**ECE 212 Lab - Introduction to Microprocessors**

**Department of Electrical and Computer Engineering**

**University of Alberta**



**Lab 2: Addressing Modes**

| **Student Name** | **Student ID** |
| --- | --- |
| Darius Fang | 1570975 |
| Marlene Gong | 1572719 |

Table of Contents

[Introduction 3](#_30j0zll)

[Design 4](#_1fob9te)

[**Part A** 4](#_3znysh7)

[**Part A Flowchart** 4](#_2et92p0)

[**Part A Assembler Code**](#_tyjcwt) 5

[**Part B**](#_3dy6vkm) 7

[**Part B Flowchart**](#_1t3h5sf) 8

[**Part B Assembler Code**](#_4d34og8) 9

[Testing](#_2s8eyo1) 10

[**Part A**](#_17dp8vu) 10

[**Part B**](#_3rdcrjn) 11

[Questions](#_26in1rg) 11

[Conclusion](#_lnxbz9) 13

Marking Sheet 14

# Introduction

For Part A of this lab, we were asked to write a program using three different addressing modes. The program would add the adjacent contents of two given arrays and would store the sum, depending on which address mode was used, in a specific memory location. Sums calculated using Register Indirect with Offset ( d16(An) ) would be stored in memory location 0x4300000C, Indexed Register Indirect ( (An, Xn\*SF) ) in 0x43000010, and Postincrement Register Indirect ( (An)+ ) in 0x43000014. Only the first three sums were required to be calculated for the method using Register Indirect with Offset since this addressing mode doesn’t allow for incrementation and thus, isn’t typically used in loops. For the other two modes, sums for the entire array had to be calculated.

In Part B, we had to write a program that used the Trapezoidal Rule

to calculate the approximate area underneath a curve. The x data points of the curve were stored in memory location 0x43000004 and the y data points were stored in 0x43000008. was conveniently one, two, or four units. Once all data points have been accounted for, the total area was calculated and the value would be stored in 0x43000010, hence ending the program.

The main objective of this lab was to familiarize students with accessing memory using different addressing modes and overall, gain more experience in coding with assembly language.

# Design

## **Part A**

Part A is used to demonstrate how to add 2 arrays together using 3 different address modes. The address to obtain the values that are going to be added will be at address 0x43A00000 and address 0x43500000. The first part of part A uses the address mode register indirect with offset, with only doing the first 3 summations of the two arrays, and the sum will be moved to address 0x43C00000. The 2nd part uses the address mode index register indirect with a loop depending on the size of the array (9 in this case) . Every time the function loops, it adds one to the counter and 4 to the address offset. Since every value is a longword, offsetting by 4 makes it so that it goes to the next memory location provided. The sum will be moved to address 0x43200000. The last part uses address mode post increment register indirect. There will be a counter to determine how many times it will loop (9 in this case). This function automatically goes to the next memory address after it has obtained the values to sum. The sum will be moved to address 0x43F00000.

## **Part A Flowchart**

## **Part A Assembler Code**

/\* DO NOT MODIFY THIS --------------------------------------------\*/

.text

.global AssemblyProgram

AssemblyProgram:

lea -40(%a7),%a7 /\*Backing up data and address registers \*/

movem.l %d2-%d7/%a2-%a5,(%a7)

/\*----------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* General Information \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* File Name: Lab2a.s \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Names of Students: Darius Fang and Marlene Gong \*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* IDs: 1570975 and 1572719 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Date: February 26, 2020 \*\*/

/\* General Description: Program adds the adjacent contents of \*\*/

/\* two arrays and places the results in a memory location \*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*Write your program here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*Part A \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Register Indirect with Offset \*/

move.l (0x43000004), %a2 /\* set a2 to point at first value in first array \*/

move.l (0x43000008), %a3 /\* set a3 to point at first value in second array \*/

move.l (0x4300000C), %a4 /\* set a4 to point at location for first sum \*/

move.l (%a2), %d2 /\* copy first value of first array to d2 \*/

add.l (%a3), %d2 /\* add first values of both arrays, store in d2 \*/

move.l %d2, (%a4) /\* copy sum to location for first sum \*/

move.l 4(%a2), %d2 /\* copy second value of first array to d2 \*/

add.l 4(%a3), %d2 /\* add second values of both arrays, store in d2 \*/

move.l %d2, 4(%a4) /\* copy sum to location for second sum \*/

move.l 8(%a2), %d2 /\* copy third value of first array to d2 \*/

add.l 8(%a3), %d2 /\* add third values of both arrays, store in d2 \*/

move.l %d2, 8(%a4) /\* copy sum to location for third sum \*/

/\*Part B \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Indexed Register Indirect \*/

move.l (0x43000010), %a4 /\* set a4 to point at location for first sum \*/

move.l (0x43000000), %d1 /\* copy size of arrays to d1 \*/

clr.l %d4 /\* clear d4 \*/

clr.l %d5 /\* clear d5, use as counter \*/

cmp.l %d1, %d5 /\* compare counter to size of arrays \*/

ble loop /\* if less than or equal to, proceed to loop \*/

loop:

move.l (%a2, %d4), %d2 /\* increment a2 pointer by d4, copy value to d2 \*/

add.l (%a3, %d4), %d2 /\* increment a3 pointer by d4, add value to content of d2 \*/

move.l %d2, (%a4, %d4) /\* increment a4 pointer by d4 and copy d2 to it \*/

add.l #1, %d5 /\* increment counter \*/

add.l #4, %d4 /\* increment d4 by 4 \*/

cmp.l %d1, %d5 /\* compare counter to size of arrays \*/

ble loop /\* if less than or equal to, proceed to loop \*/

/\*Part C \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Postincrement Register Indirect \*/

move.l (0x43000014), %a4 /\* set a4 to point at location for first sum \*/

clr.l %d4 /\* clear d4, use as counter \*/

loop1:

move.l (%a2)+, %d2 /\* copy value of first array to d2 then increment a2 \*/

add.l (%a3)+, %d2 /\* add values of both arrays, store in d2 then increment a3 \*/

move.l %d2, (%a4)+ /\* copy sum to location for sum then increment a4 \*/

add.l #1, %d4 /\* increment counter \*/

cmp.l %d1, %d4 /\* compare counter to size of arrays \*/

ble loop1 /\* if less than or equal to, proceed to loop1 \*/

/\*End of program \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* DO NOT MODIFY THIS --------------------------------------------\*/

movem.l (%a7),%d2-%d7/%a2-%a5 /\*Restore data and address registers \*/

lea 40(%a7),%a7

rts

/\*----------------------------------------------------------------\*/

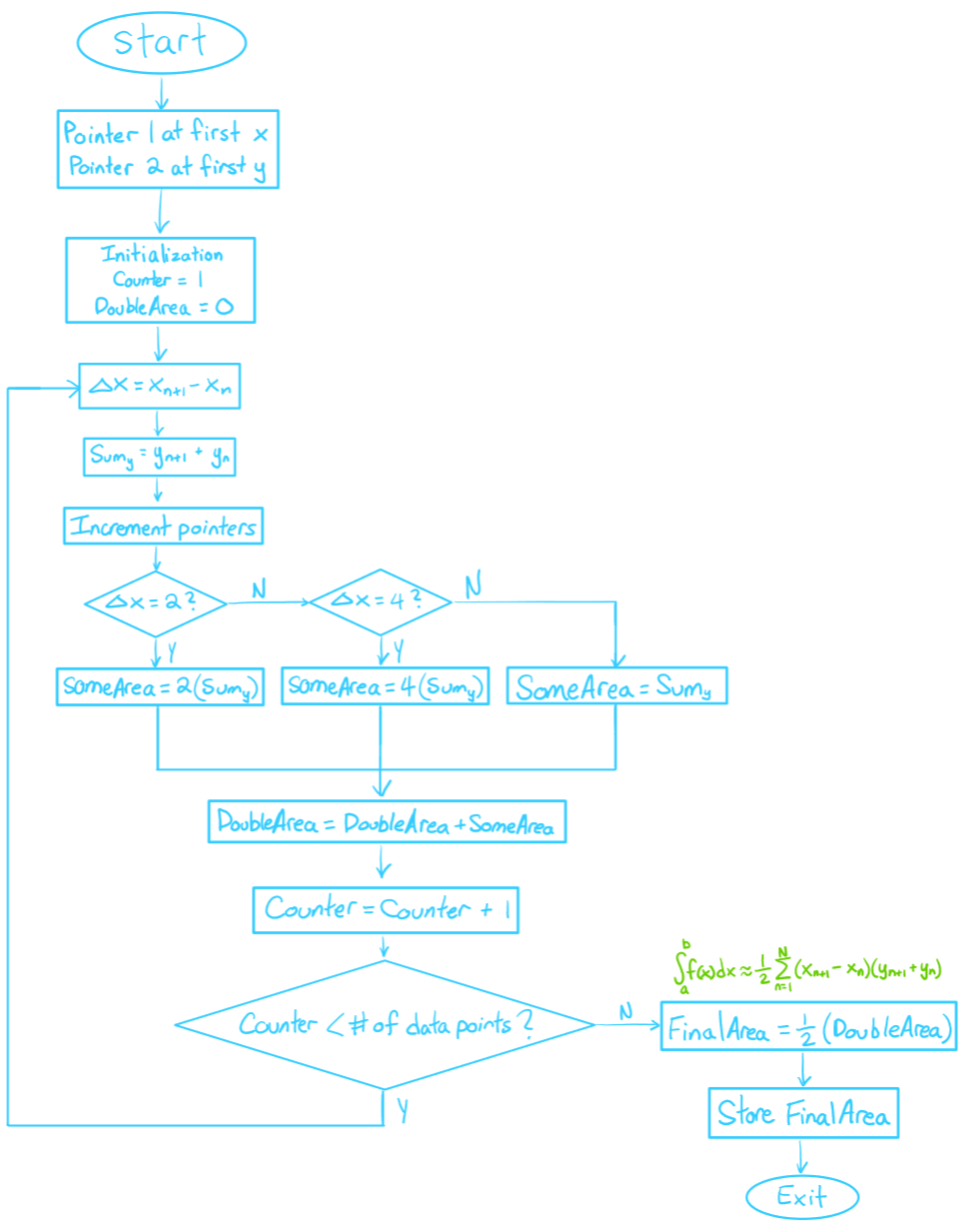
## 

## **Part B**

The trapezoidal rule uses the summation of areas to estimate the total area of the data points. The area is calculated by the summation of the two y points, and the absolute difference of two x points. The difference of x is guaranteed to be 1,2,4. We first obtain the 4 value points and do the calculations just using 2 data registers. We then shift the bits of the summation of the 2 y points. 1 shift to the left if the change in x is 2, and 2 shifts to the left if the change in x is 4, otherwise do not shift. This value is then added to a data register that stores the total. The number of times it loops is the number of points -1. For example, if there are only 2 points to be addressed, only 1 area should be calculated. Once we obtain the total area, we then shift the data register that stores the total area to the right once so that the total area is divided by 2. In our code for this design, A2 is the pointer of the first value of the x array, A3 is the pointer of the fist value of the y array, A4 is the output, the summation of the area. D2 determined the number of points, D3 determined delta x, D4 determined the sum of the 2 y points, D5 determined the sum of trapezoids, D6 was used as a counter to compare the number of points.

## 

## **Part B Flowchart**



## **Part B Assembler Code**

/\* DO NOT MODIFY THIS --------------------------------------------\*/

.text

.global AssemblyProgram

AssemblyProgram:

lea -40(%a7),%a7 /\*Backing up data and address registers \*/

movem.l %d2-%d7/%a2-%a5,(%a7)

/\*----------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* General Information \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* File Name: Lab2.s \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Names of Students: Darius Fang and Marlene Gong \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* IDs: 1570975 and 1572719 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Date: February 26, 2020 \*\*/

/\* General Description: Program calculates the area underneath a \*\*/

/\* curve using the Trapezoidal rule \*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*Write your program here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

move.l (0x43000000), %d2 /\* move the number of points to d2 \*/

move.l (0x43000004), %a2 /\* set a2 to point at first x data point \*/

move.l (0x43000008), %a3 /\* set a3 to point at first y data point \*/

move.l (0x43000010), %a4 /\* set a4 to point inside address for final area value \*/

clr.l (%a4) /\* clear contents of a4 \*/

clr.l %d5 /\* clear contents of d5, use d5 to calculate area \*/

move.l #1, %d6 /\* counter \*/

loop:

move.l 4(%a2), %d3 /\* store x\_(n+1) in d3 \*/

sub.l (%a2)+, %d3 /\* x\_(n+1) - x\_n = delta x, store delta x in d3, increment pointer\*/

move.l 4(%a3), %d4 /\* store y\_(n+1) in d4 \*/

add.l (%a3)+, %d4 /\* sum of y\_(n+1) and y\_n, store in d4, increment pointer \*/

cmpi.l #2, %d3 /\* compare delta x to 2 \*/

beq double /\* if delta x is 2, proceed to double \*/

bgt quad /\* if delta x is 4, proceed to quad \*/

bra compare /\* otherwise (delta x is 1), proceed to compare \*/

double:

lsl.l #1, %d4 /\* multiply the sum of y in d4 by 2 \*/

bra compare /\* proceed to compare \*/

quad:

lsl.l #2, %d4 /\* multiply the sum of y in d4 by 4 \*/

bra compare /\* proceed to compare \*/

compare:

add.l %d4, %d5 /\* add contents of d4 to d5 \*/

add.l #1, %d6 /\* increment counter \*/

cmp.l %d2, %d6 /\* compare counter to the number of data points \*/

blt loop /\* if less than, proceed to loop \*/

lsr.l #1, %d5 /\* divide contents of d5 to get area \*/

move.l %d5, (%a4) /\* store final area value \*/

/\*End of program \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* DO NOT MODIFY THIS --------------------------------------------\*/

movem.l (%a7),%d2-%d7/%a2-%a5 /\*Restore data and address registers \*/

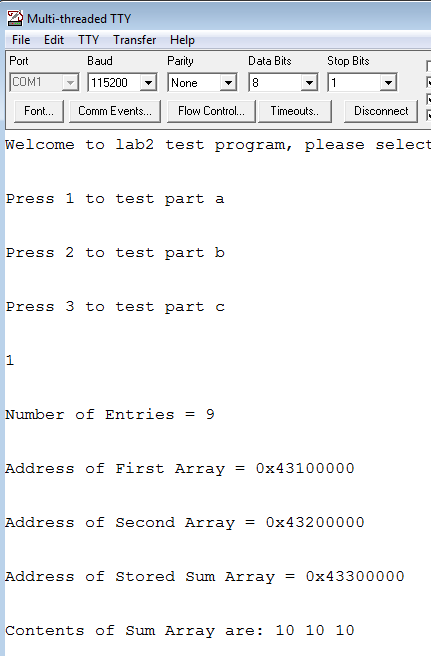
lea 40(%a7),%a7

rts

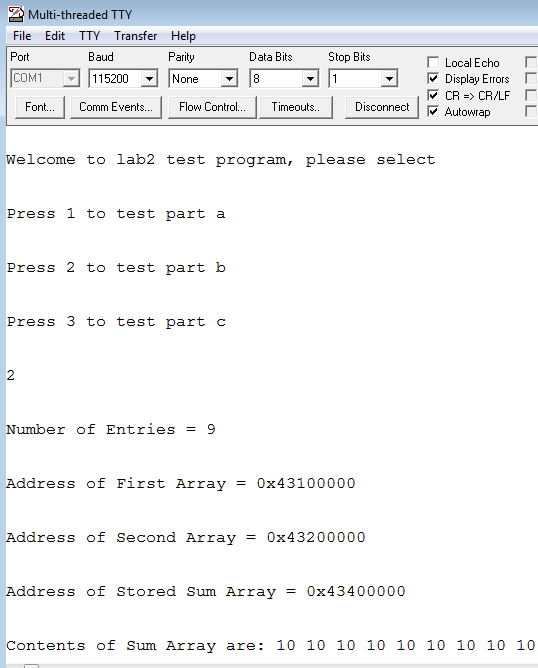
/\*----------------------------------------------------------------\*/

# Testing

## **Part A**

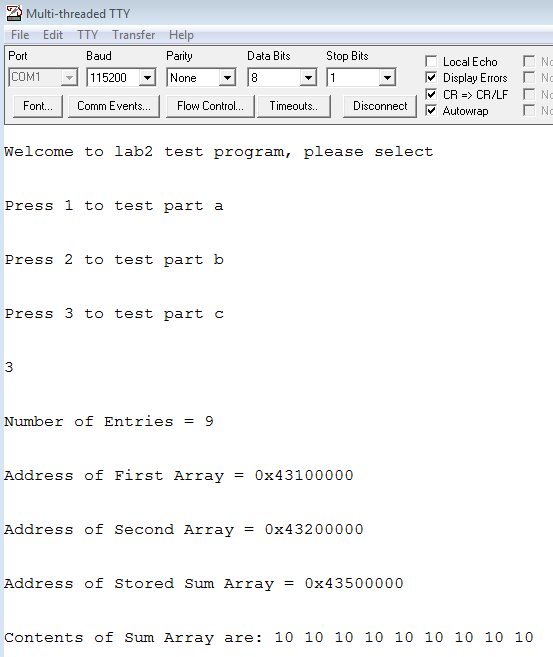


Entering 1 in part 1 of a lab. The number of entries is 9, that means that there are 9 values to be summed, but we only have summed 3 values for part 1. The address of the arrays are there to show where the values are being extracted and inserted . The sums shown are 10 10 10, from 1+9, 2+8, 3+7. This is using the offset method.



Similar to part 1, entering 2 for part 2 uses a different output address to store the array. Again the sums are all 10 but this time having a looping counting to 9. This is using the index register offset.

## 

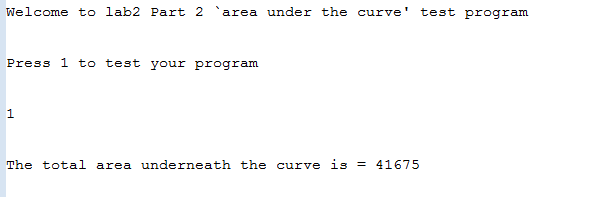
Lastly, entering 3 for part 3 gives the last output of the last stored array. Again the sums are all 9 but this time using the post increment mode.

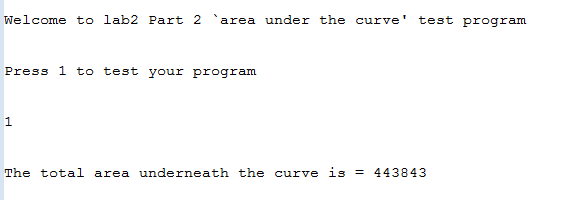
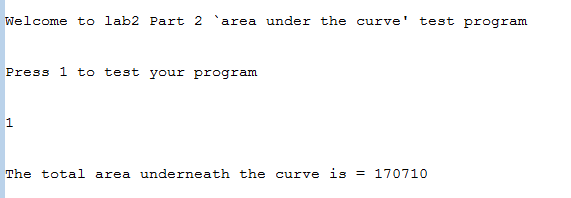
Part A only used DataStorage.s for all 3 parts.

## 

## 

## **Part B**



Part B used 3 different files to test each case. The first one used was DataStorage4, then DataStorage5, and lastly DataStorage6. All of these storages plotted a function that was . Storage4 plotted to 50, Storage5 plotted 80, and storage plotted 110. As you can see from the results, the values were close to the value that you would get if we did the calculations at that point and thus the trapezoidal rule was put into places for these 3 tests

# 

# Questions

1. *What are the advantages of using the different addressing modes covered in this lab?*

***Answer:*** The advantages of using different addressing modes is that they each give different sub instructions that can help compact and condense coding, thus make the code more legible to read and faster to compile. Address mode register indirect with offset uses an immediate value, and is usually used for only finding a few values very quickly. Index register indirect is similar to indirect with offset, but instead of using an immediate value, it uses a register. This address mode allows for greater range and is very flexible in considering values but is very bulky. Lastly, using post increment, this mode is optimally used in getting values of an array very quickly, or to be used as a pop in stack.

1. *If the difference between the X data points are not restricted to be either one or two units, how would you modify your program to calculate the area? You do not need to do this in your code.*

***Answer:*** If the difference in x points were not a number divisible by 2, then we would definitely have to use the mulu function to multiply the 2 values and add that value to the summation of areas.

mulu.l D3,D4

add.l D4, D5

This would be replaced instead of the double and quad functions created in the code.

1. *From the data points, what is the function (y=f(x))? What is the percent error between the theoretical calculated area and the one obtained in your program? Calculate all 3 functions.*

***Answer:*** The function for all three data storages is y = x2.

DataStorage4

Theoretical calculated area =

Experimental calculated area = 41675

Percent error = = 0.02%

DataStorage5

Theoretical calculated area =

Experimental calculated area = 170710

Percent error == 0.03%

DataStorage6

Theoretical calculated area =

Experimental calculated area = 443843

Percent error == 0.04%

# 

# Conclusion

Part A of the lab required us to add the contents of two arrays together and store the results in specific memory locations using three different addressing modes. The addressing modes used were Register Indirect with Offset, Indexed Register Indirect, and Postincrement Register Indirect. In Part B, we were given sets of x and y data points and we had to find the area underneath the curve created by these data points using the Trapezoidal rule. The final area was to be stored and that would be the end of the program. In order to carry out this lab, we used instructions like LSL and LSR to multiply and divide values. We then tested our codes by running them on MTTTY and ended up obtaining the expected results for both parts of the lab. For Part B, the approximation of the area ended up having a percent error of 0.02 to 0.05%.

