CIS-11 Project Documentation

**Team JGH:**

**Joshlyn Bui**

**Gray Greenridge**

**Heather Laird**

**Case B: Test Score Calculator**

**May 25, 2025**

**Advisor: Kasey Nguyen, PhD**

# Part I – Application Overview

## Objectives

The main objective of this project is to prove our team’s knowledge and understanding of computer assembly and LC-3. We are pursuing this project at the end of the course as we now have an ample amount of knowledge surrounding the basic functionalities of assembly language and LC-3. If we decide not to do this project, we lose the opportunity to apply our knowledge of significant concepts we have learned in this class. Our team will benefit from this project by gaining a greater understanding of the concepts that we have learned throughout this course and learn how to utilize them through documentation and programming. In addition, we will gain experience working and programming as a team.

## User Roles and Responsibilities

Users: Joshlyn Bui, Gray Greenridge, Heather Laird

Main roles: Each user has a role in documentation, pseudocode and coding daily with weekly check-ins.

Documentation and Pseudocode: Divided into tasks between each team member.

Joshlyn Bui - Production Rollout considerations, Functionality requirements, pseudo-code

Gray Greenridge - Pseudo-code

Heather Laird - Objectives, Scope, and pseudo-code

## Production Rollout Considerations

Our team will be working individually on assigned tasks, with the use of a shared Google Docs for the documentation part of the project, and a shared [Github repository](https://github.com/joshlynb/Test-Score-Calculator) for the programming part of the project. As everybody will be working individually, our team will schedule at least one meeting per week and schedule personal deadlines for the group. This will allow every member to be updated on the status of our individual assigned tasks and the progress of the project as a whole despite working individually.

| **05/05/25** | **05/12/25** | **05/21/25** | **05/23/25** | **05/28/25** |
| --- | --- | --- | --- | --- |
| * Conduct meeting to introduce team members * Create Github repository * Start on documentation and pseudocode | * Conduct meeting to introduce additional team members * Work on documentation * Work on program pseudocode | * Conduct meeting to delegate tasks and clearly define the role of each team member * Work on documentation * Work on program pseudocode | * Conduct meeting * Finalize documentation and pseudocode | * Conduct meeting * Finish input and data management functionalities * Continue work on other project functionalities |
| **06/04/25** | **06/06/25** | **06/09/25** | **06/11/25** | **06/12/25** |
| * Conduct meeting * Finish min,max, and avg calculation functionalities * Finish letter grade equivalence display functionalities * Continue work on project functionalities | * Finalize required project functionalities * Test program to make sure it is functional | * Finalize project details before true deadline (06/12) | * Conduct meeting to address team members’ final comments or concerns | * Finalize project (if not done already) and submit on Canvas * Submit team evaluation form * Create/fork individual Github repository for each team member |

# Part II – Functional Requirements

## Statement of Functionality

This program prompts the user to input 5 test scores, and displays the minimum, maximum, and average grade of the test score, and the corresponding letter grades to each test score.

### **Address Management:**

Origination, fill contents, array contents, and input and output must be at their appropriate addresses.

### **Calculation and Console Display:**

The calculated values for the minimum, maximum, and average test score values, and the test scores’ letter grade equivalents will be displayed to the console. The letter grade equivalents are 0 – 59 = F,

60 – 69 = D, 70 – 79 = C, 80 – 89 = B, 90 – 100 = A.

### **Labels and Comments:**

The program will include clear and understandable labels and comments for readability and comprehension of the code and the purpose of each section.

### **Program Instruction:**

The program will contain instructions for arithmetic calculations, data movement, and conditional and iterative operations.

### **Subroutine Implementation:**

The program will contain at least 2 or more subroutine procedures and implement subroutine calls.

### **Control Structure Implementation:**

The program will implement branching, conditional, and iterative control structures.

### **Overflow Management and Storage Allocation:**

The program will be able to prevent and handle underflow and overflow errors to prevent problematic situations such as data loss or data corruption. This will be done with user input validation functionality and program instruction to detect underflow and overflow.

### **Stack Management:**

The program will implement the usage of stack abstract data type, as the program will need to handle and store multiple values for program functionality, and there are a limited number of registers.

### **Save-Restore Operations:**

The program will include save-restore operations as they allow for the program to continue to operate even when experiencing external interruptions.

### **Pointer Utilization**

The program will utilize pointers to indirectly access and manipulate the necessary data for program functionality.

### **ASCII Conversion Operations**

The program will be able to convert the user-inputted data into a binary or hexadecimal format that is processable by the computer, and decode calculated data such as the minimum, maximum, and average test score values back into ASCII format for console display and human-readability.

### **System Call Directives**

The program will implement system calls as they allow for users to safely interact with the program and perform sensitive operations.

### **Testing**

The program will be tested with the following test score values: 52, 87, 96, 79, 61

The program will correctly display the test scores’ corresponding letter grades: D, B, A, C, D

The program will correctly display the minimum, maximum, and average score: 52, 96, 75

### **User Input Validation**

The program will be able to validate the user’s input, ensuring that the test scores do not contain any non-numerical values and only range from 0 – 100.

## Scope

The central goal of this project is to work together as a group to efficiently create a test score calculator program in LC-3. The first phase of this project is the planning phase. During this phase we firstly created our group, conducted a meeting to introduce the team members and started planning documentation and pseudocode of our project. We divided the tasks of the project and discussed how each member was going to contribute. We decided that we would meet once a week to review and discuss our progress on the project. The next phase in our project is the creation of the program itself. The main goal of the program is to work efficiently to display the minimum, maximum and average of the test scores given to the program, as well as display the correct letter grade associated with the test scores. The program will include the following functionalities: address management, calculation and console display, labels and comments, program instruction, implementation of subroutines, control structures, overflow management and storage allocation, stack management, save-restore operations, pointer utilization, appropriate conversion operations, and system call directives. The last phase in the project will consist of testing, reviewing, and adding additional features to improve the program. The program will be thoroughly tested for accuracy and efficiency. Upon the required functionalities being implemented successfully, the team will then evaluate requests for enhancements. The final project is projected to be finished by June 9th, with documentation to be finished by May 23rd.

# Documenting Requests for Enhancements

| **Date** | **Enhancement** | **Requested by** | **Notes** | **Priority** | **Release No/ Status** |
| --- | --- | --- | --- | --- | --- |
| 05/20/25 | Input Validation | Joshlyn Bui | Input validation for checking for non number inputs and for test scores not within the range 0-100 | Mid Priority- not a project requirement, but is generally good practice and will ensure proper data management |  |
|  |  |  |  |  |  |

# Part III – Appendices

## Flow chart or pseudo-code.

#### Main Program

* 1. Loop a prompt for user to input a test score
     1. Prompt user to input digit
     2. Push digit into stack
     3. Loop 3 times to get 3 digits for one test score
  2. Convert ASCII input to hexadecimal value
     1. Pop value from stack
        1. Add #–48 to value (Call Subroutine ENCODE)
     2. Convert digit to proper place value
        1. Set counter = #3 (“counter” is a loop counter value)
        2. When i = #3
           1. Add digit to register
        3. When i = #2
           1. Multiply digit by 10 (Call Subroutine MULT\_BY\_10)
           2. Add digit to register
        4. When i = #1
           1. Multiply digit by 100 (Call Subroutine MULT\_BY\_100)
           2. Add digit to register
        5. When i = 0, branch to step 1c.
  3. Input validation
     + 1. If 100 - input < 0, tell user that the input was bad, halt program
       2. If 0 - input > 0, tell user that input was bad, halt program
  4. Store test scores in array
     1. Store test score in R0, JSR PUSH.
  5. Display corresponding letter grade
     1. JSR GET\_LETTER
  6. Display minimum, maximum, and average letter grade.

#### Subroutine VALID (Input validation for test score )

* 1. Check if inputted character is in the range of valid ASCII characters (0-9)
     1. Take R0, subtract from x39
     2. If result is greater than or equal to 0, jump to next check
     3. If result is less than 0, send a message to user, halt the program
     4. Take x30, subtract from R0
     5. If result is greater than or equal to 0, jump back to the call
     6. If result is less than 0, send a message to user, halt the program

#### Subroutine MIN (Find & display minimum test score)

* 1. Clear register
  2. Use a loop to iterate through the array of test scores
  3. Set a register to hold minimum score
  4. Compare scores to minimum, if score is < minimum store as minimum score in minimum register

#### Subroutine MAX (Find & display maximum test score)

1. Clear register
2. Use a loop to iterate through the array of test scores
3. Set a register to hold max score
4. Compare scores to max, if score > max store as max score in max register

#### Subroutine AVG (Find & display average test score)

* 1. Clear a register
  2. Iterate through array of test scores
     1. Add each test score to specified register
     2. Divide contents of specified register by 5

#### Subroutine MULT\_BY\_10 (Multiply by 10)

* 1. R1 = R0 + R0
  2. R2 = R1 + R1
  3. R3 = R2 + R2
  4. R1 = R1 + R3
  5. Store R1 in R0, clear registers

#### Subroutine MULT\_BY\_100 (Multiply by 100)

* 1. Call MULT\_BY\_10 twice.

#### Subroutine DIV\_BY\_5

* 1. loop however many times:
     1. Subtract 5 from the number until it can’t be done anymore
     2. Take this remainder, multiply by 10
  2. Disregard remainder, store the result of division in a register

#### Subroutine DECODE (Hexadecimal → ASCII conversion)

* 1. Add #48 (ASCII offset)
     1. Add #15 to register
     2. Add #15 to register
     3. Add #15 to register
     4. Add #3 to register
  2. Return

#### Subroutine ENCODE (ASCII → Hexadecimal conversion)

* 1. Add # – 48 (ASCII offset)
     1. Add #15 to register
     2. Add #15 to register
     3. Add #15 to register
     4. Add #3 to register
     5. NOT value inside register
     6. Add #1 to register
  2. Return

#### Subroutine PUSH

* 1. Take R0, increment R6 (stack pointer) by one
  2. Store R0 into the stack address + stack pointer
  3. Return

#### Subroutine POP

* 1. Overwrite stack address + stack pointer with empty
  2. Decrement R6 (stack pointer) by one
  3. Return

#### Subroutine GET\_LETTER

* 1. If score is lesser than 59, return F
  2. If score is lesser than 69, return D
  3. If score is lesser than 79, return C
  4. If score is lesser than 89, return B
  5. If score is lesser than or equal to 100, return