**Table of Contents Page**

1. **Introduction**…......................................................................................................ii
2. **Design**

Part A…......................................................................................................................1

Part B…......................................................................................................................2

1. **Testing**

Part A…......................................................................................................................3

Part B…......................................................................................................................3

1. **Questions**….............................................................................................................4
2. **Conclusion**…..........................................................................................................6
3. **Appendix**

Part A Flowchart…................................................................................................7

Part A Assembler Code…....................................................................................8

Part B Flowchart….............................................................................................10

Part B Assembler Code….................................................................................11

1. **Marking Sheet**…................................................................................................13

**Introduction**

In this design laboratory project, we were formally introduced to assembly language programming using different addressing modes. We were instructed to write a program that would add two exact same set of array values but in three different addressing modes. We were also asked to apply our newfound knowledge of assembly language programming to write another program to calculate the area underneath a curve using the Trapezoidal rule.

Firstly, for the program in Part A we added the first three longwords from two particular memory locations and stored the result into another memory location using Register Indirect with Offset, d16(An). Next, we had to do the same again, but instead of three words, this time we added the corresponding elements for the entire length of the two arrays and stored them in a specified memory location using Indexed Register Indirect, d8(An Xn\*Sf). Lastly, we repeated this process and stored the result in the last specified memory address using Post-Increment Address Register Indirect, (An)+.

In Part B of the lab, we had to implement a program that would calculate the area under a curve (y = f(x)) using the Trapezoidal rule for the (x,y) co-ordinates stored at the memory locations stated in the lab document. Taking advantage of the fact that the iterating x values varied from each other by either 1 or 2 (Δx = 1 or 2), we wrote our program such that it would take into account both these cases. Based on the information provided in the lab session, we considered areas where Δx = 1, added up the incrementing areas to a final value. We did the same for Δx = 2 case. Lastly, we summed up the final area values for the two cases to get the overall area.

**Design**

Part A

We wrote our first program in the ‘Lab2a.s’ file provide beforehand. The program in Part A was divided into three parts where we used three different addressing modes. In the following discussion, these subsections are specified as A’, B’ and C’ respectively. Also, the specific commands/instructions used to implement our program are stated when they are first suggested in the following paragraphs and are subsequently not mentioned assuming that the reader of this report would be able to connect the instructions to further related discussions ahead.

We first used the Register Indirect With Offset d16(An) mode to sum up the first three longwords in the given two arrays of numbers. For this, we initialized the first, second and summary arrays with address registers a1, a2 and a3 respectively. with “movea.l” commands. Then, we use “move.l” to move the longwords starting at the memory addresses specified by a1 and a2 to data registers d2 and d3 respectively. Subsequently, we added the contents of d2 and d3 with “add.l” and moved the result in d3 to the memory location starting at the address in a3. It is in this final move instruction that we used the Register Indirect With Offset with 0 for d16 to start storing from the first location in memory implied by a3. We repeated this process for the next two longwords stored in the given arrays with offsets 4 and 8 for d16.

In Part B’, we made the memory address initializations again. However, this time we specified the summation array with a4 at a location different from a3 in Part A’. Also, we moved the size of the array to d1 with the ultimate motive to use it as a counter for the upcoming looping instructions and initialized d4 to 0 to use it as one of the offset causing elements in our chosen addressing mode. Next, we coded a loop with the obvious name “loop” where we repeated the process from Part A – move the memory contents from a1 and a2 to d2 and d3 respectively, add the longwords from d2 and d3 and move the summed up result from d3 to a4. Here, in order to make the move from d3 to a4, we used Indexed Register Indirect with 1 in place of d8, d3 for Xn and d4 for SF. We then incremented d4 by 4 units for the next location after filling up the current memory with a longword and decremented the counter. We repeated this process until all the longwords in the two given arrays have been summed up in which case, our counter would decrement to zero. The “bne” control sequence instruction would check whether our counter is equal to zero or not by looking at the most recent CCR Z bit. If Z = 0, the counter is not yet zero and bne would make the program go back to “loop” again. Otherwise, when Z = 0 our program would just move on to the next set of instructions.

Finally, in Part C’, we repeated the initializations from B’ with the obvious modification for the memory address specification of the summation array via a5 and the exclusion of the data register d4. We then implemented a loop called “loop2” with further repetition of the sets of commands from B’ to move array elements from memory to data registers, add them and then to move the result in memory address implied by a5. In the move instruction for the sum, we used Postincrement Register Indirect where we did not have to specify any offsets since, the “+” after (An)+ in this mode routinely points to the next ‘empty’ memory byte. This particular memory byte for this program is 4 bytes (a longword) away from the previous location pointed by a5. We then decremented our counter and repeated the process in “loop2” until we covered all the elements from the two arrays. When this was over, we just exited the program.

Part B

In order to code a program to determine the area under a curve in this part of the lab, we first consider overall formula of the Trapezoidal rule –

Area under any curve f(x) from (x = a) to (x = b) = Δx \* (f(a) + f(b))\*0.5

(Here Δx = b – a).

We started with the initializations in our ‘Lab2b.s’ file. We stored the number of data points in d1, pointed to the x and y arrays and the final storage address in memory with address registers a1, a2 and a4 respectively. We also initialized data registers d7 and d6 for use as the temporary locations for the area sum for the cases when Δx = 1 and 2 respectively. An important line of code was written where we used the “sub.l” command to subtract 1 from d1. This is because, we ultimately aspired to use d1 as a counter for the intended loop ahead where we would ultimately get x areas with x+1 points where x is any positive integer (x is 51 in our case).

Next, we made a loop with label “iteration”. With our goal of calculating Δx, we moved two successive x values to d2 and d3 respectively and determined the difference between d3 and d2. We then used the “cmp.l” command to compare d3 to the number 2. If the contents of d3 were indeed equal to 2, the “beq” instruction, which would check whether the CCR Z bit, would direct our program to the label “case\_of\_2” (Δx = 2). Otherwise, we would continue with the next set of commands for the case of 1 (Δx = 1). At this point, as our code demonstrates we consider two area scenarios where in one case we have the y values when Δx = 1 and y values for Δx = 2 in the other. Despite these differences, in both these cases we add up two successive values of y for the two x values taken into account earlier. In case\_of\_1, we add these values to d7 and to d6 in case\_of\_2. We repeat this until we have went over every single x and y in the given arrays. Next, when our counter hits zero, we direct our program to “account\_for\_fraction”. Although the case of two moves to the set of instructions spontaneously, we actually have to use the “bra” command for the case\_of\_1 so that our program our unconditionally branch to account\_for\_fraction bypassing case\_of\_2 in case the counter becomes 0 which running down Δx = 1 case.

Now, we carefully consider what happens to the formula of the area under the curve f(x) when Δx = 1 and 2 respectively. When Δx = 2, the number 0.5 in the formula and the Δx cancels each other out and the area is determined only by the summation of the y values. On the other hand, when Δx = 1 we have to divide the summed up y values by 2. This is accomplished in our program where we use the “asr.l” command to arithmetically shift the bits in d7 by in place to the right. This ultimately accounts to dividing the contents of d7 by 21. We then get the overall area values from the two scenarios by adding the contents in d6 and d7. Finally, we store the desired area value in the memory location specified by a4. Once all this is done, we exit our program.

**Testing**

Part A

We first tested our program using the ‘main.cpp’ and ‘Lab2aTest.s’ against the ‘DataStorage.s’ file provided to us by the lab instructor. We ended up getting the resultant values of 10 from the summation of each of the corresponding elements of the two arrays. When we checked the ‘DataStorage.s’ file and manually tried to sum up the values from the two arrays where two arrays/blocks had the numbers from 0 to 9 in increasing and decreasing orders respectively, we also got 10 summed up result. Next, we looked over the ‘DataStorage1.s’ file that was also provided to us. This time, the arrays were arranged such that the first block had 0xA to 0xF in increasing order and the reverse in the second block. Thus, the result for each corresponding set of numbers was 25 and that is exactly, what we say when we ran our program against DataStorage1.

We further checked the validity of our program using our own set of values in the Debugger mode. In this mode, we tapped into the memory and changed the values at the locations where the values were initially stored into simple positive integers, like – 1, 2, 3 and so on. Then we “stepped over” every line of code for five or six iterations and ended up getting the result that we manually calculated. We also ran our program to see its reliability against negative integers. However, this time, instead of changing numbers in the actual memory grid in Debugger mode, we changed the numbers in the first and second block of ‘DataStorage1.s’ into negative numbers such as 0xFF (-1 in decimal), 0xFE (-2 in decimal) and so on. Our program successfully implemented the summation operation again and displayed the results in decimal format.

Part B

As always, we started our testing stage for Part using the files in Lab2bTest Program. We initially ran our program using ‘DataStorage4.s’. When we first ran our program, we received a massive output number as the area instead of the 41675 sq. units as specified by the instructor in the lab. Subsequent investigation revealed that we missed the important design step of subtracting one from the data register d1 that was originally the counter for the number of data points in the x and y arrays and also, the counter for our program. We subtracted 1 from d1 in order to account for 50 smaller areas using 51 points in the given DataStorage4.s document. Without this subtraction, our program was tapping into the 52nd memory block that, as noticed in the Debugger mode, contained huge numbers in both the x and y arrays and ultimately invalidated our moderately small area with a gigantic iterated area. After taking care of this tiny but significant detail in our program, we ran it again and received the desired result. We also, checked our program against ‘DataStorage5.s’ and received the result as suggested by the lab instruction which as 170710 sq. units.

As a final check for our own satisfaction, we altered the first six values in each of the two memory blocks in DataStorage5, ran our program iterating only over those six modified x and y values. The area that the program predicted was slightly off and smaller than what calculated by hand. This can be accounted by the “asr.l” command via which we got rid of some low significant bits. Also, it should be noted that we only carefully considered the x values such that Δx was either 1 or 2 when we modified the DataStorage5.s file.

Although our program would work successfully against positive x values with Δx = 1 or 2, we did not check our program against negative x values. But, based on what we have learned so far we can predict that in order to suit our program to negative integers for x or y, we would first have to change the signs of those x values using the “neg.l” command. This would ultimately complicate our simple program. Thus, the lack of these specifications for the program in our lab outline document, the necessary complications and limited time constraints prevented us to modify our program to account for data values. But aside from this, our program was a success.

**Questions**

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

Although all the three different modes covered in this lab specifies memory locations, each of them do so in their own unique way. For instance, the Register Indirect With Offset specifies a memory location with an offset added to it. Controlling this offset will let us point to any memory location we want. Next, Indexed Register Indirect allows for two offsets – one with a number and the other with an address or a data register, Xn, times a scaling factor, SF, with values 1, 2, or 4. Here, instead of incrementing manually by changing the offset d8 at the front (like we did for Register Indirect with Offset), we can increment by manipulating the register Xn. Finally, with Postincrement Register Indirect as soon as we move some instructions into the memory, the address register points to the next available block of memory. Here, we won’t even have to manually set up the offset factor to increment. Other unique characteristics of Register Indirect with Offset and Indexed Register Indirect is that both of the points to the same memory location with An at any given time. These works in our advantage when we want to preserve the initial memory address in An In contrast to Postincrement Register Indirect, where An points at a different memory address after each successive use. Moreover, while we cannot use Register Indirect with Offset for looping over (since we have to specify d16 with a changing number each time), we can easily to do with the last two addressing modes discussed above. This might be handy when we want to work with only a few data points. To the contrary, if we have to manipulate a large array of numbers the two other addressing modes can be used in looping situations.

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

If x data points were not restricted to the situation where Δx could be anything including 1 and 2, then we would not have to consider the two area scenarios. Then we could just use the “muls.l” instruction to multiply the Δx values with the summation of the y values as well as the number 0.5 stated in the Trapezoidal rule formula.

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

From the data points, the function y = f(x) is x2 between 0 and 50. (For data storage 4)

The result we obtained in the lab was 41675.

From the data points, the function y = f(x) is x2. (For data storage 5)



The result we obtained in the lab was 170710.

*Percent Error* = = 0.03%

**Conclusion**

In this design project, we implemented two programs. For the first program, we added the contents of two arrays stored at two particular memory locations and then store the result using three different addressing modes – Register Indirect with Offset, Indexed Register Indirect and Postincrement Register Indirect – at three specified memory locations. We implemented another program which would calculate the area underneath a curve y = f(x) using the Trapezoidal rule and the given x and y data points of the curve. Both these programs were tested against the given template files. For the sake of completeness, we also ran these programs against our specified data values. Although the first program successfully passed the test, we had to somewhat tweaked our second program to give the right result. To summarize, the design met the required specifications. Nevertheless, this was a valuable learning experience for us with respect to writing assembly language programs using various addressing modes as well as solving an actual mathematical problem.

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: Ishtiak Ahmed (1269389) and Mohammad \*\*/

/\* Sirajee (1255986) \*\*/

/\* Date: February 26, 2013 \*\*/

/\* General Description: Addition of the contents of two arrays \*\*/

/\* from different memory locations and placement of the summed \*\*/

/\* value at another specified location using three different \*\*/

/\* addressing modes \*\*/

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

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

/\*clr.l %d2 /\*clear %d2\*/

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

/\* Initializations \*/

movea.l 0x2300004, %a1 /\* Point to the first array with a1 \*/

movea.l 0x2300008, %a2 /\* Point to the second array with a2 \*/

movea.l 0x230000C, %a3 /\* Point to the summation array with a3 \*/

/\* Addition of the first value in the two arrays \*/

move.l (%a1)+,%d2 /\* Move a longword from memory location (a1) to d2 \*/

move.l (%a2)+,%d3 /\* Move a longword from memory location (a2) to d3 \*/

add.l %d2,%d3 /\* Add the numbers contained in d2 and d3 \*/

move.l %d3,0(%a3) /\* Move the resultant number from d3 to memory location (a3) \*/

/\* Addition of the second value in the two arrays \*/

move.l (%a1)+,%d2

move.l (%a2)+,%d3

add.l %d2,%d3

move.l %d3,4(%a3)

/\* Addition of the second value in the two arrays \*/

move.l (%a1),%d2

move.l (%a2),%d3

add.l %d2,%d3

move.l %d3,8(%a3)

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

/\* Re-initializations \*/

move.l 0x2300000, %d1 /\* Move the size of the array to d1; Use d1 as a counter \*/

movea.l 0x2300004, %a1 /\* Point to the first array with a1 \*/

movea.l 0x2300008, %a2 /\* Point to the second array with a2 \*/

movea.l 0x2300010, %a4 /\* Point to the summation array with a4 \*/

move.l #0,%d4 /\* Initialize the offset-causing data register to 0 \*/

loop: move.l (%a1)+,%d2 /\* Move a longword from memory location (a1) to d2 \*/

move.l (%a2)+,%d3 /\* Move a longword from memory location (a2) to d3 \*/

add.l %d2,%d3 /\* Add the numbers contained in d2 and d3 \*/

move.l %d3,(%a4,%d4)/\* Move the resultant number from d3 to memory location (a4) \*/

add.l #4, %d4 /\* Iterate the offset to point to the next longword \*/

sub.l #1, %d1 /\* Decrement the counter \*/

bne loop /\* If counter does not equal to 0, then do

"loop" \*/

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

/\* Re-initializations \*/

move.l 0x2300000, %d1 /\* Move the size of the array to d1; Use d1 as a counter \*/

movea.l 0x2300004, %a1 /\* Point to the first array with a1 \*/

movea.l 0x2300008, %a2 /\* Point to the second array with a2 \*/

movea.l 0x2300014, %a5 /\* Point to the summation array with a5\*/

loop2: move.l (%a1)+,%d2 /\* Move a longword from memory location (a1) to d2 \*/

move.l (%a2)+,%d3 /\* Move a longword from memory location (a2) to d3 \*/

add.l %d2,%d3 /\* Add the numbers contained in d2 and d3 \*/

move.l %d3,(%a5)+ /\* Move the resultant number from d3 to memory location (a5) \*/

sub.l #1, %d1 /\* Decrement the counter \*/

bne loop2 /\* If counter does not equal to 0, then do

"loop2" \*/

exit:

/\*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 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: Lab2b.s \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Names of Students: Ishtiak Ahmed (1269389) and Mohammad Sirajee (1255986) \*\*/

/\* Date: February 26, 2013 \*\*/

/\* General Description: A program to calculate the area under a \*\*/

/\* curve (y = f(x)) using the trapezoidal rule for the specified \*\*/

/\* (x,y) co-ordinates in two different arrays \*\*/

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

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

/\*clr.l %d2 /\*clear %d2\*/

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

move.l 0x2300000,%d1 /\*Move size of the array to d1\*/

sub.l #1, %d1

movea.l 0x2300004,%a1 /\* Point to the memory address of the array x with a1\*/

movea.l 0x2300008,%a2 /\* Point to the memory address of the array y with a2\*/

movea.l 0x2300010,%a4 /\* Point to the final storage location with a4\*/

move.l #0, %d6 /\*Initialize d6 to zero\*/

move.l #0, %d7 /\*Initialize d7 to zero\*/

/\* Calculate delta\_x = x(b)-x(a), where b>a \*/

iteration:move.l (%a1)+,%d2

move.l (%a1),%d3

sub.l %d2,%d3

cmp.l #2,%d3

beq case\_of\_2

/\* Calculate f(x(b))-f(x(a)) when delta\_x = 1 \*/

/\*case\_of\_1:\*/move.l (%a2)+,%d4

move.l (%a2),%d5

add.l %d4,%d5

add.l %d5,%d7

sub.l #1,%d1

bgt iteration

bra account\_for\_fraction

/\* Calculate f(x(b))-f(x(a)) when delta\_x = 2 \*/

case\_of\_2:move.l (%a2)+,%d4

move.l (%a2),%d5

add.l %d4,%d5

add.l %d5,%d6

sub.l #1,%d1

bgt iteration

/\* Summation of the overall areas from the two cases \*/

account\_for\_fraction:asr.l #1,%d7

add.l %d7,%d6

/\* Move the final area value to the memory location (%a4) \*/

move.l %d6,(%a4)

exit:

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

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

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

lea 40(%a7),%a7

rts

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