Documentation Standards

Purpose

The purpose of this document is to outline the expectations for documentation and formatting for the lab assignments in CE 12. Clean visual presentation and thorough documentation enables readers to effectively interpret engineering design. Use this class as an opportunity to start developing quality documentation habits.

README

Each lab will require a README file. A README file is used to explain the contents of a directory. It provides context for the files in that directory, and gives instructions on how to run any program or file.

In your labs, think of your README files as notes to your future self. If you ever look back at your work, you will want to know what the purpose of the lab was, how to run any code or simulations (e.g. what program to use) and any other special instructions. You are not required to follow any specific format for the README. However, make sure to include:

- Your name & CruzID
- Year and quarter (e.g. Fall 2019)
- Lab number and description
- List of all files in the directory (except for the README)
- Instructions on how to run any code or simulations

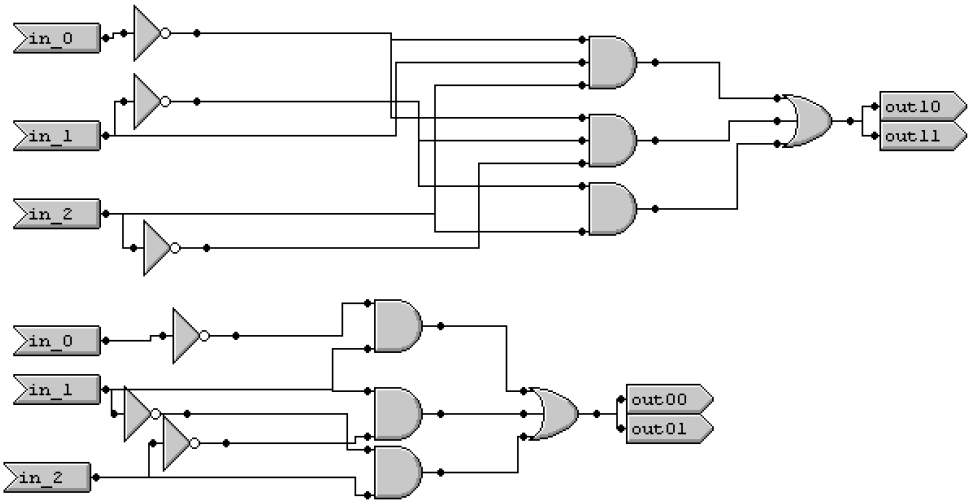
Example

```
Rebecca Rashkin
rrashkin
Fall 2019
Lab 4: Name of Project

-----------
DESCRIPTION

In this lab, the user enters a number using Roman numeral notation. The program
will print the decimal value to the screen.

-----------
FILES

-
Diagram.pdf

This file includes a block diagram of the program design. The application draw.io
was used to draft the diagram and generate the pdf.

-
Lab4.asm

This file includes the assembly code of the lab.

-----------
INSTRUCTIONS

This program is intended to be run using the MIPS Assembler and Runtime Simulator
(MARS). Enter the test case as a program argument and run using MARS.
```

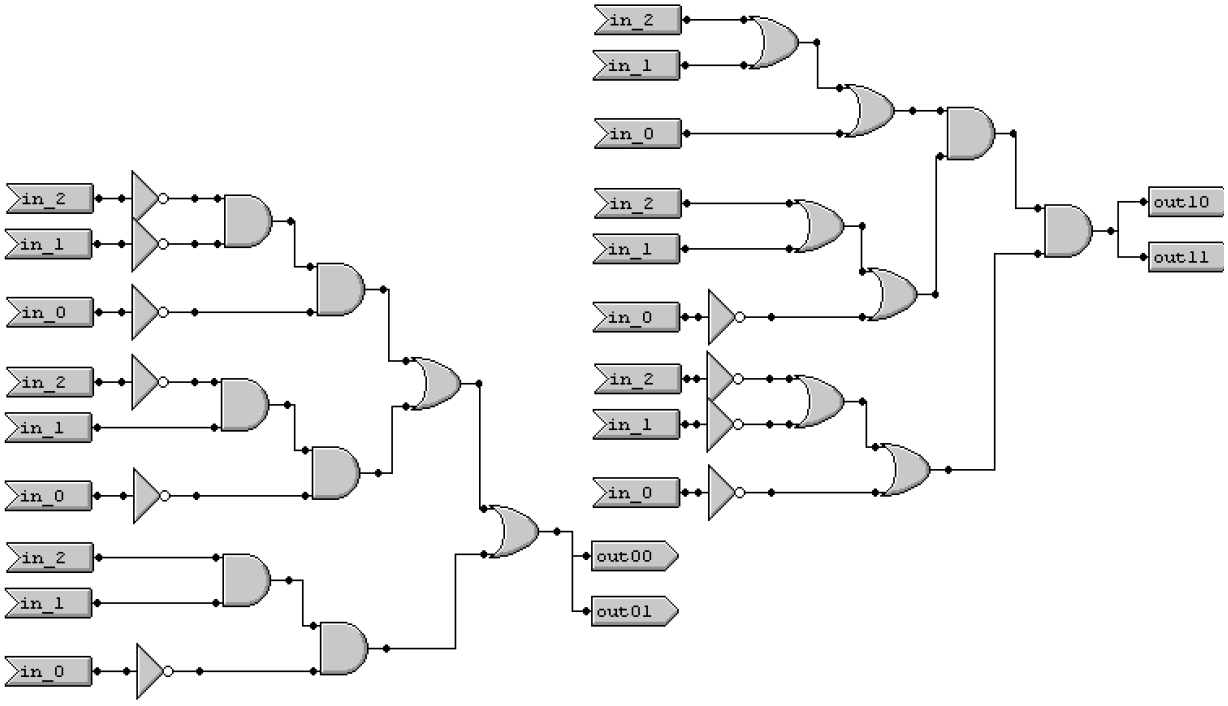
Schematic Visual Structure

Your circuits should be structured in an organized method that is easy to read and interpret. In Multimedia Logic, selecting "Snap to Grid" under the View menu makes it easy to line up components. A clean circuit uses many senders and receivers with meaningful names, and has no wires crossing over each other. Note that there may be multiple receivers for one sender. See below for examples.

Messy Circuit Example



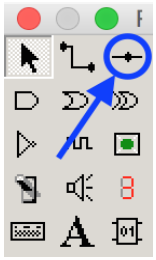
Clean Circuit Example



Comments

Each page should be labeled with your last name, first name, and CruzID (the name used in your UCSC email address). Label each circuit with a description of the functionality and the part of the lab that they are for.

Missing Wire Best Practices



MML has a known bug which causes some wires to disappear during the save process. To reduce the likelihood of this occurring, **DO NOT use the "Node" tool** (it's a black dot located at the top-right of the tool palette). This tool is particularly vulnerable to the bug.

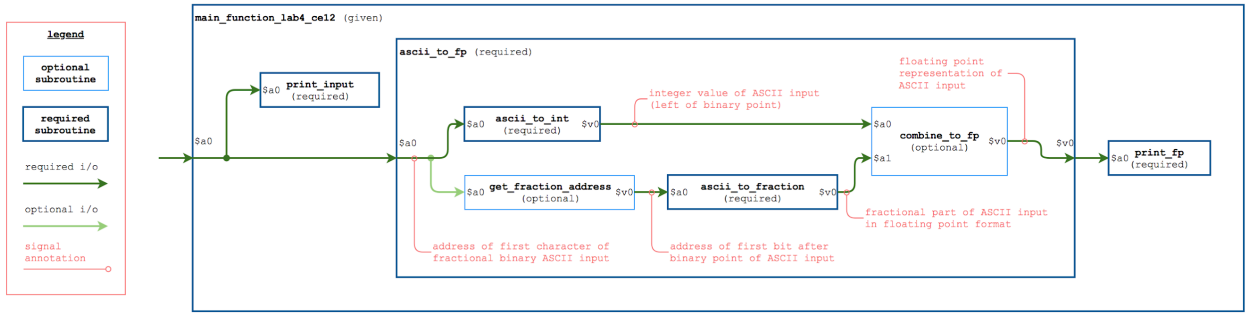
If this bug occurs, the grader will attempt to repair the missing wire in your file. This is only possible if your circuit is very readable. Make sure that wires **do not cross** whenever possible. Wire paths should be short and direct. **Use senders and receivers liberally**.

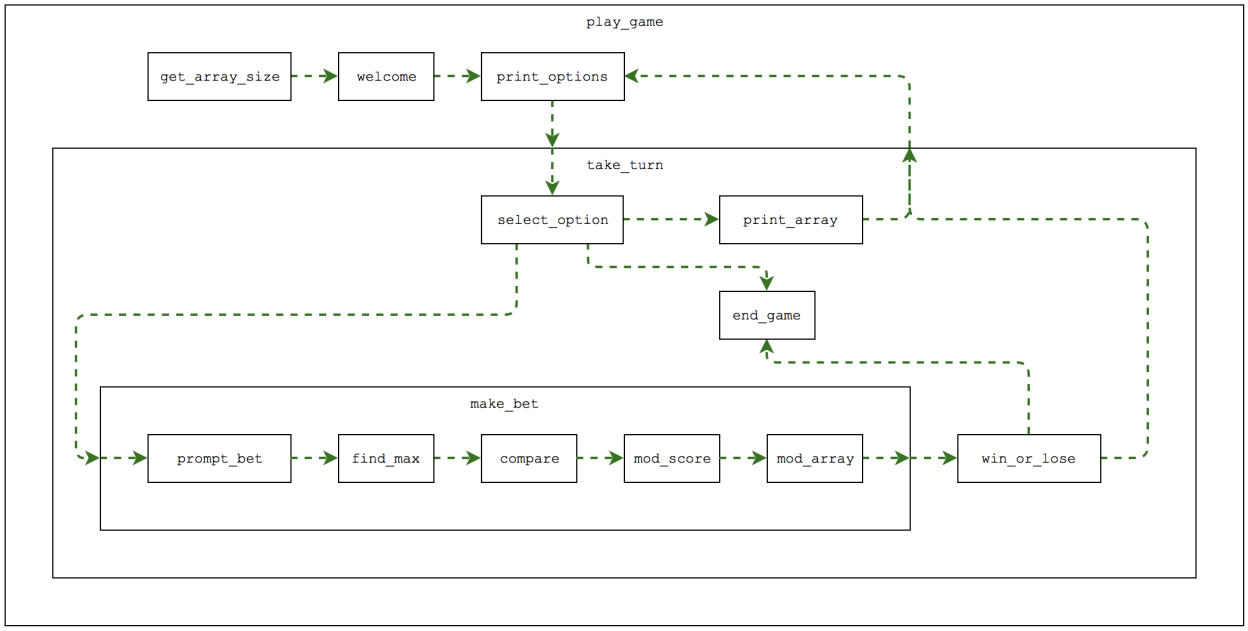
Block Diagram (For Programming Assignments)

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

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.**

Examples





Pseudocode

Pseudocode describes program functionality without language specific syntax. *General* guidelines on developing pseudocode can be found here:

https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/

Please note that it is expected that you generate pseudocode that is specific to assembly language and the context of this class.

You must include pseudocode for each lab assignment underneath the header comment. You should develop pseudocode before writing your assembly program. You may modify your pseudocode as you develop your program. Your pseudocode must be present in your final submission.

Comments

Your code should include a header comment with your name, CruzID, date, lab number, course number, quarter, school, program description and notes. Every program you write should include information like this. An example header comment is shown below.

```
###############################################################################
# Created by:  Last Name, First Name
#              CruzID
#              13 February 2019
#
# Assignment:  Lab 64: Super Hello World
#              CMPE 012, Computer Systems and Assembly Language
#              UC Santa Cruz, Winter 2019
#
# Description: This program prints 'Hello world.' to the screen.
#
# Notes:       This program is intended to be run from the MARS IDE.
###############################################################################
```

Every block or section of code should have a comment describing what that block of code is for. In-line comments should be lined up (using spaces) for ease of readability.

Register Usage

You should try to use as few registers as possible. You should only use registers for their intended purposes.

```
$zero       : anytime you need to use the value 0
$t0   - $t9 : the bulk of your program will use the temporary registers
$a0   - $a3 : only use for arguments to subroutines and for syscalls
$s0   - $s8 : only use these for values that should be saved across subroutine calls,
              or for values that will need to be referred to many times throughout
              your program like for user inputs
$ra         : only use for the return address if you implement subroutines
$a0   - $a3 : used for arguments to subroutines
$v0   - $v1 : used for return values from subroutines, $v0 is also used for syscalls
$sp         : only use to push and pop values from the stack
```

Block Comment

At the beginning of your code, and optionally at the beginning of each block of code, indicate the functionality of the registers used. For instance, if you are using $t0 and $t1 for the user input and a loop counter, respectively, your comments should include something like the following:

```
# REGISTER USAGE
# $t0: user input
# $t1: loop counter
```

White Space

Line up instructions, operands, and comments to increase readability. Code should be indented from labels.

Bad Example

```
LOOP: NOP
LI $t1 2 #initialize $t1
ADDI $t0 $t0 1 #increment $t0
BLT $t0 $t1 LOOP #determine if code should re-enter loop
```

Good Example

```
LOOP: NOP
      LI   $t1 2          # initialize $t1
      ADDI $t0 $t0 1      # increment $t0
      BLT  $t0 $t1 LOOP   # determine if code should re-enter loop
```

Notice how the instructions, operands and comments are all lined up in columns in this example. Colored pipe characters are added below to illustrate this.

```
LOOP:|NOP
     |LI   |$t1|2          |# initialize $t1
     |ADDI|$t0|$t0|1      |# increment $t0
     |BLT  |$t0|$t1|LOOP   |# determine if code should re-enter loop
```

A Note About Tabs

It is preferable to line up comments using spaces as opposed to tabs. Text editors can have different standards for the width of one tab character. For this reason, it is preferable to line up comments using spaces, not tabs, so that the code appears the same regardless of text editor.