CIS-11 Project Documentation

Team Effort
Alexa Torres
Joshua Hernandez
Test Score Calculator
5/25/2025

Advisor: Kasey Nguyen, PhD

Part I – Application Overview

The test score calculator is designed to help users input 5 test scores, compute the minimum, maximum, and average score, and then determine the letter grade for each. This utility supports educational environments or grading systems where consistent evaluation of numeric grades and corresponding letters is needed. This application supports automated grade analysis in academic institutions or small scale educational software. By lessening human error in calculating grades and simplifying grade conversion, it helps us make sure there is fairness and accuracy in student evaluations.

Objectives

The primary objective of the test score calculator project is to develop an interactive, low level educational tool using LC-3 assembly language that automates the process of evaluating and displaying student test scores. This tool is intended for academic use in teaching fundamental concepts of computer organization, low level programming, and systems architecture.

Why are we doing this?

- **What objectives will this project help achieve?** ***It provides a practical, hands-on experience in LC-3 assembly language, reinforcing theoretical knowledge through applied problem-solving.***
- **Why are we doing this project?** ***We are aiming to improve our technical fluency in low level operations, especially in how systems can handle memory, input/output, and arithmetic. This project also provides exposure to concepts that are foundational for advanced topics.***
- **Who will benefit from this project?** ***The larger objective is to create a reusable, student friendly LC-3 project. This calculator will serve as a good beginning for more complex future assignments involving data structures or memory management.***

Business Process

Students in introductory computer architecture or systems programming courses learn LC-3 through theoretical examples or small, isolated instruction level exercises. Grading logic and arithmetic operations are usually taught conceptually or performed manually on paper or calculators.

Student Tasks:

- *Manually compute min, max, and average from sample inputs.*
- *Write small, unconnected LC-3 programs without real-world application.*
- *Learn basic input/output without practical integration.*

With the introduction of the test score calculator, the process is enhanced through structured application of multiple LC-3 concepts in a cohesive and meaningful program. The project integrates user input, memory management, conditional logic, arithmetic operations, and output display in a real world grading scenario.

Student Tasks:

- *Write and test a complete LC-3 application using subroutine and stack operations.*
- *Convert user-entered ASCII characters into numeric values.*
- *Compute and display the minimum, maximum, average, and letter grade.*
- *Gain practical exposure to debugging and testing real time I/O in a constrained system.*

User Roles and Responsibilities

Objective:

- *Learn and demonstrate competency in LC-3 assembly programming, systems level logic, and low level data handling through a hands on project.*

Tasks:

- *Input data: Enter five test scores in ASCII format via the console.*
- *Debug code: Test the program for correct input, branching, and computation.*
- *Verify output: Ensure that minimum, maximum, average, and letter grades are accurately calculated and displayed.*
- *Submit Assignment: Upload the completed and tested program for grading.*

Production Rollout Considerations

Since the test score calculator is an educational software project developing in LC-3 assembly, the rollout is academic rather than commercial. It will be deployed as a part of a course module, lab assignment, or term project in a systems programming or computer architecture course.

Terminology

Minimum score - The lowest of the five test scores entered.

Maximum score - The highest of the five test scores entered.
Average score - The sum of the five test scores divided by five, using integer division.
Letter grade - The academic grade corresponding to the average score, based on predefined ranges.
ASCII - Standard for representing characters as numeric codes. Used to interpret keyboard input and display output in LC-3 program.
Subroutine - Reusable block of assembly instruction that performs a specific task and can be called multiple times using JSR.
Stack - A last in first out data structure used to store temporary values such as return addresses and saved register values.
PUSH/POP - Assembly level operations that store or retrieve data from the stack.
Save/Restore - Process of saving register values before a subroutine modifies them and restoring them afterward to maintain state consistency.
Overflow - Condition where a calculated value exceeds the range that can be represented in the LC-3's 16-bit signed integer format.
Branching - Assembly control flow operations used to conditionally jump to different parts of the program.
System call (TRAP) - Predefined instruction in LC-3 that invokes an operating system like service.
Instruction - Single operation performed by the LC-3 CPU, such as load, store, add, or jump.
Register - A small, fast storage location inside LC-3 CPU usd to temporarily hold data.
Pointer - A register or memory location that holds the address of another value or structure, such as the top of the stack.
Simulator - A software tool that emulates the LC-3 computer, allowing students to run and test their programs.
Input - The section of code responsible for reading user input from the keyboard.
Output - The section of code that formats and displays results to the console.

Part II – Functional Requirements

Statement of Functionality

This section precisely defines the functionality of the LC-3 Test Score Analyzer program. The system will implement only the functions listed here and will not include any additional functions not described. This specification serves as the binding agreement between the development team and stakeholders.

The system is designed for a single user category:

- *User category: General user (student or tester)*
- *Purpose: Input, process, and view a summary of five test scores*

The system shall wait for the user to input exactly five test scores, one at a time.

Each test score shall be a single-digit value (0–9), entered from the keyboard.

The system shall echo each character input back to the display to provide visual confirmation.

Note: The system shall not support multi-digit inputs, negative numbers, or non-numeric inputs.

2 Score Conversion and Storage

- *For each character input, the system shall:*
  - *Convert the ASCII character ('0' to '9') into its numeric integer value (0–9).*
  - *Store this value temporarily in a register for further processing.*

3 Score Processing

- *After each input, the system shall:*
  - *Update the minimum score if the new score is less than the current minimum.*
  - *Update the maximum score if the new score is greater than the current maximum.*
  - *Add the new score to the running sum for later average calculation.*
- *Initial conditions:*
  - *Minimum score is initialized to 100 (a high placeholder).*
  - *Maximum score is initialized to 0.*
  - *Sum is initialized to 0.*

4 Average Calculation

- *After all five scores are entered:*
  - *The system shall compute the average by dividing the sum of scores by 5.*
  - *The result shall be stored in a designated memory location for later display.*
- *Note: Due to LC-3 architecture limitations, division may be implemented using loops or other non-division instructions.*

5 Results Display

The system shall display the following result section:

makefile
CopyEdit
Results:

- *Then, it shall display:*

Minimum Score in the format:
php-template
CopyEdit
Minimum Score: <value>

Maximum Score in the format:
php-template
CopyEdit
Maximum Score: <value>

Average Score in the format:
php-template
CopyEdit
Average Score: <value>

- *Note: Each displayed value shall be a single-digit integer, as stored in the program.*

6 Grade Assignment

- *Based on the computed average, the system shall assign and display a letter grade as follows:*
  - *A if average ≥ 90*
  - *B if average ≥ 80 and < 90*
  - *C if average ≥ 70 and < 80*
  - *D if average ≥ 60 and < 70*
  - *F if average < 60*
- *The system shall display only the single letter (e.g., A), without any additional explanation.*

7 System Termination

- *After displaying all results and the letter grade, the system shall halt execution.*

Limitations and Exclusions

- *The system shall not handle invalid or non-numeric inputs.*
- *The system shall not handle input errors, backspacing, or correction.*
- *The system shall not support entering more or fewer than five scores.*
- *The system shall only work within the LC-3 simulated environment; it is not intended for deployment on other architectures.*

Scope

This project will deliver a functional LC-3 assembly program that allows a user to input five test scores, computes the minimum, maximum, and average, and then assigns a letter grade based on the average.

Because the development is divided into multiple phases, the scope is broken down as follows:

Phase 1: Core Input & Processing

- *Prompt the user to enter five test scores.*
- *Read and store each score as numeric input.*
- *Convert ASCII character input into integer values.*
- *Keep a running sum of scores for later average calculation.*
- *Track the minimum and maximum scores as input is processed.*

Phase 2: Computation & Storage

- *After all inputs, compute the average score.*
- *Store computed values (minimum, maximum, average) in memory locations.*
- *Prepare thresholds and divisors for later grade computation.*

Phase 3: Display Results

- *Output the minimum score to the console.*
- *Output the maximum score to the console.*
- *Output the average score to the console.*

Phase 4: Grade Evaluation

- *Compare the computed average against preset grade thresholds (A, B, C, D, F).*
- *Display the correct letter grade based on the average score.*

Out of Scope

- *Error handling for non-numeric input (this version assumes correct input).*
- *Dynamic score count (fixed to five scores in this version).*
- *Weighted averages or advanced grading curves.*

Notes

If the project is later expanded, the functionality statement section can include planned features like:

- *Handling a variable number of inputs.*
- *Allowing re-entry or correction of scores.*
- *Providing summary statistics (median, standard deviation).*

Performance

Input Processing

- *Each individual test score input should be read, converted, and stored using no more than 100 instructions per score.*
- *Total time to process all five scores (excluding user typing time) should be less than 500 instructions.*

Score Computation

- *The minimum, maximum, and sum updates should complete in < 20 instructions per score, i.e., total under 100 instructions for all five scores.*
- *The average computation should be completed in < 50 instructions after all scores are collected.*

Memory Usage

- *The total memory footprint (including data storage and code) should not exceed 512 words (x0200 to x03FF), ensuring the program stays within the LC-3's available user memory space.*

Result Display

- *Each result output (min, max, average, grade) should display on the console using ≤ 50 instructions per output, totaling ≤ 200 instructions for all results.*

Overall Execution

- *The entire program, from first prompt to final grade output, should complete in ≤ 1000 instructions (excluding any time the CPU waits for user keystrokes).*

User Interaction

- *Console outputs (prompt, results, grades) should appear immediately after the corresponding action, i.e., < 1 instruction delay after computation, to avoid perceptible lag in the simulated environment.*

I/O Assumptions

- *User input speed is not controlled by the program; the program will wait indefinitely for keystrokes but will process each keystroke in O(1) time after it is entered.*

Notes

- *Instruction count serves as the best proxy for time performance in this system because LC-3 runs on a fixed clock per instruction.*
- *These performance targets are designed to ensure the program remains lightweight and efficient, suitable for use in educational or constrained embedded environments.*
- *Meeting these requirements can be verified through instruction-level simulation (e.g., using the LC-3 simulator's step-through and count tools).*

Usability

Prompt Clarity

- *The program must display a clear and unambiguous prompt:*

→ “Enter five test scores:”

- *100% of users should be able to understand from this prompt that they need to input five numeric scores, one at a time.*

Input Feedback

- *After each keystroke (score entry), the character should be echoed to the console within 1 instruction cycle so the user sees what they typed without noticeable delay.*

Error Prevention (User Scope)

- *Since the program only reads single-character inputs (due to GETC), the system assumes users will input single-digit scores (0–9).*
- *To avoid user confusion, instructions or documentation must clearly state this limitation to 100% of users.*

Result Presentation

- *Final results should be labeled and structured so they are readable, e.g.:*
  → “Minimum Score: [value]”
  → “Maximum Score: [value]”
  → “Average Score: [value]”
  → Letter grade shown separately (A/B/C/D/F).
- *90% of users should be able to understand the results without additional explanation.*

Navigation Simplicity

- *There is no menu or navigation; the flow is strictly linear:*
  → Prompt → Input → Compute → Display.
- *Users should be able to complete one full run (entering five scores, receiving results) in under 2 minutes, assuming average human typing speed.*

Consistency

- *All text outputs (prompts, labels, grades) must use consistent formatting and terminology across runs to reduce user confusion.*

Learnability

- *New users (first-time LC-3 users) should be able to correctly complete the input and interpret the results after no more than one practice run.*

Notes

- *Usability goals are framed to match the minimal, text-only interface constraints of LC-3 assembly programs.*
- *Any improvements to usability (e.g., supporting multi-digit scores or error handling for invalid inputs) would require future feature expansion beyond the current scope.*

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**
5/21/25	Formatting	Joshua Hernandez	some things aren’t capitalized	not high	close
5/22/25	Name Correlation	Joshua Hernandez	names don’t correlate to their variables, (temporary fix added)	a bit higher	far
5/23/25	“Why are there 20 errors?”	Alexa Torres	it literally doesn’t run , who wrote this?!!	running/should be focused on	hopefully fixed soon

Part III – Appendices

Identified Issues in Code

1 ASCII Conversion Problem

The line:

```
assembly
CopyEdit
ADD R5, R6, R0  ; Convert ASCII to integer
```

should subtract x30 to turn the ASCII character ('0'–'9') into the numeric value. But here, it adds instead.

Possible fix:

```
assembly
CopyEdit
LD R6, ASCII_CONV
NOT R6, R6
ADD R6, R6, #1
ADD R5, R0, R6  ; R5 = R0 - '0'
```

2 SUM and AVERAGE Mismatch

In COMPUTE_AVERAGE, we are looping again over the sum:

```
assembly
CopyEdit
AVG_LOOP
ADD R7, R7, R4  ; Accumulate sum
ADD R1, R1, #-1
BRp AVG_LOOP
```

But we already have the total sum in R4, so multiplying again by SCORE_COUNT is wrong.

Possible fix:
Simply divide:

```
assembly
CopyEdit
LD R1, DIVISOR
AND R7, R7, #0
ADD R7, R4, #0  ; Copy sum to R7

; Repeated subtraction for division (basic integer divide)
AND R5, R5, #0  ; R5 = quotient
DIV_LOOP
   ADD R7, R7, #-5
   BRn DONE_DIV
   ADD R5, R5, #1
   BR DIV_LOOP
DONE_DIV
ST R5, AVG_SCORE
```

3 Outputting Raw Integer as Character

We are loading:

```
assembly
CopyEdit
LD R0, MIN_SCORE
OUT
```

But this outputs the raw numeric (like x5) as a character, which may not print as '5' but some random symbol.

Possible Fix:
Convert numeric value to ASCII:

```
assembly
CopyEdit
LD R0, MIN_SCORE
ADD R0, R0, x30  ; Convert digit to ASCII
OUT
```

Summary of Next Steps

Fix the ASCII conversion
Correct the average calculation
Fix how numbers are displayed as characters

Flow chart or pseudo-code.

```
start
    PROMPT ask user for test scores
    initialize score_count, max_score, min_score, sum_score

    for each test score:
        read_input
        process_score
        decrement score_count
        repeat until all scores are processed

    compute_average

    display results:
        show min_score
        show max_score
        show avg_score
        show grade
END

subroutine read_input:
    get user input as character
    convert to integer
    return

subroutine process_score:
    if input < min_score then update min_score
    if input > max_score then update max_score
    add input to sum_score
    return

subroutine compute_average:
    divide sum_score by score_count
    store result in avg_score
    return

subroutine display_min:
    print “minimum score:”
    print min_score
    return

subroutine display_max:
    print “maximum score:”
    print max_score
    return

subroutine display_avg:
    print “Average score:”
    print ave_score
    return
```

```
subroutine display_grade:
    if avg_score >= 90 then print “A”
    else if avg_score >= 80 then print “B”
    else if avg_score >= 70 then print “C”
    else if avg_score >= 60 then print “D”
    else print “F”
    retur
```