Lab 4: Sorting Integers

Due Friday, 22 May, 11:59 PM

Minimum Submission Requirements

- Ensure that your Lab4 folder contains the following files (note the capitalization convention):
  - Diagram.pdf
  - Lab4.asm
  - README.txt
- Completed Google Form
- Commit and push your repository

Objectives

1. Take user input from program arguments.
2. Parse a numeric character ASCII string to an integer.
3. Sort a list of up to 8 integers.

Functionality

The program will accept up to 8 program arguments in HEX format. The numbers will range from 0x000 up to 0xFFF. These numbers will be converted to decimals and numerically sorted in ascending order. The sorted numbers will be printed on screen in decimal format.

Example Output

```
Program arguments:
0x0A 0x0B 0x0A 0x0A 0x0A 0x0C 0x00 0xFE

Integer values:
10 11 10 10 10 12 0 254

Sorted values:
0 10 10 10 10 11 12 254

-- program is finished running --
```

Preparation

Read and understand everything in this section before attempting the assignment.

Links

Read
Program Arguments

Introduction To MIPS Assembly Language Programming
sections 9.2, 9.3

Documentation Standards

ASCII reference table

Optional (for floating point values - not required in this lab!)
MIPS Integer to Floating Point

Program Arguments

A program argument is an argument passed to the program when it is called. For instance, when you run a C program, you might call it on the command like:

```
./program arg1 arg2
```

The ASCII strings "arg1" and "arg2" are the program arguments that are stored in memory. In C, to use the program arguments, the programmer needs to access the array argv[] (you'll get this in CSE 13!)

In MARS, program arguments are also read as ASCII strings, and they are stored in memory. Please read through this document for an explanation of how to find the program arguments in memory.

In this lab, **YOU MUST USE PROGRAM ARGUMENTS** to take in user input. **DO NOT USE A SYSCALL FOR USER INPUT.** The grading script is automated, and sends input to the program using program arguments.

Specification

Input

You will be using program arguments instead of a syscall to take user inputs. See this document on how to use program arguments.

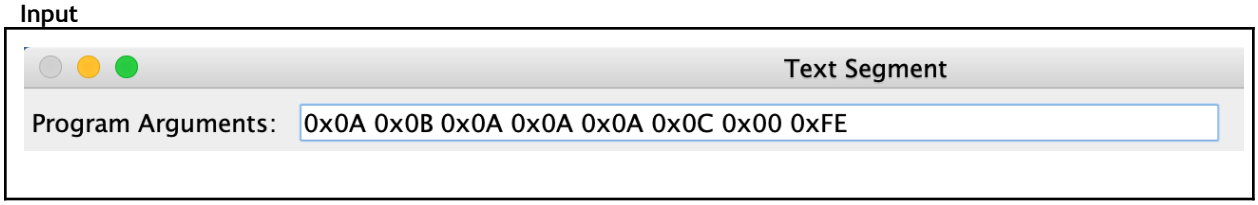
Part A

Create a block diagram and pseudocode to aid in the development of your MIPS assembly program. Generate code to print the program arguments to the console.

The diagram and pseudocode should completely describe the process for reading program arguments, converting the ASCII strings to integer values, sorting and printing.

Part A Example Use Case:

Input



Output

Print the program arguments to the screen in the order they were entered.

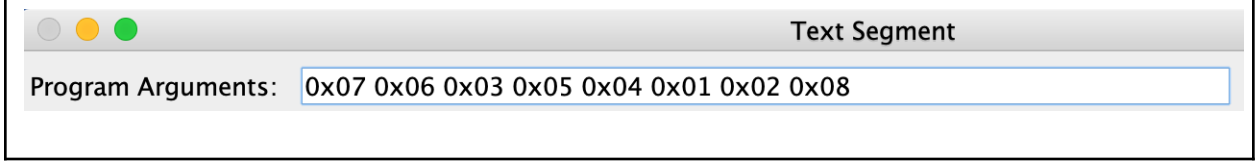
```
Program arguments:
0x0A 0x0B 0x0A 0x0A 0x0A 0x0C 0x00 0xFE
-- program is finished running --
```

Part B

Convert the program arguments to integer values. These values should be printed in the order they were obtained and then in numerical order with the smallest number first and the largest number last. You may use syscalls 1 to print the in-order values.

Part B Example Use Case:

Input



Output

Print the program arguments to the screen, the unsorted order in decimal format, and the sorted order in decimal.

```
Program arguments:
0x07 0x06 0x03 0x05 0x04 0x01 0x02 0x08

Integer values:
7 6 3 5 4 1 2 8

Sorted values:
1 2 3 4 5 6 7 8

-- program is finished running --
```

Format

Input

Each value will be given in the following format: zero, lowercase x, followed by up to 3 hexadecimal digits, with A-F capitalized. There will be exactly one space in between each value.

Output

Program Arguments

Program arguments should be printed in the format they were entered: '0x' followed by two hexadecimal digits, with A-F capitalized and one space in between each value.

Sorted/unsorted Values

Sorted and unsorted values should be printed in space-separated decimal format. You may use syscall 1 for this purpose.

Summary

To receive all possible points, **the output should match the stated format exactly**. Take note of the:

1. Wording, spelling, and capitalization of descriptions
2. New line characters after:
    - Printed program arguments
    - Sorted values in decimal format
3. Hex digit capitalization
    - Capital A-F in printed program arguments
4. Spaces
    - One space between each program argument
    - One space between each value in sorted and unsorted list.

Input Types

You may assume all input values will be valid and will not include NaN. No error checking is required.

Syscalls

You may not use syscalls 5 - 8. You must use **syscall 10** to exit your program. See below for a summary of syscalls and their uses. You may use either syscall 4 or 11 to print program arguments.

For extra credit, you may choose to print the sorted values using syscall 4 or syscall 11 instead of syscalls 1.

SYSCALL #	FUNCTION	PURPOSE
1	Print integer	Print integer values in decimal
4	Print string	Print descriptions, print program arguments
5, 6, 7, 8	Read *	**FORBIDDEN! DO NOT USE**
10	Exit	Exit program cleanly
11	Print character	Print program arguments, print decimals (extra credit)

Table: Syscalls

Turn Off Delayed Branching

Make sure delayed branching is turned OFF. See the appendix for more information.

MIPS Memory Configuration

Your program must run in the default memory configuration. See the appendix for more information.

Automation

Note that our grading script is automated, so **it is imperative that your program's output matches the specification exactly**. Output that deviates from the spec will cause point deduction. Remember, **YOUR PROGRAM MUST ACCEPT USER INPUT ONLY THROUGH PROGRAM ARGUMENTS.**

Extra Credit

Don't Use Syscall 1

For this extra credit option, use only syscall 11 and/or syscall 4 to print the sorted/unsorted values in decimal format. Do not use syscall 1.

Submission

This assignment will be submitted in two parts.

Part A: Block Diagram, Pseudocode, and Program Arguments Print

Diagram.pdf

This diagram should completely describe the process for reading program arguments, converting the ASCII strings to decimal values, sorting and printing. Instructions for the block diagram can be found here.

Lab4.asm

This file should contain full pseudocode for reading program arguments, converting to decimal values, sorting, and printing. It should contain code to print the program arguments to the screen. Include a header comment as indicated in the documentation guidelines here.

Your program must assemble and run to get credit for printing the program arguments. You will not be penalized if your program has additional functionality.

Part B: Sorted Values in Decimal, README, Google Form

Diagram.pdf

Update if necessary.

Lab4.asm

Keep pseudocode or update if necessary. Should contain full functionality of the program.

README.txt

Instructions for the README can be found here.

Google Form

You are required to answer questions about the lab in this Google Form. Question answers, excluding the ones asking about resources used and collaboration should total at the very least 150 words.

https://forms.gle/aTXH9L62hNjd32bW7

A Note About Academic Integrity

Please review the syllabus on acceptable and unacceptable collaboration.

Grading Rubric (80 points - subject to change)

Part A (5 points)

```
1 pt pseudocode & block diagram
4 pts program arguments
```

Part B (75 points)

15 pts general

```
8 pts assembles without errors
1 pt  updated pseudocode & block diagram
1 pt  program output match specifications
2 pts README
3 pts Google Form
```

60 pts test cases (4 test cases, 15 pts per test case)

One test case point breakdown

```
1 pt   HEX format values printed in order of appearance.
4 pts  unsorted decimal values printed in order of appearance.
10 pts sorted decimal values printed in ascending order.
```

10 pts extra credit

```
10 pts no syscall 1
```

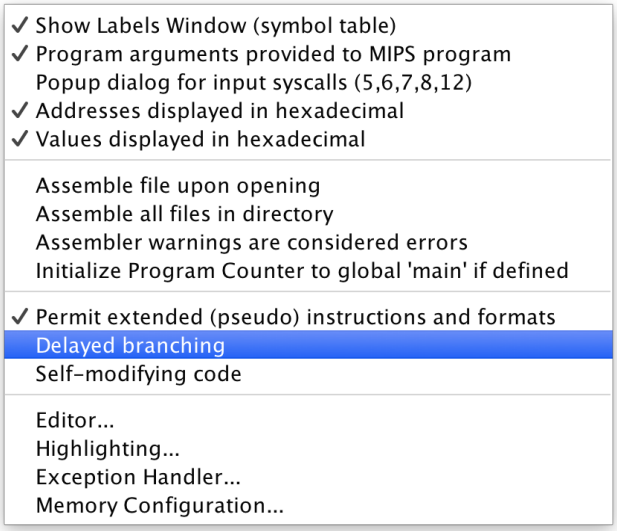
Point Deductions

```
-15 pts program only runs in a specific memory configuration or memory addresses for
       the program arguments are hard-coded (see the appendix for more information)

-15 pts program uses syscalls 5 - 8

-3  pts program sort values, but in descending order
```

Appendix

Turn Off Delayed Branching

Make sure delayed branching is turned off. From the settings menu, make sure **Delayed branching is unchecked**



Checking this option will insert a "delay slot" which makes the next instruction after a branch execute, no matter the outcome of the branch. To avoid having your program behave in unpredictable ways, make sure **"Delayed branching"** is turned **OFF**. In addition, add a NOP instruction after each branch instruction. The NOP instruction guarantees that your program will function properly even if you forgot to turn off delayed branching. For example:

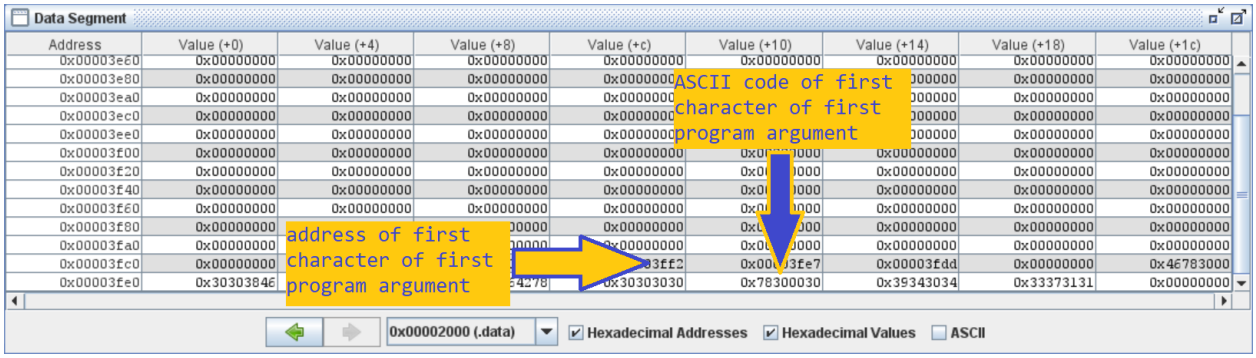
```
      LI   $t1 2

LOOP: NOP
      ADDI $t0 $t0 1
      BLT  $t0 $t1 LOOP
      NOP                    # nop added after the branch instruction
      ADD  $t3 $t5 $t6
```

MIPS Memory Configuration

To find the program arguments more easily in memory, you may choose to develop your program using a compact memory configuration (Settings -> Memory Configuration). However, your program **MUST** function properly using the **Default** memory configuration. You should not run into issues as long as you **do not hard-code any memory addresses** in your program. Make sure to test your program thoroughly using the **Default** memory configuration.

What does it mean to "hard-code" an address?

Let's say after compiling our program, we discover that the address of the first character of the first program argument is 0x00003ff2 as shown below.



With this knowledge, we *could* access the first character of the first program argument like this:

```
li $t1 0x00003ff2
lb $t2 ($t1)
```

However, in this code snippet, we are hard-coding the address to $t1. When changing the memory configuration back to default, this code will no longer work, because the address 0x00003ff2 is inaccessible by programmers in the default MIPS memory configuration. In addition, program arguments may be assigned to different memory locations depending on which machine the program is running on.

For your program to work as expected on all machines, use the values of $a0 and $a1 to access program arguments as described in the program arguments document.