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_05

Lab_No_05 Addressing Modes

Professor: Dr. Rubén Velásquez, Ph.D.

Spring 2021

Student's Name: Ashahi Shafin

Student's ID: 23607352

Preparation date: *March 10, 2021*

Due date: *March 17, 2021*

1. Table of content

2. OBJECTIVE

Laboratory No 5. To be familiar with the operation of each data-addressing mode. To use and select the appropriate addressing mode for a given task. To exam different data-addressing modes including register addressing. Immediate addressing, direct addressing, register indirect addressing, and register relative addressing.

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_05 Example 5.2

Source Code

```
#include <iostream>
using namespace std;

int main()
{
	printf(" Lab_No_05_Addressing_Modes\n");
	printf(" Module: C++ progrmming \n");
	printf(" Ashahi Shafin, ID#23607352\n");
	printf(" CET3510-OL25\n");
	printf(" Presentation date: March 10, 2021\n");
	printf(" Due date: March 17, 2021\n");
	printf(" Example 5.2 addressingModesTemp.cpp\n");
	printf(" file name: 5.2 addressingModesTemp.cpp\n");
	printf("--------------------------------------------------------------------------------
\n");

	const int n = 5; //the dimension of an array
	//more delclaration and initilization here
	unsigned int u32_arr[n] = { 0x00000041, 0x00000052, 0xabcd1169, '1234', 'aABc'};
	unsigned int u0, u1, u2, u3, u4;
	unsigned int *uPtr;
	int i;  // used as index
	int u0_addr, u1_addr, u2_addr, u3_addr, u4_addr;
```

```
	signed int i32_arr[n] = { 0x00000041, 0x00000052, 0xabcd1169, '1234', 'aABc' };
	signed int i0, i1, i2, i3, i4;
	signed int *iPtr;
	int i0_addr, i1_addr, i2_addr, i3_addr, i4_addr;

	cout <<
"+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n";
	//Display hxadecimal and decimal value each element of the unsigned int array
	cout << "The value of each element of 32 bit unsigned int array" << endl << endl << endl;
	for (i = 0; i < n; i++)
{
		cout<<"The value and the size of element" <<dec<<i;
		cout << " in an array are 0x" << hex << u32_arr[i]
			<< "and " << sizeof(u32_arr[i]) << "byte(s)" << endl;
}
	printf("-------------------------------------------------------------------------------
\n");
	uPtr = u32_arr;
	//Display the 32-bit address in hexadecimal format
	cout << "The memory address of the array is 0x" << hex << uPtr << endl;

	//The retrieved value from different address modes
	_asm
	{
		//EBX holds the address of element 0 in the array
		mov EBX, uPtr;

		/*u0 holds the retrieved value from the indirect address contained
		in the register EBX*/
		mov EAX, [EBX];
		mov u0, EAX;
		mov u0_addr, EBX;

		/*EBX is increased by 4 due u32_arr is an unsigned int array (4 bytes)
		EBX holds the address of element 1 in the array*/
		mov ECX, 4H;
		add EBX, ECX;

		/*u1 holds the retrieved value from the indirect address contained
		in the register EBX*/
		mov EAX, [EBX];
		mov u1, EAX;
		mov u1_addr, EBX;

		/*EBX+4H holds the address of element 2 in the array
		u2 holds the retrieved values from the register relative addressing*/
		mov EAX, [EBX + 4H];
		mov u2, EAX;

		//EDI hold the address of element 2 in an array
		mov EDI, EBX;
		add EDI, 4H;
		mov u2_addr, EDI;

		/*EBX+8H holds the address of element 3 in the array
		u3 holds the retrieved value from the register relative addressing*/
		mov EAX, [EBX + 8H];
		mov u3, EAX;
```

```
		//EDI holds the address of element 3 in the array
		mov EDI, EBX;
		add EDI, 8H;
		mov u3_addr, EDI;

		/*EBX+12H holds the address of element 4 in the array
		u4 holds the retrieved value from the register relative addressing*/
		mov EAX, [EDI + 4H];
		mov u4, EAX;

		//EDI holds the address of element 4 in the array
		mov ESI, EDI;
		add ESI, 4H;
		mov u4_addr, ESI;
	}

	//Display the retrieved value from different address modes
	cout << "-----------------------------------------------------------------" << endl;
	cout << "The retrieved values 0x" << hex << u0 << "\t at the address of 0x" << hex <<
u0_addr << endl;
	cout << "The retrieved values 0x" << hex << u1 << "\t at the address of 0x" << hex <<
u1_addr << endl;
	cout << "The retrieved values 0x" << hex << u2 << "\t at the address of 0x" << hex <<
u2_addr << endl;
	cout << "The retrieved values 0x" << hex << u3 << "\t at the address of 0x" << hex <<
u3_addr << endl;
	cout << "The retrieved values 0x" << hex << u4 << "\t at the address of 0x" << hex <<
u4_addr << endl;
	cout <<
"++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n";

	//Display hexadecimal and decimal values each element of the signed int array
	cout << "--- The value of each element of 32-bit signed int array ----" << endl << endl
<< endl;
	for (i = 0; i < n; i++)
	{
		cout << "The value and the size of element " << dec << i;
		cout << " in an array are 0x" << hex << i32_arr[i] << " and " <<
sizeof(i32_arr[i]) << " byte(s) " << endl;
	}
	printf("-----------------------------------------------------------------\n");
	iPtr = i32_arr;

	//Display the 32-bit address in hexadecimal format
	cout << "The memory address of the array is 0x" << hex << iPtr << endl;

	//The retrieved value from different address modes
	_asm
	{
		//EBX holds the address of element 0 in the array
		mov EBX, iPtr;

		/*i0 holds the retrieved value from the indirect address contained
		in the register EBX*/
		mov EAX, [EBX];
		mov i0, EAX;
		mov i0_addr, EBX;

		/*EBX is increased by 4 due i32_arr is an signed int array (4 bytes)
```

```
                EBX holds the address of element 1 in the array*/
                mov ECX, 4H;
                add EBX, ECX;

                /*i1 holds the retrieved value from the indirect address contained
                in the register EBX*/
                mov EAX, [EBX];
                mov i1, EAX;
                mov i1_addr, EBX;

                /*EBX+4H holds the address of element 2 in the array
                u2 holds the retrieved values from the register relative addressing*/
                mov EAX, [EBX + 4H];
                mov i2, EAX;

                //EDI hold the address of element 2 in an array
                mov EDI, EBX;
                add EDI, 4H;
                mov i2_addr, EDI;

                /*EBX+8H holds the address of element 3 in the array
                i3 holds the retrieved value from the register relative addressing*/
                mov EAX, [EBX + 8H];
                mov i3, EAX;

                //EDI holds the address of element 3 in the array
                mov EDI, EBX;
                add EDI, 8H;
                mov i3_addr, EDI;

                /*EBX+12H holds the address of element 4 in the array
                i4 holds the retrieved value from the register relative addressing*/
                mov EAX, [EDI + 4H];
                mov i4, EAX;

                //EDI holds the address of element 4 in the array
                mov ESI, EDI;
                add ESI, 4H;
                mov i4_addr, ESI;
        }

        //Display the retrieved value from different address modes
        cout << "------------------------------------------------------------" << endl;
        cout << "The retrieved values 0x" << hex << i0 << "\t at the address of 0x" << hex <<
i0_addr << endl;
        cout << "The retrieved values 0x" << hex << i1 << "\t at the address of 0x" << hex <<
i1_addr << endl;
        cout << "The retrieved values 0x" << hex << i2 << "\t at the address of 0x" << hex <<
i2_addr << endl;
        cout << "The retrieved values 0x" << hex << i3 << "\t at the address of 0x" << hex <<
i3_addr << endl;
        cout << "The retrieved values 0x" << hex << i4 << "\t at the address of 0x" << hex <<
i4_addr << endl;
        cout << "\n\n\n\n\n";


        system("pause");
        exit(0);
        return 0;
```

```
}
```

5. SOURCE CODE LINE DESCRIPTION

Laboratory No_05 Example 5.2

Line	Source Code Description
01	#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.	
02	using namespace std;
is an abbreviation for standard. So that means we **use** all the things with in "**std**" **namespace**.	

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	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	const int n = 5; unsigned int u32_arr[n] = { 0x00000041, 0x00000052, 0xabcd1169, '1234', 'aABc'}; unsigned int u0, u1, u2, u3, u4; unsigned int *uPtr; int i; int u0_addr, u1_addr, u2_addr, u3_addr, u4_addr; signed int i32_arr[n] = { 0x00000041, 0x00000052, 0xabcd1169, '1234', 'aABc' }; signed int i0, i1, i2, i3, i4;

	signed int *iPtr; int i0_addr, i1_addr, i2_addr, i3_addr, i4_addr;
containers for storing data values. In C++, there are different types of **variables** (defined with different keywords), for example: int - stores integers (whole numbers), without decimals, such as 123 or -123. bool - stores values with two states: true or false.	
06	cout << "++\n"; cout << "The value of each element of 32 bit unsigned int array" << endl << endl << 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.	
07	for (i = 0; i < n; i++)
A loop is used for executing a block of statements repeatedly until a particular condition is satisfied. For example, when you are displaying number from 1 to 100 you may want set the value of a variable to 1 and display it 100 times, increasing its value by 1 on each loop iteration.	
08	cout<<"The value and the size of element" <<dec<<i; cout << " in an array are 0x" << hex << u32_arr[i]<< "and " << sizeof(u32_arr[i]) << "byte(s)" << 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("--\n");
his is mainly used in C language. It is a formatting function that prints to the standard out.	
10	cout << "The memory address of the array is 0x" << hex << uPtr << 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.	

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

_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.	

12	mov EBX, uPtr;
	
	mov EAX, [EBX];

```
mov u0, EAX;
mov u0_addr, EBX;

mov ECX, 4H;
add EBX, ECX;

mov EAX, [EBX];
mov u1, EAX;
mov u1_addr, EBX;

mov EAX, [EBX + 4H];
mov u2, EAX;

mov EDI, EBX;
add EDI, 4H;
mov u2_addr, EDI;

mov EAX, [EBX + 8H];
mov u3, EAX;

mov EDI, EBX;
add EDI, 8H;
mov u3_addr, EDI;

mov EAX, [EDI + 4H];
mov u4, EAX;

mov ESI, EDI;
add ESI, 4H;
mov u4_addr, ESI;
```

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

13

```
cout << "-------------------------------------------------------" << endl;
cout << "The retrieved values 0x" << hex << u0 << "\t at the address of 0x" << hex << u0_addr
<< endl;
cout << "The retrieved values 0x" << hex << u1 << "\t at the address of 0x" << hex << u1_addr
<< endl;
cout << "The retrieved values 0x" << hex << u2 << "\t at the address of 0x" << hex << u2_addr
<< endl;
cout << "The retrieved values 0x" << hex << u3 << "\t at the address of 0x" << hex << u3_addr
<< endl;
cout << "The retrieved values 0x" << hex << u4 << "\t at the address of 0x" << hex << u4_addr
<< endl;
cout <<
"++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++\n";
```

	cout << "--- The value of each element of 32-bit signed int array ----" << endl << endl << 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.	
14	for (i = 0; i < n; i++)
A loop is used for executing a block of statements repeatedly until a particular condition is satisfied. For example, when you are displaying number from 1 to 100 you may want set the value of a variable to 1 and display it 100 times, increasing its value by 1 on each loop iteration.	
15	cout << "The value and the size of element " << dec << i; cout << " in an array are 0x" << hex << i32_arr[i] << " and " << sizeof(i32_arr[i]) << " byte(s) " << 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.	
16	printf("----------\n");
his is mainly used in C language. It is a formatting function that prints to the standard out.	
17	cout << "The memory address of the array is 0x" << hex << iPtr << 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.	

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

_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	_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 EBX, iPtr;
	
	mov EAX, [EBX];
	mov i0, EAX;
	mov i0_addr, EBX;

```
mov ECX, 4H;
add EBX, ECX;

mov EAX, [EBX];
mov i1, EAX;
mov i1_addr, EBX;

mov EAX, [EBX + 4H];
mov i2, EAX;

mov EDI, EBX;
add EDI, 4H;
mov i2_addr, EDI;

mov EAX, [EBX + 8H];
mov i3, EAX;

mov EDI, EBX;
add EDI, 8H;
mov i3_addr, EDI;

mov EAX, [EDI + 4H];
mov i4, EAX;

mov ESI, EDI;
add ESI, 4H;
mov i4_addr, ESI;
```

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

21

```
cout << "------------------------------------------------------------" << endl;
cout << "The retrieved values 0x" << hex << i0 << "\t at the address of 0x" << hex << i0_addr << endl;
cout << "The retrieved values 0x" << hex << i1 << "\t at the address of 0x" << hex << i1_addr << endl;
cout << "The retrieved values 0x" << hex << i2 << "\t at the address of 0x" << hex << i2_addr << endl;
cout << "The retrieved values 0x" << hex << i3 << "\t at the address of 0x" << hex << i3_addr << endl;
cout << "The retrieved values 0x" << hex << i4 << "\t at the address of 0x" << hex << i4_addr << endl;
cout << "\n\n\n\n\n";
```

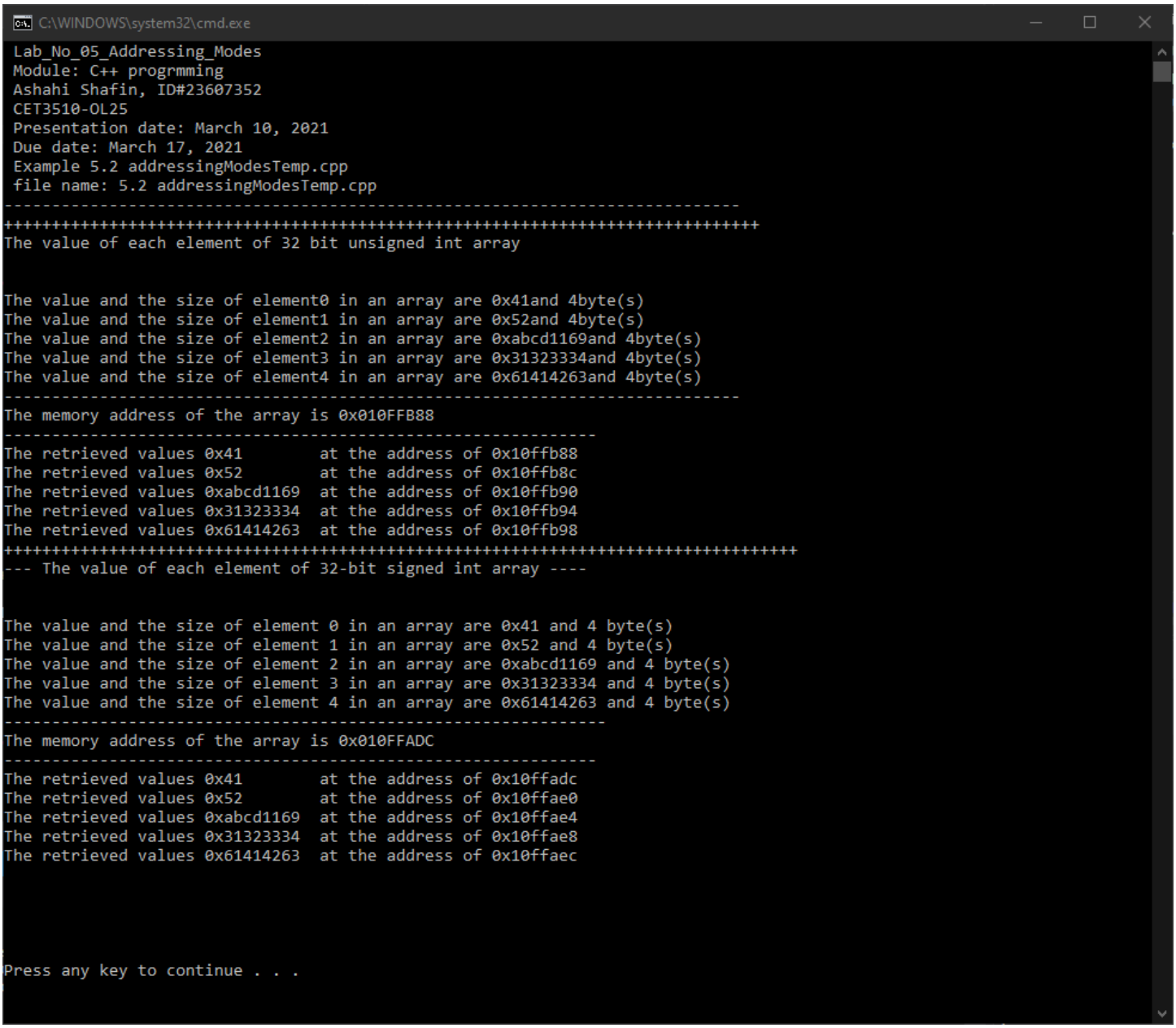
The **cout** object in **C++** is an object of class ostream. It is used to display the output to the standard output device i.e. monitor. It is associated with the standard C output stream stdout.

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	exit(0);
he value supplied as an argument to exit is returned to the operating system as the program's return code or exit code.	
24	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).	

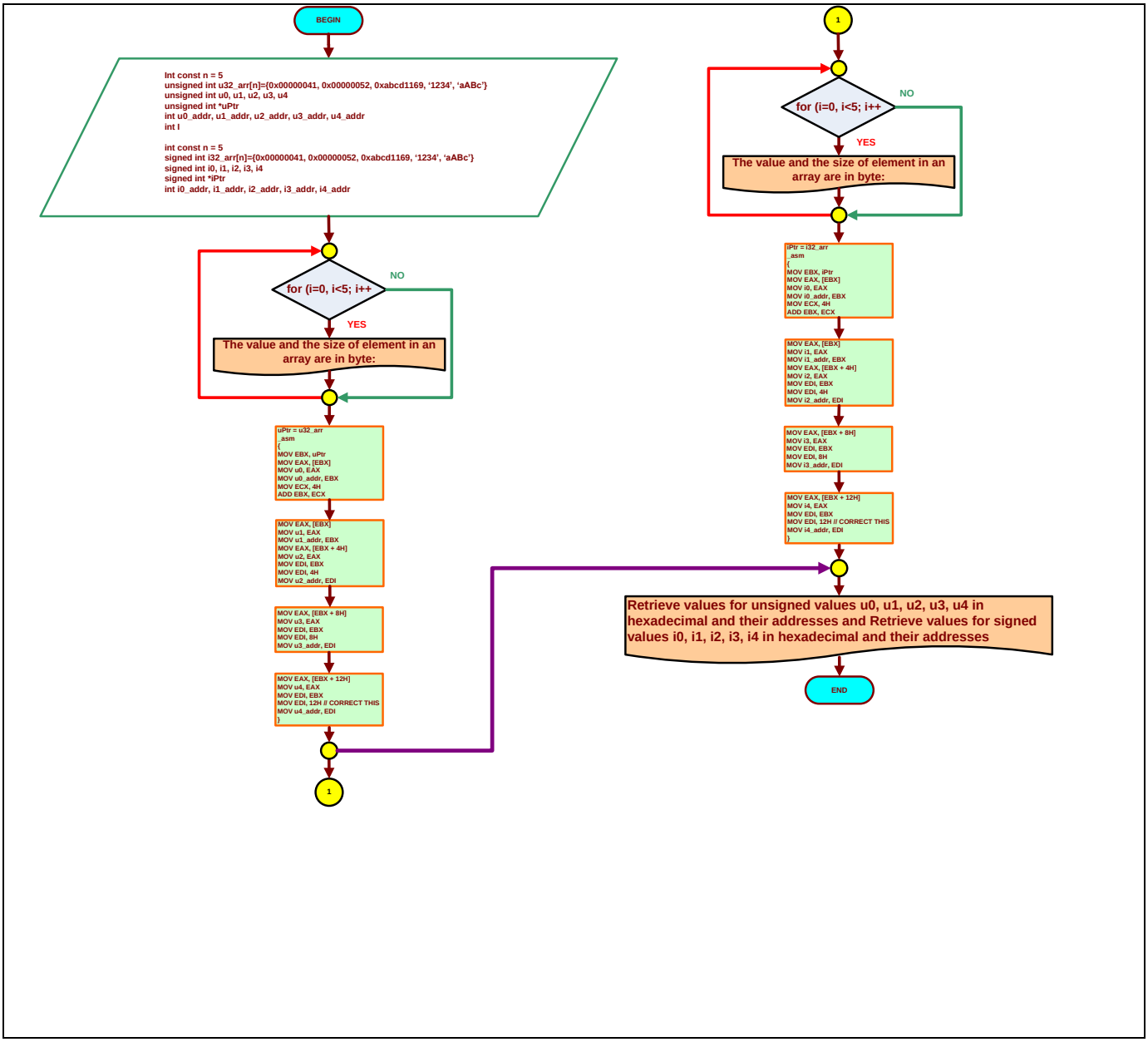
6. EXPLANING OUTPUT RESULTS

Laboratory No_05 Example 5.2

i. This is the output console of example 5.2 for laboratory 5. It shows the unsigned and signed memory in the hex and then show the addresses for them. This is done by the array and the variables that is used to make this work.



7. PROGRAM FLOW CHART:



8. *Table No 5.2 data type, memory contents, and memory addresses*

Data type	*Variable name*	*Memory content (hexadecimal)*	*Address (hexadecimal)*
unsigned int	*u32_arr[0]*	*0x41*	*0xf7fe98*
	u32_arr[1]	*0x52*	*0xf7fe9c*
	u32_arr[2]	*0xabcd1169*	*0xf7fea0*
	u32_arr[3]	*0x31323334*	*0xf7fea4*
	u32_arr[4]	*0x61414263*	*0xf7fea8*
signed int	*i32_arr[0]*	*0x41*	*0xf7fdec*
	i32_arr[1]	*0x52*	*0xf7fdf0*
	i32_arr[2]	*0xabcd1169*	*0xf7fdf4*
	i32_arr[3]	*0x31323334*	*0xf7fdf8*
	i32_arr[4]	*0x61414263*	*0xf7fdfc*

9. COMMENTS

Comments on Outputs and Tables show. Is that they show how the hex can be different when it is signed or unsigned and the addresses to in the array

10.CONCLUSION

In this experiment, the module shows that when you have signed and unsigned variable in 32 bit it can make the hex and hex address different from each other. They use also move in the registers to make the output different. The hardest part of the lab was making the cod because it use unsigned and signed variables which made it the code much longer but they followed the same format which mean I had to just copy and paste after the first part of the lab.

A real-life application for laboratory No 5 would be in a computer because they are always transferring data and files and with this, they can communicate with each other. Not only that they can use the addresses to

find each other if someone was hacking from another place and then the police and get them. Another is in sever system because they need to comminate with a lot of computers and user in order to send data and the addresses.