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_04

Lab_No_04 Data Memory Address

Professor: Dr. Rubén Velásquez, Ph.D.

Spring 2021

Student's Name: Ashahi Shafin

Student's ID: 23607352

Preparation date: *March 03, 2021*

Due date: *March 10, 2021*

1. Table of content

2. OBJECTIVE

Laboratory No 4. To access and explain the computer memory addresses. To develop a program for moving data between the CPU and RAM and find the contents and addresses for each memory variable. To exam the address relationship among the memory variables and declared pointers. To find the size of a variable in bytes.

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_04 Example 4.3

Source Code

```
#include <stdio.h>
#include <iostream>
int main()
{
	printf(" Lab_No_04_Memory_Addresses\n");
	printf(" Module: C++ progrmming \n");
	printf(" Ashahi Shafin, ID#23607352\n");
	printf(" CET3510-OL25\n");
	printf(" Presentation date: March 03, 2021\n");
	printf(" Due date: March 10, 2021\n");
	printf(" Example 4.3 addressContentTemp32bits.cpp\n");
	printf(" file name: 4.3 addressContentTemp32bits.cpp\n");
	printf("-------------------------------------------------------------------------------
\n");

	int const size = 4;
	int i32_arr[size] = { 0x8000000, 0x0000ffff, -660000, 6600000}; //32 bit array
	int *ivPtr0, *ivPtr1, *ivPtr2, *ivPtr3;                          //they are used to store
the addresses
	int md[4], madd[4];
	int i;                                                           // used as index

	//address for each element in an array i32_arr[4]
	ivPtr0 = &i32_arr[0];
	ivPtr1 = &i32_arr[1];
	ivPtr2 = &i32_arr[2];
	ivPtr3 = &i32_arr[3];
```

```
	//Display Hax value, decimal value, and char value for each element of t array
	printf("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n"
);
	printf("----------The address of each element of 32 bit array--------\n\n");
	for (i = 0; i<size; i++)
{
		printf("The value of element %d in an array is 0x%X (hex), %d (decimal)\n",
			i, i32_arr[i], i32_arr[i], i32_arr[i]);
}
	//Display address in hexadecimal for each element of the array
	printf("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n"
);
	printf("----------The address of each element of 32 bit array--------\n\n");
	for (i = 0; i<size; i++)

{
	printf("The memory address of element %d in an array is 0x%x (hexidecimal),\n", i,
ivPtr0+i);
}
	printf("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n"
);

	//Find the total numbers in byte of array i32_arr[4]
	printf("----------The size information in bytes of an 32 bit array--------\n");
	printf("The total numbers in bytes of an 32 bit array with 4 elements in %d bytes\n",
		sizeof (i32_arr));

	//Find the number of bytes of each element
	for (i = 0; i<size; i++)
{
		md[i] = sizeof(i32_arr[i]);
		madd[i] = sizeof(ivPtr0+i);
		printf("---------------------------------------\n");
		printf("The size of element %d is %d bytes\n", i, md[i]);
		printf("The size of the address of the element %d is %d bytes\n ", i, madd[i]);
}

	system("pause");
		return 0;

}
```

5. *SOURCE CODE LINE DESCRIPTION*

Laboratory No_04 Example 4.3

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");
printf("----------The address of each element of 32 bit array--------\n\n");
```

Output console to indicate separation marked with asterisks within parentheses.

05

```
int const size = 4;
int i32_arr[size] = { 0x8000000, 0x0000ffff, -660000, 6600000};
int *ivPtr0, *ivPtr1, *ivPtr2, *ivPtr3;
int md[4], madd[4];
int i;
```

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

06

```
ivPtr0 = &i32_arr[0];
ivPtr1 = &i32_arr[1];
ivPtr2 = &i32_arr[2];
ivPtr3 = &i32_arr[3];
```

the **array**, which stores a fixed-size sequential collection of elements of the same type. An **array** is used to store a collection of data, but it is often more useful to think of an **array** as a collection of variables of the same type.

07

```
for (i = 0; i<size; i++)
```

A for **loop** is a repetition control structure that allows you to efficiently write a **loop** that needs to execute a specific number of times.

08

```
printf("The value of element %d in an array is 0x%X (hex), %d (decimal)\n",
i, i32_arr[i], i32_arr[i], i32_arr[i]);
```

Output console to indicate separation marked with asterisks within parentheses.

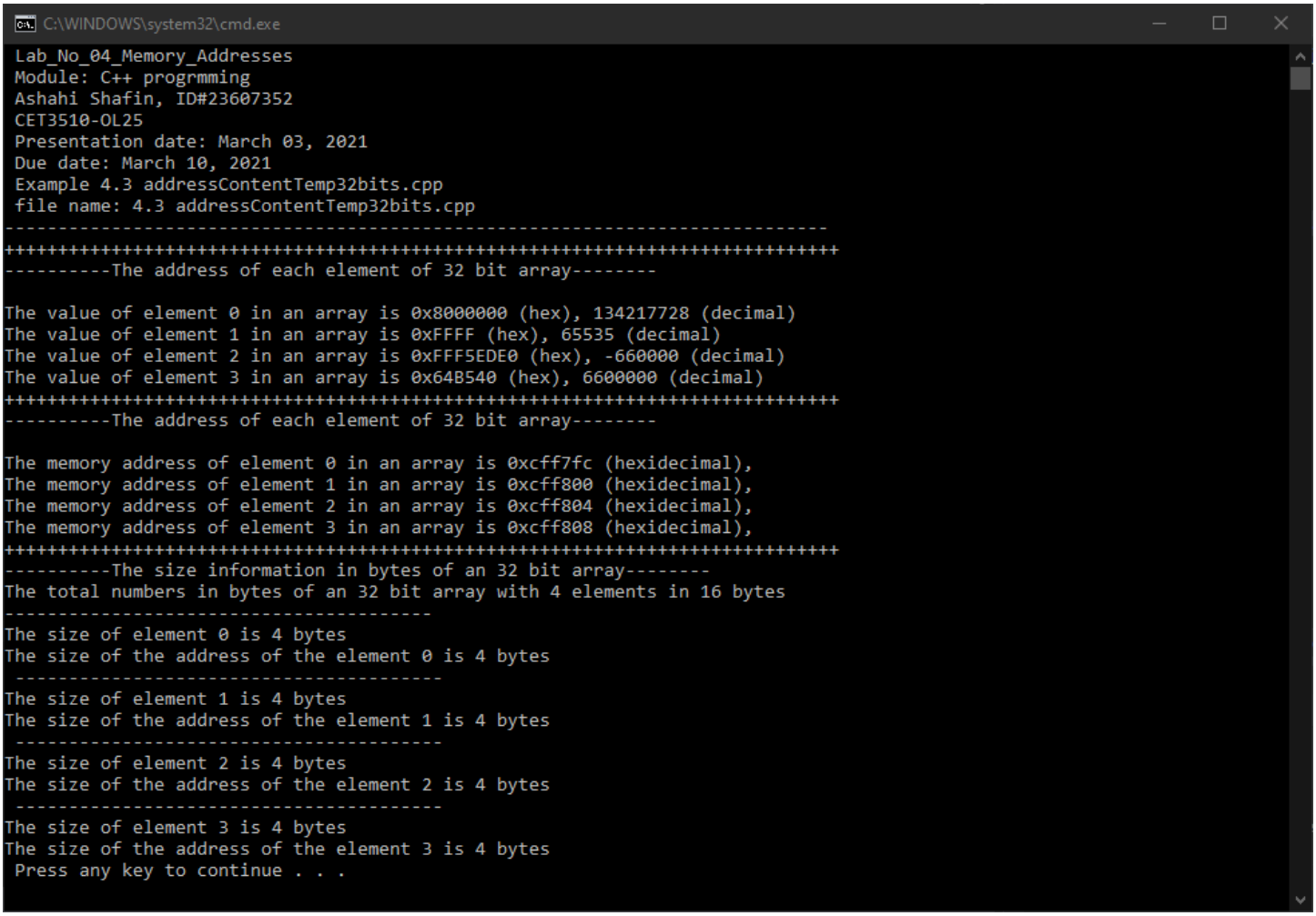
09	printf("++\n"); printf("----------The address of each element of 32 bit array--------\n\n");
Output console to indicate separation marked with asterisks within parentheses.	
10	for (i = 0; i<size; i++)
A for **loop** is a repetition control structure that allows you to efficiently write a **loop** that needs to execute a specific number of times.	
11	printf("The memory address of element %d in an array is 0x%x (hexidecimal),\n", i, ivPtr0+i); printf("++\n"); printf("----------The size information in bytes of an 32 bit array--------\n"); printf("The total numbers in bytes of an 32 bit array with 4 elements in %d bytes\n",
Output console to indicate separation marked with asterisks within parentheses.	
12	sizeof (i32_arr));
The **sizeof** is a keyword, but it is a compile-time operator that determines the **size**, in bytes, of a variable or data type. The **sizeof** operator can be used to get the **size of** classes, structures, unions and any other user defined data type. The syntax of using **sizeof** is as follows − **sizeof** (data type)	
13	for (i = 0; i<size; i++)
A for **loop** is a repetition control structure that allows you to efficiently write a **loop** that needs to execute a specific number of times.	
14	md[i] = sizeof(i32_arr[i]); madd[i] = sizeof(ivPtr0+i);
The **sizeof** is a keyword, but it is a compile-time operator that determines the **size**, in bytes, of a variable or data type. The **sizeof** operator can be used to get the **size of** classes, structures, unions and any other user defined data type. The syntax of using **sizeof** is as follows − **sizeof** (data type)	
15	printf("---------------------------------------\n"); printf("The size of element %d is %d bytes\n", i, md[i]); printf("The size of the address of the element %d is %d bytes\n ", i, madd[i]);
Output console to indicate separation marked with asterisks within parentheses.	
16	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.	
17	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.

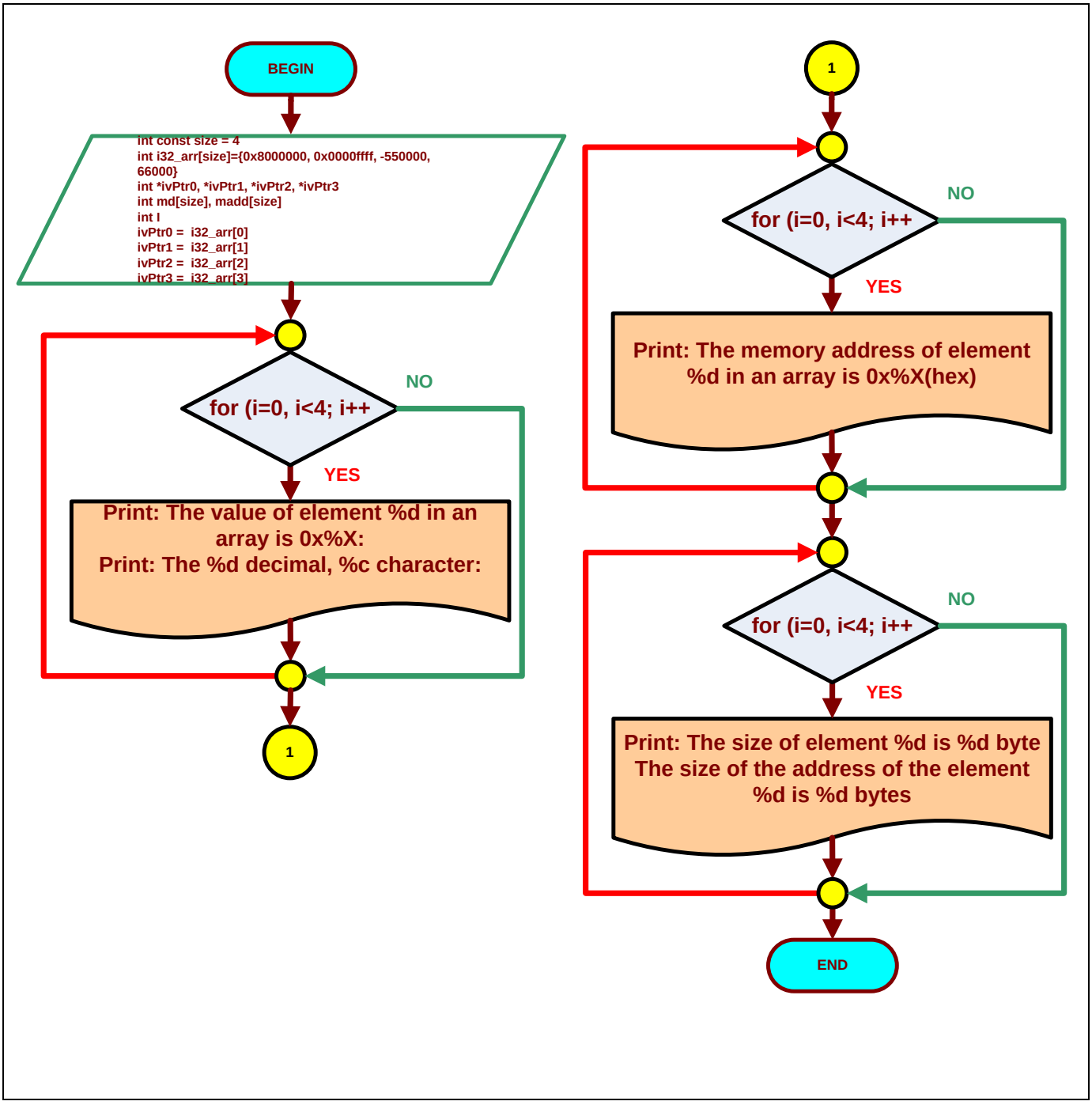
6. *EXPLANING OUTPUT RESULTS*

Laboratory No_03 Example 4.3

i. This is the output console of example 4.3 for laboratory 4. It shows 32-bit array with 4 elements. It also shows the hex and decimal value of the 4 elements. After that it show the address of the 4 elements. Then it shows the size information in bytes. Also we did not do 8 or 16 bit because than it would be too long.



7. *PROGRAM FLOW CHART:*



8. Table No 4.3 data type, memory contents, and memory addresses

Data type	*Variable name*	*Hexadecimal*	*Decimal*	*Address*
int (an array with 4 elements)	*i32_arr[0]*	*0x8000000*	*134217728*	*0x3dfed0*
	i32_arr[1]	*0xFFFF*	*65535*	*0x3dfed4*
	i32_arr[2]	*0xFFF5EDE0*	*-660000*	*0x3dfed8*
	i32_arr[3]	*0x64B540*	*6600000*	*0x3dfedc*

9. COMMENTS

Comments on Outputs shows how Elements are getting the hex and decimal value and getting the address and storge information as well.

10.CONCLUSION

In this experiment, the module shows when you add elements to your variable it can find the hexadecimals, decimals, addresses, and size information. These are also in 32 bits because if we did 8 and 16 bit this lab would be much longer, and it will have more variable. That will make the output too long and then take a lot of time and explaining. What I had a hard time toing in this lab was making the code because I put 3 extra semicolon and didn't realized I put them since I'm used to putting it at the end of every line.

A real-life application for laboratory No 4 is a computer because the computer they use these variables to commutate with each other this is also very helpful with data transferring when it decode and decode it. With the addresses you tell which computer it is and what size they can have.