**SPL-1 Project Report , 2022**

**Assemulation**

**A MIPS Assembly Simulator**

Submitted by

***Swarna Islam***

**BSSE Roll No. : 1214**

**BSSE Session: 2019-2020**

Submitted to

***Dr. Zerina Begum, Professor***

****

**Institute of Information Technology**

**University of Dhaka**

[29-05-22]

**1. Introduction:**

MIPS Assembly Simulator, a software project, simulates the state of the memory and register after executing instructions written in MIPS assembly language.

For this project I have mostly followed MARS simulator.

After executing the program six segments will be simulated:

1. Static memory or data memory that is stored in data segment.
2. Binary representation of each label in text segment.
3. Text segment where:
4. Translation of each pseudo instructions will be shown.
5. Unique address in memory for each instruction.
6. The state of 32 registers of 32-bit after execution each instruction.
7. To help user visualize the concept of procedure calling in MIPS, the state of stack memory after each frame is added will be demonstrated.
8. Around 26 types of unique bug will be reported to the user.

Today it is most unlikely that a device has a CPU that uses the MIPS architecture. Most people have computers with either ARM or x86. Hence, we need a simulator to run MIPS code.

While learning MIPS assembly language, many students struggle to understand what is happening behind all of those instructions and how are they interacting with the memory and registers. The objective of this project is to help user learn about the interaction of the instructions with memory and registers by simulating their state after each execution.

# **2.Background Study:**

Some prior study was necessary to implement the project:

## **2.1.****MIPS Architecture:**

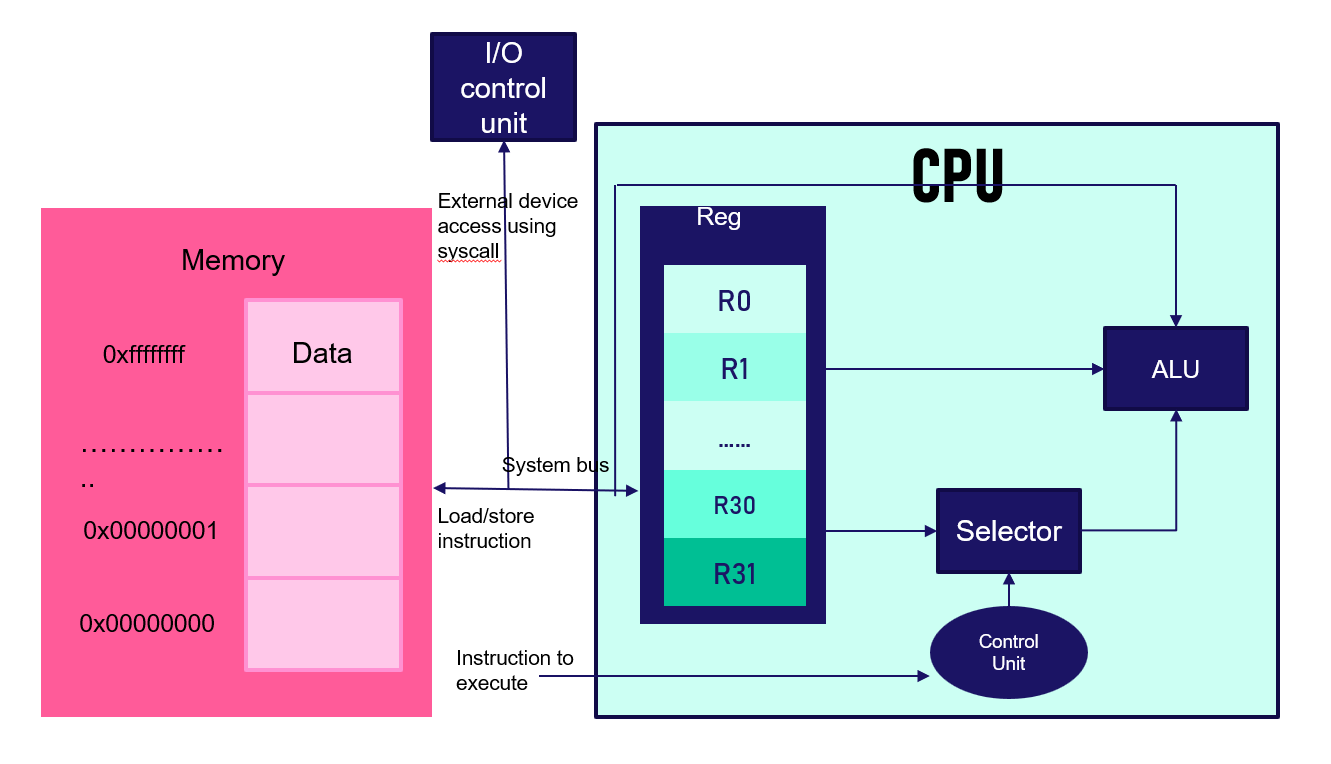


Figure 3.1: Interaction between memory and CPU(Source [Image](https://eng.libretexts.org/@api/deki/files/23411/Screen_Shot_2020-06-28_at_10.33.56_PM.png?revision=1&size=bestfit&width=1080&height=772))

## **2.1.1. Registers:**

MIPS has 32 general purpose registers which are used for memory address or data whenever needed. In order to do anything useful with data values in memory, they must be first loaded into registers. HI and LO registers are separate from the 32 general purpose registers and are not directly addressed. With special instructions mfhi and mflo (Move From HI/LO), their contents are accessed. They are 32-bits each and are present in the Multiply Unit. They hold the 64-bit full result of a 32x32-bit integer for mult instruction. For div instruction, LO holds the value of the quotient and HI holds the value of remainder.

## **2.1.2.Flat memory:**

MIPS architecture is based on a 32-bit flat memory model. All of that memory isn’t available for programmers use. For this project I have implemented:

**Static data** which we will get from the data segment of the program. The size of the elements in this section are assigned when the program is created, thus cannot be changed during the execution of the program.

**Stack** which is used for dynamic data allocation for subprograms or procedure calling by push and pop operations.

**Program text** which is used for storing instructions(excluding code representation of the instructions for simplicity of the project) as a word.

## **2.1.3.Program Counter:**

The Program Counter (PC) contains the address pointer value of the currentinstruction. The value at the pointer is read into the instruction decoder and the program counter is updated to point to the next instruction in each cycle.

## **2.2.Lexical Analysis:**

After taking source code from the given file path, syntaxes are broken into series of tokens, removing comma, whitespace, comments from the source code. These token are further examined to detect valid label, valid source and destination register, valid integer within range for a specific instruction, valid instruction and to report error if the tokens are not valid.

## **2.3. Syntax analysis:**

Syntax analysis is performed to determine whether the tokens created during lexical analysis are in proper order as per their usage. The correct sequence of a set of keywords, which can yield the desired result, is called syntax.

## **2.4. Different Types of Instructions:**

Layout of an instruction is called the instruction format. Only 3 different formats exist. R-type, J-type and I-type. About 41 instructions are implemented in this project. Subtle differences from one instruction to another is also considered. For example, overflow trap, sign and zero extension, translation of pseudo instructions etc.

## **2.5.Algorithm:**

## **2.5.1.Binary Search tree:**

In computer science, a binary search tree, is a rooted binary tree data structure whose internal nodes store a key greater than all the keys in the node's left subtree and less than those in its right subtree. BST is used to detect if a label was declared before by matching the already inserted label in the BST and to see if an instruction is valid.

## **2.5.2.Hash Table:**

In computing hash table is a data structure that can map keys to values. A hash table uses a hash function to calculate an index(hash code) into an array of buckets, from which the desired value can be found. Hash table is used to store the data segment’s static variables and to store the corresponding memory address denoted by each labels in text segment. Polynomial rolling hash function is used to calculate the hash value of the labels.

# **3.Project Description:**

After getting the source code it will search for labels in data and text segment and also detect if there is any repetition of declaration. Then for each line of instructions in text segment comments, comma, white space will be removed and each instruction will be trimmed. Then the trimmed instruction will be preprocessed for prior validity check.

After getting the fully trimmed and translated version of the previous source code execution of the instructions will be simulated. During this further error will be reported.

## **4.Implementation and Testing:**

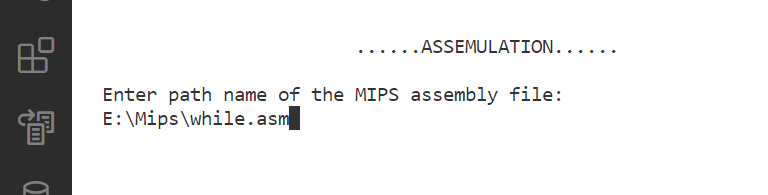
This project is dependent on different modules. Each module has their respective works to perform.

Split module is used for tokenizing. Algorithm module is used to store different types of algorithm implementation on which data structure will act. Definition is used to define basic parameters to asses perform other modules work. Preprocess module is used for preprocessing instructions before actually execute them. This module will perform the job of register validity check, pseudo translation etc. These are the some of the important modules.

# **5.User Interface:**

## **5.1.Input from the user:**

User simply have to give the file name of the existing MIPS assembly program. Then the project will start tokenize the source code at the given file path.



## **5.2.Output for the given source code:**

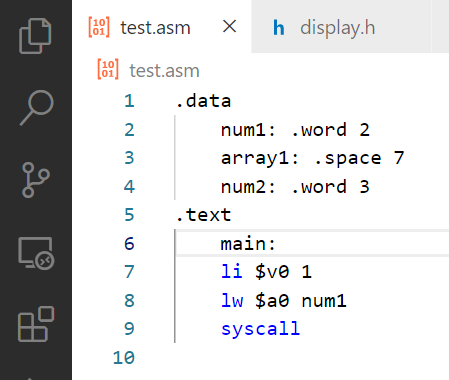
Users have to write both the text and data segment in the source code.

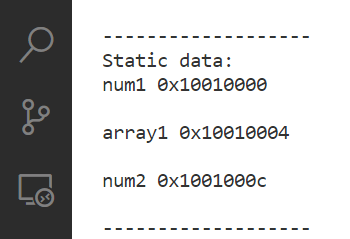
If the source code is written without any bug, after displaying the memory address of the static variables in the data segment and the unique memory address denoted by the labels in the text segment, the state of the registers and stack memory will be simulated for the currently executing instruction.

## **5.2.1.Bug report:**

It will first check for the valid format of label both in data in text segment. Also repetition of the same label will also be detected. Incorrect format of an instruction will be reported. This includes integer range for a particular instruction, invalid source or destination register, address out of bound, invalid word alignment, invalid keyword etc.

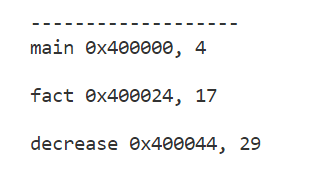
## **5.2.2.Data segment:**



Memory address of each static data will be shown following correct word alignment. 

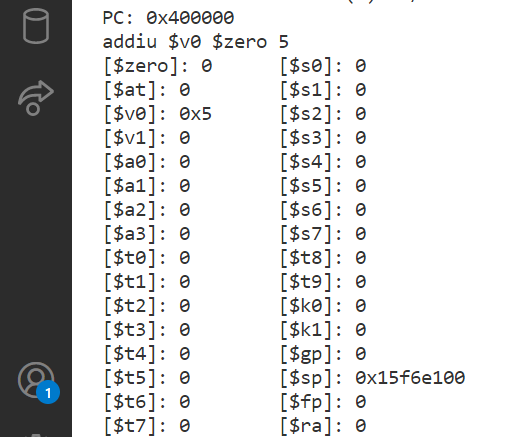
## **5.2.3.Text segment:**

After removing every blank space, unnecessary white space, comma, comments each of the remaining instructions(each fully translated) will be given a unique memory address starting from 0x400000(the start of text segment). Labels of text segment will be mapped with their corresponding instruction address. Instruction line in the source code will be also displayed.



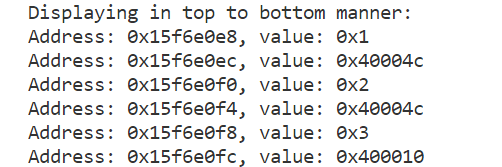
## **5.2.3.Register:**

After executing each instruction, state of the 32 registers will be simulated.



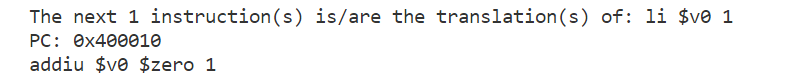
## **5.2.4.Memory stack:**

After each push operation, memory will be allocated in the stack memory. Stack grows downwards. For memory limitation stack memory has started from 0x15f6e100 until it reaches program text(heap memory is avoided in this project for simplicity). The address of the top stack memory is stored in stack memory. After pushing into the stack it decreases, and after popping from stack it increases.



## **5.2.5.Translation of pseudo instruction:**

The actual instruction of each pseudo instruction will be displayed.



# **6.Challenges:**

Working with multiple header files was challenging as there are some restrictions connecting them. For which I have to learn about include guarding.

Translation of a pseudo instruction can vary according to different criteria. For example, for 16-bit unsigned integer, ori will be simply executed as ori, but for 32-bit immediate integer ori will be translated into three different instructions(lui, ori and or) because of zero extension.

Tokenization was also challenging as in this project it is ensured as much as possible that user is comfortable using this simulator so they can write their source code without a strict manner.

Implementing binary search tree was hard because I have to work with type non-specific data using template. Implementing hash table to get the most out of it was also challenging.

**7.Conclusion:**

To make this project more versatile, more instructions can be included. Graphics user interface can be provided. I had learnt a lot about MIPS instruction architecture, memory-register interaction etc.

# **Reference:**

[1] <https://eng.libretexts.org/Bookshelves/Computer_Science/Programming_Languages/Introduction_To_MIPS_Assembly_Language_Programming_(Kann)/02%3A_First_Programs_in_MIPS_Assembly/2.02%3A_MIPS_and_Memory>

[2]

<https://stackoverflow.com/questions/29284428/in-mips-when-to-use-a-signed-extend-when-to-use-a-zero-extend#:~:text=2)%20ANDI%2C%20ORI%2C%20XORI%20both%20use%20zero%2Dextend.&text=It%20clearly%20mentioned%20the%2016%2Dbit%20immediate%20is%20signed%2Dextend>.

[3]

<https://stackoverflow.com/questions/72038539/exact-difference-between-mul-and-mulu/72046543#72046543>

[4]

<http://matthews.sites.truman.edu/files/2019/11/pseudoinstructions.pdf>