Lisandro Nunez
Benjamin Chau

Help was received from other classmates and TAs during office hours about some of the specific requirements for certain components such as the Opcode operations.

We primarily stayed with the design plan that we initially brainstormed for the assignment. We correctly implemented several components of our program such as taking in input, representing the Universal Machine using standard types and structs, and the Opcode instructions such as conditional move, multiplying, loading, loading, and dividing.

The architecture of our program is defined as follows:

- ➔ Universal Machine **struct** Key data structure
  - ◆ Public **registers** vector of type 32 Key data structure
  - ◆ Public **segments** 2D vector of type u32 Key data structure
  - ◆ Public program **counter** of type u32
  - ◆ `new() -> UniversalMachine struct`
    - instantiates new UniversalMachine struct
- ➔ Rumload
  - ◆ `load(input) -> Vec<u32>`
    - Uses std::io to read instructions from .um files
    - Returns a vector containing instructions
- ➔ Disassembler
  - ◆ `mask(bits) -> u32`
  - ◆ `get(field, instruction) -> u32`
  - ◆ `disassemble(&UniversalMachine, instruction) -> String`
    - reads opcodes and values from word
    - Calls appropriate instruction from opcode, and passes in values
    - returns instruction called as String
- ➔ Instructions
  - ◆ All functions that perform operations modify the public **registers** vector and the public **segments** 2D vector in a **UniversalMachine struct**
  - ◆ UniversalMachine structs are passed in by reference
  - ◆ `mov(&UniversalMachine, A, B, C)`
    - if r[C] != 0, then set r[A] = r[B]
  - ◆ `load(&UniversalMachine, A, B, C)`
    - Load word at segments[r[B]][r[C]] into r[A]
  - ◆ `store(&UniversalMachine, A, B, C)`
    - Store word at r[c] into segments[r[A]][r[B]]
  - ◆ `add(&UniversalMachine, A, B, C)`
    - Compute r[B] + r[C] and store sum in r[A]

- `mul(&UniversalMachine, A, B, C)`
  - Compute r[B] * r[C] and store product in r[A]
- `div(&UniversalMachine, A, B, C)`
  - Compute r[B] * r[C] and store product in r[A]
- `nand(&UniversalMachine, A, B, C)`
  - Compute ~(r[B] ^ r[C]) and store result in r[A]
- `halt()`
  - Stop the universal machine
- `map(&UniversalMachine, B, C)`
  - Create new segment with number of words equal to value in r[C]
  - Initialize new segments with zeroes
- `nand(&UniversalMachine, C)`

For timing how long it takes to run 50 million instructions, we used the bash command `time` for Linux. We ran it on a file containing 80 million instructions and the terminal outputted 0.74 seconds. Since 50 million is 62.5% of 80 million, we took 62.5% of 0.74 seconds to get 0.4625 seconds to run 50 million instructions.

We spent approximately 8 hours analyzing the assignment, 10 hours preparing the design as well as the document associated with it, and 25 hours on programming and solving the problems after our analysis.