

## Assignment 4 (100 marks) – Lab Week Eleven – Final Assignment

See the Hand-In Sheet for details about program demonstration and documentation submissions.

**APPROVED**

By David Haley at 4:30 pm, Mar 31, 2018

Early Demos and  
Submissions are  
Most Welcome!

**Late demos or submissions will not be accepted and will receive a mark of zero (0).**

Note that if you missed your demo deadline, you can still get partial marks for the assignment by meeting the Documentation Submission deadline.

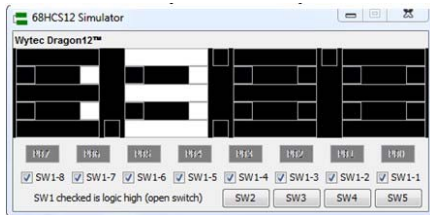
**Note: If your name is not on the submitted code listings or the code is not adequately documented, no credit will be given for the code.**

This lab exercise may be optionally performed by up to **THREE** students in the same lab section working as a group (students pick their own partners). Also, ensure all student names/numbers are on all program listings and documentation in order for all students to receive credit for the work (No name, no credit). **There is only ONE submission required per group of students.**

### Task One – The BCD Counter – Demo Solution on the Wytec HCS12 Dragon12-Plus board (40 marks)

For this lab assignment, you are required to code a BCD counter using Assembly Language that continually counts from 00 → 99 → 00 → 99, etc. and correctly displays the count on the HEX Displays as illustrated below. Note that you must use **only one** of the Accumulators to hold your count; the ideal one to use based upon lecture discussions regarding the instruction **daa**. Then, on an as-required basis, you will push and pull its value to and from the stack to save the count at the appropriate time(s) in your program.

To assist in your learning, I have included, a flowchart for the “main” Assembly Language program, some “skeleton code”, which you may use to build upon for this assignment, and a couple of videos that illustrate the problem solution.



Instead of using the LEDs to display our count from 00→99→00→99 etc, we display the results on the 7-Segment HEX Displays of the Simulator to confirm that the display portion of our solution is correct. (LEDs' colour adjusted for printing purposes only).

Finally, once we have debugged our software and have confirmed that is FULLY functional, we will download it to the Dragon12-Plus HCS12 Trainer Board and run it there for demonstrations purposes.

**You may wish to change the delay from 250 ms to 125 ms for this demonstration**

### Development Requirements and Constraints

To be considered for full credit for this portion of the assignment, you are to write a “main” Assembly Language program called **Counter\_00\_99\_BCD\_HEX\_Display.asm** that solves the problem specification detailed in this document and illustrated in **Counter\_00\_99\_BCD\_HEX\_Display Flowchart**, which is located on Blackboard. (20 marks)

Here are the constraints for **Counter\_00\_99\_BCD\_HEX\_Display.asm**

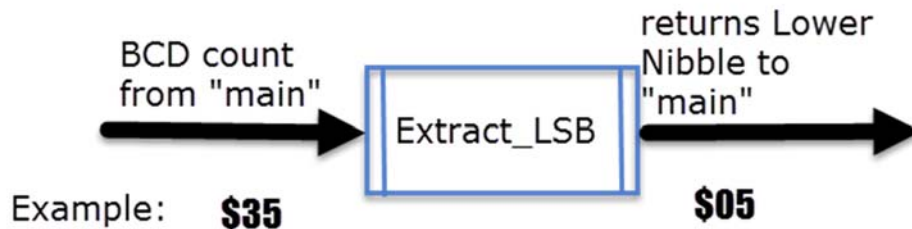
- ✓ Use the supplied “skeleton code” for the assignment as a starting point for your coding solution. Note the “(DO NOT CHANGE ANY OF THE FOLLOWING LINES OF CODE)” statement in the skeleton
- ✓ Make use of CONSTANTS ***DIGIT3\_PP0*** and ***DIGIT2\_PP1*** as well as ***DELAY\_VALUE***.
- ✓ **You MUST use ONLY one Accumulator as the BCD count within this BCD Counter; otherwise little credit will be given for your solution.**
- ✓ **You MUST use the stack to temporarily store and retrieve the value of the BCD count. YOU MAY NOT USE any other methods to temporarily store and retrieve values; otherwise, little credit will be given for your solution.**
- ✓ Use AsmIDE to develop the program code and the Simulator to debug your solution.
- ✓ Use the Wytec Dragon12 Plus board to demonstrate your solution.
- ✓ Re-use library subroutine ***Delay\_ms***, which should be found in the C:\68HCS12\Lib folder, in your solution. This file was supplied to you in a previous lab. Do not change any code in that file.
- ✓ Appropriately use the supplied library subroutine ***Config\_HEX\_Displays***, placing ***Config\_Hex\_Displays.asm*** in C:\68HCS12\Lib. Do not change any code in that file.
- ✓ Fully document **Counter\_00\_99\_BCD\_HEX\_Display.asm** as per other previously supplied documented programs.

In support of **Counter\_00\_99\_BCD\_HEX\_Display.asm**, write the following Assembly Language subroutines:

- **Extract\_MSB** – Subroutine to extract MSB (the upper nibble of the Accumulator) from an 8-bit Accumulator as per the following illustration: (5 marks)



- **Extract\_LSB** – Subroutine to extract LSB (the lower nibble of the Accumulator) from an 8-bit Accumulator as per the following illustration: (5 marks).



**Notes:**

Each subroutine is to be in a separate file (use the subroutine name + .asm as the filename – e.g. **Extract\_MSB.asm**) and fully document your subroutines as per the documentation standards contained in the supplied **Config\_Hex\_Displays.asm** file.

In support of **Counter\_00\_99\_BCD\_HEX\_Display.asm**, use the supplied **HEX\_Display** subroutine to display the MSB and LSB of the BCD count on the HEX displays, using **Register-Offset Addressing** and the procedure outlined in the supplied flowchart. **Do NOT change any of the code in ANY supplied subroutine.** You should note that the Wytec hardware board's HEX Displays are "dumb" devices. As such, **HEX\_Display** does not contain any iterative and/or decision-making statements (e.g. no loops or if-then-else type of statements) within the subroutine.

**IMPORTANT PROGRAMMING CONCEPT**

**Subroutines NEVER call other subroutines.** That is, a subroutine has one and only one function – e.g. the **HEX\_Display** subroutine **cannot** call the provided **Delay\_ms** subroutine because this subroutine knows nothing about other subroutines. Rather, a subroutine only accepts parameters (as applicable) from the calling "main" routine, which provides it arguments (as applicable) and returns values (as applicable) just like in any other higher-level programming language. "main" is the only portion of a program that may call subroutines.

See the Hand-In sheet for particulars of demos and submissions.

**TASK Two – Pass-Fail Calculator – Demo Solution on the Wytec HCS12 Dragon12-Plus board (60 marks)**

**Congratulations!** You have just received the contract to provide an assembly language program (**Pass\_Fail\_Calc.asm**) that evaluates a class of six students' CST8216 marks and displays the results on the Wytec Dragon12+ Demo board.



In part, the course outline for CST8216 stipulates the following:

*To pass this course, the student must have a grade of at least at least 50% or 'D- on*

*Lab Exercises/Assignments/Hybrid activities (25/50) **and** at least 50% or 'D-' on Term Tests/Final Exam (25/50).*

You are to write a fully documented solution that determines if each of the six students have passed CST8216 or not, noting that 50% represents an integer values  $\geq 5$  (e.g.  $36 \setminus 5$  on labs = 7 or 70%, which is a pass, while  $22 \setminus 5 = 4$ , or 40%, which is a failure).

**Part A – The Test Plan (6 marks)**

**The first thing you must do before writing any code**, is to use the marks file appropriate to your lab section (located on Blackboard) and complete the test plan on the hand-in sheet so that you will know WHAT results your software solution should realize. Ensure you record which test plan you used (e.g. Wed 10—12) so that your expected results can be compared against the correct marks file during your program demo.

Once the test plan has been created, take one student and "walk" their marks through the supplied program logic so that you know what all intermediate results as well as the result should be when you code the solution.

Once you have one student's results calculated, go through all the other students' values so that you know what they should be as well. This will greatly assist you in the program debugging process! I recommend that you use **demo.txt** and the **Demo\_of\_Pass\_Fail\_Calculator\_1\_sec\_delay.mp4** video as a basis for your software development. Both items are on Blackboard in the same folder as this document.

**Part B – The Program Logic**

Using the following **mandatory** Program Logic, you are to write a fully documented "main" HCS12 Assembly Language program (**Pass\_Fail\_Calc.asm**) that implements the following program flow (**in this EXACT order**) for solving this problem. You may wish to optionally create a high-level flowchart if that would assist you in better understanding the problem solution

- a. Configuring program constants;
- b. Reading the file of marks, which contains marks for six students;
- c. Calling subroutine **Calculate\_Average** that calculates the integer average 0 – 10 for one student's Theory marks;
- d. Calling subroutine **Pass\_Fail** to determine from c. above if that student has passed or failed the Theory – "main" stores this result;
- e. Calling subroutine **Calculate\_Average** that calculates the integer average 0 – 10 for one student's Practical marks;
- f. Calling subroutine **Pass\_Fail** to determine from e. above if that student has passed or failed the Practical – "main" stores this result
- g. Looping back to c. to do the remaining students' marks calculations – **ALL calculations must be performed and the results of Pass\_Fail must be stored for each student before proceeding to h.**
- h. Calling subroutine **Config\_HEX\_Displays**, a subroutine to configure the Wytec Dragon12+ Demo board hardware to use the 7-segment Hex Displays;
- i. **Now, for each student, "main" will retrieve the previously stored Theory and Practical results and determine if a "P" (for pass) or "F" (for fail) for the overall course should be displayed**
- j. Calling subroutine **PF\_HEX\_Display** that causes the Wytec Dragon12+ Demo board hardware to display **P** or **F** for an overall Pass or Fail of the Course on the right-most 7-segment Hex Display, noting that there is only **one** output per student. Additionally, display the **P** or **F** values for **1 sec**, then **use the blank code** to blank the display for **1 sec**, which means that there is **nothing** displayed;
- k. Looping back to i. to display the remaining results; and
- l. Continually looping back to this statement (l.) to maintain the **blanked** Display.

**Part C – Coding the Implementation (Total of 44 marks)**

What you are to do for this part:

- I. Write a fully documented "main" HCS12 Assembly Language program (**Pass\_Fail\_Calc.asm**) that implements the given **mandatory** Program Logic to this problem. **(20 marks)**
  - Use the supplied skeleton code **Pass\_Fail\_Calc.asm** as your starting point, noting the instructions in that code
  - Make maximum use of **CONSTANTS** in your program.
  - Ensure that you use iteration in your solution. The **mandatory** Program Logic indicates **3 separate loops** within "main." Note that the last loop is just a line of code that branches to itself.
  - The library file **Config\_HEX\_Displays.asm** has been provided to you – see the instructions in that file for its use

- Your solution must be **structured** – e.g. **Pass\_Fail\_Calc.asm** MUST use the identified subroutines rather than having ALL of the code in **Pass\_Fail\_Calc.asm**. For example, it is **not** the responsibility of “main” to average values or calculate the Theory/Lab portion Pass or Fail. Rather, “main” calls subroutines to do its work by passing the subroutines values, receiving a result back from the subroutine. “main” can then later evaluate the results to determine if a “P” or “F” for the entire course should be displayed.

II.

- **Important Note:** Only “main” can store values in “main” – e.g. subroutines cannot globally store data back in “main”; rather, data from subroutines must be returned in registers. The result received by “main” could then be directly passed to another subroutine, used as part of a calculation, stored in memory or used for some other action.
- Ensure that all non-library files are written as separate files in the same folder as **Pass\_Fail\_Calc.asm**

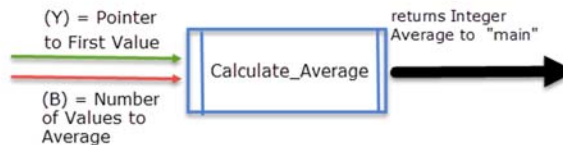
**Helpful Hints:**

- When you are ready to code, and I recommend that you code the subroutines one-by-one and test them with some of the student data (e.g. from demo.txt). Once you know that the subroutine is functional, incorporate it into the main program;
- Continue with this approach until you complete all of the subroutines that calculate the results; and
- Now, work on the program output.

III. Write a fully documented subroutine **Calculate\_Average** (using the skeleton code provide to you), that accepts two parameters as per the following illustration: **(14 marks)**

- a pointer (use *Index Register Y*) to the first value to use; and
- the number of values (in Accumulator B) to average

then adds up the values by using iteration, then divides the total by 5 and returns the integer average.



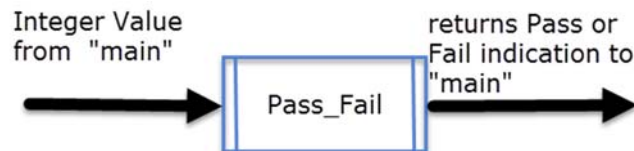
Note: The term “value” refers to the first mark associated with either the Theory or Practical array of marks

Make maximum use of CONSTANTS within the subroutine, especially the DIVISOR constant in the supplied source code.

Do **not** use an if-then-else structure in this subroutine as it is meant to serve **any** number of values, not just the three or five values in this application. Additionally, subroutines do not contain **org** statements, but may contain multiple **rts** statements.

Note that your solution for **Calculate\_Average** must work regardless of the number of values passed to the subroutine to average. As such, Calculate\_Average must be used for both the Theory and the Practical marks.

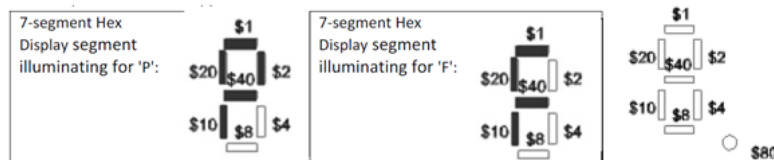
- Write a fully documented subroutine **Pass\_Fail** (using the skeleton provided to you that determines if the supplied integer average is a passing or failing average and returns the results **(7 marks)**
- Make maximum use of CONSTANTS within the subroutine.



- Complete the supplied skeleton code for subroutine **PF\_HEX\_Display** that converts the supplied value to a ‘P’ or ‘F’ segment value or **blanks** the display, which means that there is **nothing** displayed. (There is only one line of code that you need to complete). **(3 marks)**

**Helpful Information**

Here are the 7-segment illustrations for 'P' and 'F' and the "blank."

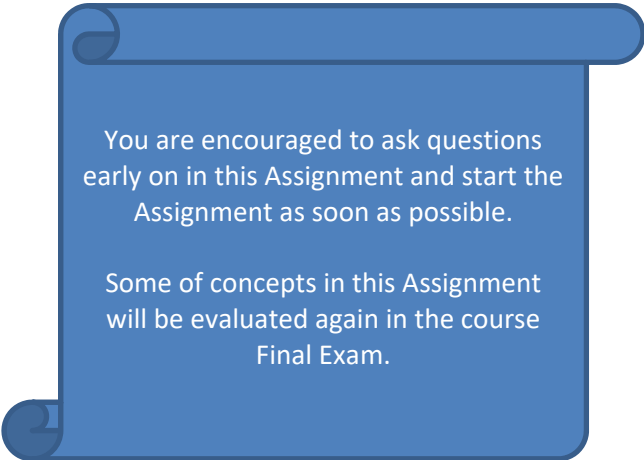


**IMPORTANT PROGRAMMING CONCEPT**

**Subroutines NEVER call other subroutines.** That is, a subroutine has one and only one function – e.g. the Calculate\_Average subroutine **cannot** call the Pass\_Fail subroutine because they know nothing about each other. Rather, a subroutine only accepts parameters (as applicable) from the calling “main” routine, which provides it arguments (as applicable) and returns values (as applicable) just like in any other higher-level programming language. “main” is the only portion of a program that may call subroutines.

Final Note: Do not change any of the supplied Library files that I have provided to you.

**See the Hand-In sheet for particulars of demos and submissions.**



You are encouraged to ask questions early on in this Assignment and start the Assignment as soon as possible.

Some of concepts in this Assignment will be evaluated again in the course Final Exam.

**Assignment 4 (100 marks) – Lab Week Eleven – Hand-In Sheet for BCD Counter – Page One of Two****Notes:**

1. If your name is not on the submitted code and/or the code is not adequately documented, no credit will be given for the code.
2. If your name is not on the demo'd code prior to the demo and/or you are not present for the demo, no credit will be given to you for the demo.

**Due Dates**

**Demos** (ensure student names are on the code – you may complete the code documentation by the Code submission due date if you have note already done so).

- During your lab period the Week of 15 – 21 April 2018
- Additionally, in WB170 on Wed, Apr 18 2017 (without penalty) – No lecture; lab period.

Please  
staple the  
pages  
together.



Early Demos and  
Submissions are  
most welcome!

**Code Submission for ALL students:** *On Blackboard by 1159 pm on Fri, Apr 20, 2108*

*This sheet contains details about program demonstration and documentation submissions. Ensure you have this sheet with you when you demonstrate your solutions and that the sheet is submitted to your portfolio folder once all demonstrations have been completed or by the end of Apr 18's lecture period (which will be a lab) even if you haven't demo'd and you wish feedback on your submission. Otherwise, you will just receive an undocumented mark for the assignment.*

**Late demos or submissions will not be accepted and will receive a mark of zero (0).**

Note that you do not have to demonstrate Task One and Task Two at the same time and that if you missed your demo deadline, you can still get partial marks for the assignment by meeting the Documentation Submission deadline.

This lab exercise may be optionally performed by up to **THREE** students in the same lab section working as a group (students pick their own partners). Also, ensure all student names/numbers are on all program listings and documentation in order for all students to receive credit for the work (No name, no credit).

Name: \_\_\_\_\_ Student Number: \_\_\_\_\_

Name: \_\_\_\_\_ Student Number: \_\_\_\_\_

Name: \_\_\_\_\_ Student Number: \_\_\_\_\_

**Assessment – It is recommended that you check your solution against the following marking rubric BEFORE the lab demo.**

**Indicate Your Lab Period**

***Task One – The BCD Counter – Demo Solution on the Wytec HCS12 Dragon12-Plus board (40 marks)***

A. Demo of your solution on the Hardware Board (**Are your names on the code?**) Professors Initials: \_\_\_\_\_ /10

Notes: For full demo marks, your solution must correctly implement the assignment instructions, specifically using the correct Digits on the Hardware Board, the correct count and the display of only **valid** BCD counts.

Credit for the demonstration will only be given to students in your group who are present for the demonstration and whose names are on the code prior to the demo.

You are to set the delay time for your hardware demonstration to 125 ms **AND** I recommend that you test your solution in the student version of the simulator (before demoing on the hardware board) using a delay of 250 ms.



## Assignment 4 (100 marks) – Lab Week Eleven – Hand-In Sheet for BCD Counter – Page Two of Two

Remember that there is only one demo permitted, so test your software in the Simulator first; then, test it on **the** Wytec HCS12 Dragon 12-Plus board to ensure that it works there **before calling me over to demo the software**. If it doesn't work on the hardware board, then load the software back into the simulator and check to ensure that you initialized all of your constants/variables and memory storage addresses at the beginning of the program.


- B. Submit the following files to the [Assignment Four – BCD Counter Link](#) on Blackboard **by 1159 pm on Fri, Apr 20, 2108**


Place following files depicted to the right of this text into a single compressed .zip file using the following Naming Convention for the file name:


BCD\_<Lab\_Day\_Time>\_<Student\_Last\_Names>.zip


e.g. *BCD\_Wed\_10\_12\_Gosling\_Hooper\_Kernighan.zip*

Name

 Counter\_00\_99\_BCD\_HEX\_Display.asm

 Extract\_LSB.asm

 Extract\_MSB.asm

 HEX\_Display.asm

Please  
staple the  
pages  
together.



- C. Post-Lab Code Inspection – your submitted code will be evaluated as follows:

I. Counter\_00\_99\_BCD\_HEX\_Display.asm (See note below on the use of **daa**) / 20

II. Extract\_MSB.asm / 5

III. Extract\_LSB.asm / 5

All code will be assessed as follows in descending priority:

- Functionality – does the code correctly solve the problem per the assignment instructions, including use of valid BCD theory; wise use of iteration; have the registers been correctly used?
- Maintainability – use of valid BCD theory, formatting of code; use of assembler directives, labels and CONSTANTS versus hardcoding values other than 0, 1, -1
- Documentation: program title, header; meaningful comments that do not merely explain the instruction set
- Optimal use of the instruction set - **You must correctly use the instruction *daa*, as per in-class discussions and as detailed in the programmer's manual; otherwise your Task One mark will be reduced by 20/40 marks.**

- D. Post-Lab Program Run – I will assemble your files using the library files from their required locations to ensure that your solution works on the hardware board when independently assembled, downloaded and run.

/5

Note that the above 5 marks are intended to supplement your mark if you did not have time to demonstrate the program run by the deadlines. These marks, however, may be counted regardless of the status of your previous demonstration. So, if your original demo was correct, incorrect or not performed and the program runs correctly Post-Lab, then you will be awarded the five marks, conditional to having student names and program documentation within your code.

**Your Assignment 4 – Task One Mark /40 + a possible 5 bonus marks if you have already correctly demo'd your solution and it correctly functions when evaluated post-lab.**

## Assignment 4 (100 marks) – Lab Week Eleven – Hand-In Sheet for Pass-Fail Calculator – Page One of Three

### Notes:

1. If your name is not on the submitted code and/or the code is not adequately documented, no credit will be given for the code.
2. If your name is not on the demo'd code prior to the demo and/or you are not present for the demo, no credit will be given to you for the demo.

**Demos** (ensure student names are on the code – you may complete the code documentation by the Code submission due date if you have note already done so).

- During your lab period the Week of 15 – 21 April 2018
- Additionally, in WB170 on Wed, Apr 18 2017 (without penalty) – No lecture; lab period.

Please  
staple the  
pages  
together.



Early Demos and  
Submissions are  
most welcome!

**Code Submission for ALL students: On Blackboard by 1159 pm on Fri, Apr 20, 2108**

*This sheet contains details about program demonstration and documentation submissions.*

*Ensure you have this sheet with you when you demonstrate your solutions and that the sheet is submitted to your portfolio folder once all demonstrations have been completed or by the end of Apr 18's lecture period (which will be a lab) even if you haven't demo'd and you wish feedback on your submission. Otherwise, you will just receive an undocumented mark for the assignment.*

### Late demos or submissions will not be accepted and will receive a mark of zero (0).

Note that you do not have to demonstrate Task One and Task Two at the same time and that if you missed your demo deadline, you can still get partial marks for the assignment by meeting the Documentation Submission deadline.

This lab exercise may be optionally performed by up to **THREE** students in the same lab section working as a group (students pick their own partners). Also, ensure all student names/numbers are on all program listings and documentation in order for all students to receive credit for the work (No name, no credit).

Name: \_\_\_\_\_ Student Number: \_\_\_\_\_

Name: \_\_\_\_\_ Student Number: \_\_\_\_\_

Name: \_\_\_\_\_ Student Number: \_\_\_\_\_

**Assessment – It is recommended that you check your solution against the following marking rubric BEFORE the lab demo.**

**Indicate Your Lab Period**

Assessment – It is recommended that you check your solution against the following marking rubric BEFORE the lab demo.

### TASK Two – Pass-Fail Calculator – Demo Solution on the Wytec HCS12 Dragon12-Plus board (60 marks)

A. Demo of your solution on the Hardware Board (**Are you names on the code?**) Professors Initials: \_\_\_\_\_ /10

#### Notes:

1. For full demo marks, your solution must correctly implement the assignment instructions, specifically using the correct Digits on the Hardware Board and displaying only **one** Pass/Fail result each of the six students' marks you have been assigned by lab period. **Your program should display the same results if the program is run again either in the Simulator or the Hardware board, without reloading the software.**
2. Ensure that you have completed the Test Plan before your demo on the hardware board (otherwise you are not ready to demo) and that the display delay time is set to 125 ms.



## Assignment 4 (100 marks) – Lab Week Eleven – Hand-In Sheet for Pass-Fail Calculator – Page Two of Three

Please  
staple the  
pages  
together.



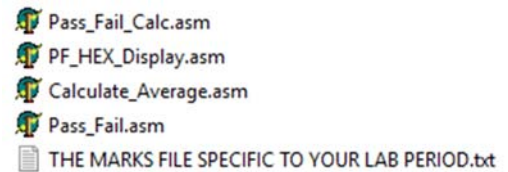
3. Credit for the demonstration will only be given to students in your group who are present for the demonstration and whose names are on the code prior to the demo.
4. Remember that there is only one demo permitted, so test your software in the Simulator first; then, test it on **the** Wytec HCS12 Dragon 12-Plus board to ensure that it works there **before calling me over to demo the software**. If it doesn't work on the hardware board, then load the software back into the simulator and check to ensure that you initialized all of your constants/variables and memory storage addresses at the beginning of the program. **Then call me over to demo the software.**

Submit the following files to the [Assignment Four – Pass-Fail Calculator Link](#) **on Blackboard by 1159 pm on Fri, Apr 20, 2108**

Place following files depicted to the right of this text into a single compressed .zip file using the following Naming Convention for the file name:

PF\_<Lab\_Day\_Time>\_<Student\_Last\_Names>.zip

e.g. PF\_Wed\_10\_12\_Gosling\_Hooper\_Kernighan.zip



- B. Complete the following Test Plan for the set of six students' marks you have been assigned by lab period and present it to me before your demo

/6

**Note:** Without this file, your solution cannot be fully evaluated off-line.

Marks File Name: \_\_\_\_\_

Students	Passed Theory (Y/N)	Passed Practical (Y/N)	Value to Display ( P or F )
Student A			
Student B			
Student C			
Student D			
Student E			
Student F			

**Note:** If you did not submit the Test Plan into your portfolio folder, then include it with your Blackboard submission as part of the single compressed file.

**Assignment 4 (100 marks) – Lab Week Eleven – Hand-In Sheet for Pass-Fail Calculator –  
Page Three of Three**

Please  
staple the  
pages  
together.



C. Post-Lab Code Inspection – your submitted code will be evaluated as follows:

I. Pass\_Fail\_Calc.asm / 20

II. Calculate\_Average.asm / 14

III. Pass\_Fail.asm / 7

IV. PF\_Hex\_Display.asm / 3

All code will be assessed as follows in descending priority:

- a. Functionality – does the code correctly solve the problem per the assignment instructions; wise use of iteration; have the registers been correctly used, does the program give the same results if run more than once without reloading it?
- b. Maintainability – formatting of code; use of assembler directives, labels and CONSTANTS versus hardcoding values other than 0, 1, -1
- c. Documentation: program title, header; meaningful comments that do not merely explain the instruction set
- d. Optimal use of the instruction set

D. Post-Lab Program Run – I will assemble your files using the library files from their required locations to ensure that your solution works on the hardware board when independently assembled, downloaded and run.

/5

Note that the above 5 marks are intended to supplement your mark if you did not have time to demonstrate the program run by the deadlines. These marks, however, may be counted regardless of the status of your previous demonstration. So, if your original demo was correct, incorrect or not performed and the program runs correctly Post-Lab, then you will be awarded the five marks.

**Your Assignment 4 – Task Two Mark /60 + a possible 5 bonus marks if you have already correctly demo'd your solution and it correctly functions when evaluated post-lab.**

## HCS12 Instructions you are likely to use in this Lab Assignment

S12CPUV2 Reference Manual, Rev. 4.0

Table 5-1. Load and Store Instructions

Mnemonic	Function	Operation
<b>Load Instructions</b>		
LDAA	Load A	$(M) \Rightarrow A$
LDAB	Load B	$(M) \Rightarrow B$
LDD	Load D	$(M : M + 1) \Rightarrow (A:B)$
LDS	Load SP	$(M : M + 1) \Rightarrow SP_H : SP_L$
LDX	Load index register X	$(M : M + 1) \Rightarrow X_{16} : X_L$
LDY	Load index register Y	$(M : M + 1) \Rightarrow Y_{16} : Y_L$

Table 5-2. Transfer and Exchange Instructions

Mnemonic	Function	Operation
<b>Transfer Instructions</b>		
TAB	Transfer A to B	$(A) \Rightarrow B$

Table 5-6. Decrement and Increment Instructions

Mnemonic	Function	Operation
<b>Decrement Instructions</b>		
DEC	Decrement memory	$(M) - \$01 \Rightarrow M$
DECA	Decrement A	$(A) - \$01 \Rightarrow A$
DECB	Decrement B	$(B) - \$01 \Rightarrow B$
<b>Increment Instructions</b>		
INC	Increment memory	$(M) + \$01 \Rightarrow M$
INCA	Increment A	$(A) + \$01 \Rightarrow A$
INCB	Increment B	$(B) + \$01 \Rightarrow B$
INX	Increment X	$(X) + \$0001 \Rightarrow X$
INY	Increment Y	$(Y) + \$0001 \Rightarrow Y$

Table 5-9. Clear, Complement, and Negate Instructions

Mnemonic	Function	Operation
CLR	Clear memory	$\$00 \Rightarrow M$
CLRA	Clear A	$\$00 \Rightarrow A$
CLRB	Clear B	$\$00 \Rightarrow B$

Table 5-10. Multiplication and Division Instructions

Mnemonic	Function	Operation
<b>Division Instructions</b>		
IDIV	16 by 16 integer divide (unsigned)	$(D) \div (X) \Rightarrow X$ Remainder $\Rightarrow D$

Table 5-12. Shift and Rotate Instructions

Mnemonic	Function	Operation
<b>Logical Shifts</b>		
LSR	Logic shift right memory	
LSRA	Logic shift right A	
LSRB	Logic shift right B	

Table 5-21. Jump and Subroutine Instructions

Mnemonic	Function	Operation
JSR	Jump to subroutine	$SP - 2 \Rightarrow SP$ $RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ Continue address $\Rightarrow PC$
RTS	Return from subroutine	$M_{(SP)} : M_{(SP+1)} \Rightarrow PC_H : PC_L$ $SP + 2 \Rightarrow SP$

Table 5-23. Index Manipulation Instructions

Mnemonic	Function	Operation
<b>Addition Instructions</b>		
ABX	Add B to X	$(B) + (X) \Rightarrow X$
ABY	Add B to Y	$(B) + (Y) \Rightarrow Y$
<b>Compare Instructions</b>		
CPX	Compare X to memory	$(X) - (M : M + 1)$
CPY	Compare Y to memory	$(Y) - (M : M + 1)$

Table 5-24. Stacking Instructions

Mnemonic	Function	Operation
<b>Stack Operation Instructions</b>		
PSHA	Push A	$(SP) - 1 \Rightarrow SP; (A) \Rightarrow M_{(SP)}$
PSHB	Push B	$(SP) - 1 \Rightarrow SP; (B) \Rightarrow M_{(SP)}$
PSHX	Push X	$(SP) - 2 \Rightarrow SP; (X) \Rightarrow M_{(SP)} : M_{(SP+1)}$
PSHY	Push Y	$(SP) - 2 \Rightarrow SP; (Y) \Rightarrow M_{(SP)} : M_{(SP+1)}$
PULA	Pull A	$(M_{(SP)}) \Rightarrow A; (SP) + 1 \Rightarrow SP$
PULB	Pull B	$(M_{(SP)}) \Rightarrow B; (SP) + 1 \Rightarrow SP$

Table 5-5. BCD Instructions

Mnemonic	Function	Operation
ABA	Add B to A	$(A) + (B) \Rightarrow A$
ADCA	Add with carry to A	$(A) + (M) + C \Rightarrow A$
DAA	Decimal adjust A	$(A)_{10}$

Your thought-process, coupled with flowcharts and your test plan should lead you to the various instructions that are typically used to code this assignment. For example, the given algorithm states that you must divide a certain result to see if the answer is evenly divisible by 10 – e.g. that the remainder is 0. Given that we are working with unsigned integer values, which should lead us to the *idiv* instruction. To see how this, or any instruction works, make up a small test program, set up the registers appropriately, execute the instruction and observe the results – e.g. make wise use of the simulator's debugging capabilities.

The same holds true for any code you develop – test it in the simulator – if you don't get the expected results, single step the code to discover where you went wrong.