CIS11 Course Project Part 1: Documenting the Project

1. Introduction

1.1 Purpose

The purpose of this program is to perform a bubble sort to sort an array of eight numbers in ascending order. The requirements are that the program:

1. Contain appropriate addresses: origination, fill, array, input and output.
2. Display sorted values in console.
3. Use appropriate labels and comments.
4. Contain appropriate instructions for arithmetic, data movement and conditional operations.
5. Comprise of 2 or more subroutines and implement subroutine calls.
6. Use branching for control: conditional and iterative.
7. Manage overflow and storage allocation.
8. Manage stack: include PUSH-POP operation on stack.
9. Include save-restore operations.
10. Include pointer.
11. Implement ASCII conversion operations.
12. Use appropriate system call directives.
13. Testing.

1.2 Intended Audience and Users

Users familiar with LC-3 Simulate.

1.3 Product Scope

The bubble sort program is intended to sort an array of eight numbers input by the user in ascending order from left to right using a series of comparative operators. The program is intended to only sort numbers from 0 - 999.

1.4 Reference

Source Documents for the Program Requirements and Specification

1.) LC-3 Manual

http://faculty.lasierra.edu/~ehwang/courses/cptg245/LC3/LC-3%20Instructions.pdf

2.) LC-3 Guide

https://www.cis.upenn.edu/~milom/cse240-Fall05/handouts/lc3guide.html

3.) Bubble Sort Algorithm Explanation Video

https://www.youtube.com/watch?v=Jdtq5uKz-w4

4.) Bubble Sort in C++

https://www.youtube.com/watch?v=IFJi_P68iKk

5.) Bubble Sort LC-3 Template

https://stackoverflow.com/questions/43735625/how-to-make-a-sorting-algorithm-in-assembly-code-in-lc3

6.) Looping Input Data Example

http://lc3assembly.blogspot.com/2013/12/now-create-array-of-10-values-starting.html

7.) Inputting Multi-Digit Long Numbers Inspiration

https://stackoverflow.com/questions/43641736/how-do-i-display-double-digit-numbers-in-lc3#

8.) Multiplication in LC-3

http://www.lc3help.com/tutorials.htm?article=Multiplication/

Companion Application Requirements Documents (If applicable)

Documents:

1.) CIS11_Course_Project_Part1_FINAL-1.docx
2.) LC-3 Instructions.pdf

Companion Applications:

1.) Java 1.4 or newer
2.) Windows Operating System

2. Overall Description

2.1 Product Perspective

The program includes:

- Input from the user
- Sort subroutine which resets/sets the pointer and counter then sends the array through inner and outer loops for the bubble sort
- Outer For-Loop to run through the list of numbers multiple times
- Inner For-Loop to run through the list of numbers once swapping values where applicable
- Swapped label where If the values are already swapped, then the counters and pointers are incremented/decremented.
- Sorted label which once the array is sorted returns it from the subroutine
- Output subroutine to display values to the console

2.2 Product Functions

The overall description of functionality:

1. Input user data to create an array of eight numbers through a looped subroutine.
2. A sort subroutine which comprises the primary functions of the program. The initial pointer location is set then the array is run through a series of for-loops repeated through the entire array via the swapped label and when complete returns to main via the sorted label.
3. Outer for-loop that runs through the array multiple times (the inner for-loop) until the array is sorted in ascending order.
4. Inner for-loop that runs through the array of numbers once, swapping values as necessary.
5. A swapped label that increments the pointer and decrements the counters for the outer for-loop subroutine and inner for-loop subroutine.
6. A sorted label is used to return the sorted array back from the subroutine to be output.
7. An overflow subroutine that verifies that the numbers input are in the range of 0 - 999, preventing overflow within the program.
8. Output the sorted values to the console using a looped subroutine, outputting one value at a time.

Technical functionality

- Several for-loops that run through a subroutine for a set number of times.
- Several conditional branches that help control the flow of the program.

2.3 User Classes and Characteristics

Facility:

Moreno Valley College (MVC)

16130 Lasselle Street

Moreno Valley, CA 92551

(951) 571-6100

Professor/Instructor:

Professor Kasey Nguyen, PhD.

Department: Computer Information Systems

Students:

Kyle Janosky

Alyssa Wilcox

2.4 Operating Environment

LC-3 is developed for use on Windows or Unix systems.
LC-3 Edit and LC-3 Simulate are specifically designed to edit and run the assembly program.

2.5 Design and Implementation Constraints

Constraints include:

- Due to how LC-3 functions, user input is not very friendly. To input multi-digit number, the program has to handle the digits individually, and therefore the digits have to be input one at a time on a new line. This can become confusing when inputting small numbers such as 3, because the number has to be input as 003.
- The program is hardcoded to only sort eight numbers.
- The program can only sort numbers ranging from 0 to 999.

2.6 Assumptions and Dependencies

The program needs LC-3 Simulate to run and LC-3 Edit to view and edit the code.

3. External Interface Requirements

3.1 User Interfaces

The user is required to use LC-3 Simulate as well as the object file (.obj) and assembler file (.asm) associated with this program in order to interact with it. Within LC-3 Simulate, the user see both menus and icons to start the program, and will then be prompted with a console to input data through a keyboard.

3.2 Hardware Interfaces

Windows PC or Unix.

Input and output (I/O) devices such as a keyboard, mouse, and monitor.

3.3 Software Interfaces

LC-3 Edit and LC-3 Simulate.

3.4 Communications Interface

Not required for this program.

4. Detailed Description of Functional requirements

4.1 Type of Requirement (summarize from Section 2.2)

Initialization and Program Flow:

- Purpose: Used to set up the correct values for the counters and establish program flow between the main calling program and the subroutines
- Inputs: None
- Processing:
    - Set the counters
    - Load the pointer
    - Jump to the input subroutine
    - Reset the pointer and counter values
    - Jump to the outer for-loop sorter
    - Jump to the inner for-loop sorter
    - Jump to the output loop
    - Pause the program
- Outputs: None
- Data: Values within registers

Input User Data:

- Purpose: Take in user input of eight numbers and store them in an array.
- Inputs: A two digit number. Digits are inputted separately
- Processing:
    - Create the input loop subroutine
    - Load the input prompt and display on console
    - Input the first digit
    - Convert to ASCII character to hex value
    - Multiply first digit by 100
    - Input second digit
    - Convert to ASCII character to hex value
    - Multiply second digit by 10
    - Input third digit
    - Convert to ASCII character to hex value
    - Add first, second, and third digit together to get real inputted number
    - Store value using the pointer
    - Increment pointer
    - Decrement loop counter
    - Branch back to the input loop for as long as the counter is positive
    - Return to calling program
- Outputs: Input user prompt
- Data: Input numbers stored in locations x4000 - x4007

Data Section:

- Purpose: Used to establish the prompt strings, ASCII offset values, pointer starting location, and counter values
- Inputs: None
- Processing:
    - Set up input prompt

    - Set up output prompt
    - Set up space for output
    - Set up negative ASCII offset
    - Set up positive ASCII offset
    - Set multiple of ten for second digit
    - Set multiple of hundred for third digit
    - Set up pointer starting location
    - Set up counter value
    - Set up space for first digit
    - Set up space for second digit
    - Set up space for third digit
- Outputs: None
- Data: Labels

Sort Subroutine:

- Purpose: Set/Reset the pointer value in reference to the array then send the aforementioned through the Outer For-Loop Sort and Inner For-Loop Sort a process repeated by the Swapped label until the array has been sorted in ascending order at which point it will be returned to the calling program via the Sorted label.
- Inputs: None
- Processing:
    - Reset pointer location to start of array
    - Set outer for-loop sorter counter
    - Set inner for-loop sorter counter
    - Outer for-loop sorter documented below
    - Inner for-loop sorter documented below
    - Swapped label documented below
    - Sorted label documented below
- Outputs: None
- Data: Counter and pointer values held in registers

Outer For-Loop Sorter:

- Purpose: This causes the program to loop through the inner for-loop sorter multiple times. In other words, this causes the program to run through the array of numbers multiple times.
- Inputs: None
- Processing:
    - Create the outer for-loop sorter
    - Decrement the outer for-loop sorter counter
    - If the counter is negative or zero, return to the calling program
    - Copy the outer counter to the inner counter to keep both counters synchronized
    - Set the pointer to the beginning of the array
    - Return to the calling program
- Outputs: None
- Data: Counter and pointer values held in registers

Inner For-Loop Sorter:

- Purpose: This loop runs through the array of numbers once, swapping values as necessary
- Inputs: None
- Processing:
    - Create the inner for-loop sorter
    - Load the first value using the pointer
    - Load the second value using the pointer
    - Negate the second value
    - Subtract the negated second value from the first
    - If the result is negative or zero, move to the swapped subroutine
    - Perform the swap
    - Return to the calling program
- Outputs: None
- Data: Numbers from memory addresses x4000 - x4007 loaded onto registers

Swapped Label:

- Purpose: Once the values have been swapped or are already in ascending order, increment/decrement the counters and pointer
- Inputs: None
- Processing:
    - Create the swapped label
    - Increment the pointer
    - Decrement the inner for-loop sorter counter
    - If the counter is positive, go back to the inner for-loop sorter
    - If the counter is positive or zero, go back to the outer for-loop sorter
- Outputs: None
- Data: Counter and pointer values held in registers

Sorted Label:

- Purpose: Once the entire array has been sorted in ascending order return the sorted array to calling program.
- Inputs: None
- Processing:
    - Create sorted label.
    - Return to calling program from the subroutine.
- Outputs: None
- Data: Uses register seven in a jump to the calling program

Output to Console:

- Purpose: Output the sorted values to the console
- Inputs: None
- Processing:
    - Create the output subroutine

    - Load and display the output prompt
    - Reset the pointer value
    - Reset the counter value
    - Create the output loop
    - Load a value using the pointer
    - Break up the value into individual digits
    - Convert value from hex to ASCII character using ASCII offset
    - Output the value to console
    - Increment the pointer
    - Decrement the counter
    - Branch back to the output loop for as long as the counter is positive
    - Return to calling program
- Outputs: User prompt and the sorted list of numbers
- Data: Numbers from memory addresses x4000 - x4007 loaded onto registers

4.2 Performance requirements

4.2.1 The program should enable the user to input an array of eight values ranging from 0 - 999, and the program will then sort these values, left to right, in ascending order.

4.2.2 The program should be accessible to users of LC-3 Simulate.

4.2.3 Since the program is bound to only sort eight individual values, the response time for a particular set of input should not be greater than a few seconds.

4.2.4 Overflow error handling should be implemented to prevent the program from attempting to handle a number that is too large for it.

4.3 Pseudocode and Flowchart

Bubble Sort Pseudocode

Starting Location:
Origination address of program
- Originate at x3000

Initialize Registers:
Use designated registers for specific tasks (these are just comments, explaining the main use of these registers throughout the program)
- R0 as the first item to be manipulated
- R1 as the second item to be manipulated
- R2 as a temporary work variable
- R3 as our pointer, used to access different elements of the array
- R4 as the Outer For-Loop Counter
- R5 as the Inner For-Loop Counter
- R6 as input/output loop counter

Initialization and Program Flow:

Set up the correct value for the counters. Establish program flow between the main calling program and the subroutines

- Initialize the pointer
  - LD R3, PT
- Initialize the I/O counter
  - LD R6, COUNT
- Input Loop
  - See input loop below
- Jump to the sort subroutine
  - JSR SORT
- Jump to the output subroutine
  - JSR OUTPUT_LOOP
- Pause the program
  - HALT

Input User Data:

Creating out array of 8 numbers through our subroutine INPUT_LOOP.

To input multi-digit numbers, we have to handle one digit at a time. To work around the limitations of LC-3's IN (TRAP x23), handle the number like so:

The number 143 can be rewritten as: $(1 \times 10^2) + (4 \times 10^1) + (3 \times 10^0) = 100 + 40 + 3 = 143$. Handle each character separately, then add at the end to get the real number.

- Display the prompt to input the characters one at a time
  - LEA, R0, INPROMPT
  - PUTS
- Load the label SPACE that causes the console to go to the next line, clear R0 first
  - AND R0, R0, #0
  - LD R0, SPACE
- Output to console
  - OUT
- Load the example prompt, clear R0 first
  - AND R0, R0, #0
  - LEA R0, EXAMPLE
- Display on console
  - PUTS
- Begin the input loop using the label INPUT
  - INPUT
- Input the first digit. This digit is the number in the 100s place, so we will need to multiply it by 100
  - IN
- Load 100 into R5 using the label HUNDRED, clear R5 first
  - AND R5, R5, #0
  - LD R5, HUNDRED
- Load the ASCII offset into our temporary work variable (R2), first clearing out R2
  - AND R2, R2, #0
  - LD R2, ASCIINEG
- Add ASCII offset to the first digit
  - ADD R0, R0, R2

- Clear our work variable and move the character to it
    - AND R2, R2, #0
    - ADD R2, R0, #0
- Clear R0 to use it for multiplication
    - AND R0, R0, #0
- Enter the multiplication loop, distinguished by the label FIRST_DIGIT
    - FIRST_DIGIT
- Add the first digit to itself, using R0 and R2 (R0 = R2 at the beginning), storing in R0
    - ADD R0, R0, R2
- Decrement the counter
    - ADD R5, R5, #-1
- Condition check. If the counter is positive, continue multiplying. This multiplies the first digit by 100
    - BRp FIRST_DIGIT
- Once multiplied, move R0 to R1. R1 now contains the first digit
    - ADD R1, R0, #0
- Clear R0 for use of the second digit
    - AND R0, R0, #0
- Input the first digit. This digit is the number in the 10s place, so we will need to multiply it by 10
    - IN
- Load 10 into R5 using the label TEN, clear R5 first
    - AND R5, R5, #0
    - LD R5, TEN
- Load the ASCII offset into our temporary work variable (R2), first clearing out R2
    - AND R2, R2, #0
    - LD R2, ASCIINEG
- Add ASCII offset to the second digit
    - ADD R0, R0, R2
- Clear our work variable and move the character to it
    - AND R2, R2, #0
    - ADD R2, R0, #0
- Clear R0 to use it for multiplication
    - AND R0, R0, #0
- Enter the multiplication loop, distinguished by the label SECOND_DIGIT
    - SECOND_DIGIT
- Add the second digit to itself, using R0 and R2 (R0 = R2 at the beginning), storing in R0
    - ADD R0, R0, R2
- Decrement the counter
    - ADD R5, R5, #-1
- Condition check. If the counter is positive, continue multiplying. This multiplies the second digit 10
    - BRp SECOND_DIGIT
- Copy R0 to R4. F4 now contains the second digit
    - ADD R4, R0, #0
- Clear R0
    - AND R0, R0, #0
- Input the third digit. This number is in the 1s place, so we do not need to multiply it
    - IN

- Load the ASCII offset into our temporary work variable (R2), first clearing out R2
    - AND R2, R2, #0
    - LD R2, ASCIINEG
- Add the ASCII offset to the third digit
    - ADD R0, R0, R2
- Clear R2
    - AND R2, R2, #0
- Now get the real number by adding the first digit (R1) to the second digit (R4), then add their sum to the third digit (R0). Store in R2
    - ADD R2, R1, R4
    - ADD R2, R0, R2
- Store the result from R2 into the array using the pointer
    - STR R2, R3, #0
- Increment the pointer so that it points to the next location to store
    - ADD R3, R3, #1
- Decrement the INPUT loop counter
    - ADD R6, R6, #-1
- Condition check. If the INPUT loop counter is positive, continue looping. The loop will loop eight times
    - BRp INPUT

Data Section:

Used to establish our prompt strings, ASCII offset values, pointer starting location, and counter values.

- Prompt to input the values, using the label INPROMPT
    - INPROMPT .STRINGZ “Please input 8 numbers (0 - 100) to be sorted:”
- Prompt that gives an example of the format used to input the numbers
    - EXAMPLE .STRINGZ “Example: Input 3 as 003”
- Prompt to display the output values, using the label OUTPROMPT
    - OUTPROMPT .STRINGZ “The sorted list: “
- Spaces the output on the console by causing it to jump to the next line
    - SPACE .FILL x000A
- Negative ASCII offset, which is -48 decimal, xFFD0 hex. Used when inputting values
    - ASCIINEG .FILL xFFD0
- Positive ASCII offset, which is 48 decimal, x30 hex. Used when outputting values
    - ASCIIPOS .FILL x0030
- To multiply by 100, which is x64 in hex
    - HUNDRED .FILL x0064
- To multiply by 10, which is x000A in hex
    - TEN .FILL x000A
- Array starting point. Start at location x4000
    - PT .FILL x4000
- Counter values to deal with eight numbers
    - COUNT .FILL #8
- Set up the storage location for the first output digit
    - DIGIT1 .FILL x400A
- Set up the storage location for the second output digit
    - DIGIT2 .FILL x400B
- Set up the storage location for the third output digit

- DIGIT3 .FILL x400C

Sort Subroutine:

This subroutine resets/sets the pointer and counter values and contains the outer for-loop sorter, the inner for-loop sorter, swapped label, and sorted label.

- Reset the pointer, clearing R3 first
  - AND R3, R3, #0
  - LD R3, PT
- Set the outer for-loop sorter counter, clearing R4 first
  - AND R4, R4, #0
  - LD R4, COUNT
- Set the inner for-loop sorter counter, clearing R5 first
  - AND R5, R5, #0
  - LD R5, COUNT
- Outer for-loop sorter counter
  - See below
- Inner for-loop sorter counter
  - See below
- Swapped label
  - See below
- Sorted label
  - See below
- Return to calling program
  - RET (JMP R7)

Outer For-Loop Sorter:

This causes the program to loop through the inner for-loop sorter multiple times (running through the array multiple times) until the array is sorted.

- Create the outer for-loop sorter, with the label OUTER_LOOP
  - OUTER_LOOP
- Decrement the outer for-loop sorter counter
  - ADD R4, R4, #-1
- Condition check, If the outer for-loop sorter counter is negative or zero, then the array is sorted (run through 8 times) and branch to the SORTED label
  - BRnz SORTED
- Copy the outer for-loop sorter counter to the inner for-loop sorter counter to keep both synchronized
  - ADD R5, R4, #0
- Set the pointer to the beginning of the array
  - LD R3, PT

Inner For-Loop Sorter:

This loop runs through the array of numbers once, swapping values as necessary.

- Create the inner for-loop sorter, with the label INNER_LOOP
  - INNER_LOOP
- Load the first value using the pointer (R3), store in R0
  - LDR R0, R3, #0
- Load the second value using the pointer (R3), store in R1

    - LDR R1, R3, #1
- Clear R2
    - AND R2, R2, #0
- Use two's complement to negate the second value. Store in R2
    - NOT R2, R1
    - ADD R2, R2, #1
- Subtract the negated second value from the first value, store in R2
    - ADD R2, R0, R2
- Condition check. If R2 (R0 - R1; first minus second) is negative or zero, that means the first item is smaller or the same as the second. No need to swap, so branch to SWAPPED subroutine
    - BRnz SWAPPED
- Perform the swap:
    - STR R1, R3, #0 ;Second item is now stored in the first slot
    - STR R0, R3, #1 ;First item now stored in the second slot

Swapped Label:

If the values are already in ascending order, increment/decrement the counters and pointers.

- Create the swapped label
    - SWAPPED
- Increment the pointer (R3) to look at the next set of elements
    - ADD R3, R3, #1
- Decrement the inner for-loop sorter counter (R5)
    - ADD R5, R5, #-1
- Condition check. If the inner for-loop sorter counter (R5) is positive, continue going through the inner loop
    - BRp INNER_LOOP
- Condition check. If the inner for-loop sorter counter (R5) is positive or zero, branch back to the outer for-loop sorter
    - BRzp OUTER_LOOP

Sorted Label:

Once the array is sorted, return to the calling program

- Create the sorted label
    - SORTED
- Return to the calling program
    - RET (JMP R7)

Output Subroutine:

Output the sorted values into the console. We cannot load large numbers to the registers to output, or else we will run into overflow errors. To work past this, we will need to break up the real number into its three single digits and display those digits instead. To do this, use multiplication and modulus.
For example, use 142. Subtract 100 until the result becomes negative. We will subtract once before the result becomes negative; this becomes the first digit 1. With the remainder as 42 - 100, add 100 back to make 42 positive. Subtract this remainder by 10 until it becomes negative. We will subtract 4 times before it becomes negative; this becomes the second digit 4. With the remainder as 2 - 10, add 10 back to make it positive. This now becomes the third digit, 2.

- Create the output subroutine, with the label OUTPUT_LOOP
    - OUTPUT_LOOP

- Load the output prompt to R0
    - LEA R0, OUTPROMPT
- Display the prompt onto the console
    - PUTS
- Reset the pointer value so that it starts x4000, the beginning of our array
    - LD R3, PT
- Reset the value for the output counter
    - LD R6, COUNT
- Move to the output loop, using the label OUTPUT
    - OUTPUT
- Clear R0, R1, R4, and R5. R1 will be used to hold the first digit. R4 will be used to hold the second digit. R5 will be used to hold the third digit
    - AND R1, R1, #0
    - AND R4, R4, #0
    - AND R5, R5, #0
    - AND R0, R0, #0
- Load the label SPACE that causes the console to go to the next line and output to console
    - LD R0, SPACE
    - OUT
- Load a number from the array to R0, clearing R0 first
    - AND R0, R0, #0
    - LDR R0, R3, #0
- Set up to divide by 100. Clear R2 and load our 100 label to it
    - AND R2, R2, #0
    - LD R2, HUNDRED
- Use two's complement to negate 100 in R2
    - NOT R2, R2
    - ADD R2, R2, #1
- Enter the first subtraction loop using the label SUBTRACT1
    - SUBTRACT1
- Use R1 as a counter to keep track of how many times we have subtracted
    - ADD R1, R1, #1
- Subtract 100 from the number in R0
    - ADD R0, R0, R2
- Condition check. Continue to subtract until the result becomes negative
    - BRzp SUBTRACT1
- Enter the first remainder using the label REMAINDER1
    - REMAINDER1
- Clear R2 and load the label HUNDRED to it
    - AND R2, R2, #0
    - LD R2, HUNDRED
- Get the correct positive remainder by adding 100 to it
    - ADD R0, R0, R2
- Subtract 1 from R1, our subtraction counter, to get the real number of times we subtracted before the remainder became negative. This becomes the first digit
    - ADD R1, R1, #-1
- Store the first digit in R1 to the label DIGIT1
    - STI R1, DIGIT1

- Set up to divide by 10 to find the second digit. Clear R2 and load the TEN label to R2
    - AND R2, R2, #0
    - LD R2, TEN
- Use two's complement to negate 10 in R2
    - NOT R2, R2
    - ADD R2, R2, #1
- Set up the second subtraction loop using the label SUBTRACT2
    - SUBTRACT2
- Use R4 as a counter to keep track of how many times we have subtracted. This will become the second digit
    - ADD R4, R4, #1
- Subtract 10 from the remainder, which is in R0
    - ADD R0, R0, R2
- Condition check. Continue to subtract until the remainder becomes negative
    - BRzp SUBTRACT2
- Enter the second remainder, using the label REMAINDER2
    - REMAINDER2
- Clear R2 and load the TEN label to it
    - AND R2, R2, #0
    - LD R2, TEN
- Add 10 to the remainder (R0), store in R5. R5 becomes the third digit
    - ADD R5, R0, R2
- Store the third digit in R5 to the label DIGIT3
    - STI R5, DIGIT3
- Find the real number of times we subtracted before the remainder became negative by subtracting 1 from R4
    - ADD R4, R4, #-1
- Store the second digit in R4 to the label DIGIT2
    - STI R4, DIGIT2
- Set up to display the first digit. Clear R0 and load the first digit to it
    - AND R0, R0, #0
    - LDI R0, DIGIT1
- Clear R2 and load the positive ASCII offset to it
    - AND R2, R2, #0
    - LD R2, ASCIIPOS
- Add the ASCII offset to the first digit
    - ADD R0, R0, R2
- Output to console
    - OUT
- Set up to display the second digit. Clear R0 and load the second digit to it
    - AND R0, R0, #0
    - LDI R0, DIGIT2
- Clear R2 and load the positive ASCII offset to it
    - AND R2, R2, #0
    - LD R2, ASCIIPOS
- Add the ASCII offset to the second digit
    - ADD R0, R0, R2
- Output to console

    - OUT
- Set up to display the third digit. Clear R0 and load the third digit to it
    - AND R0, R0, #0
    - LDI R0, DIGIT3
- Clear R2 and load the positive ASCII offset to it
    - AND R2, R2, #0
    - LD R2, ASCIIPOS
- Add the ASCII offset to the third digit
    - ADD R0, R0, R2
- Output to console
    - OUT
- Increment the pointer R3
    - ADD R3, R3, #1
- Decrement the output loop counter
    - ADD R6, R6, #-1
- Condition check. If the loop counter is positive, continue to loop
    - BRp OUTPUT
- Return to the calling program
    - RET (JMP R7)

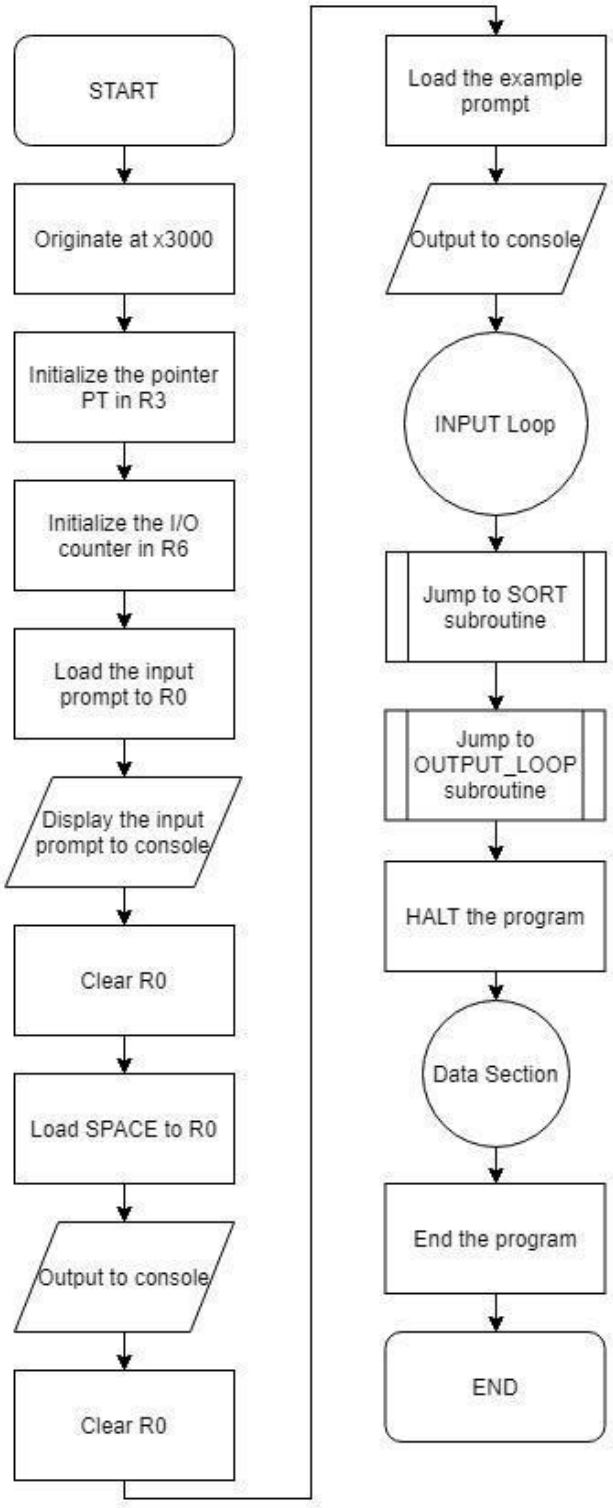
End of Program:

End the program

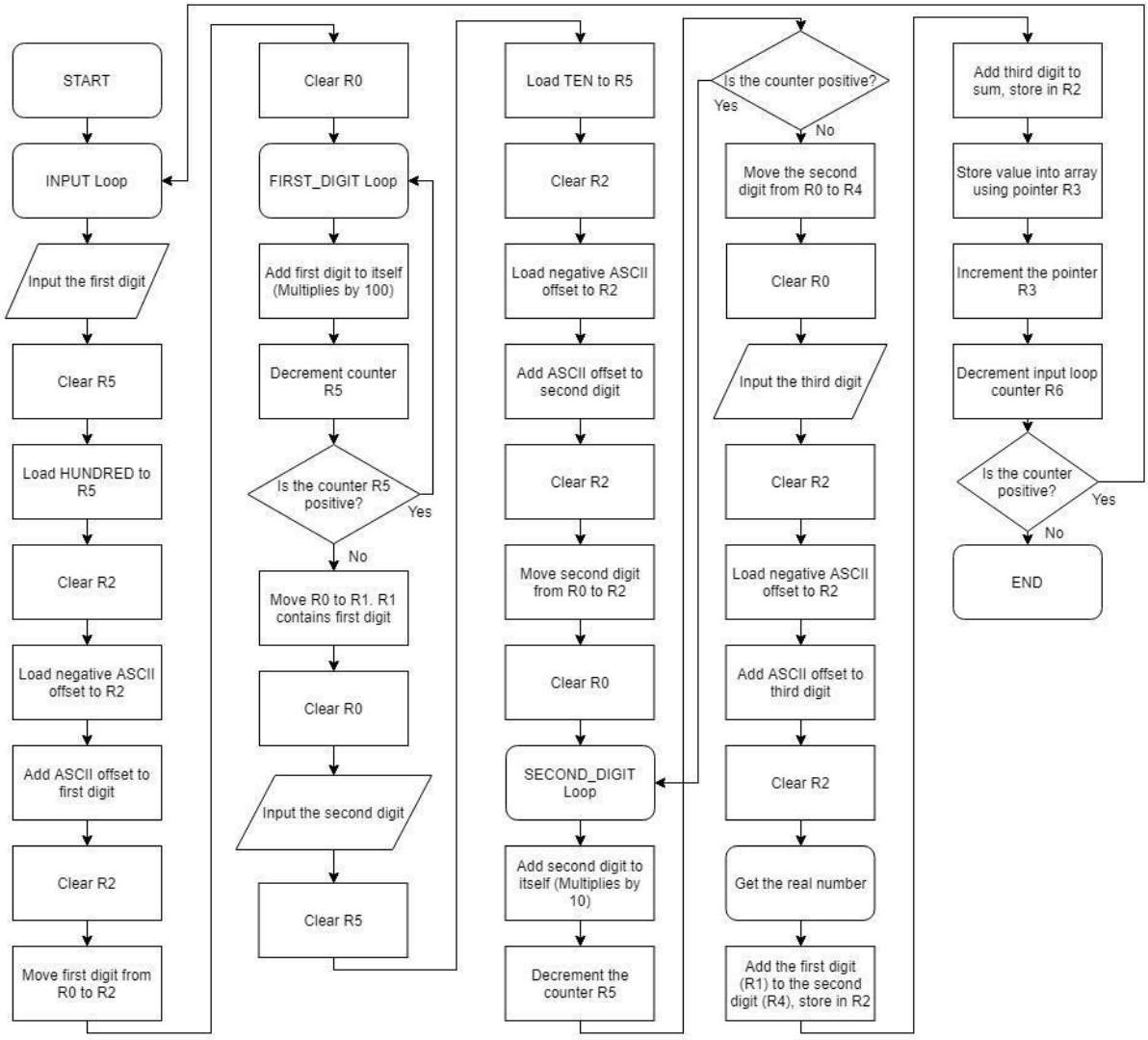
- .END

Bubble Sort Flowcharts

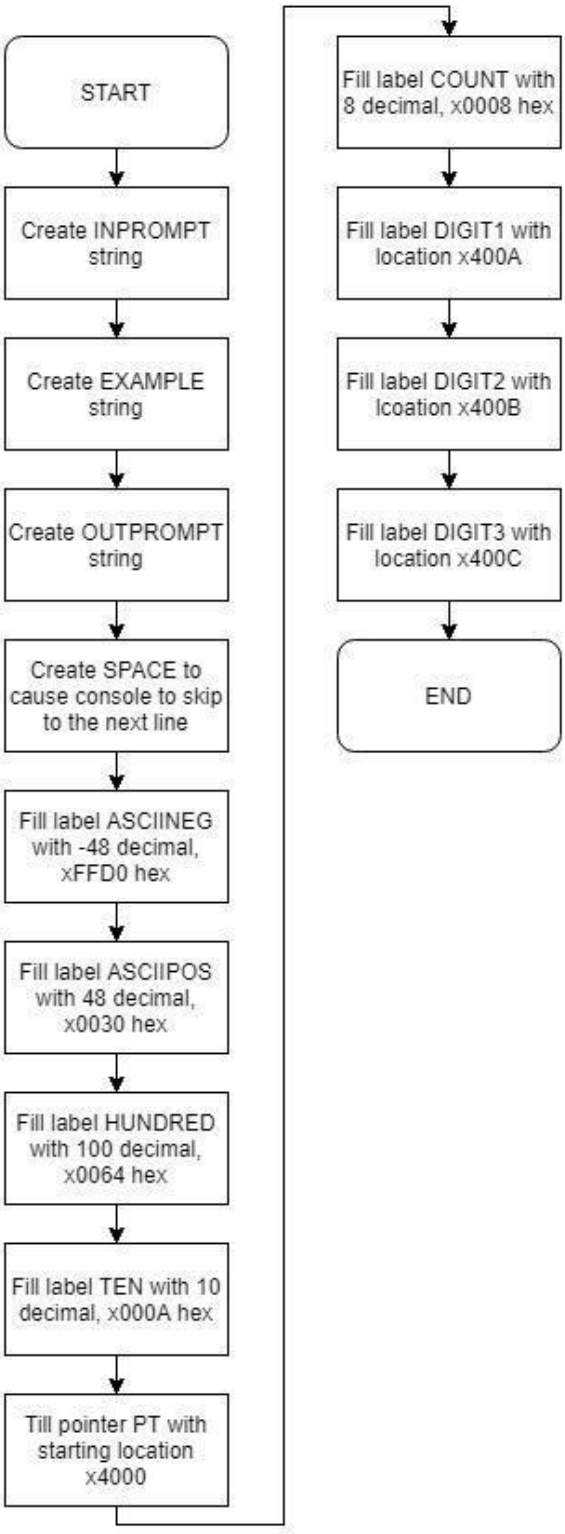
Initialization and Program Flow:



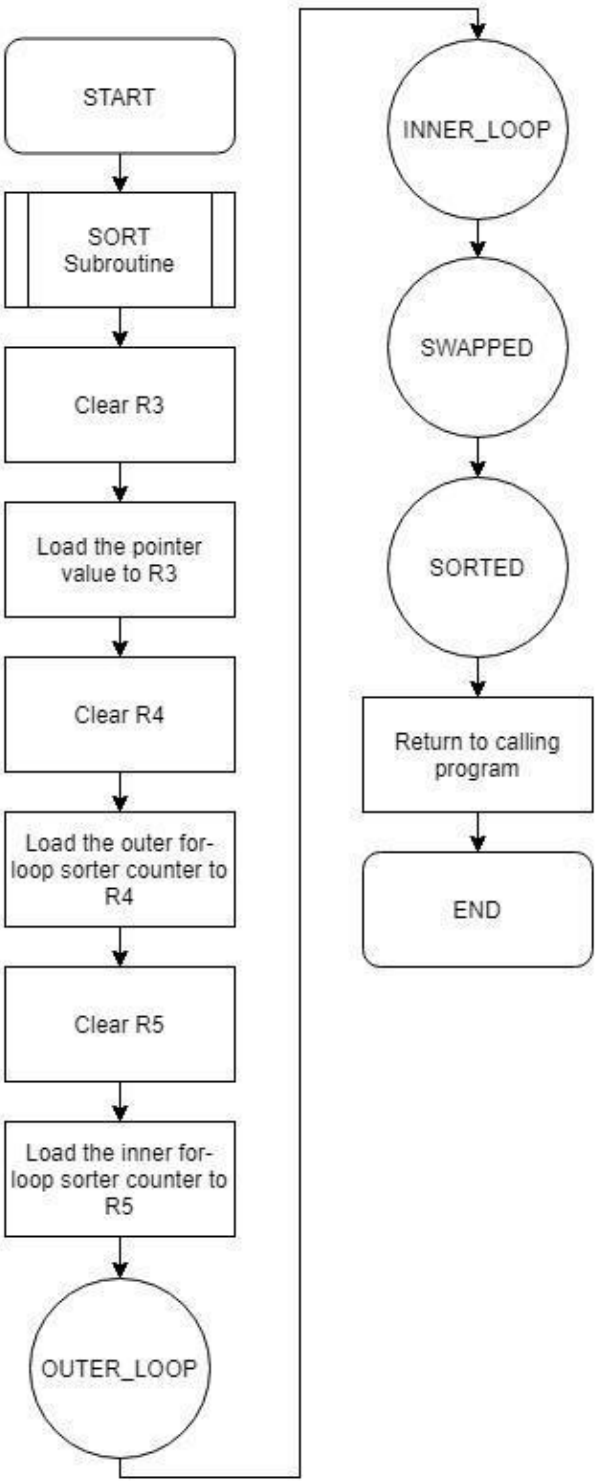
INPUT Loop:


START
INPUT Loop
Input the first digit
Clear R5
Load HUNDRED to R5
Clear R2
Load negative ASCII offset to R2
Add ASCII offset to first digit
Clear R2
Move first digit from R0 to R2
Clear R0
FIRST_DIGIT Loop
Add first digit to itself (Multiplies by 100)
Decrement counter R5
Is the counter R5 positive?
Yes
No
Move R0 to R1. R1 contains first digit
Clear R0
Input the second digit
Clear R5
Load TEN to R5
Clear R2
Load negative ASCII offset to R2
Add ASCII offset to second digit
Clear R2
Move second digit from R0 to R2
Clear R0
SECOND_DIGIT Loop
Add second digit to itself (Multiplies by 10)
Decrement the counter R5
Is the counter positive?
Yes
No
Move the second digit from R0 to R4
Clear R0
Input the third digit
Clear R2
Load negative ASCII offset to R2
Add ASCII offset to third digit
Clear R2
Get the real number
Add the first digit (R1) to the second digit (R4), store in R2
Add third digit to sum, store in R2
Store value into array using pointer R3
Increment the pointer R3
Decrement input loop counter R6
Is the counter positive?
Yes
No
END

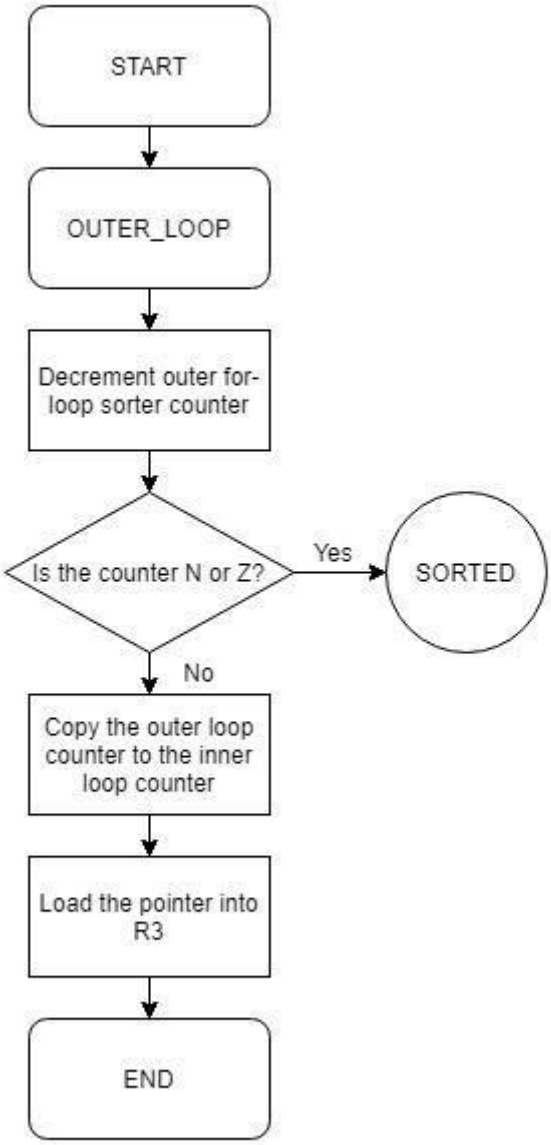
Data Section:



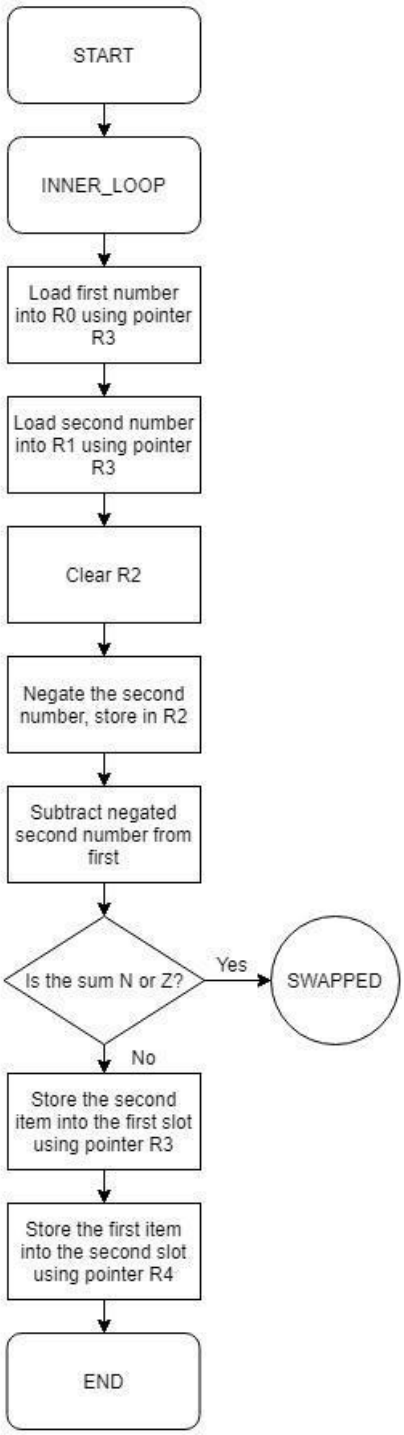
SORT Subroutine:



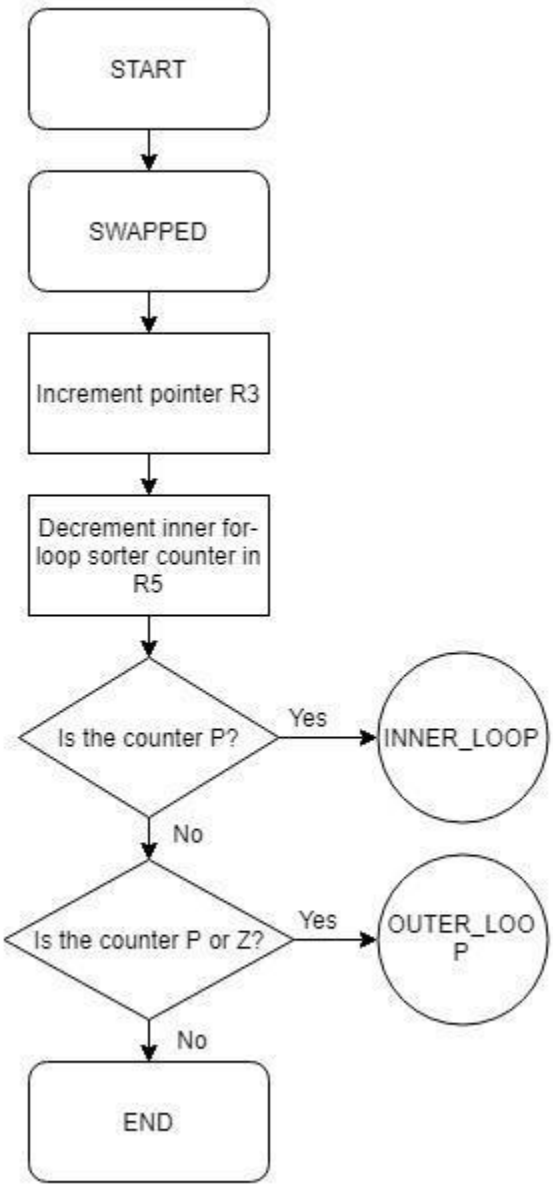
OUTER_LOOP:


START
OUTER_LOOP
Decrement outer for-loop sorter counter
Is the counter N or Z?
Yes
SORTED
No
Copy the outer loop counter to the inner loop counter
Load the pointer into R3
END

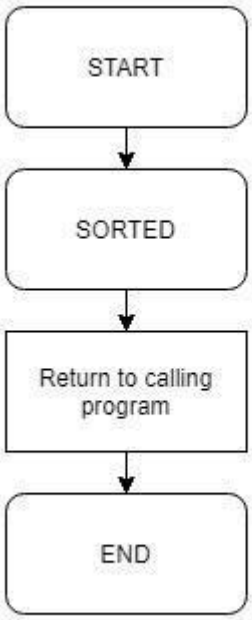
INNER_LOOP:


START
INNER_LOOP
Load first number into R0 using pointer R3
Load second number into R1 using pointer R3
Clear R2
Negate the second number, store in R2
Subtract negated second number from first
Is the sum N or Z?
Yes
SWAPPED
No
Store the second item into the first slot using pointer R3
Store the first item into the second slot using pointer R4
END

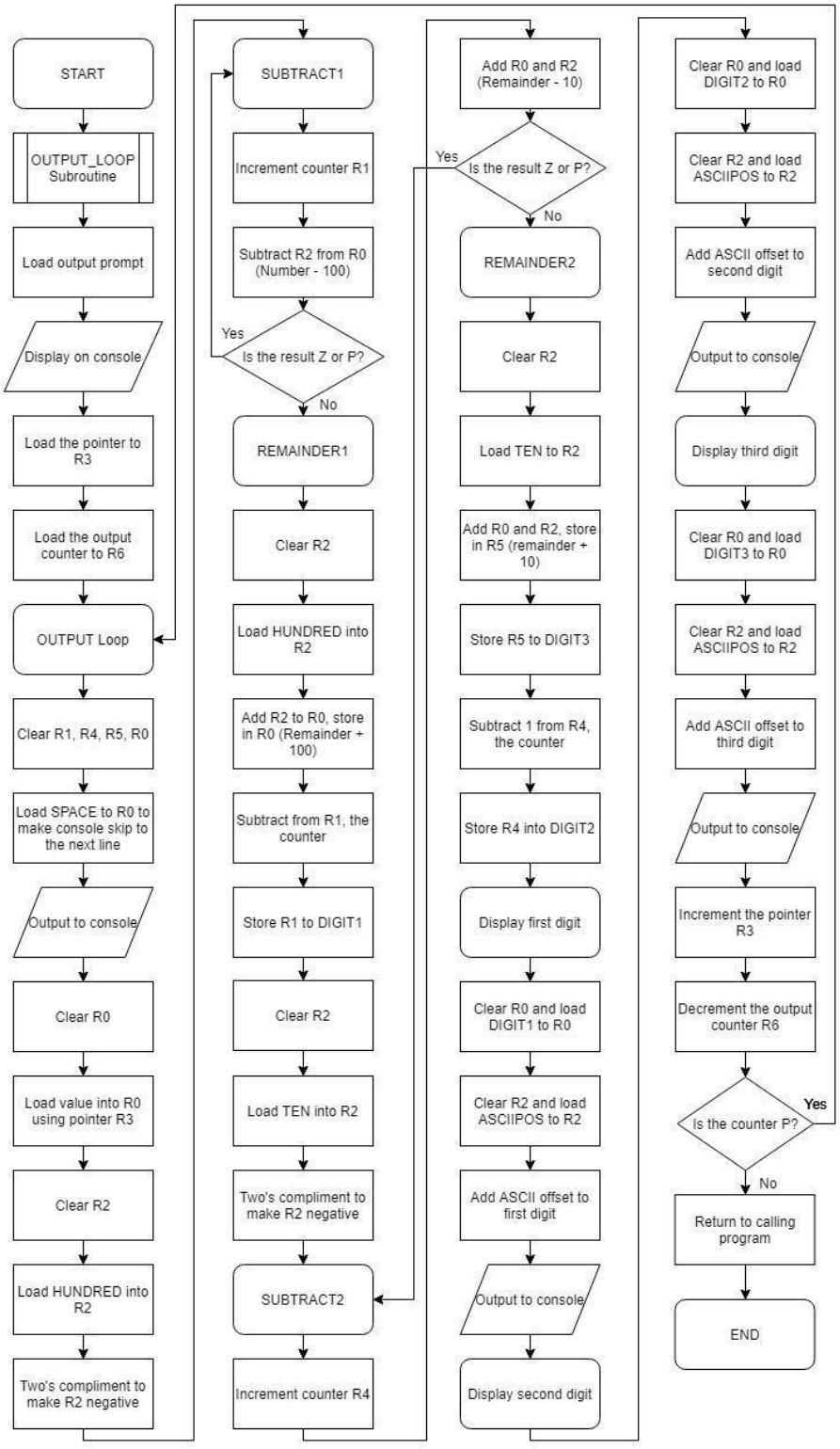
SWAPPED:



SORTED:



OUTPUT_LOOP:


START
OUTPUT_LOOP Subroutine
Load output prompt
Display on console
Load the pointer to R3
Load the output counter to R6
OUTPUT Loop
Clear R1, R4, R5, R0
Load SPACE to R0 to make console skip to the next line
Output to console
Clear R0
Load value into R0 using pointer R3
Clear R2
Load HUNDRED into R2
Two's compliment to make R2 negative
SUBTRACT1
Increment counter R1
Subtract R2 from R0 (Number - 100)
Yes
Is the result Z or P?
No
REMAINDER1
Clear R2
Load HUNDRED into R2
Add R2 to R0, store in R0 (Remainder + 100)
Subtract from R1, the counter
Store R1 to DIGIT1
Clear R2
Load TEN into R2
Two's compliment to make R2 negative
SUBTRACT2
Increment counter R4
Add R0 and R2 (Remainder - 10)
Yes
Is the result Z or P?
No
REMAINDER2
Clear R2
Load TEN to R2
Add R0 and R2, store in R5 (remainder + 10)
Store R5 to DIGIT3
Subtract 1 from R4, the counter
Store R4 into DIGIT2
Display first digit
Clear R0 and load DIGIT1 to R0
Clear R2 and load ASCIIPOS to R2
Add ASCII offset to first digit
Output to console
Display second digit
Clear R0 and load DIGIT2 to R0
Clear R2 and load ASCIIPOS to R2
Add ASCII offset to second digit
Output to console
Display third digit
Clear R0 and load DIGIT3 to R0
Clear R2 and load ASCIIPOS to R2
Add ASCII offset to third digit
Output to console
Increment the pointer R3
Decrement the output counter R6
Yes
Is the counter P?
No
Return to calling program
END