CIS-11 Project Documentation Template

Construction Crew

Mario Ruiz, Vincent Rosales, Domingo Montoya

Case 1 (Bubble Sort)

May 18, 2024

Advisor: Kasey Nguyen, PhD

Overall Description:

This project aims to develop an LC-3 assembly language program that implements the Bubble Sort algorithm to sort a list of eight user-input numbers in ascending order. Bubble Sort is a straightforward sorting technique where adjacent elements are repeatedly compared and swapped if they are in the wrong order, ensuring that the largest values "bubble" to the end of the list with each pass. The program will prompt the user to enter eight numbers, store these numbers in an array, sort the array using Bubble Sort, and then display the sorted numbers on the console. The program will include features such as memory management, stack operations, subroutine calls, ASCII conversion, and appropriate use of LC-3 instructions for arithmetic, data movement, and conditional operations. The project will also include comprehensive testing to verify that the program meets all specified requirements and functions correctly.

Purpose:

The purpose of this program is to implement and demonstrate the Bubble Sort algorithm on the LC-3 simulator. Bubble Sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted. The algorithm gets its name because smaller elements "bubble" to the top of the list while larger elements "sink" to the bottom, much like bubbles rising in water. This LC-3 program is designed to accept 8 numerical inputs from the user, each within the range of 0 to 100. These numbers are then stored in an array. The program will sort this array in ascending order using the Bubble Sort algorithm and subsequently display the sorted numbers on the console.

Functionality/Technical Requirements:

Overall function

1. **Input Routine**: The program will prompt the user to enter 8 numbers. Each number will be converted from its ASCII representation to an integer and stored in an array.

2. **Bubble Sort Algorithm**: The sorting process involves multiple iterations over the array, where each element is compared with the next one, and they are swapped if they are out of order. This process is repeated until the array is sorted.

3. **Output Routine**: Once the array is sorted, the numbers will be converted back to their ASCII representations and displayed on the console in ascending order.

Applications:

While Bubble Sort is not commonly used in real-world business applications because of its inefficiency compared to other sorting algorithms, it has several niche uses. It serves as an educational tool in programming courses and training sessions to teach basic sorting concepts and algorithmic thinking as seen in many of the data structure classes in Moreno Valley. In other scenarios involving small data sets, quick prototyping, or legacy systems, Bubble Sort can be sufficient due to its simplicity. It may also be used in business simulation games and interactive learning modules for educational purposes. Additionally, Bubble Sort can be utilized for simple administrative tasks, small automation scripts, and benchmarking to compare the efficiency of different sorting algorithms. Despite its limitations, Bubble Sort's straightforward implementation makes it valuable in specific contexts.

Technical breakdown:

Input Handling

- The application will prompt the user to enter exactly eight numbers.
- The numbers entered must be within the range of 0 to 100.
- The input will be accepted as ASCII characters and converted to their integer equivalents.
- The application will store these integers in an array for processing.

Bubble Sort Algorithm

- The application will implement the Bubble Sort algorithm to sort the array of eight integers.
- It will repeatedly iterate through the array, comparing each pair of adjacent elements.
- If a pair of elements is found in the wrong order (i.e., the first element is greater than the second), the elements will be swapped.
- This process will continue until the array is sorted in ascending order.

Output Handling

- Once the array is sorted, the application will convert each integer back to its ASCII representation.
- The sorted values will be displayed on the console in ascending order.

Memory and Storage Management

- The application will utilize specific memory addresses for origination, fill, array storage, input handling, and output handling.
- It will efficiently manage storage allocation and handle any potential overflow issues.

Stack Operations

- The application will manage the stack using PUSH and POP operations to store and retrieve data during subroutine execution.
- It will include save-restore operations to preserve the state of the program during subroutine calls.

Subroutines

- The application will be modular, comprising at least two subroutines: one for input handling and another for sorting the array.
- Additional subroutines may be implemented as needed for specific tasks such as swapping elements and handling output.

Control Flow

- The application will use conditional and iterative branching to control the flow of execution.
- It will ensure that the sorting process correctly iterates through the array until it is fully sorted.

System Call Directives

- The application will use appropriate system call directives to manage input and output operations.

ASCII Conversion

- The application will include operations to convert between ASCII characters and their integer equivalents for both input and output.

Testing

- The application will be tested using specified values to ensure correctness and reliability.
- Testing will involve entering various sets of eight numbers to verify that the sorting algorithm functions correctly and the sorted values are displayed accurately.

Pseudocode / Flowchart

Pseudocode

Main Program

1. **Initialize**
    - Set up stack pointer
    - Set up memory locations for array
2. **Input Routine**
    - Prompt user for 8 numbers
    - Convert ASCII input to integer
    - Store integers in array
3. **Bubble Sort Routine**
    - Iterate through array
    - Compare adjacent elements
    - Swap if necessary
    - Repeat until array is sorted
4. **Output Routine**
    - Convert integers to ASCII
    - Display sorted numbers on console
5. **End Program**

Subroutines

1. **Input Subroutine**
    - Get user input
    - ASCII to integer conversion
    - Store in array
2. **Output Subroutine**
    - Convert integer to ASCII
    - Output to console
3. **Swap Subroutine**
    - Swap two elements in the array

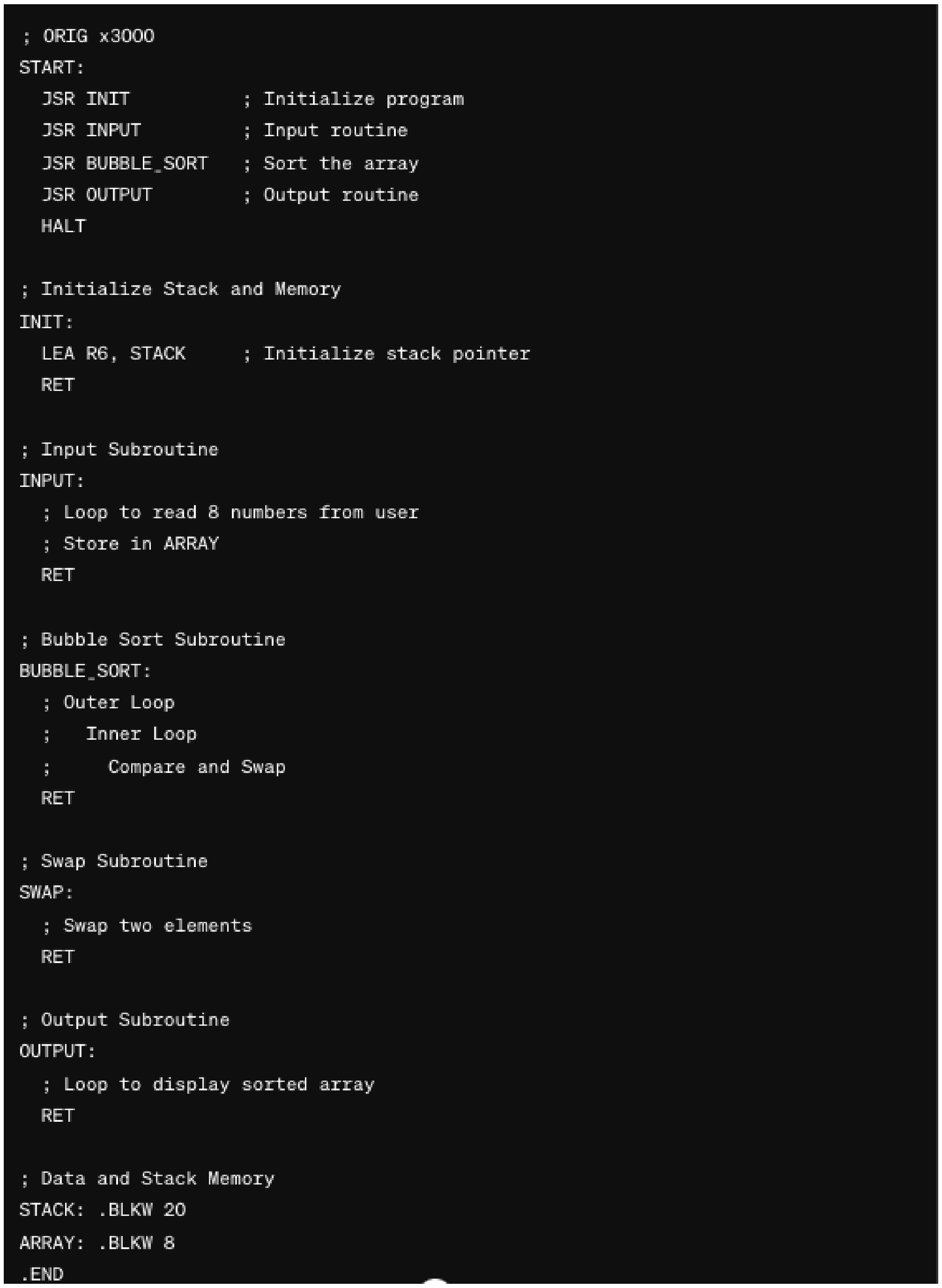
Flowchart

1. **Start**
2. **Initialize Memory and Stack**

3. **Input Subroutine**
    - Loop to get 8 numbers
    - Store in array
4. **Bubble Sort Subroutine**
    - Outer loop (N-1 times)
        - Inner loop (N-1 times)
            - Compare and Swap
5. **Output Subroutine**
    - Loop to convert and display sorted array
6. **End**

Prototype of code/ comments:

- The program includes placeholders for key sections, such as initialization, input, sorting, and output.
- Specific LC-3 instructions for arithmetic operations, data movement, conditional branching, and other requirements will be filled in greater detail as per the LC-3 instruction set when project is finalized



```
; ORIG x3000
START:
  JSR INIT          ; Initialize program
  JSR INPUT         ; Input routine
  JSR BUBBLE_SORT   ; Sort the array
  JSR OUTPUT        ; Output routine
  HALT

; Initialize Stack and Memory
INIT:
  LEA R6, STACK     ; Initialize stack pointer
  RET

; Input Subroutine
INPUT:
  ; Loop to read 8 numbers from user
  ; Store in ARRAY
  RET

; Bubble Sort Subroutine
BUBBLE_SORT:
  ; Outer Loop
  ;   Inner Loop
  ;     Compare and Swap
  RET

; Swap Subroutine
SWAP:
  ; Swap two elements
  RET

; Output Subroutine
OUTPUT:
  ; Loop to display sorted array
  RET

; Data and Stack Memory
STACK: .BLKW 20
ARRAY: .BLKW 8
.END
```

Statement of Functionality

In this section you state **precisely** *what the application will do.*

This part, more than anything else in the requirements document, spells out your contract with your customers. The application will ***include all functions listed here and will not include any of the functions not listed****.*

In this section you must use as precise language as you can since the developers will use it to code the application. When reviewing this part with other people you should pay extreme attention to removing any possibility for ambiguous interpretation of any of the requirements.

If your application has several distinct categories of users, you can list the requirements by user category. User categories may be defined in terms of their job title (clerk, manager, administrator), the frequency with which they will use the system (heavy or casual), the purpose for which they will use the system (operational decisions, long-term decisions), etc. If each category of users uses the system in its own way, structuring the requirements based on user category will make sense.

If your application deals with several kinds of real-world objects, you can list the requirements by object. For example, for a reservation system a booking is an important object, and you may want to list all requirements pertaining to bookings in one sub-section.

One of the most common approaches is to list the requirements by feature. For example, features of a word processing application are file management, formatting, editing, etc.

Scope

In this section you state what functionality will be delivered and in which phase.

You should include this section if your development consists of multiple phases. As an alternative to this section, you can note the planned project phase for each feature in the functionality statement section. Usually, it is better to include a separate scope section for easy reference and communication.

Performance

In this section you describe any specific performance requirements.

You should be very specific and use numeric measures of performance. Stating that the application should open files quickly is not a performance requirement since it is ambiguous and cannot be verified. Stating that opening a file should take less than 3 seconds for 90% of the files and less than 10 seconds for every file is a requirement.

Instead of providing a special section on performance requirements, you may include the relevant information for each feature in the statement of functionality.

Usability

In this section you describe any specific usability requirements.

You need to include this section only if there are any "overarching" usability goals and considerations. For example, the speed of navigation of the UI may be such a goal. As in the previous section, use numeric measures of usability whenever possible.

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	Reques ted 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.

Program Description

Purpose:
The purpose of this program is to implement the Bubble Sort algorithm on the LC-3 (Little Computer 3) simulator to sort an array of 8 user-input numbers (ranging from 0 to 100) in ascending order.

Functionality:
1. Accepts 8 numerical inputs from the user.
2. Stores these numbers in an array.
3. Sorts the array using the Bubble Sort algorithm.
4. Displays the sorted numbers on the console.

Technical Requirements:
1. **Addresses:**
   - Origination
   - Fill
   - Array
   - Input
   - Output

2. **Output:**
   - Sorted values displayed in the console.

3. **Labels and Comments:**
   - Appropriately used throughout the program.

4. **Instructions:**
   - Arithmetic operations
   - Data movement
   - Conditional operations

5. **Subroutines:**
   - Contains at least 2 subroutines with subroutine calls.

6. **Branching:**
   - Utilizes conditional and iterative branching.

7. **Overflow Management:**
   - Manages overflow and storage allocation.

8. **Stack Management:**
   - Includes PUSH-POP operations on the stack.

9. **Save-Restore Operations:**
   - Implements save-restore operations for subroutines.

10. **Pointer:**
    - Uses pointers for array and data handling.

11. **ASCII Conversion:**
    - Includes operations to handle ASCII conversions for input/output.

12. **System Call Directives:**
    - Uses appropriate system call directives.

13. **Testing:**
    - Tests the program using specified values.

Pseudocode / Flowchart

Pseudocode

Main Program
1. **Initialize**
   - Set up stack pointer
   - Set up memory locations for array

2. **Input Routine**
   - Prompt user for 8 numbers
   - Convert ASCII input to integer
   - Store integers in array

3. **Bubble Sort Routine**
   - Iterate through array
   - Compare adjacent elements
   - Swap if necessary
   - Repeat until array is sorted

4. **Output Routine**
   - Convert integers to ASCII
   - Display sorted numbers on console

5. **End Program**

Subroutines
1. **Input Subroutine**
   - Get user input
   - ASCII to integer conversion
   - Store in array

2. **Output Subroutine**
   - Convert integer to ASCII
   - Output to console

3. **Swap Subroutine**
   - Swap two elements in the array

Flowchart
1. **Start**
2. **Initialize Memory and Stack**
3. **Input Subroutine**

  - Loop to get 8 numbers
  - Store in array
4. **Bubble Sort Subroutine**
  - Outer loop (N-1 times)
    - Inner loop (N-1 times)
      - Compare and Swap
5. **Output Subroutine**
  - Loop to convert and display sorted array
6. **End**

Sample LC-3 Code (Pseudocode style)

```assembly
; ORIG x3000
START:
  JSR INIT          ; Initialize program
  JSR INPUT         ; Input routine
  JSR BUBBLE_SORT   ; Sort the array
  JSR OUTPUT        ; Output routine
  HALT

; Initialize Stack and Memory
INIT:
  LEA R6, STACK     ; Initialize stack pointer
  RET

; Input Subroutine
INPUT:
  ; Loop to read 8 numbers from user
  ; Store in ARRAY
  RET

; Bubble Sort Subroutine
BUBBLE_SORT:
  ; Outer Loop
  ;   Inner Loop
  ;     Compare and Swap
  RET

; Swap Subroutine
SWAP:
  ; Swap two elements
  RET

; Output Subroutine
OUTPUT:
  ; Loop to display sorted array
  RET

; Data and Stack Memory
STACK: .BLKW 20
ARRAY: .BLKW 8
.END
```

Comments
- The program includes placeholders for key sections, such as initialization, input, sorting, and output.
- Specific LC-3 instructions for arithmetic operations, data movement, conditional branching, and other requirements should be filled in as per the LC-3 instruction set.

This template provides a structured outline to develop the LC-3 Bubble Sort program with clear pseudocode and flowchart components, ensuring all technical requirements are met.