Project Documentation

AND

Test Cases

For

MIPS - PARSER SIMULATOR Using Java Script

Prepared By Yash Kudesia

PART - 1

Documentation

Prepared By Yash Kudesia

1: Introduction

The MIPS Parser Simulator, is JavaScript version of the Simulator written in JAVA.

It works only for the MIPS assembly language and try to simulate the control flow of MIPS program with the use of Registers and Memory Visualization.

Languages Used:

HTML and CSS

JavaScript

External Libraries:

File Saver .js

2:Overall Description

Top Menu	Tab Menu
● File	Editor
● Edit	Execute
● Run	Codes
● Help	Help

2.1)-File:

It contains following options:

a)-New:

It is used to create the new editable file with the predefined name “mips1.asm”.

It is implemented as:

It uses a counter named “count” which is used to assign the name of the file and gets incremented on every click of ‘New’ option. After the file is shown on the page you have to click on it to enable the text editor. The text editor is linked with the files with a variable named

“lastbuttonID” which gets updated on on-change event of text-Area and on-click of file name.

b)-Open:

It is used to open a file from the local disk space of User.

It is implemented as:

It uses an input file tag with type= ‘file’ and the display attribute is set to none for look and feel of the ‘Open’ option. On the click of Open option the input filed is set to ‘clicked’ using JavaScript. Inside the on-click listener of input tag, File Reader API (Inbuilt) is used for fetching the file properties and the file content to properly set the values of variables required.

c)-Close and Close All:

These options are used to close file/files currently opened in the tab. All the files which are closed can be opened on Reloading or refreshing the tab.

It is implemented as:

It uses the DOM property of Java-Script for removing the child file that is selected, using it’s parent. The text of file remains saved, to get restored on refreshing or reloading.

d)-Save and Save As:

These options are used to save a file to the local disk space of User.

It is implemented as:

It uses an save As function inbuilt in JavaScript to save the file that is selected with its proper content in the local disk. The MIME used here is type/asm. Also the encoding used to save the file is UTF-8.

2.2)-Edit:

It contains following options:

a)-Undo & Redo:

These options are used to forward or rewind the changes made by the user on a particular file.

It is implemented as:

It uses an stack and two counters associated with each file you create which stores value of text-Area on-change. Every time the text-Area value gets changed, the changed value gets pushed on stack and the stack top incremented and undo_redo_counter sets to stack top ,for tracking the proper and latest changes for undo/ redo operation. On Undo/Redo button click, the value of text-Area value will get over written by the stack [undo_redo_counter] entry and the counter either decreases (in undo) or increases (in redo).

b)-Select All:

It is used to copy the whole text inside the text-Area.

It is implemented as:

It uses Java-Script inbuilt execute(“copy”) option. First the text-Area is selected using document.select() and then the execute command.

c)-Cut Copy and Paste:

These options are used to cut copy and paste the text selected on the text-Area.

It is implemented as:

It uses an copy_text_buffer for storing the value of text selected by user on cut or copy and then this value will get pasted at the desired cursor location.

Both cut and paste option uses the get selected area function to get the area selected by user to perform the respective operation.

2.3)-Run:

It contains following options:

a)-Assemble:

It is used to read the file and split it into valid instruction (leaving comments) and initialize the

text-Segment, data-Segment with the initial values as per program specification.

It is implemented as:

It split the program line by line and stores in the array and then the array is processed character by character and the pure code is stored n another array for calculation purposes. After this the setTextSegment is called and the new array with pure code is processed and the new table entries is created with the valid corresponding ID’s.

b)-Run:

It is used to run the code at once and highlight the result.

It is implemented As:-

The Step function is called inside the for loop.

c)-Step:

It is used to run the code provided step by step.

It is implemented as-:

It , one by one calls each ID assigned in Assemble option and process it and highlight the specified row element of table and the result part in Registers.

d)-Reset:

It is used to reset the registers and flags.

It is implemented as-:

The flags and Registers sets to zero.

e)-Clear and toggle Break-Points:

It is used to set the breakpoints on the code lines of which we want to verify the intermediate Result.

It is implemented as-:

It is not yet implemented.

2.4)-Help:

It contains following options:

a)-Help:

Here the short description is given to each function using tables and list.

b)-About:

Here the CSS modal is used for the short info about the MIPS Parser application.

It is implemented As:-

Here after the about option a container is present with display attribute set to none. The container on clicking the about option became visible and it takes the whole container width with z-index 1 and the opacity of background is set to 0.5 to focus the modal.

2.5)-Editor:

This tab mainly contains the text Area and run IO messages.

2.6)-Execute:

This tab mainly contains the tables for data and memory visualization.

2.7)-Codes:

This tab mainly contains three read-only files which can be used as an example code for MIPS program with the function to copy them.

2.8)-Help:

This tab contains the same contents as the help sub-option in Help option.

3 : Working Description

Here functionalities of all the functions defined inside the program is discussed.

3.1)-Index.js->>

MydropdownFunction

assemblerlistner

fileonload

Constructor

init

myfunctionlist

clicklistner

textareachanges

about_popup

close_popup

codes

init_codes

codes_show

copy_code

selectAll

getSelectedText

paste

typeInTextArea

Copy

Cut

removeOutTextArea

saveAs_function

close_fun

closeAll_fun

open_window

setinputfiletextArea

exit_window

updateStack

undo_fun and redo_fun

3.2)-Execute.js→>

reset

step_util Run_execute

process Result

init_execute

setTheInitialArea

ClearParserText

3.1.1)-MydropdownFunction:-

Function Introduction:-

This function is on-click listener of Top Menu. The function is used to make the drop-down visible or invisible. The parameter passed here signifies the drop-down menu which is to be toggled for display

Function Implementation:-

Global Variable , lockDropDown ,used as a lock for the function and its default value is -1. LockDropDown sets to parameter passed. LockDropDown ensures that only one drop-down remains active at once.

Local variable ,status, is used to close the drop-down on the second click of the same option.

Working:-

Let’s say the File option triggers the call to this function, so initially the lockDropDown is -1 and according to parameter passed, the drop-down

became visible and and lockDropDown value sets to passed parameter. Now on second trigger of this function, the passed parameter is checked against the lockDropDown and if true, means that Same option (File) is clicked again and the file drop-down that is currently active becomes deactive again and sets value of status to 1 which ensures that in the same call the drop-down will not get activated again.

3.1.2)-Assembler-listener:-

Function Introduction:-

This function is on-click listener of Assemble sub-menu of Run option. It is responsible for activating the other Run options.

Function Implementation:-

The function fetches all the other option with their ID’s and set their style attributes to visible and finally call another function to initialize the table.

3.1.3)- File On load:-

Function Introduction:-

This is the function that triggers whenever the window loads. This function is whole sole responsible for restoring the data written on files and required variables before reloading.

Function Implementation:-

The function Just restores the following values using session storage

Restores-:

- Copy_text_buffer (copied material from code section)
- lastbuttonID (Id of file accessed)
- user_file_counter (number of files user has opened)
- user_file_index (index from where next user file details to store)

3.1.4)-Constructor:-

Function Introduction:-

This function creates a structure and initialize the value with the parameter passed.

Function Implementation:-

This function initialize the following variables.

Initializes:-

- name (name of file)
- text (content of file)
- stack (undo_redo stack)
- undo_redo_counter
- stack top (for undo redo stack)

3.1.5)-init:-

This function is used to create the file variable and passes the name string to Constructor for initialization. It also initialize the file content within itself and pushes the basic entry in undo redo stack.

Here the count of files (user opened or created through application) is stored on Session Storage.

3.1.6)-My function List:-

Function Introduction:-

This function is on-click listener of New sub-menu of File option. It also triggered by file on load function while restoring.

It mainly creates New file (both User opened or created through application)

Function Implementation:-

It uses the global variable count to check whether the no. of file is in the current capacity of container or not. It then calls init which initialize the file with the parameter passed to it. In the end it creates the file as Button, sets its Id ,sets it on-click listener and append it to the fixed located container.

3.1.7)- Click Listener:-

Function Introduction:-

This function is a on-click listener for the file name display as buttons in application. It sets the lastbuttonID variable to the ID passed and update

the text value of text-Area to the content of button/File text.

3.1.8)-text Area Changed:-

Function Introduction:-

This function is a on-input listener for the text-Area, which triggers on each change in the value of text-Area. And on each undo /Redo function.

It updates the text of files with the current text-Area value and store it in session storage.

At last it calls update Stack if text Area Changed is not called through undo/redo functions.

3.1.9)-About popup:-

Function Introduction:-

It triggers when about sub-menu in Help option is clicked.

It sets the click listener for the close buttons on about popup.

3.1.10)- Close Popup-:

Function Introduction:-

It closes the ABOUT POPUP on click of close buttons on the popup itself.

3.1.11)- Code:-

Function Introduction:-

This is a constructor for the code text examples

3.1.12)- Init Codes:-

Function Introduction:-

This function initializes the examples codes and push them in an array foe further use.

3.1.13)- Codes Show:-

Function Introduction:-

It initializes he codes array using init codes function and shows the codes in text-area.

Function Implementation:-

It uses the parameter passed to it, to determine which code example to display in text-Area.

3.1.14)-Copy Code:-

Function Introduction:-

This function is the on-click listener of the copy button in the execute tab at the bottom left corner.

Function Implementation:-

It uses the global variable copy text buffer to store the values copied (so that it can be restored on page reloading). After storing the value, inbuilt copy command is executed to copy the content of text-Area.

3.1.15)-Select-All:-

Function Introduction:-

this function is on-click listener of select All sub-menu in Edit option. It is used to select All the content currently inside the text-Area.

Function Implementation:-

It uses the global variable copy text buffer for storing the value of text-Area to perform further options.

3.1.16)- Get selected Text:-

Function Introduction:-

This function is used in cut and copy options. It is used to get the value of text that is selected.

Function Implementation:-

It takes the element from which the selected text is to be returned as a parameter. It checks, if the selection starting point index is number or not. So if it is a number, it returns the selection and if not it checks it against undefined value. So if it is undefined return the empty string and if not return the range created (default).

3.1.17)- Paste:-

Function Introduction:-

This function in the on-click listener of paste sub-menu of Edit option. It just get the stored text (either form cut or copy) and paste on the cursor position or overwrite the selected text.

As the text-Area gets changed after this, so the text area change method is called.

3.1.18)- type In text Area:-

Function Introduction:-

This is a utility function for paste option. It takes the selection string and append the pasted string in between the start and end of selection.

Function Implementation:-

It takes the element (that s to be processed) and text (that is to be pasted) as a parameter. It split the content of element is three parts (1- before selected part 2-selected part 3- after selected part) and concat the string in between the before and after part.

3.1.19)-Copy:-

Function Introduction:-

This function in the on-click listener of copy sub-menu of Edit option.

Function Implementation:-

It just take the selected text using get Selected text and stores it in the global variable copy_text_buffer.

3.1.20)- Cut:-

Function Introduction:-

This function in the on-click listener of cut sub-menu of Edit option.

Function Implementation:-

It just take the selected text using get Selected text and stores it in the global variable copy_text_buffer (also in local variable for calculation). It then removes the text that is selected using remove Text area function.

3.1.21)- Remove out Text Area:-

Function Introduction:-

This is the utility function for removing text area that is selected and is used in the cut sub-menu of Edit option.

3.1.22)- Save As function:-

Function Introduction:-

It uses the functionality of File Saver .js file and save the file on the local disk space of user.

Function Implementation:-

It create a Blob object and pass the text and MIME structure to it. It then uses the inbuilt save As function of JavaScript to save the file.

3.1.23)- Close fun :-

Function Introduction:-

It takes the file and remove it from the container it is placed.

Function Implementation:-

It makes use of global variable last Button id and remove the file id in lastbuttonID by fetching it parent But it doesn’t remove the text and the file name from the storage, to get reloaded on windows reloading or refreshing.

3.1.24)- Close All Fun:-

Function Introduction:-

It takes all the files and remove it from the container it is placed.

Function Implementation:-

It iterate over the container containing the files and removes it one by one. But it doesn’t remove the text and the file name from the storage, to get reloaded on windows reloading or refreshing.

3.1.25)- Open Window:-

Function Introduction:-

This is a on-click listener of open sub-menu of File option. It makes the input file tag to become active to proceed the file opening process.

3.1.26)- set input file in text-area:-

Function Introduction:-

It is the on-click listener for the input tag which is responsible for opening the file.

Function Implementation:-

First, the file selected by user get stored in files variable and then the first entry in file variable (first entry means first file, there can be multiple file selected by user). The reader from File Reader API sets it’s events and read file as text. On loading the reader by file, the file contents stored in local variable which is then passed to myfunctionlist , which creates user file with file name passed and initialize the content with text passed.

3.1.27)-Exit Window:-

This functionality cannot be implemented in the application. As we cannot close the window using JavaScript which is not opened by it.

3.1.28)-Update Stack:-

Function Introduction:-

The function is triggered by text-area-changed method It stores the latest content of text-Area in stack.

Function Implementation:-

If the content of text-area after changed is same as previous value, then storing it will create an overhead. So if, content is not same, then store it in the stack.

After storing, the new value needed to be saved in file, so call the text-area-changed method again with a parameter which ensures that again the update stack will not be called.

3.1.29)- Undo/ Redo function:-

Function Introduction:-

These function are the on-click listener for the undo/ redo sub-menu in Edit option. It undo or redo the content in text-area

Function Implementation:-

Here after restoring the respective value (for undo and redo function) the text-area-changed method called to ensure that the data gets saved in files.

3.2.1)- Reset:-

Function Introduction:-

All the necessary global variables gets restarted here

3.2.2)- Clear Parser Text:-

Function Introduction:-

Here the content of parser text area gets removed or cleared

3.2.3)- init_execute:-

Function Introduction:-

This is used to find the index of data and text segment .It also performs the separation of defined variables in program and stores it in global variables.

Function Implementation:-

Here the set The Initial Area is called to initiate the tables in Execute tab

3.2.4)-set the Initial Area:-

Function Introduction:-

Here the code in the text segment is evaluated and each line get pushed in stack for further use.

Function Implementation:

Here each line of code is processed to slice out the pure code (opcode and operands). First all the comments and white spaces is removed and the pure code is pushed into global stack text_segment_code. After this, the entry for table is created ,ID’s are assigned and then appended to the table. At last some initialization is done to ensure proper functioning of variables.

3.2.5)- step util :-

Function Introduction:-

This function is a on-click listener for step sub-menu and option in Run option and Execute tab respectively.

It is used to get the code line by line , process the result, update the program counter and highlight the PC and the

current execution line and their corresponding result.

Function Implementation:-

The function process Result is called for processing the given line of code and sent back the status which signifies whether the line has been executed properly or not.

3.2.6)- Process Result:-

Function Introduction:-

This function is used by step_util for processing the result. It takes each word and process it with switch case.

3.2.7)- Run_execute:-

Function Introduction:-

It is a on-click listener of Run sub-menu of Run option.

Function Implementation:-

It calls the step util repeatedly until processing is finished, which is ensured by status local variable

PART - 2

Test Cases

Prepared By Yash Kudesia

Test Name	Execute Tab	**Test Case ID**	GUI_1
Test Case Description	Responsiveness of Design	**Test Priority**	High
Pre-Requisite	NA	**Post-Requisite**	NA

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Browser**	**Test Result**
1	Launch application	Execute.html	Execute Tab	Execute Tab	FF 66.0.4	Pass
2	Change the Screen Size	1271*637	*Text and Registers Segment takes 40% (508) of width * Data Segment Takes 80% (762) of width *Step and Run buttons aligned in center in container	*Text and Registers Segment takes 40%(508) of width * Data Segment Takes 80% (762) of width *Step and Run buttons aligned in center in container	FF 66.0.4	Pass
3	Change the Screen Size	784*384	*Text and Registers Segment takes 40% (300) of width * Data Segment Takes 80% (467.2) of width *Step and Run buttons aligned in center in container	*Text and Registers Segment takes 40%(300) of width * Data Segment Takes 80% (467.2) of width *Step and Run buttons aligned in center in container	FF 66.0.4	Pass
4	Change the Screen Size	984*384	*Text and Registers Segment takes 40% (380.8) of width * Data Segment Takes 80%(587.2) of width *Step and Run buttons aligned in center in container	*Text and Registers Segment takes 40% (380.8) of width * Data Segment Takes 80% (587.2) of width *Step and Run buttons aligned in center in container	FF 66.0.4	Pass

Test Name	About Option	**Test Case ID**	GUI_2
Test Case Description	Responsiveness of About popup	**Test Priority**	High
Pre-Requisite	NA	**Post-Requisite**	NA

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Browser**	**Test Result**
1	Launch application	Index.html	Editor Tab	Editor Tab	FF 66.0.4	Pass
2	Change the Screen Size	1271*637	About popup takes 70% of width (892.5)	About popup takes 70% of width (892.5)	FF 66.0.4	Pass
3	Change the Screen Size	784*384	About popup takes 70% of width (560)	About popup takes 70% of width (560)	FF 66.0.4	Pass
4	Change the Screen Size	984*384	About popup takes 70% of width (688)	About popup takes 70% of width (688)	FF 66.0.4	Pass

Test Name	Data Persistency - 1	**Test Case ID**	Functionality_1
Test Case Description	File data should persists on reloading or after switching between tabs	**Test Priority**	High
Pre-Requisite	Should be in editor tab	**Post-Requisite**	NA

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Browser**	**Test Result**
1	Launch application	Index.html	Editor Tab	Editor Tab	FF 66.0.4	Pass
2	Create a New File	Click on New Option in File menu	New File with default name (mips1.asm) with default text (Add your code here!) is created.	New File with default name (mips1.asm) with default text (Add your code here!) is created.	FF 66.0.4	Pass
3	Type In some information	Type something in text-area provided	Text-Area reads the input	Text-Area reads the input	FF 66.0.4	Pass
4	Reload the page	Either switch to another tab and come back or reload page directly	Page Reloaded	Page Reloaded	FF 66.0.4	Pass
5	Access the File	Click on the file name	Content written before reloading in the file gets saved in file and is now displayed in text-Area under file name	Content written before reloading in the file gets saved in file and is now displayed in text-Area under file name	FF 66.0.4	Pass

Test Name	Data Persistency-2	**Test Case ID**	Functionality_2
Test Case Description	File Data should persists on (In case of multiple files) reloading or after switching between tabs	**Test Priority**	High
Pre-Requisite	Should be in editor tab	**Post-Requisite**	NA

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Browser**	**Test Result**
1	Launch application	Index.html	Editor Tab	Editor Tab	FF 66.0.4	Pass
2	Create a New File	Click on New Option in File menu	New File with default name (mips1.asm) with default text (Add your code here!) is created.	New File with default name (mips1.asm) with default text (Add your code here!) is created.	FF 66.0.4	Pass
3	Open a User File	Click on Open Option in File menu	File gets opened and temprary stored	File gets opened and temprary stored	FF 66.0.4	Pass
4	Type In some information	Type something in text-area provided	Text-Area reads the input	Text-Area reads the input	FF 66.0.4	Pass
5	Reload the page	Either switch to another tab and come back or reload page directly	Page Reloaded	Page Reloaded	FF 66.0.4	Pass
6	Check the Files	Click on each file name	Content written before reloading in the files gets saved in respective files and is now displayed in text-Area under its file name	Content written before reloading in the files gets saved in respective files and is now displayed in text-Area under its file name	FF 66.0.4	Pass

END OF DOCUMENTATION

References:-

Software Testing Help Website

Source Code of MIPS-PARSER USING JS

Prepared By:-

Yash Kudesia

Project Details:-

Project Domain – CSO

Issue Number – 207

Project Topic – MIPS PARSER -2

Submitted to:-

SRIP@IIITH