CIS-11 Project Documentation Template

Team Name: Beans and Rice
Team Member: Melanie Guerra & Kawin Khamkathok
Project Name: Test Score Calculator
Date: May 24, 2025

Advisor: Kasey Nguyen, PhD

Part I – Application Overview

The purpose of this project is to create an LC-3 Assembly program that determines the maximum, minimum, and average of 5 test scores for the user inputs. The program will also determine a letter grade (A-F) based off the average score. All the results will be displayed to the user on the console. This project demonstrates key lc-3 assembly concepts like arithmetic operations, data storage with arrays, stack management with subroutines, ASCII conversion, etc.

Objectives

From a business perspective, the objectives include:

• create a program that can be used as a grading assistant tool

• lay groundwork for how such logic could be used in larger systems like school grading platforms

Why are we doing this?

To elicit the objectives, ask the business expert, the development manager, and the project sponsor the following questions:

The purpose of this project is to provide a lightweight grade processing tool to assist educators in analyzing student's exam grades efficiently. From an educational and business perspective, this project helps achieve the following:

- Improve accuracy in grade calculation
- replace outdated methods of grading
- shows how assembly can be used in user-friendly programs

We are doing this project now because it aligns with current learning goals in Assembly programming and provides students with an opportunity to apply technical concepts to a meaningful, real-world-inspired use case. Delaying this project would reduce the opportunity to reinforce recent course material and skipping it altogether would miss a chance to demonstrate the practicality of LC-3 and reinforce subroutine, memory, and branching logic in a complete application.

The primary beneficiaries are:

- Students, who learn to write structured, efficient, real-world Assembly code
- Instructors, who can use this logic as a starting point for more complex grade-processing examples
- Educational software teams, who may use similar logic in backend systems to generate quick grade evaluations

Business Process

To calculate grades it usually requires manual entry into some sort of document or grading software but with this project the Test Score Calculator it asks the user to enter five test scores via the console, the program then calculates and displays the minimum, maximum, and average scores, then maps the average to a letter grade and displays it to the user. Although it is limited to five test scores with improvements it could grow to fully automate the grading process and make it easy.

User Roles and Responsibilities

I am responsible for the core structure and data processing of the program. This included setting up the LC-3 environment with. ORIG, defining memory labels, and allocating space for the five test scores. I'll implement the user input functionality, handle ASCII-to-binary conversion and store the values using pointers or indexed addressing. I'll also develop the subroutines for calculating the minimum, maximum, and average scores and ensure proper use of stack operations with PUSH, POP, and save/restore instructions. Melanie will also start on the initial flowchart and pseudocode outlining the calculation logic.

Kawin's role is more focused on grade evaluation, output formatting, and documentation. Kawin will write the logic and subroutine for assigning a letter grade based on the average score, using conditional branching to match the appropriate range. Kawin will also create the output display routines to show the minimum, maximum, average, and letter grade, managing ASCII conversion and system calls for proper console output. They ensured input values were validated and managed memory handling to prevent overflow. Kawin will finalize the flowchart and pseudocode
Both partners collaborated on integrating and testing the program, using sample inputs such as 52, 87, 96, 79, and 61 to validate functionality. Working together to debug the code, write comments, and refine the documentation and flowchart. The teamwork ensured a complete, tested, and well-documented project that met all requirements

Production Rollout Considerations

Deployment Plan
The program will be loaded onto the LC-3 simulator environment for execution and testing. Deployment consists of assembling and loading the .asm file into the LC-3 IDE or virtual machine.

Initial Data Population
Since the program prompts for live user input, no preloaded data is required. Users will enter their test scores at runtime.

Expected Data and Transaction Volume
This is a lightweight educational application. The program handles a small, fixed number of inputs (5 test scores) and performs basic arithmetic operations. Transaction volume is

minimal and restricted to one user session at a time.

Scalability Consideration
While this version handles only 5 inputs, future versions could be expanded to support more scores or even multiple students. However, given the limitations of LC-3 Assembly, scalability is intentionally limited.

Support Plan
Users (students and instructors) can rerun the program in the simulator for additional test runs. Debugging and maintenance will be handled through source code updates and reassembly.

Terminology

Terminology:
LC-3: A simplified instruction set architecture used for educational purposes.

ASCII Conversion: Converting numeric characters (e.g., '5') to binary integers (e.g., 5).

Subroutine: Reusable section of code called from the main program.

Stack: Memory used for storing temporary data (like return addresses and register values).

PUSH/POP: Stack operations to save/restore data during subroutine calls.

Overflow Management: Preventing errors due to arithmetic operations exceeding limits.

Part II – Functional Requirements

This part of the requirements document states in a detailed and precise manner what the application will do.

Statement of Functionality

Prompt the user five times for test scores.
Convert each score from ASCII to integer.
Store each score in memory.
Compute the minimum score from the inputs.
Compute the maximum score from the inputs.
Compute the average score using integer division.
Convert the average score into a letter grade using defined ranges:
0–59 = F
60–69 = D
70–79 = C
80–89 = B
90–100 = A
Display all of the above results to the user using ASCII output.
The program uses:
Subroutines for modularity.
Branches and loops for input collection and comparison.
Stack operations for subroutine state management.
ASCII conversion to display readable output.
Scope
Input handling
Arithmetic operations
Output and display logic
Stack usage
Overflow handling
ASCII conversion
Performance
Input and output operations respond within 1 second each.
The entire program executes in under 5 seconds.
Performance measured using the LC-3 simulator.
Tested for edge conditions like max (100) and min (0) scores.

Scope

Input handling
Arithmetic operations
Output and display logic
Stack usage
Overflow handling
ASCII conversion

Performance

Input and output operations respond within 1 second each.
The entire program executes in under 5 seconds.
Performance measured using the LC-3 simulator.
Tested for edge conditions like max (100) and min (0) scores.

Usability

The user interface is console-based with clear prompts.

Labels accompany each output value (Min, Max, Avg, Grade).

Input and output are designed to be user-friendly and self-explanatory.

Documenting Requests for Enhancements

There does come a time when the requirements for the initial release of your application are frozen. Usually, it happens after the system acceptance test which is the last chance for the users to lobby for some changes to be introduced in the upcoming release.

Currently, you need to begin maintaining the list of requested enhancements. Below is a template for tracking requests for enhancements.

Date	Enhancement	Requested by	Notes	Priority	Release No/ Status

Part III – Appendices

Appendices are used to capture any information that does not fit naturally anywhere else in the requirements document yet is important. Here are some examples of appendices.

Supporting and background information may be appropriate to include as an appendix – things like results of user surveys, examples of problems to be solved by the applications, etc. Some of the supporting information may be graphical – remember all those charts you drew trying to explain your document to others?

Appendices can be used to address a specialized audience. For example, some information in the requirements document may be more important to the developers than to the users. Sometimes this information can be put into an appendix.

Flow chart or pseudo-code.

Include branching, iteration, subroutines/functions in flow chart or pseudocode.

MAIN PROGRAM
CALL INIT,
CALL READ_INPUTS,
CALL FIND_MIN,
CALL FIND_MAX,
CALL CALC_AVG,
CALL GET_GRADE,
CALL DISPLAY_RESULTS,
HALT,
SUBROUTINE: READ_INPUTS
Loop 5 times:
Prompt user
Read ASCII input
Convert ASCII to binary
Store in memory
SUBROUTINE: FIND_MIN
Set first score as min
Loop through array:
If score < min, update min
SUBROUTINE: FIND_MAX
Set first score as max
Loop through array:
If score > max, update max
SUBROUTINE: CALC_AVG
Sum all scores
Divide sum by 5
Store average
SUBROUTINE: GET_GRADE
Compare average to grade brackets
Store letter grade
SUBROUTINE: DISPLAY_RESULTS

Convert results to ASCII
Output min, max, avg, grade
STACK USAGE
PUSH/POP R0–R7 as needed in subroutines
Use temporary stack space to store return addresses and values
REGISTER SAVE/RESTORE
At subroutine entry: PUSH used registers
At subroutine exit: POP and restore
ASCII CONVERSION
Convert between binary numbers and characters for input/output