CIS-11 Project

Documentation

**Assemblers**

**Joseph Medina, Arianna Balderram, Emma Wuysang**

**Case B (Test Score Calculator)**

**Sunday, May 19**

**Advisor: Kasey Nguyen, PhD**

# Part I – Application Overview (Arianna)

### This part of the requirements document serves to present the “big picture” of the application. Here you lay out the objectives of the application, how it fits into the business process of the company, and how it relates to other software systems.

### The sections listed below should be included in this part of the requirements document.

## Objectives/ Business Process

**What business objectives of the company will this project help achieve? Possible objectives might be reducing costs, improving customer service, simplifying the workflow, replacing obsolete technology, piloting a new technology, and many others. Also, make sure you understand exactly how the proposed project will help accomplish the stated objective.**

* This project provides multiple benefits for our team. Coming up with the proper programming for this calculator gives all of us a better understanding of assembly and the concepts that are included within the project. What we will be working on for this project would include algorithms, managing data, controlling program flow, and problem-solving. And when it comes to displaying outputs for the test score calculator, it will provide the user with the minimum test score, maximum test score, the average test score, and the grade equivalent from A to F.

**Why are we doing this project now? What will happen if we do it later? What if we do not do it at all?**

* Completing this project ahead of time gives us a better opportunity for our team to succeed and do everything the way that we want before it is due. Giving ourselves enough time to go over everything and make sure that it is done properly is a very important part of the process for our team to be most confident and successful with this project.
* If we were to start later then it would show a lack of interest in doing this project. And if we were to do it at a later point, we would be causing ourselves unnecessary stress with other assignments that are due around the same period. It would cause a lot of inconveniences along with making it harder to improve our code in a time crunch.
* If we chose to not do the projects at all, we would simply get no credit for lack of work and miss a very beneficial coding opportunity for the class. And since it’s a required assignment for this course, not doing it would impact our grade in a negative way because this is worth a large number of points. But overall doing this project is a good opportunity for everyone in the class to get along and work as a team to improve our overall coding experience.

**Who will benefit from this project? Do the people who will benefit from it consider it the most important improvement that can possibly be made at this time? Should we be doing a different project instead?**

* Everyone in the team would benefit from this project because it helps improve all of our understanding of using assembly along with learning more concepts and how to use it. We also would benefit from it if we ended up doing everything correctly because we would get a good grade for the work that we did. At this time, this is most likely one of the bigger projects that everyone in the team has done and it is important for all of us because it will improve our coding skills. I believe that this is a good project for all of us to work on because it goes over the material that we learned throughout this semester and allows us to work as a team to share different perspectives of the project and how we can do our overall best.

## User Roles and Responsibilities

* For this assignment we split the workload into separate parts for each individual team member to work on their own time and to complete before the due date. Arianna was assigned to do Part 1 which lists the application overflow, objectives, process, and user roles. Emma was assigned Part 2 for functional requirements and listing a total summary of the purpose and things that we will incorporate into our project. And Joseph will be working on Part 3 for the appendices and providing the pseudocode to give an overall idea of how this project will come along.

## Production Rollout Considerations

* Everything will be taken care of within a timely manner, and with each of us being assigned a certain portion of the project, it will also open up potential corrections from one another which will help correct any problems that we may encounter later on.

# 

# Part II – Functional Requirements

### This portion will detail the functions and process of the calculator program.

## Statement of Functionality

### GENERAL BASIS:

### This application will work on creating a user input-based program of five test scores. With these numbers, the tester can see the maximum, minimum, and average. After conducting these three routines, the user will display the given letter grade based on the input scores. After, the console should display these values ranging from F (0-50), D (60-69), C (70-79), B (80-89), A (90,100).

**Origination -**

The LC-3 calculator program will originate at memory address x3000 as practiced, maximizing memory space, etc.

**Memory Allocation -**

Prioritize making space for allocated memory addresses utilized for storing input values, array elements, and output data. An address for input scores should be stored, array elements (including 5 for division of the test scores) are followed precedingly after the input scores, and output values commence after, to ensure proper management.

**Process (Calculation and Display) -**

After taking the user’s input for test five scores, the program efficiently calculates and displays the minimum, maximum, and average scores, along with the letter grades.

**Instruction -**

The calculator program makes use of a wide set of instructions, from arithmetic operations for the score calculations (will utilize a ton of two’s complements, etc), data movement instructions for proper data usage, and conditional operations for branching (if statements, for loops, etc) to utilize functionality.

**Comments -**

The code should include clear labels (max, min, avg, etc) and corresponding comments, which will help with readability and understanding of each portion’s purpose, this part is particularly significant for outsider comprehension.

**Subroutines and Branching -**

This program will use multiple subroutines in order to regulate the flow of the program, and it is required in order to have a functioning calculator project. Branches will need to be assigned to perform the operations, such as division, continuous addition of the test scores and a considerable number of for loops.

**Overflow Handling -**

The program handles potential overflow scenarios (entering more than the program can hold) during arithmetic calculations, which is all for correct computations and preventing data corruption/loss/infinite loops. Make sure there is input validation to also factor into avoiding data overflow.

**Stack Management -**

Utilize PUSH and POP operations for efficient stack management, this is for subroutine calls and having flow.

**State Preservation -**

Save and restore operations are meticulously integrated to safeguard register states, enabling seamless context switching and preserving critical data across subroutine calls and program execution phases.

**Pointer Utilization -**

Pointers should be used for keeping access to array elements or memory locations, this will optimize space and data retrieving.

**ASCII Operations -**

ASCII conversion will be needed for output operation, as you will work with assigning the letter grades.

**System Calls -**

Significantly essential for the tasks of getting user input or displaying output. Displays the grades, etc.

**Continuous Testing -**

The program should be able to work with various inputs, aside from the desired test scores (52, 87, 96, 79, 61). If not, refer to the functionality steps above and review where the program fails to meet needed steps.

## Scope

## PHASE 1:

Prompt the user for 5 numbers, these will be acting as the test scores.

## PHASE 2:

Take those numbers and perform the desired operations within the assembly.

## PHASE 3:

Assign min, max, average and letter grade to numbers.

## PHASE 4:

Output these numbers on the console and release the letter grade to each test score.

UTILIZE:

Subtraction, Addition, Multiplication, Division, Loops, Branching

MEMORY:

Storing and recalling required data, clearing registers

USER INTERFACE:

Simple question on console to input numbers, data validation too.

ALGORITHMS:

Min -> found through checking all 5 test scores for lowest number (PUSH-POP, etc possible)

Max -> found through checking all 5 test scores for highest number (PUSH-POP, etc possible)

Average -> Add the numbers, account for overflow, and divide by the amount (5).

LIMITATIONS:

Overflow and ASCII Conversion

## Performance

### Application should take,

### less than 2 seconds to initialize

### less than 3 to produce the letter grade to the 5 assigned scores

### Arithmetic and other operations should take less than a few (100) milliseconds to perform

### Complex operations only 500 milliseconds longer

### Program should be less than 5 kb, if not 4.

## Usability

* Must be able to handle overflow of test scores.
* Straight forward question, handling data validation
* Branch efficiently and utilize desired pointer(s)

# Documenting Requests for Enhancements

*areas to improve the program,*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Date** | **Enhancement** | **Requested by** | **Notes** | **Priority** | **Release No/ Status** |
| 5/15 | Have a brief introduction within the string before asking the user the question for the five test scores. | Emma W. | Adds to the user interface | Mid Priority, does not fall on requirements, but is reasonable practice. | Final Launch |
| 5/16 | Add a type of message to the letter grade, failing or passing? | Joseph M. | Adds to the user score personalization | Low Priority, does not fall on requirements, but is reasonable practice. | Final Launch |
| 5/17 | Handle ASCII and check for areas to free space with pointers, along with managing overflow | Arianna B. | For functionality and efficient program | High Priority, falls on the requirements and is reasonable practice. | Final Launch |

# Part III – Appendices (Joseph)

### If a developer were to modify the program, we expect it would be to change the program where you can add an unlimited amount of test scores. Implementing this might be difficult considering the use and elimination of data with registers. Another possible addition to this program could be to implement extra credit where the user is allowed to score higher than 100.

## Flow chart or pseudo-code.

### Pseudocode

1. Main Program:
   1. Clear and Load addresses for scores and letter grades (if using static data)
   2. Loop 5 times asking for 5 different test scores
      1. Validate that the numbers are positive and less than 101
         1. If negative, prompt “Enter a valid test score.”
            1. Re-enter loop
         2. If greater than 100, prompt “Enter a valid test score.”
            1. Re-enter loop
      2. Store user input once the user has entered 5 test scores where it can be stored in an array if necessary
         1. Exit loop
      3. Display “The test scores you entered are: 52 87 96 79 61.”
   3. Jump to subroutine named “MIN” for the minimum grade
      1. Push-pop: implemented by saving the return address at the beginning of the subroutine
      2. Loop through each test score
         1. Load each score into a register
         2. Conduct a comparison between test scores
            1. If the test score being compared is lower, update the new minimum score
         3. Handle overflow if it occurs
            1. Conduct overflow handling in case the score is greater than the max at the memory location

Branch to overflow subroutine

* + - * 1. If there is no overflow, continue.
      1. Once iterated 5 times, exit loop.
    1. Push-pop: restore the return address
    2. Display “Your minimum value is 52.”
  1. Jump to subroutine named “MAX” for the maximum grade
     1. Push-pop: implemented by saving the return address at the beginning of the subroutine
     2. Loop through each test score
        1. Load each score into a register
        2. Conduct a comparison between test scores
           1. If the test score being compared is higher, update the new maximum score
        3. Handle overflow if it occurs
           1. Conduct overflow handling in case the score is greater than the max at the memory location

Branch to overflow subroutine

* + - * 1. If there is no overflow, continue.
    1. Push-pop: restore the return address and loop until the minimum value is determined.
    2. Display “Your maximum value is 96.”
  1. Jump to subroutine named “AVG” for the average grade
     1. Add all the test scores together
     2. Store the sum of the test scores in a register
     3. Call subroutine named “DIV” to calculate the average
        1. Push-pop: implemented by saving the return address at the beginning of the subroutine
        2. Divide the sum by using a repeated subtraction method. Subtract the divisor (five) until the result is zero or negative.
           1. If positive, continue the loop
           2. If the negative or zero, division is complete, exit the loop
        3. Clear quotient register
     4. Display “Your average test score is: 75
     5. Display “Your grade is: C”
  2. Stop/Halt the program
  3. Fill in data (WITH designated LABELS min,max,avg,etc) if we are using static data.