CSE 141L Milestone 1

Student Name 1, PID; Student Name 2, PID

# Academic Integrity

Your work will not be graded unless the signatures of all members of the group are present beneath the honor code.

To uphold academic integrity, students shall:

* Complete and submit academic work that is their own and that is an honest and fair representation of their knowledge and abilities at the time of submission.
* Know and follow the standards of CSE 141L and UCSD.

Please sign (type) your name(s) below the following statement:

I pledge to be fair to my classmates and instructors by completing all of my academic work with integrity. This means that I will respect the standards set by the instructor and institution, be responsible for the consequences of my choices, honestly represent my knowledge and abilities, and be a community member that others can trust to do the right thing even when no one is watching. I will always put learning before grades, and integrity before performance. I pledge to excel with integrity.

Student Name 1

Student Name 2

# 0. Team

TODO. List the names of all members of your team. Note: in this report, whenever an instruction/question is prepended with "TODO", please delete the instruction/question in your submission.

# Introduction

TODO. Name your architecture. What is your overall philosophy? What specific goals did you strive to achieve? Can you classify your machine in any of the standard ways (e.g., stack machine, accumulator, register-register/load-store, register-memory)? If so, which? If not, devise a name for your class of machine. Word limit: 200 words.

# Architectural Overview

TODO. This must be in picture form. What are the major building blocks you expect your processor to be made up of? You must have data memory in your architecture. (Example of MIPS: <https://www.researchgate.net/figure/The-MIPS-architecture_fig1_251924531>)

# Machine Specification

## Instruction formats

Two example rows have been filled for you. When you submit, do not include the example types. Add rows as necessary. In your submission, please delete this paragraph.

|  |  |  |
| --- | --- | --- |
| **TYPE** | **FORMAT** | **CORRESPONDING INSTRUCTIONS** |
| R | 1 bit type, 3 bits opcode, 1 bit funct, 1 bit operand register, 1 bit operand register, 2 bit destination register | and, add, sub, etc. |
| B | 1 bit type, 3 bits opcode, 1 bit operand register, 1 bit operand register, 3 bit address | beq, bne, etc. |
|  |  |  |

## Operations

An example row has been filled for you. When you submit, do not include the example type. In the name column, be sure to also add the definition of what the example actually does. For example, "lsl = logical shift left" would be an appropriate value to put in the name column. In the bit breakdown column, add in parenthesis what specific values the bits should be in order. X indicates that it will be specified by the programmer's instruction itself (i.e. specifying registers). In the example column, give an example of an “assembly language” instruction in your machine, then translate it into machine code. Add rows as necessary. In your submission, please delete this paragraph.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NAME** | **TYPE** | **BIT BREAKDOWN** | **EXAMPLE** | **NOTES** |
| and = logical and | R | 1 bit type (0) bits opcode (010), 1 bit funct (1), 1 bit operand register (X), 1 bit operand register (X), 2 bit destination register (XX) | # Assume R0 has 0b0001\_0001  # Assume R1 has 0b1001\_0000  and R0, R1, R2 ⇔ 0\_010\_1\_0\_1\_10  # after and instruction, R2 now holds 0b0001\_0000 | This is a completely bogus example, since this implies that there are only 2 possible operand registers and 4 possible destination registers.  Mention special things like implied destination register (i.e. stack) or special notes here. |
|  |  |  |  |  |

## Internal Operands

TODO. How many registers are supported? Is there anything special about any of the registers (e.g. constant, accumulator), or all of them general purpose?

## Control Flow (branches)

TODO. What types of branches are supported? How are the target addresses calculated? What is the maximum branch distance supported? How do you accommodate large jumps?

## Addressing Modes

TODO. What memory addressing modes are supported, e.g. direct, indirect? How are addresses calculated? Give examples.

# Programmer's Model [Lite]

TODO. 4.1 How should a programmer think about how your machine operates? Provide a description of the general strategy a programmer should use to write programs with your machine. For example, one could say that the programmer should prioritize loading in the necessary values from memory into as many registers as possible, then perform calculations. Another approach could be loading and writing to memory in between every calculation step. Word limit: 200 words.

TODO. 4.2 Can we copy the instructions/operation from MIPS or ARM ISA? If no, explain why not? How did you overcome this or how do you deal with this in your current design? Word limit: 100 words.

# Program Implementation

An example Pseudocode and Assembly Code has been filled out for you. When you submit, please delete the example along with this paragraph.

## Example Pseudocode

# function that performs division

mul\_inverse(operand):

divisor = operand

dividend = 1

result = 0

counter = 0

while counter != 16:

if dividend > divisor:

dividend -= divisor

result = (result << 1) || 1

else:

result = (result << 1)

dividend <<= 1

counter += 1

return result

## Example Assembly Code

# Do not try to understand this code. It is bogus code, but a good example of what to submit.

# loading divisor

load R0, %0010 # 0010 = location of the divisor in memory

load R1, %0100 # 0100 = location of the dividend in memory

add R0, R1, R2 # R0 + R1 => R2 adding the divisor and the dividend together

...

# more assembly code

...

# note that this may be several pages long. The teaching staff will not be verifying correctness of your assembly code for Milestone 1.

## Program 1 Pseudocode

TODO Note: this will be completed for Milestone 3, but start thinking about it actively now.

## Program 1 Assembly Code

TODO Likewise, Milestone 3.

## Program 2 Pseudocode

TODO

## Program 2 Assembly Code

TODO

## Program 3 Pseudocode

TODO

## Program 3 Assembly Code

TODO