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_03

Lab_No_03 Data Movement between General Purpose Registers

Professor: Dr. Rubén Velásquez, Ph.D.

Spring 2021

Student's Name: Ashahi Shafin

Student's ID: 23607352

Preparation date: *February 24, 2021*

Due date: *March 04, 2021*

1. Table of content

2. OBJECTIVE

Laboratory No 3. To practice move instructions involving data transfer and basic CPU architecture. The data transfer involves register, immediate-data-to-register, memory-to-register, reister-to-memory, and RAM. Exam the functions and contents of general registers of AL, AH, AX, and EAX; BL, BH,, BX and EBX; CL, CH, CX AND ECX; DL, DH, DX and EDX. The examples involve MOV instryctions: MOV EAX, EBX; MOV AX, 0xA234; MOV AH,15; MOV AL, 'A'.

3. LABORATORY TOOLS TO PERFORM THE TASK

Computer NZXT
Microsoft visual studio 2015 Software
Microsoft Word 2019 Software
Microsoft Notepad Software

4. SOURCE CODE

Laboratory No_03 Example 3.3

Source Code

```
#include <stdio.h>
#include <iostream>
int main()
{
	printf(" Lab_No_03_Data_Movement_between_General_Purpose_Registers\n");
	printf(" Module: C++ progrmming \n");
	printf(" Ashahi Shafin, ID#23607352\n");
	printf(" CET3510-OL25\n");
	printf(" Presentation date: February 24, 2021\n");
	printf(" Due date: February 03, 2021\n");
	printf(" Example 3.3 SwapXBytesErr.cpp\n");
	printf(" file name: 3.3 SwapXBytesErr.cpp\n");
	printf("-------------------------------------------------------------------------------\n");

	char temp;
	char r1, r2;
	short int r;

	//A. Assignning bytes in two general registers BL, BH
	_asm
	{
		MOV BL, 'a';
		MOV BH, 'A';
		MOV r1, BL;
		MOV r2, BH;
```

```
		MOV r, BX;
	}
	std::cout <<
"================================================================================" << std::endl;
	std::cout << "Before swapping" << std::endl;

	//To display characters
	printf("BH = %c, BL = %c, BX = %c\n", r2, r1, r);
	//Display the hexadecimal value of each character
	printf("BH = 0x%x, BL = 0x%x, BX = 0x%x\n", r2, r1, r);

	//Swap bytes in two general registers BL, BH
	_asm
	{
		XCHG BL, BH;
		MOV r1, BL;
		MOV r2, BH;
		MOV r, BX;
	}
	std::cout <<
"================================================================================" << std::endl;
	std::cout << "After swapping" << std::endl;

	//To display characters
	printf("BH = %c, BL = %c, BX = %c\n", r2, r1, r);
	//Display the hexadecimal value of each character
	printf("BH = 0x%x, BL = 0x%x, BX = 0x%x\n", r2, r1, r);
	/*********************************************************************************
*****************/
	//B. Assignning bytes in two general registers AL, AH
	_asm
	{
		MOV AL, 'b';
		MOV AH, 'B';
		MOV r1, AL;
		MOV r2, AH;
		MOV r, AX;
	}
	std::cout <<
"================================================================================" << std::endl;
	std::cout << "Before swapping" << std::endl;

	//To display characters
	printf("AH = %c, AL = %c, AX = %c\n", r2, r1, r);
	//Display the hexadecimal value of each character
	printf("AH = 0x%x, AL = 0x%x, AX = 0x%x\n", r2, r1, r);

	//Swap bytes in two general registers AL, AH
	_asm
	{
		MOV AL, 'b';
		MOV AH, 'B';
		MOV temp, AH;
		MOV AH, AL;
		MOV AL, temp;
		XCHG AL, AH;
		MOV r1, AL;
		MOV r2, AH;
		MOV r, AX;
```

```
	}
	std::cout <<
"================================================================================" << std::endl;
	std::cout << "After swapping" << std::endl;
	//To display characters
	printf("AH = %c, AL = %c, AX = %c\n", r2, r1, r);
	//Display the hexadecimal value of each character
	printf("AH = 0x%x, AL = 0x%x, AX = 0x%x\n", r2, r1, r);

	system("pause");
		return 0;

}
```

5. *SOURCE CODE LINE DESCRIPTION*

Laboratory No_03 Example 3.3

Line	***Source Code Description***
01	*#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.	
02	#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.	

03	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.	
04	printf("**********\n")
Output console to indicate separation marked with asterisks within parentheses.	
05	char temp; char r1, r2;

```
short int r;
```

A **variable's type** determines the values that the **variable** can have and the operations that can be performed on it.

06

```
_asm
        {
         For mnemonic assembly instructions declaration
         explained in 7.
        }
```

_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.

07

```
MOV BL, 'a';
MOV BH, 'A';
MOV r1, BL;
MOV r2, BH;
MOV r, BX;
```

Mnemonic MOV instructions load an operand source memory hexadecimal value 0 to an operand destination BL BH BX registers to be initialized to zero.

08

```
std::cout << "================================================================================"
<< std::endl;
std::cout << "Before swapping" << std::endl;
```

The **cout** object in **C++** is an object of class ostream. It is used to display the output to the standard output device i.e. monitor. It is associated with the standard C output stream stdout.

09

```
printf("BH = %c, BL = %c, BX = %c\n", r2, r1, r);
printf("BH = 0x%x, BL = 0x%x, BX = 0x%x\n", r2, r1, r);
```

Output console to indicate separation marked with asterisks within parentheses.

10

```
_asm
        {
         For mnemonic assembly instructions declaration
         explained in 11.
        }
```

_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.

11

```
XCHG BL, BH;
MOV r1, BL;
MOV r2, BH;
MOV r, BX;
```

Mnemonic MOV instructions load an operand source memory hexadecimal value 0 to an operand destination BL BH BX registers to be initialized to zero.

12	`std::cout << ==========================================================================="`
	`<< std::endl;`
	`std::cout << "After swapping" << std::endl;`

The **cout** object in **C++** is an object of class ostream. It is used to display the output to the standard output device i.e. monitor. It is associated with the standard C output stream stdout.

13	`printf("BH = %c, BL = %c, BX = %c\n", r2, r1, r);` `printf("BH = 0x%x, BL = 0x%x, BX = 0x%x\n", r2, r1, r);`

Output console to indicate separation marked with asterisks within parentheses.

14	_asm
	{
	For mnemonic assembly instructions declaration
	explained in 15.
	}

_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.

15	`MOV AL, 'b';`
	`MOV AH, 'B';`
	`MOV r1, AL;`
	`MOV r2, AH;`
	`MOV r, AX;`

Mnemonic MOV instructions load an operand source memory hexadecimal value 0 to an operand destination AL AH AX registers to be initialized to zero.

16	`std::cout << "==========================================================================="`
	`<< std::endl;`
	`std::cout << "Before swapping" << std::endl;`

The **cout** object in **C++** is an object of class ostream. It is used to display the output to the standard output device i.e. monitor. It is associated with the standard C output stream stdout.

17	`printf("AH = %c, AL = %c, AX = %c\n", r2, r1, r);` `printf("AH = 0x%x, AL = 0x%x, AX = 0x%x\n", r2, r1, r);`

Output console to indicate separation marked with asterisks within parentheses.

18	_asm
	{
	For mnemonic assembly instructions declaration
	explained in 19.
	}

_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.

19	`MOV AL, 'b';`
	`MOV AH, 'B';`
	`MOV temp, AH;`
	`MOV AH, AL;`

```
MOV AL, temp;
XCHG AL, AH;
MOV r1, AL;
MOV r2, AH;
MOV r, AX;
```

Mnemonic MOV instructions load an operand source memory hexadecimal value 0 to an operand destination AL AH AX registers to be initialized to zero.

20

```
std::cout << "================================================================================"
<< std::endl;
std::cout << "After swapping" << std::endl;
```

The **cout** object in **C++** is an object of class ostream. It is used to display the output to the standard output device i.e. monitor. It is associated with the standard C output stream stdout.

21

```
printf("AH = %c, AL = %c, AX = %c\n", r2, r1, r);
printf("AH = 0x%x, AL = 0x%x, AX = 0x%x\n", r2, r1, r);
```

Output console to indicate separation marked with asterisks within parentheses.

22

```
system("pause");
```

This is a Windows-specific command, which tells the OS to run the **pause** program. This program waits to be terminated, and halts the exceution of the parent **C++** program. Only after the **pause** program is terminated, will the original program continue.

23

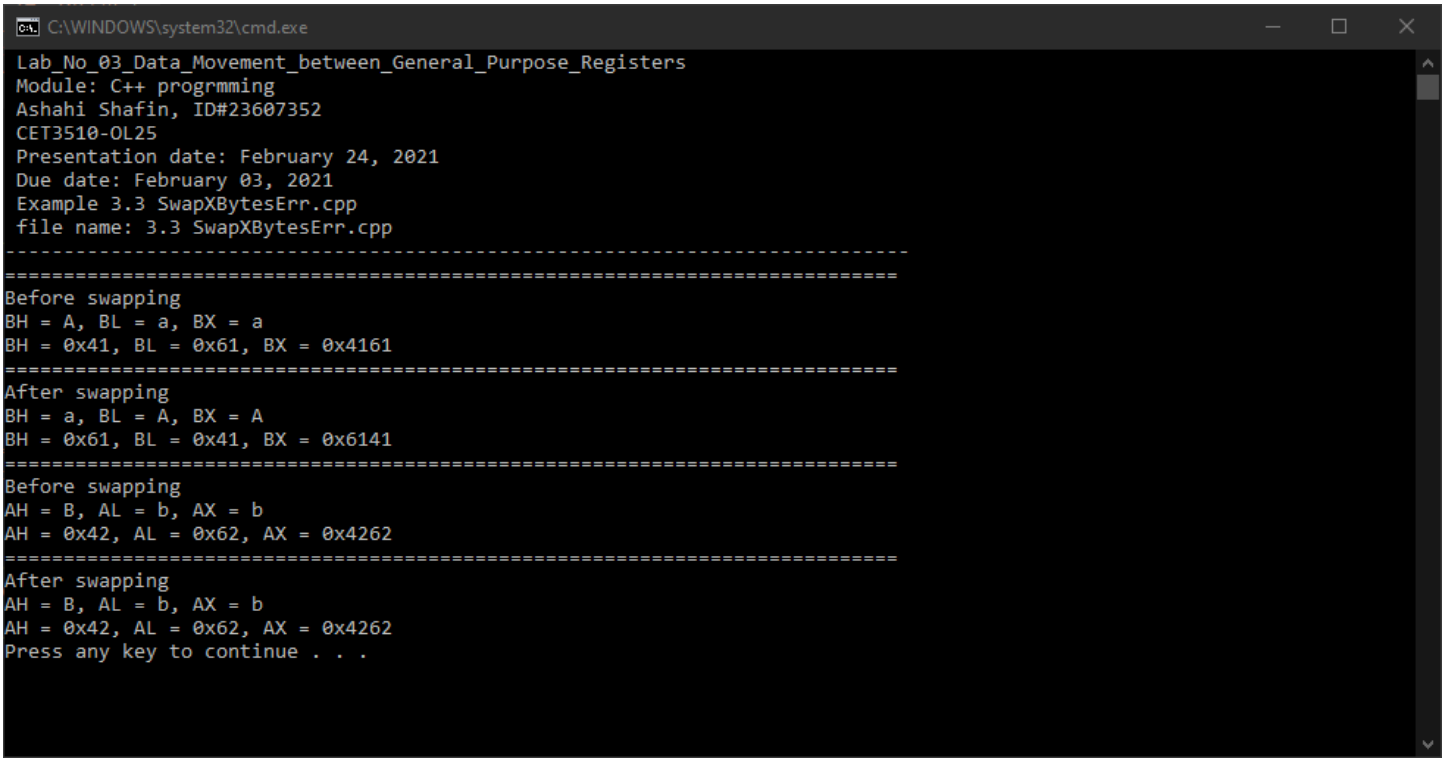
```
return 0;
```

Terminates the execution of a function and **returns** control to the calling function (or to the operating system if you transfer control from the main function). Execution resumes in the calling function at the point immediately following the call.

6. EXPLANING OUTPUT RESULTS

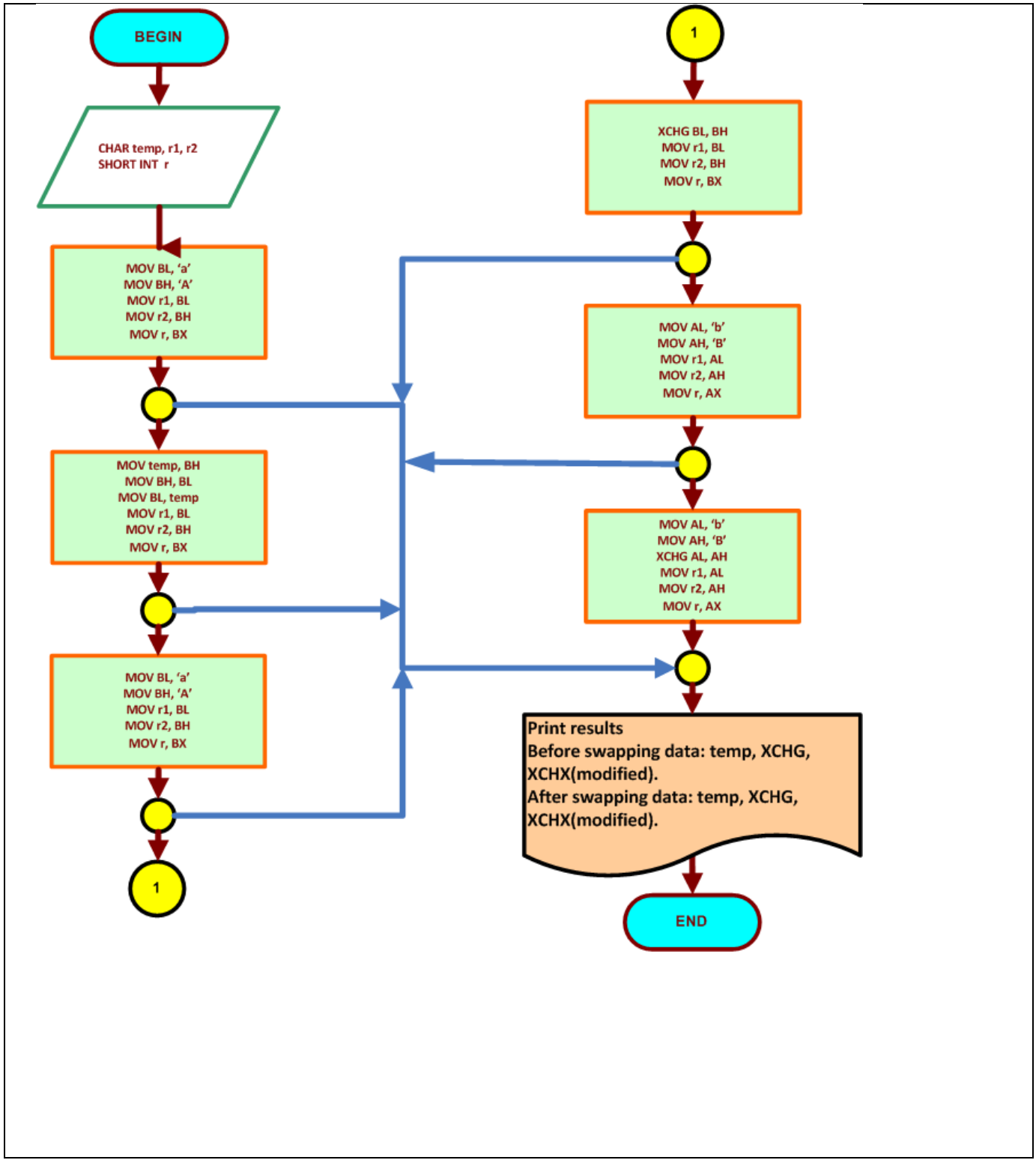
Laboratory No_03 Example 3.3

i. This is the output console of example 3.3 for laboratory 3. It shows the BL BH BX and AL AH AX value and their outputs. They also show the hexadecimal value to and AX and BX is the two number combined and not added or multiplied.


C:\WINDOWS\system32\cmd.exe
Lab_No_03_Data_Movement_between_General_Purpose_Registers
Module: C++ progrmming
Ashahi Shafin, ID#23607352
CET3510-OL25
Presentation date: February 24, 2021
Due date: February 03, 2021
Example 3.3 SwapXBytesErr.cpp
file name: 3.3 SwapXBytesErr.cpp

==
Before swapping
BH = A, BL = a, BX = a
BH = 0x41, BL = 0x61, BX = 0x4161
==
After swapping
BH = a, BL = A, BX = A
BH = 0x61, BL = 0x41, BX = 0x6141
==
Before swapping
AH = B, AL = b, AX = b
AH = 0x42, AL = 0x62, AX = 0x4262
==
After swapping
AH = B, AL = b, AX = b
AH = 0x42, AL = 0x62, AX = 0x4262
Press any key to continue . . .

7. *PROGRAM FLOW CHART:*



8.

9. COMMENTS

Comments on Outputs shows how swapping the values can show different hex values and with different swapping method are used in order to make outputs.

10.CONCLUSION

In this experiment, the module shows what happens when you swap value and what happens when you combine them together. It also showed us how when you use more than one method to swap it move the registers into normal state. What was hard about this lab was when doing the second part figuring out how to make the output work since the code was not correct. We also had to add AX an BX to 4 outputs because they were not there in the lab manual. Then the swapped worked how it should because we need the temp variable to make it work.

A real-life application for laboratory No 3 is in a computer because they are always switching data. And this can help when you are transferring big files and it will make sure to combine them together. This will also help with the cup making the process faster and it will tell them how to encode it and decode it. Which these methods you coded a lot.