CIS11 Course Project Part 1: Documenting the Project

Fill in the following areas (purple).

Introduction

1.1 Purpose

The purpose of this program is to output 8 numbers that the user has input and output them from smallest to greatest. Meaning that the program will sort the numbers accordingly.

1.2 Intended Audience and Users

Any programmer looking to learn or make use of sorting algorithms or other rudimentary algorithms in assembly language.

1.3 Product Scope

What is the intention of this program?

The bubble sort program is designed to compare sets of numbers adjacent to one another in an array and, if necessary, re-order them so that the final order of the numbers is from the number with the lowest value to the number with the highest value..

1.4 Reference

Source Documents for the Program Requirements and Specification

Reference Project requirements and LC-3 specifications.

1.) ASCII Table

https://www.asciitable.com/

2.) LC-3 simulator

https://highered.mheducation.com/sites/0072467509/student_view0/lc-3_simulator.html

3.) What is bubble sort?

https://www.bbc.co.uk/bitesize/guides/z2m3b9q/revision/2#:~:text=A%20bubble%20sort%20algorithm%20goes,data%20is%20sorted%20into%20order

4.) Program references

https://github.com/oc-cs360/s2014/blob/master/lc3/bubblesort.asm

https://github.com/AlliyahMunir/LC-3-Project/blob/master/finalprogram.asm

https://github.com/AAWilcox/CIS11-Bubble-Sort/blob/master/BubbleSort.asm

https://github.com/lumix103/CIS-11-Final-Project/blob/master/main.asm

2. Overall Description

2.1 Product Perspective

Primary program objectives

- Successfully ask user to input 8 numbers ranging from 0-100
- Run without an error or looping more than needed
- Output numbers from the smallest value to the largest.

2.2 Product Functions

The overall description of functionality:

Highlight the program functionality: Identify tasks and subtasks of the program in summary.

1. Create an array of eight random numbers ranging from 0 to 100 in value.
2. First pass: compare array elements [0] and [1], or in other words, the first and second elements of the array. If the value of element [0] is greater than the value of element [1], swap the location of the elements. This is done using a temp value (described in technical functionality section). This comparison is done using subtraction of the second element from the first, where the locations of the first and second elements will be swapped if the difference between said elements is positive.
3. Second pass: compare array elements [1] and [2]. If the value of element [1] is greater than the value of element [2], swap the location of the elements. Repeat this step until the array has been completely sorted, or until the array has been checked for sorting error (if already sorted). The swap process involves a triangular swap using the two memory locations occupied by the data being swapped and a temporary memory location, used to store the first value until the second value has been swapped into the location of the first value.
4. Print the completed array using a loop to trace through all aray elements and output each element to the console.

Technical functionality

A configurable toolkit of functions including:

What are the technical functions of the program? Subroutines and operations.

JSR PROMPT_INPUT: Used to notify the user to supply numerical inputs in as variables

JSR BUBBLE_SORT: Enables the program to select each adjacent set of data within an array

JSR PUSH: Places one piece of data on top of the data stack

JSR POP: Removes one piece of data from the data stack

JSR PRINT_NUMBERS: Enables the program to display output to console

JSR SAVE_REG: Saves data from each register in the program necessary before service routines

JSR GET_INPUT: Retrieves the user-supplied numerical inputs prompted from the PROMPT_INPUT subroutine

HALT: Stops program runtime and execution

2.3 User Classes and Characteristics

Who are involved in this development process? Include business and technical personnel and their tasks.

Samantha Placito Melendrez

-Program documentation, programming, and flowchart.

Noah Curtis

- Program documentation and programming

Shawn Chacko

-program documentation and programming

2.4 Operating Environment

What type of system will the application be operated on? Operating system? System types? Development platform?

This program will be operated on LC-3 simulation and github.

2.5 Design and Implementation Constraints

Note any constraints or limitation to the application.

Access to a web text editor is required if the user is running the program on a MacOS or Chrome OS device.

Developer constraints: This program might end if two of the same value are input back to back.

2.6 Assumptions and Dependencies

Note any dependencies

It is assumed the user is familiar with an internet browser, a text editing software, and handling the keyboard and mouse. It is also assumed that the user is familiar with the LC3 simulator, editor, and output software's.

3. External Interface Requirements

3.1 User Interfaces

How will the user interface with your program? Menus? Access prompt? Links? Icons?

The user does not need to access anything other than a program that can successfully support .asm files. They will need to follow the prompt already provided within the program (which appears as an output message) which states that they would need to input 8 numbers within the given range.

3.2 Hardware Interfaces

Specify hardware interface – computer types? Terminal types?

Windows computer is strongly recommended (MAC OS can also be used, but it would take much more time for it to process the program compared to WINDOWS) along with a operational keyboard and mouse.

3.3 **Software Interfaces**

Specify additional software interface – if any. What type of software will the application require to run?

Both the LC3 Simulator application/website (Mac) and a text editor or LC3 Editor application (Windows) are required for the program to run.

3.4 Communications Interface

Does your application require web, Internet or network connectivity? If so, which browser? What type of network connection?

For MacOS it does require web access and a WiFi connection to run and build the LC-3 program

4. Detailed Description of Functional requirements

4.1 Type of Requirement (summarize from Section 2.2)

What are the functions? Their purposes? Inputs? Outputs? Data? Where is the data stored (internal or external to the application)?

Purpose: Sort numbers in ascending order.

Inputs: Inputs are through the keyboard (should input 8 numbers between the given range)

Processing: The input is run through the program, is looped, stored in the arrays, and then goes through different subroutines. It checks if the input value is within the range 0 to 100, if not it will release a message and if it is, it will continue to run through the program. Then all elements will be stored into one stack before it loops. During the loop more elements after pushed into the 2nd stack until it is smaller than the first stack. If the first stack is smaller than the second, they will be swapped.

Outputs: The 8 numbers input by the user will be output in ascending order.

Data: Internal sorting algorithm (internal data).

Memory allocation, arrays

Subroutines used (& their purposes):

- *JSR PROMPT_INPUT: Used to notify the user to supply numerical inputs in as variables*
- *JSR BUBBLE_SORT: Enables the program to select each adjacent set of data within an array*
- *JSR PUSH: Places one piece of data on top of the data stack*
- *JSR POP: Removes one piece of data from the data stack*
- *JSR PRINT_NUMBERS: Enables the program to display output to console*
- *JSR SAVE_REG: Saves data from each register in the program necessary before service routines*
- *JSR GET_INPUT: Retrieves the user-supplied numerical inputs prompted from the PROMPT_INPUT subroutine*
- *HALT: Stops program runtime and execution*

4.2 Performance requirements

What is the expected performance level of the program?

The program is expected to successfully output the message "Enter a number from within the range 0-100:" 8 times and store the input values into its selected arrays/registers. It should also be able to run the input through the program, meaning that it should successfully loop until the result comes out in ascending order. It is also expected of it to take in any input outside of the stated range and output a message.

4.3 Flow Chart and Pseudocode.

PSEUDOCODE:

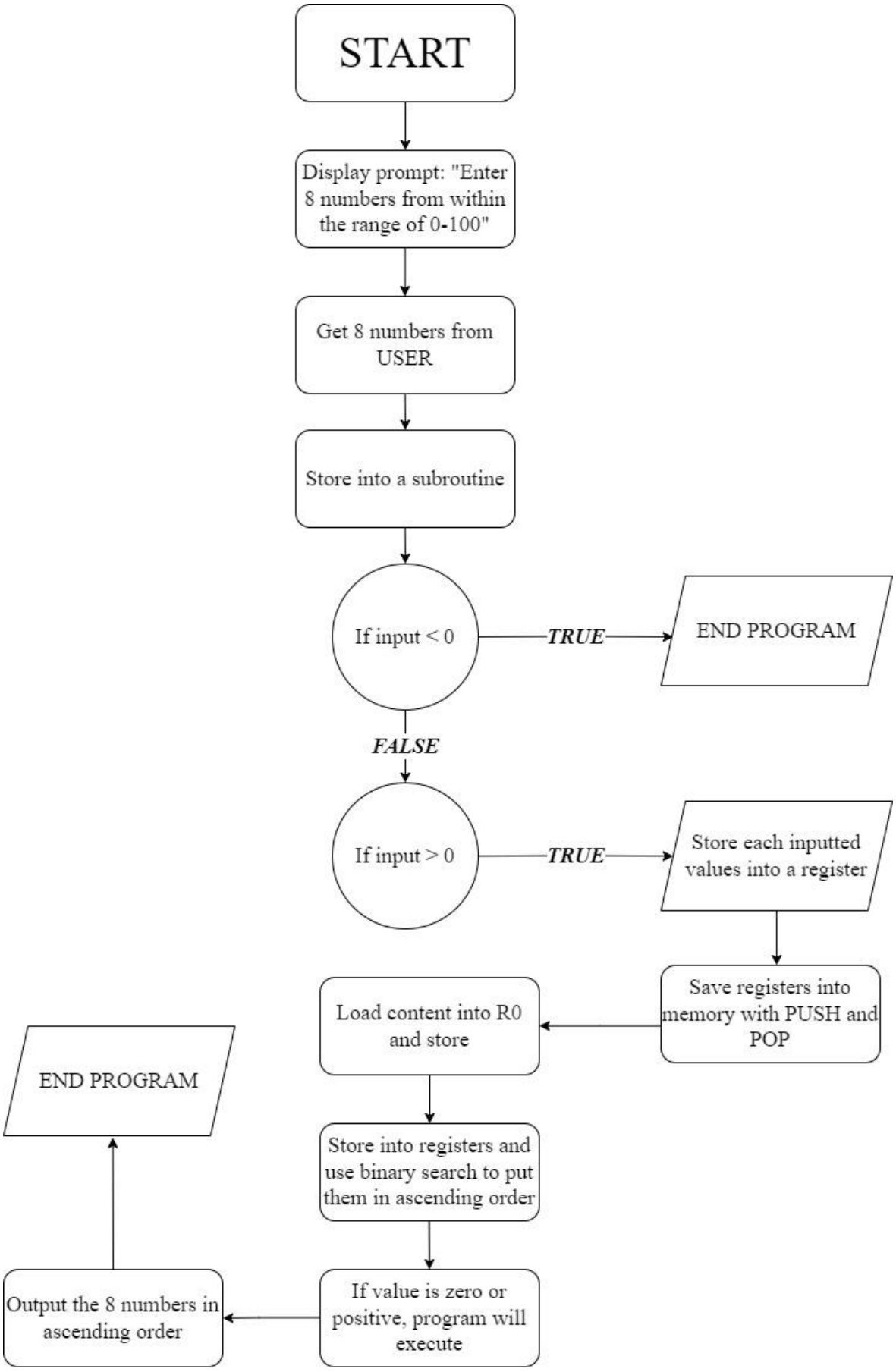
1. Origination Address; save registers initialized
2. Program should be able to display the prompt "Input 8 numbers within the range of 0-100" by using PUTS
3. Loop used for input validation
4. Store the input values into its respected registers
    4.1. Clear each register first so that the values are stored within them
5. Store each register's contents into its respected subroutine
6. Program should be able to differentiate the input while it loops them. Meaning that if a number is less than 0 or bigger than 100 the program should exit.

6.1. If the input is within the range the program should be able to continue (push value onto stack if it fits within range)
6.2. Input HEX 48 to create the range (ASCII conversion before displayed on console)

7. Break input validation loop (w/ PUSH function)
8. POP loop
9. Use 2s' compliment to be able negate 100 into its respected register
10. Reinitialize certain register that contains the inputs
11. ASCII offset (allows machine to use these values properly; must be re-converted at end of program)
12. Load HEX values
13. Clear registers
14. Break POP loop
15. Load registers
16. Program sorts the input in ascending order
    16.1. Initialize register
    16.2. Loop
    16.3. Increment counter
    16.4. Counter loop
    16.5. Store values into registers
    16.6. Initialize inner and outer loop
    16.7. Get input values
    16.8. Negate (for comparison of two adjacent values), swap (using triangular swapping method), and store
    16.9. End inner and out loop
    16.10. Halt
17. Fill every listed subroutine's data in their respected storage locations
18. Output Loop
19. Print numbers
    19.1. Clear registers
    19.2. Print numbers to array
    19.3. Pointer to array and offset
    19.4. ASCII offset
    19.5. Counter
    19.6. Return
20. Break Loop
21. HALT execution

22. End program
    1. Reinitialize certain register that contains the inputs
1. ASCII offset (allows machine to use these values properly; must be re-converted at end of program)
1. Load HEX values
1. Clear registers
1. Break POP loop
1. Load registers
1. Program sorts the input in ascending order
    1. Initialize register
    1. Loop
    1. Increment counter
    1. Counter loop
    1. Store values into registers
    1. Initialize inner and outer loop
    1. Get input values
    1. Negate (for comparison of two adjacent values), swap (using triangular swapping method), and store
    1. End inner and out loop
    1. Halt
1. Fill every listed subroutine's data in their respected storage locations
1. Output Loop
1. Print numbers
    1. Clear registers
    1. Print numbers to array
    1. Pointer to array and offset
    1. ASCII offset
    1. Counter
    2. Return
1. Break Loop
1. HALT execution
1. End program

FLOWCHART


START
Display prompt: "Enter 8 numbers from within the range of 0-100"
Get 8 numbers from USER
Store into a subroutine
If input < 0
TRUE
END PROGRAM
FALSE
If input > 0
TRUE
Store each inputted values into a register
Save registers into memory with PUSH and POP
Load content into R0 and store
Store into registers and use binary search to put them in ascending order
If value is zero or positive, program will execute
Output the 8 numbers in ascending order
END PROGRAM