Lab 4: Roman Numeral Conversion

Part A: Due Sunday, 19 May 2019, 11:59 PM
Part B: Due Friday, 24 May 2019, 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
- Commit and push your repository

Lab Objective

In this lab, you will develop a more detailed understanding of how data is represented, stored, and manipulated at the processor level. In addition to strengthening your understanding of MIPS coding, you will use program arguments to read a string input and learn how to convert Roman numerals into binary and how to print digits as ASCII characters.

Lab Preparation

1. Read sections 6.4, 9.2 - 9.3 from Introduction To MIPS Assembly Language Programming.
2. Learn how to interpret Roman numerals here

Specification

You will write a program in the MIPS32 language using the MARS integrated development environment to convert a Roman numeral input into an integer. You will then print the integer as a binary ASCII string.

Input

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

Sample Outputs

Valid Inputs

```
You entered the Roman numerals:
XXVI

The binary representation is:
0b11010

-- program is finished running --
```


© 2019, Computer Engineering Department, University of California - Santa Cruz

```
You entered the Roman numerals:
CXCIX

The binary representation is:
0b11000111

-- program is finished running --
```

```
You entered the Roman numerals:
CCCLXXXVIII

The binary representation is:
0b110000100

-- program is finished running --
```

Non Valid Inputs

```
You entered the Roman numerals:
CIC

Error: Invalid program argument.

-- program is finished running --
```

```
You entered the Roman numerals:
IIIV

Error: Invalid program argument.

-- program is finished running --
```

```
You entered the Roman numerals:
V111

Error: Invalid program argument.

-- program is finished running --
```

Extra Credit

In this example, the input is non-minimal, so it is counted as a non valid input.

```
You entered the Roman numerals:
VIIII

Error: Invalid program argument.

-- program is finished running --
```


© 2019, Computer Engineering Department, University of California - Santa Cruz

```
You entered the Roman numerals:
DCXXIV

The binary representation is:
0b1001110000

-- program is finished running --
```

For full credit, **the output should match this format exactly**. Take note of the:

1. Exact wording of the statements.
2. Program argument printed on a new line.
3. Blank line under the program argument.
4. Binary value printed on a new line.
5. Blank line under the final result or invalid program argument error.

Functionality

The functionality of your program will be as follows:

1. Read a Roman numeral input from the program arguments.
    - Valid ASCII characters are I, V, X, L, and C
    - Assume program argument input will have a max of 20 characters
2. Print the user's input.
3. Convert the Roman numeral input to an integer and store in $s0.
    - Your program should parse through the ASCII string and compare adjacent characters to decide if you must **add** or **subtract** or if the **input is invalid**.
    - Basic conversion algorithm:
        i. Iterate over each character of the string
            1. Compare value of the current character to the value of the next character
                a. If current value is greater or equal, add value to running sum
                b. Else, subtract value from running total
4. Print an **ERROR** message if:
    - The user entered invalid ASCII characters
    - An invalid Roman numeral string has been entered
5. Exit the program cleanly using syscall 10.
6. Extra credit options
    - Include D as a valid input from the user
    - Print error message for non-minimal Roman numeral input

Part A: Block Diagram, Pseudocode, and Program Arguments

In part A, you must create a block diagram and pseudocode to aid in the development of your MIPS assembly program. In addition, you must generate code to print the program argument to the console.

Block Diagram

Before coding, create a top level block diagram or flowchart to show how the different portions of your program will work together. Start by creating a flow chart

© 2019, Computer Engineering Department, University of California - Santa Cruz

of the steps that were previously described. Next, break each step into its own block diagram.

Use https://www.draw.io or a similar drafting program to create this document. This diagram will be contained in the file Diagram.pdf. **This diagram must be computer generated to receive full credit.**

Pseudocode

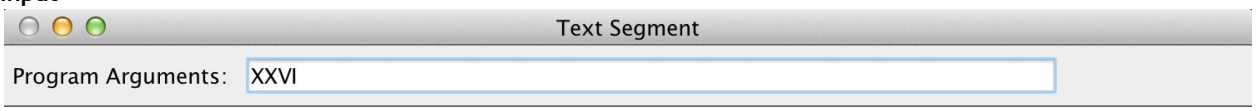
Next, you will create pseudocode that outlines your program. Your pseudocode will appear underneath the header comment in Lab4.asm. Guidelines on developing pseudocode can be found here: https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/

You may modify your pseudocode as you develop your program. **Your pseudocode must also be present in your final submission (part B).**

Program Arguments

Your code must read and print the program argument to the screen. For example:

Input



Output

```
You entered the Roman numerals:
XXVI

-- program is finished running --
```

Part B: Assembly Code

Implement the functionality of the program previously listed.

Automation

Note that part of 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.

Your code should end cleanly without error. **Make sure to use the exit syscall** (syscall 10).

Files

Diagram.pdf

This file will contain a block diagram or flowchart of how the different components of your code work together.

Lab4.asm

This file contains your pseudocode and assembly code. Follow the code documentation guidelines here.

© 2019, Computer Engineering Department, University of California - Santa Cruz

README.txt

This file must be a plain text (.txt) file. It should contain your first and last name (as it appears on Canvas) and your CruzID. Your answers to the questions should total at least 150 words. Your README should adhere to the following template:

```
------------------------
Lab 4: Roman Numeral Conversion
CMPE 012 Spring 2019

Last Name, First Name
CruzID
------------------------

Can you validly represent the decimal value 1098 with Roman numerals using
only I, V, X, and C?
Write the answer here.

What was your approach for recognizing an invalid program argument?
Write the answer here.

What did you learn in this lab?
Write the answer here.

Did you encounter any issues? Were there parts of this lab you found
enjoyable?
Write the answer here.

How would you redesign this lab to make it better?
Write the answer here.

What resources did you use to complete this lab?
Write the answer here.

Did you work with anyone on the labs? Describe the level of collaboration.
Write the answer here.
```

Syscalls

When printing the integer values, you may use syscall system services 4 (print string) and 11 (print character). **You <u>may not</u> use the following syscalls:**

1 (print integer)
5 (read integer)
12 (read character)
34 (print integer as hexadecimal)
35 (print integer as binary)
36 (print integer as unsigned)

You will lose a significant number of points if you use any of these syscalls. See the rubric for more details.

Make sure to exit your program cleanly using syscall 10.

© 2019, Computer Engineering Department, University of California - Santa Cruz

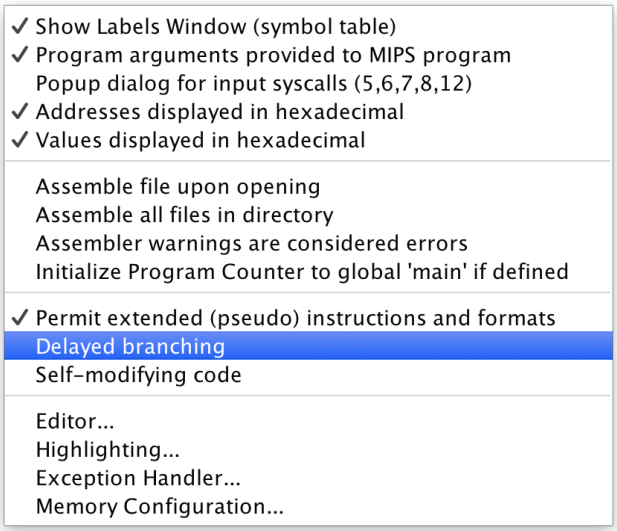
Note

It is important that you **do not hard-code** the values for any of the **addresses** in your program.

Other Requirements

Turn Off Delayed Branching

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.

© 2019, Computer Engineering Department, University of California - Santa Cruz

Submission Instructions

This assignment will be submitted in two parts. Late hours will not be used for Part A of the assignment. If you do not submit a diagram or pseudocode by the Part A deadline, you can submit it with your Part B submission for less points.

© 2019, Computer Engineering Department, University of California - Santa Cruz

Grading Rubric

point values to be determined

Part A

pseudocode
block diagram
program arguments

Part B

pseudocode quality
block diagram quality
assembles without errors
correct value in $s0
statements match specification
invalid input detection
test cases
extra credit

-50% if program only runs in a specific memory configuration or memory addresses are hard coded

-25% incorrect naming convention

No credit if you use a forbidden syscall (1, 5, 12, 34, 35, 36)

6 pt style and documentation
- 1 pt comment on register usage
- 1 pt useful and sufficient comments
- 1 pt labels, instructions, operands, comments lined up in columns
- 2 pt readme file complete (should total at least 150 words)
- 1 pt complete headers for code and README

  Note: program header must include name, CruzID, date, lab name, course name, quarter, school, program description, note on program argument usage
  README must include name, CruzID, lab name, course name

© 2019, Computer Engineering Department, University of California - Santa Cruz