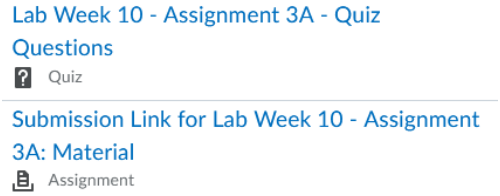
Assignment 3A (50 marks) – Lab Week 10 – More Assembly Language Analysis and Coding
Approximate time to complete: 3 – 4 hours.
Due Dates:
Part A – By 11:30 PM of Friday of Week 11 – Assignment 3A – Lab Week 10 – Quiz Questions
Part B – By 11:30 PM of Friday of Week 11 – Submission Link for Assignment 3A – Lab Week 10 **GradesII.asm** and **FX.asm** – Code Submissions (**use 2 separate files – DO NOT place them into a compressed file**)
There is no requirement to submit Grades.txt



Purpose of the LAB:

The purpose of this lab is to confirm your problem solving skills by implementing solutions to two problems using 68HCS12 Assembly Language, based upon Week 9 and 10's Lectures and various Hybrid Lectures.

Part A (20 marks) – Lab Week 10 - Assignment 3A Quiz Questions

A3A Quiz – Questions 1 – 10 (1 mark each) test your understanding of tracing through an Assembly Language program that uses various Addressing Modes by evaluating the contents of Registers and Memory Address locations, including various Addressing Modes used within the code listing.

A3A Quiz – Questions 11 – 15 (2 marks each) test your understanding of simple Assembly Language branch instructions as we have discussed in various lectures. A reminder to use the "$" sign in your answers in this quiz.

Structured Programming in Assembly Language

This lab exercise challenges your problem solving abilities to determine a well-structed solution to two given problems. You are advised to think through these problems and solve them on paper prior to coding them as "jumping into" Assembly Language coding without an already workable solution can be very frustrating and likely cause you to spend an inordinate amount of time attempting a solution, which may not solve the given problem(s).

GradesII.asm (15 marks)

Week 9's Assignment 2B saw us convert Marks into Grades using an Iterative Selection Switch/Case Structure in 68HCS12 Assembly Language. This lab exercise will see us create **GradesII.asm** in 68HCS12 Assembly Language that iterates through a provided Grades.txt file (applicable to your lab section) to Tally (count) the number of As, Bs, Cs, Ds, and Fs associated with those grades.

Implementation Details

The following information expands upon the Marking Rubric provided for this assignment.

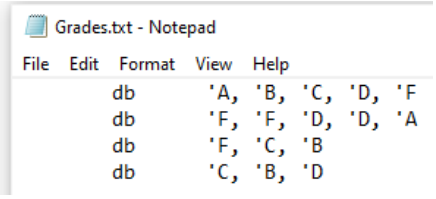
- Starter Code **GradesII.asm** has been provided for your use in AsmIDE
- Use $2000 as the program code origin and initialize the Stack Pointer to $2000 as your first instruction.
- Use the **Grades.txt** file supplied for your lab section in your solution. It **must** be loaded into the Grades array in the memory locations illustrated in the example program run screen shot.
- The Tally of the Grades **must** appear in the memory locations illustrated in the example program run screen shot. To save a lot of complex coding, do **not** use a Tally array in your solution. Rather, for each grade use a unique Label and **define storage** for the grade

- All Labels must follow course standards
  - e.g. Num_As, Num_Bs, Grades, End_Grades, and so on
- The Expected Results of the program run must be included within the code, ideally immediately following the Grades Array as per the supplied example
- Appropriate documentation must be used both in the Program Header and within the code so that the program functionally is clear. Describe WHAT the program is doing, NOT the instruction set.

Very Important

- Iteration must be used in your solution by either using a Loop Counter or the beginning and ending Memory Addresses of the Grades. Note, that using a Loop Counter may result in more complex code.
- Your code must produce identical results if it is run more than once – e.g. you run the code in the Simulator, observe the results and then use File→Reset and run the code again by pressing "Go", to observe identical results.

GradesII.asm Expected Results, Hints and Sample Program Run



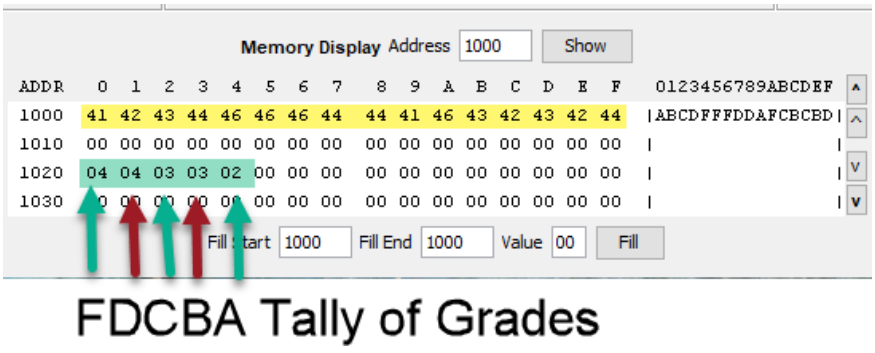
For the Expected Results, document your code as per the following example:

```
Grades
#include Grades.txt
EndGrades

; Expected Result
;                  F     D     C     B     A
;              $1020 $1021 $1022 $1023  $1024
;                  4     4     3     3     2
; as shown in the sumulator
```

The following **example** screen shot illustrates the Memory Map that you must use in your program. Note the memory locations used for the Grades array (highlighted in Yellow) and the memory locations used for the Tally of each grade occurrence (highlighted in Green).

(Annotations and the highlighting of the example results added for emphasis).



FDCBA Tally of Grades

FX.asm (15 marks)

In this part of the assignment, we will create **FX.asm**, which iterates through a given mathematical formula, calculates and stores both the evaluated number and the results of the calculation in Memory.

Implementation Details

Week 9's lecture included a review of Constant Offset Indexed Addressing, which was also discussed in an earlier Hybrid Lecture. This Addressing Mode must be used in your solution to store the results of each calculation.

The following information expands upon the Marking Rubric provided for this assignment.

- Starter Code **FX.asm** has been provided for your use in AsmIDE
- The supplied 21F Flowchart for Grades II document contains a Nested-IF structure that you are to use for the programming logic of your solution.
- Use $2000 as the program code origin and initialize the Stack Pointer to $2000 as your first instruction.
- Solve the equation f(x) = 5x + 3 for x = 0 to 9, using Iteration, 8-bit Multiplication
- During each iteration of your loop to solve the equation, you are to do the following, using
    a. place the x value from an Accumulator into an X_Values array using the Constant Offset Addressing mode with Index Register X or Y
    b. calculate f(x)
    c. place the calculated f(x) value from an Accumulator into the Results array using the Constant Offset Addressing mode using the same Index Register in step a.
- Array addresses are provided in the provided program run
- All Constants and Labels must follow course standards
    - e.g. FIVE, ARRAY_LEN, Loop, Results, End_Results, and so on
    - Note that ARRAY_LEN must be defined as ARRAY_LEN equ 10 for this assignment and that the Results array must be created using the correct Assembler directive for **defined storage**, followed by ARRAY_LEN
- The Expected Results of the program run must be included within the code, ideally immediately following the X_Values array for the x values and the Results array for the calculated values of f(x)
- Appropriate documentation must be used both in the Program Header and within the code so that the program functionally is clear. Describe WHAT the program is doing, NOT the instruction set.
- I recommend that you use the Memory Address approach for iteration versus using loop counter

Very Important

- Do **not** initially create an array and fill it with 0 – 9 values
- Iteration must be used in your solution by using the beginning and ending Memory Address of any of the program's array. Do **not** use a Loop Counter in your solution.
- Your code must use the 8-bit **mul** instruction in its solution.
- Your code must use **Constant Offset Indexed Addressing Mode** to store the results of both the x and the f(x) values. (This will save you a LOT of unnecessary coding if you think through the problem before coding its solution).

FX.asm Hints and Sample Program Run

f(x) = 5x + 3 where x = 0

x	5x	5x + 3
0	0	3
1	5	8
2	10	13
3	15	18
4	20	23
5	25	28
6	30	33
7	35	38
8	40	43
9	45	48

Solving mathematical equations may seem to be an overwhelming task to do in a programming language; however, I find that if I layout the problem using a tool such as Excel, then it makes the problem easier to understand and implement. Here is the problem solved in Excel in base 10.

Here we should be able to extract the concept of a loop counter from the values in the x column, the multiplication of the x value by 5 in the 5x column and the Expected Results after adding 3 in the 5x + 3 column.

All that remains is for the solution to be coded. Then we confirm the program's output against the Excel solution, which forms our Test Plan.

Note: Just before using the **mul** instruction, you may have to temporarily store whatever register you are using to store the value of x in f(x) and then retrieve it back after the **mul** instruction.

Expected Program Output

Note that your solution must exactly match this program snap-shot of the output. (Annotation of x and f(x) and the highlighting of the example value added for emphasis).

Memory Display Addre

	ADDR	0	1	2	3	4	5	6	7	8	9
	1000	00	00	00	00	00	00	00	00	00	00
	1010	00	00	00	00	00	00	00	00	00	00
x	1020	00	01	02	03	04	05	06	07	08	09
f(x)	1030	03	08	0d	12	17	1c	21	26	2b	30

Assessment

Note: If your Name and Student Number are missing in the Program Header on any Code Submission, you will receive a mark of zero for the submission, as you are not indicating that it is your original, independently created work.

You will be assessed in accordance with the following marking rubrics, which are also available on the Assignment link.

GradesII.asm (15 marks)

GradesII.asm	Meets Expectations (1) 1 point	Below Expections (0.5) 0.5 points	Missing / Poor (0) 0 points	Criterion Score
Grades II - Constants and labels	Submitted code illustrates that all constants and labels correctly defined and used	Submitted code illustrates that some constants and labels correctly defined and used	Submitted code illustrates that constants not defined or used and/or labels not correctly defined	/ 1

Grades II - Criteria	Meets Expectations (1) 2 points	Below Expections (0.5) 1 point	Missing / Poor (0) 0 points	Criterion Score
Grades II - Submitted Code matches Assignment Requirements	Submitted code illustrates that: • Documentation explains code's purpose • Most lines of code explain what the program is doing, NOT the instruction set	Submitted code illustrates that one of the following is incorrect/missing: • Documentation explains code's purpose • Most lines of code explain what the program is doing, NOT the instruction set	Submitted code illustrates that two of the following are incorrect/missing: • Documentation explains code's purpose • Most lines of code explain what the program is doing, NOT the instruction set	/ 2

Criteria	Meets All Expectations (4) 4 points	Meets Some Expectations (3) 3 points	Below Expections (1) 1 point	Missing / Poor (0) 0 points	Criterion Score
Grades II - Iteration	Submitted code illustrates that Iteration correctly uses: • a dynamically created loop-counter or • the beginning and ending Memory Addresses of the Grades array • Indexed Addressing (IMM) Mode	Submitted code illustrates that one of the following is incorrect/missing: • a dynamically created loop-counter or • the beginning and ending Memory Addresses of the Grades array • Indexed Addressing (IMM) Mode	Submitted code illustrates that one or two of the following is incorrect/missing: • a dynamically created loop-counter or • the beginning and ending Memory Addresses of the Grades array • Indexed Addressing (IMM) Mode	Submitted code illustrates that all of the following is incorrect/missing: • a dynamically created loop-counter or • the beginning and ending Memory Addresses of the Grades array • Indexed Addressing (IMM) Mode	/ 4
Grades II - Tally of Grades	Selection Structure correctly implemented			Selection Structure not used or incorrectly implemented	/ 4
Grades II - Off-Line Testing of Code	Submitted code: • Assembles without intervention from Professor • Grades array at correct Memory Address • Tally of Grades at correct address • Tally of Grades contains correct expected results for 2 or more program runs • Has no hard coded Memory Addresses other than those indicated in the Assignment	Submitted code is missing one of the following: • Assembles without intervention from Professor • Grades array at correct Memory Address • Tally of Grades at correct address • Tally of Grades contains correct expected results for 2 or more program runs • Has no hard coded Memory Addresses other than those indicated in the Assignment	Submitted code is missing two or more one of the following: • Assembles without intervention from Professor • Grades array at correct Memory Address • Tally of Grades at correct address • Tally of Grades contains correct expected results for 2 or more program runs • Has no hard coded Memory Addresses other than those indicated in the Assignment	Submitted code does not assemble and/or code is deemed non-functional because the following are missing/incorrect: • Grades array at correct • Memory Address Tally of Grades at correct address • Tally of Grades contains correct expected results for 2 or more program runs • Has no hard coded Memory Addresses other than those indicated in the Assignment	/ 4

FX.asm (15 marks)

FX.asm	Meets Some Expectations (3) 1 point	Below Expections (1) 0.5 points	Missing / Poor (0) 0 points	Criterion Score
FX - Constants and labels	Submitted code illustrates that all constants and labels correctly defined and used	Submitted code illustrates that some constants and labels correctly defined and used	Submitted code illustrates that constants not defined or used and/or labels not correctly defined	/ 1
FX - Submitted Code matches Assignment Requirements	Submitted code illustrates that: • Documentation explains code's purpose • Most lines of code explain what the program is doing, NOT the instruction set	Submitted code illustrates that one of the following is incorrect/missing: • Documentation explains code's purpose • Most lines of code explain what the program is doing, NOT the instruction set	Submitted code illustrates that two of the following are incorrect/missing: • Documentation explains code's purpose • Most lines of code explain what the program is doing, NOT the instruction set	/ 1

Criteria	Meets All Expectations (3) 3 points	Missing / Poor (0) 0 points	Criterion Score
FX - Iteration	Submitted code controls Iteration by using the beginning and ending Memory Addresses of any of the program's arrays	Submitted code does not control Iteration by using the beginning and ending Memory Addresses of any of the program's arrays	/ 3
FX - Calculation and Storage of x values in the f(x) equation into the X_Values array	Submitted code correctly calculates the x values and implements Constant Offset Indexed Addressing Mode to store values	Submitted code: • read x values from an array of x values or • incorrectly calculates the x values or • does not correctly implement Constant Offset Indexed Addressing Mode to store values	/ 3
FX - Calculation and Storage of f(x) values in the f(x) equation into the Results array	Submitted code correctly calculates f(x) using the **mul** instruction and implements Constant Offset Indexed Addressing Mode to store values	Submitted code : • does not use the **mul** instruction or incorrectly calculates f(x) or • does not correctly implement Constant Offset Indexed Addressing Mode to store values	/ 3

Criteria	Meets All Expectations (3) 4 points	Meets Some Expectations (3) 3 points	Below Expections (1) 1 point	Missing / Poor (0) 0 points	Criterion Score
FX - Off-Line Testing of Code	Submitted code: • Assembles without intervention from Professor • X_Values array at correct Memory Address • Results array at correct Memory Address • Both arrays contain correct expected results	Submitted code is missing one of the following: • Assembles without intervention from Professor • X_Values array at correct Memory Address • Results array at correct Memory Address • Both arrays contain correct expected results	Submitted code is missing two or more one of the following: • Assembles without intervention from Professor • X_Values array at correct Memory Address • Results array at correct Memory Address • Both arrays contain correct expected results	Submitted code does not assemble and/or code is deemed non-functional because the following are missing: • X_Values array at correct Memory Address • Results array at correct Memory Address • Both arrays contain correct expected results	/ 4

Total	/ 30

Should you have any questions regarding this assignment, please ask your Lab Professor or myself during a Lab period or during the Lecture Professor's Consultation Hours or at the end of any Lecture period.