register to an operand destination MEMORY variable sum3.	
78	SUB CL, BL;
Mnemonic SUB instruction adds an operand source 8-bit EBX register to an operand destination 8-bit CL register where the data is stored.	
79	MOV diff3, CL;
Mnemonic MOV instruction loads an operand source 8-bit ECX register to an operand	

destination MEMORY variable diff3.	
80	printf("Subtraction result = %d\n", diff3);
Mnemonic MOV instruction loads an operand source 8-bit ECX register to an operand destination MEMORY variable diff3.	
81	sytem(“pause”);
System("**pause**") runs the Windows command-line "**pause**" program and waits for that to terminate before it continues execution of the program , the console window stays open so you can read the output.	
82	return(0);
The main function means that the program executed successfully. return 1 in the main function means that the program does not execute successfully and there is some error. In user-defined function. return 0 means that the user-defined function is returning false.	
83	
84	
85	
86	
87	
88	
89	
90	
91	

92	
93	
94	
95	
96	
97	
98	
99	
100	
101	
102	
103	
104	

105	
106	
107	
108	
109	
110	
111	
112	
113	
114	
115	
116	
117	

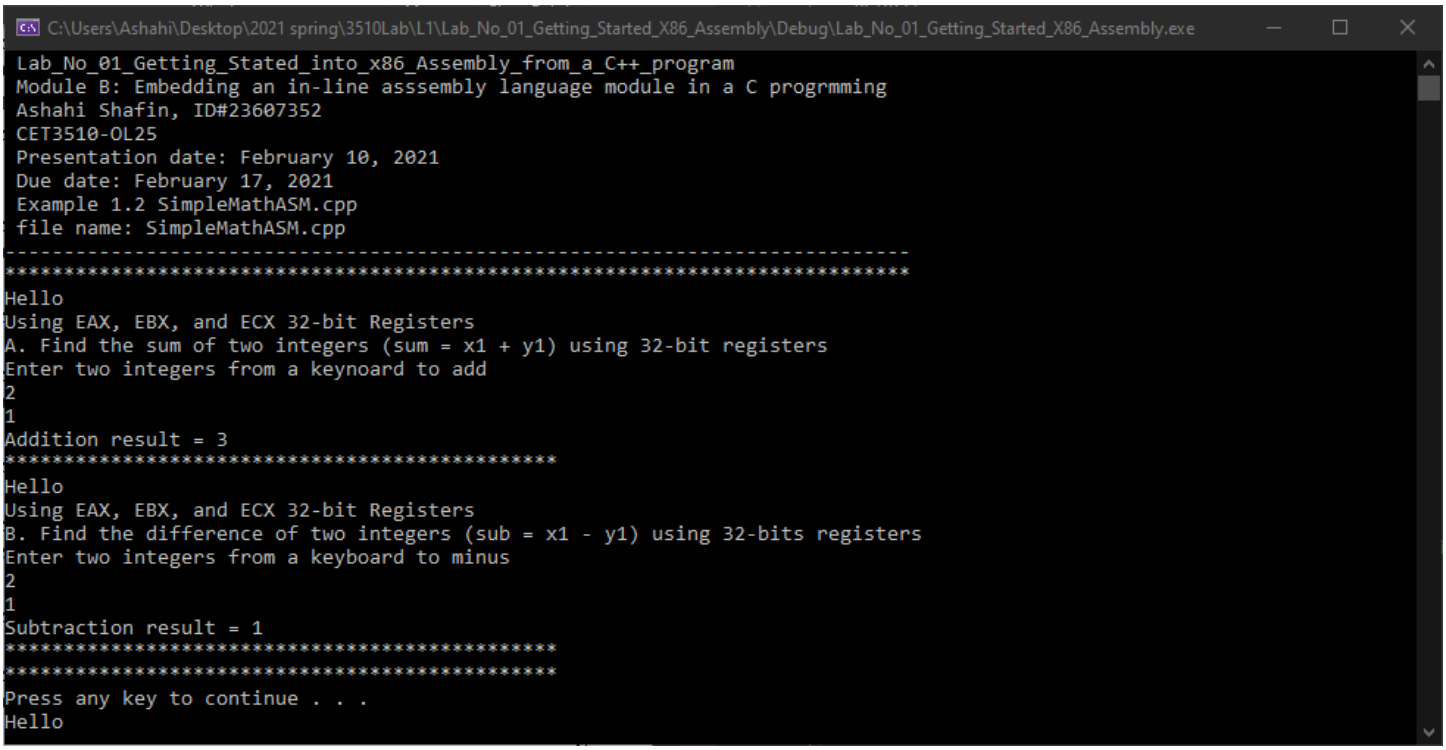
118	
119	
120	
121	
122	
123	
124	
125	
126	
127	
128	
129	
130	

131	
132	
133	
134	
135	
136	
137	
138	
139	
140	

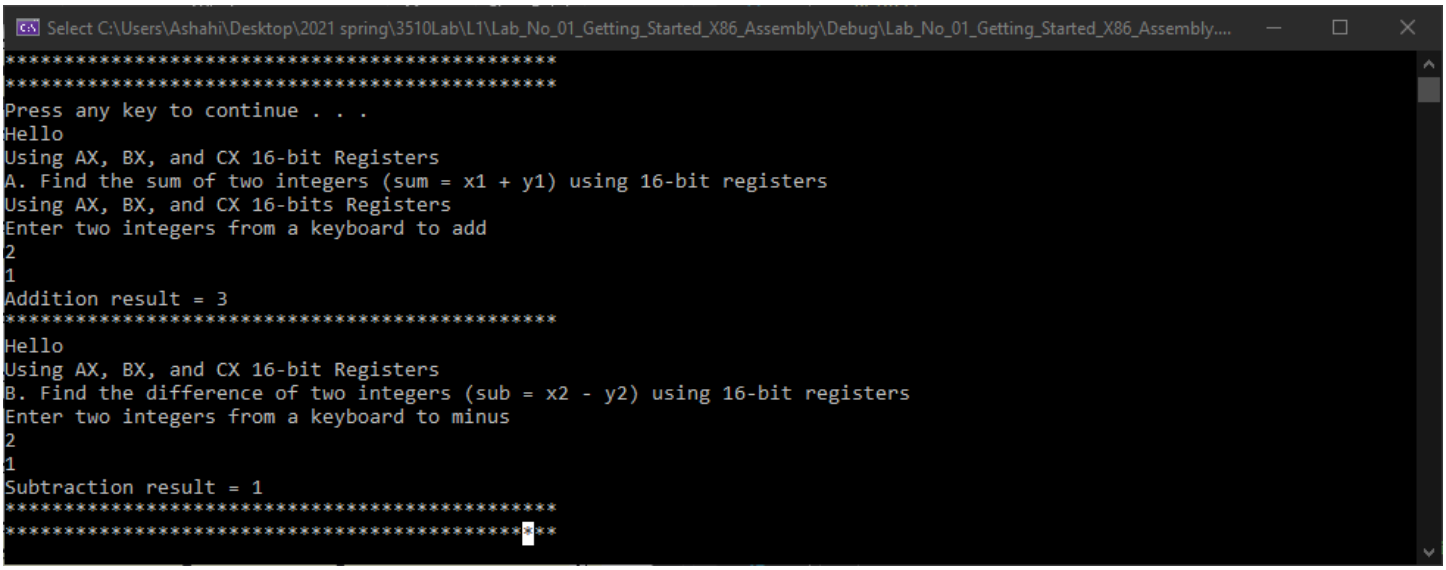
6. EXPLANING OUTPUT RESULTS

Laboratory No_01 Example 1.2

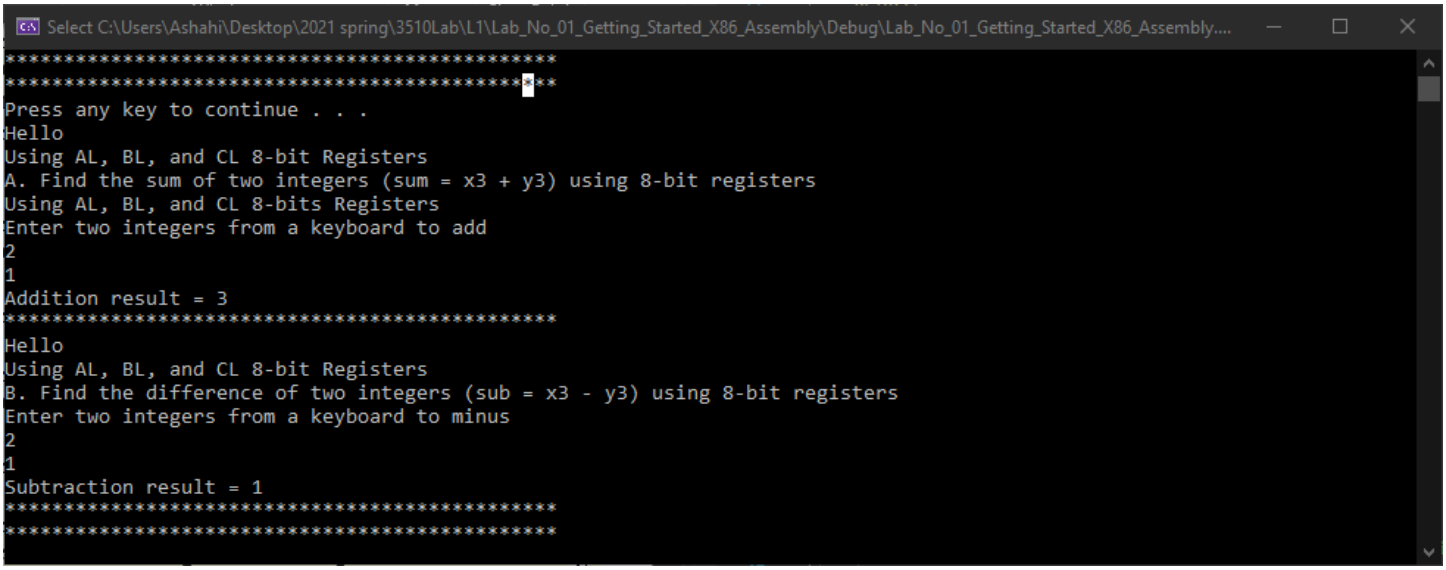
i. This is the output console of example 1.2 for laboratory 1. It shows both operations addition and subtraction for 32-bit EAX, EBX and ECX registers. For this type of operations in a 32-bit platform the variables x1 and y1 need to be declared as “int” otherwise these two operations does not work. This is because they need to call the variable for it to work. And as you see it shows the sum and the difference



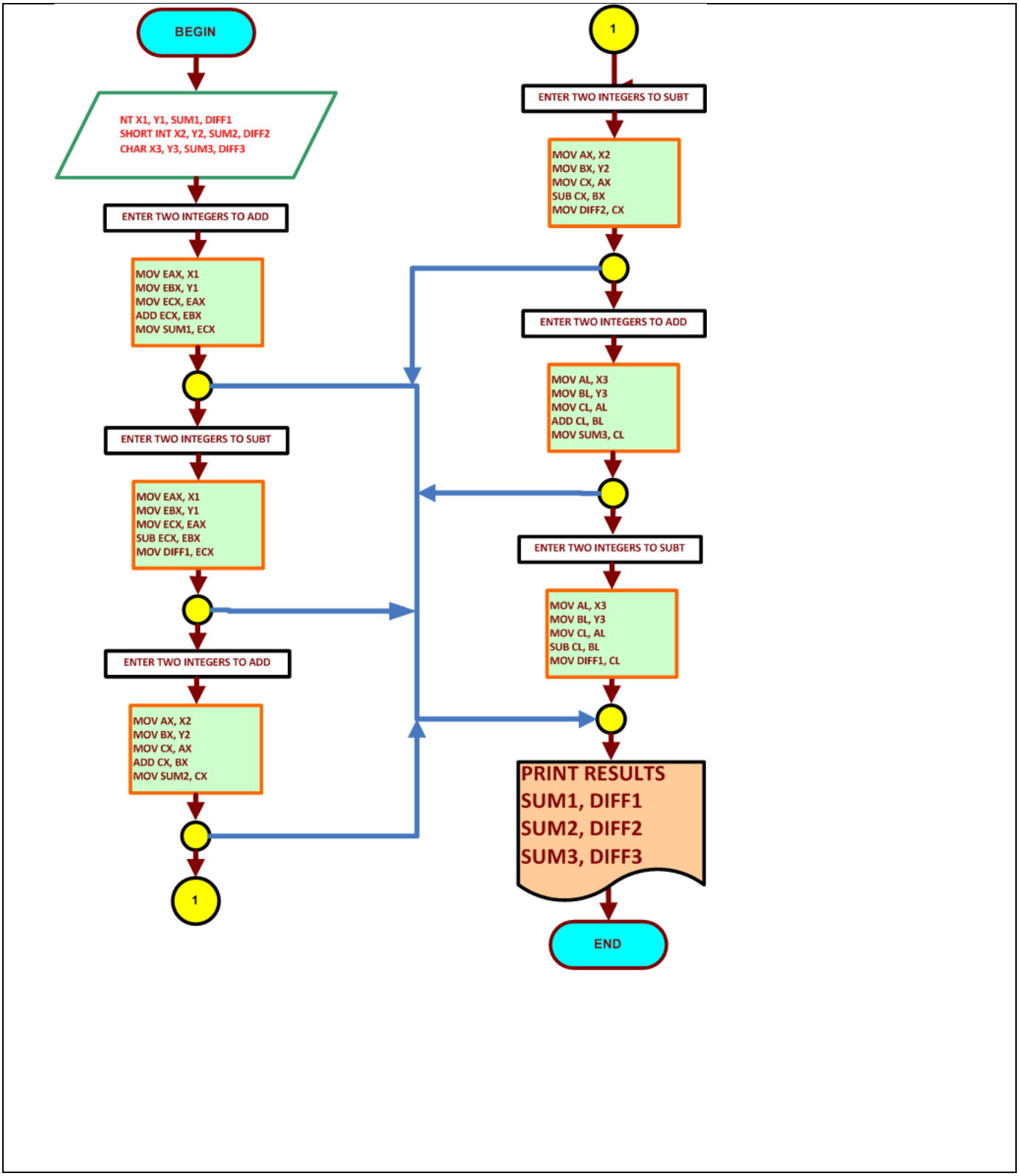
ii. This is the output console of example 1.2 for laboratory 1. It shows both operations addition and subtraction for 16-bit AX, BX and CX registers. For this type of operations in a 16-bit platform the variables x2 and y2 need to be declared as “short” otherwise these two operations does not work. This is because they need to call the variable for it to work. And as you see it shows the sum and the difference



iii. This is the output console of example 1.2 for laboratory 1. It shows both operations addition and subtraction for 16-bit AL, BL and CL registers. For this type of operations in a 8-bit platform the variables x3 and y3 need to be declared as "char" otherwise these two operations does not work. This is because they need to call the variable for it to work. And as you see it shows the sum and the difference



7. PROGRAM FLOW CHART:



8.

9. COMMENTS

Comments on Outputs shows how 32 16 8 bit method of register work. They also separate the diff and sum for someone who looking at it really quick.

10.CONCLUSION

In this experiment, the module asked the user to select 2 number in 32 16 8 bit and showed you the sum and difference this was done by using c++. What hard about this was that making the code the code take time to make and I have hard time understanding it and it took me a while figuring out the errors. Luckily when I did fix the errors it worked afterward. The output was correct too which mean everything worked. What I thought that was interesting was that how the module still worked after doing one section.

The 32 16 8 bit can be used in register because in real life the register is always adding up whatever is being scan and it then subtract the total from what you pay them and then it show you if you need to get changes or not. This can also be useful in bank atm because you take out money and put in money so it will basically work the same way like a register worked but your taking money from your account or putting in money to your account