ARM SIMULATOR

Project report submitted in partial fulfillment of the requirements for the

Award of the degree of

Bachelor of Computer Applications

by

P SAI SUJAYEENDRA KULKARNI (Regd.No. 164406)

&

S VINOD KUMAR (Regd.No. 164414)

DEPARTMENT OF MATHEMATICS & COMPUTER SCIENCE

SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING, PRASHANTI NILAYAM

MARCH 2019

Dedicated at the Lotus feet of our beloved Bhagawan

SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING

(Deemed to be University)

DEPARTMENT OF MATHEMATICS AND COMPUTE SCIENCE

CERTIFICATE

This is to certify that this project report entitled **ARM SIMULATOR** being submitted by Sri. **P Sai Sujayeendra Kulkarni** and **S Vinod Kumar** in partial fulfillment of the requirements for the award of the degree **Bachelor of Computer Applications** is a record of bonafide project work carried out by them under my supervision and guidance during the academic year 2018-19 in the Department of Mathematics and Computer Science, Sri Sathya Sai Institute of Higher Learning, Muddenahalli campus.

Place: Muddenahalli

Date: 20-03-2019

Asst. Prof Dr. Ajith Padyana

Department of Mathematics and Computer Science,

Sri Sathya Sri Sathya Institute of Higher Learning,

Muddenahalli Campus.

ACKNOWLEDGEMENTS

Our sincere gratitude to the one who has helped us in each and every step, our chancellor BHAGAWAN SRI SATHYA SAI BABA.

Our sincere gratitude towards the Department of Mathematics and Computer Science(DMACS) Department for providing us with an opportunity for developing a project so that we can explore in the field of Computer Science.

We deeply express our gratitude to the HOD of the DMACS Department who has been playing a vital role by giving his valuable support.

We deeply express our sincere thanks to the Director and Deputy Director of the Muddenahalli Campus, for granting us permission to make use of the lab during free timings and also during study hours.

We thank our Project guide Sri. Dr. Ajith Padyana and all other faculty members for supporting, encouraging and also guiding us throughout the development of this project.

We express our sincere gratitude towards our Computer Lab administrator for allowing us to work in the lab during our free time.

We would also like to thank our Co-classmates who have been correcting and giving us critical suggestions throughout the project preparation.

We would also like to thank our senior brother who never turned back our doubts and questions and helped us to their fullest possible.

We also express our gratitude to our parents who has helped us indirectly by motivating us to complete our project work.

TABLE OF CONTENTS

OVERVIEW

INTRODUCTION

ARM processors are a particular type of processors which are made by the ARM holdings PLC. This processor is also known as **RISC (Reduces Instruction Set Computing)** which has a simple central processing unit producing a high quality for the users.

Nowadays ARM processor are widely used in a number of electronic products such as mobile phones, tablets, multi-media players etc…

There are only 25 basic instruction types in this designed processor. All instructions have some condition on them. There can be multiple modes of addressing. They are cheap compared to other processors.

This project is a simple ARM Simulator which performs only arithmetic operations, Load and Store operations. This is a humble attempt to understand the ARM processor with some basic knowledge about 8085 microprocessor. This project has come into shape by keeping 8085 simulator as a base.

Since the instruction set of ARM is not licensed, it can be modified or instruction can be added to it, which makes ARM more popular compared to X86. It is preferred over X86 in smart devices because of its less power consumption.

It is dual-instruction-format architecture which switch between ARM Instruction Set and Thumb Instruction Set which are 32-bit and 16-bit respectively. Every instruction in Thumb architecture has an alternate in ARM Instruction Set.

TOOLS AND TECHNIQUES USED

OPERATING SYSTEM:

- Ubuntu 18.04 LTS (64 bit)

SYSTEM PROPERTIES:

- Memory: 7.7 GB
- Processor: intel core i3-3220 CPU @ 3.30GHZ

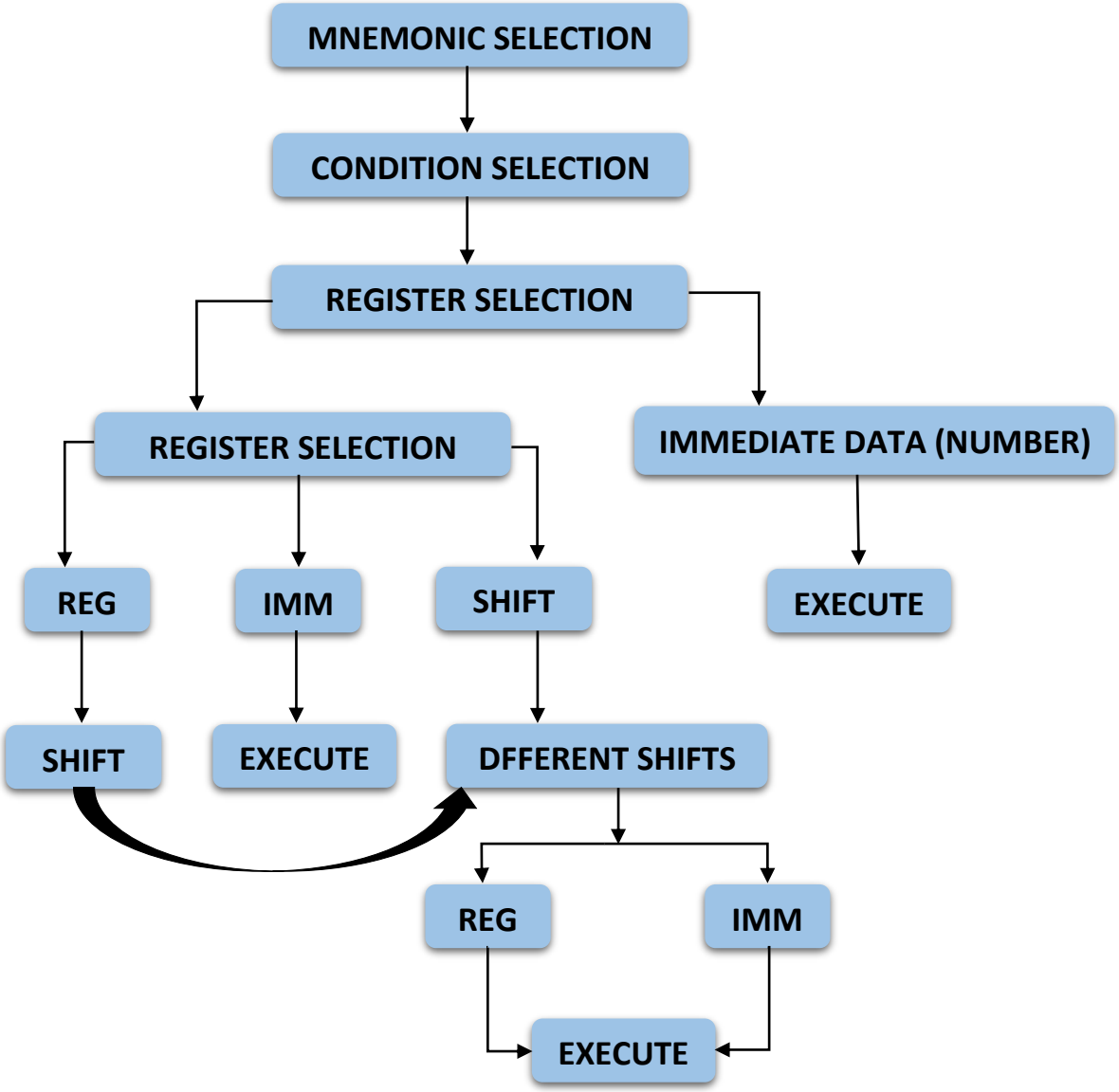
PROGRAMMING LANGUAGES AND TECHNOLOGIES USED:

- HTML
- JAVASCRIPT (as a main tool for achieving the goals of the project)
- CASCADING STYLESHEET
- BOOTSTRAP (used for plugins like modals, responsive tables)

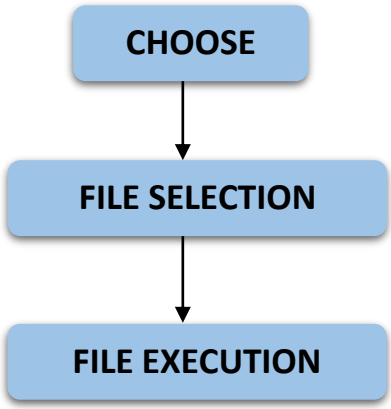
CONTROL FLOW

CONTROL FLOW DIAGRAM:

1) Normal Execution



2) File Execution



FUNCTIONALITIES

Normal Execution Functionalities:

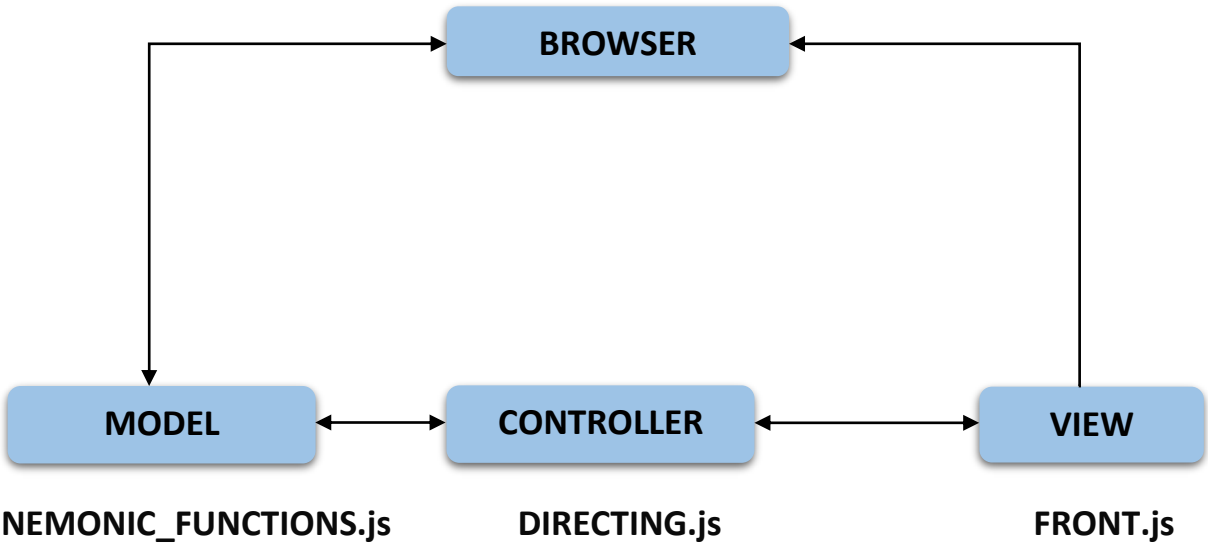
- The user is made sure that he/she doesn't enter an instruction that doesn't fall under the ARM Instruction Set.
- On hover over an element a popup appears explaining the functionality of the particular instruction.
- User has an option of deleting the mnemonic by clicking the back button.
- On execution of an instruction the instruction table (list of instructions that has been executed till that point) pops up.
- If the conditional execution fails, there will be an alert informing that the instruction has not been executed.

File Execution Functionalities:

- The simulator accepts only files of .txt format.
- The instructions have to be written in the way as mentioned

  MOV, NE, R0, #100
  MOV, NE, R1, R0
  ADD, AL, R2, R0, R1
  #
- In the above code each mnemonic and its operands are succeeded with comma and a space (,) except the last one.
- The end of file is recognized with (#).
- All the instructions in the file must be uppercase.
- The immediate value must be preceded with (#).
- By clicking the choose button in the simulator, the option for selecting files gets opened and on clicking at the file, it gets executed.
- If the conditional execution fails, there will be an alert informing that the instruction has not been executed.

ARCHITECTURE



MVC provides a layer of abstraction on top of the core language. Their goal is to help structure the code-base and separate the concerns of an application into three parts:

- **MODEL** (**Business logic**)- Represents the data of the application. This matches up with the type of data a web application is dealing with, such as a user, video, picture or comment. Changes made to the model notify any subscribed parties within the application.

- **VIEW (Presentation logic)** - The user interface of the application. Most frameworks treat views as a thin adapter that sits just on top of the DOM. The view observes a model and updates itself should it change in any way.

- **CONTROLLER (changing state of model and view)** - Used to handle any form of input such as clicks or browser events. It's the controller's job to update the model when necessary (i.e. if a user changes their name).

MVC can be used if

- The same data is being rendered in different ways on the page
- Your application has many trivial interactions that modify data (buttons, switches)
- Much of the viewing or manipulation of data will be within the browser rather than on the server

DESCRIPTION OF FUNCTIONS

Description of Functions in FRONT.js

NAME OF THE FUNCTION	DESCRIPTION ABOUT THE FUNCTION
mnemonicPage()	Displays the mnemonic operations available.
mnemonic(ins)	This function takes ins as a parameter and it contains the mnemonic name. This function is called when the mnemonic is clicked. It writes the mnemonic (ADD,LDR/MUL….) in the instruction panel and creates a track.
changeToConditional(mcins)	This function takes mcins as a parameter and it contains the instruction that is to be displayed in the instruction panel. This function is called by the mnemonic(). It displays the conditional execution instructions.
condition(condition)	This function takes condition as a parameter and it contains the conditional execution mnemonic that's been clicked by user. This function is called when the condition is clicked. It writes the condition(NE/EQ/AL….) in the instruction panel and creates a track.
changeToRegister()	This function is called by the condition(). It displays the Register set.
register(register)	This function takes register as a parameter and it contains the register name. This function is called when a Register is clicked. It writes the register (R0/R1…) in the instruction panel and creates a track. This function plays an important role in directing user in a proper of entering the instruction.
shifter(mcins)	This function takes mcins as a parameter which contains the instruction that's to be written in the instruction panel. This function is called by the register().This displays the button of shifter.
changeToShifter()	This function is called when the shifter button is clicked. This function displays different shifts available. (LSL/ASR…).
shifterValues(shif)	This function takes shif as a parameter it contains the type of shifter that's been chosen by the user. This function gives the option for the user to decide whether the shifter value is an immediate value or a value from the register.

changeToNumAndReg()	This function displays two buttons NUMBER, REGISTER.
changeToNumRegAndShift(mcins)	This function takes mcins as a parameter which contains the instruction that's to be written in the instruction panel. This function displays three buttons NUMBER, REGISTER, SHIFTER.
changeToNumber()	This function allows the user to enter the number that he wishes to store in a register. This function goes through validation where it accepts only numeric values.
ChangeToAddress()	This function allows the user to enter an address where the value of the register specified in the instruction will be stored. This function goes through validation where it accepts only numeric values.
openFile(event)	This function takes in event as a parameter which is created onclick of choose button. This function gets invoked when the user wants to execute a file by clicking the CHOOSE option in the simulator.
number(num)	This function takes num as a parameter which contains the number entered by the user. This function adds the number to instruction panel and modifies the track.
address(addr)	This function takes addr as a parameter which contains the address entered by the user. This function adds the address entered by the user to instruction panel and modifies the track.
back()	This function gets invoked when the BACK button is clicked in the simulator. This removes the last word in the instruction panel and removes the last word in the track. EXAMPLE: ADD, EQ, R0 onclick of BACK button the instruction panel become ADD, EQ, . The track becomes M01, C00, from M01, C00, R.
clr()	This function gets invoked when the CLEAR button is pressed in the simulator. This clears the whole instruction that is present in the instruction panel an d calls the mnemonicPage() function which displays the mnemonic operations available.
validation(num)	This function takes num as parameter which contains a number entered by user.This function gets called when an address or a number is entered. This does a validation where it allows only numeric values to be entered.

removePopover()	Since the popover of the elements are activated on hover. Onclick of an element activates the popover and it doesn't get removed. To solve this problem, the element is captured by its class name and is removed in the next function.
tour()	This function is a kind of reminder for the user who uses the simulator. This gets called onload() of the body. This displays the Do's and Don'ts of the simulator.

Description of Functions in DIRECTING.js

NAME OF THE FUNCTION	DESCRIPTION ABOUT THE FUNCTION
executeFile(ARM)	This function takes a parameter called ARM which is a string array. This variable contains the instruction that's present in the file. This function first creates the track for each instruction and directs each instruction to the direct() function. On successful execution of the file there will be an alert indicating the file has been executed successfully.
createTrack(ARM[i])	This function takes a parameter called ARM[i] which is a string, which contains instruction. There is a kind of unique code for each part of the instruction. The instruction is executed by recognizing this code. In manual execution onclick of each button the track is created. In file execution track is not created therefore this function creates the track for the instruction.
direct()	This is a main function in the simulator which directs to different functions. It checks the conditional execution by calling checkExecution(), it fetches the register values by calling getRegisterValue(), does a shift modification by calling shift(), last it passes the operands the mnemonic function().
operands(ltrack, lword)	The parameters ltrack and lword are the particular unique code and the mnemonic for that code respectively. This function differentiates between register and immediate operands returns the value of it.
getRegisterValue(lword)	The parameter lword contains the part of instruction. In this case it contains the register (R0, R1, …). This function using document. getElementById () accesses the GPR table and fetches the value of the particular register mentioned.

shift(track, word)	The parameters track and word is a string which contains the unique code for the whole instruction and the whole instruction respectively.
mnemoniCall(ltrack, val1, val2)	After fetching the operand values this function directs the values and the operation unique code to the respective mnemonic function. ADD (ltrack, val1, val2). The parameters ltrack is the unique code for the mnemonic operation, val1 and val2 are the two operands for the operation.
writeToTable(val, word)	This function takes the value that is to be written to the register and the register to which it is to be written. This function accesses the GPR table and write the value to the particular register mentioned.
checkExecution(cond)	This function takes cond as parameter which is the conditional execution instruction. This function checks the condition of the conditional execution with the flag registers.

Description of Functions in MNEMONIC_FUNCTIONS.js

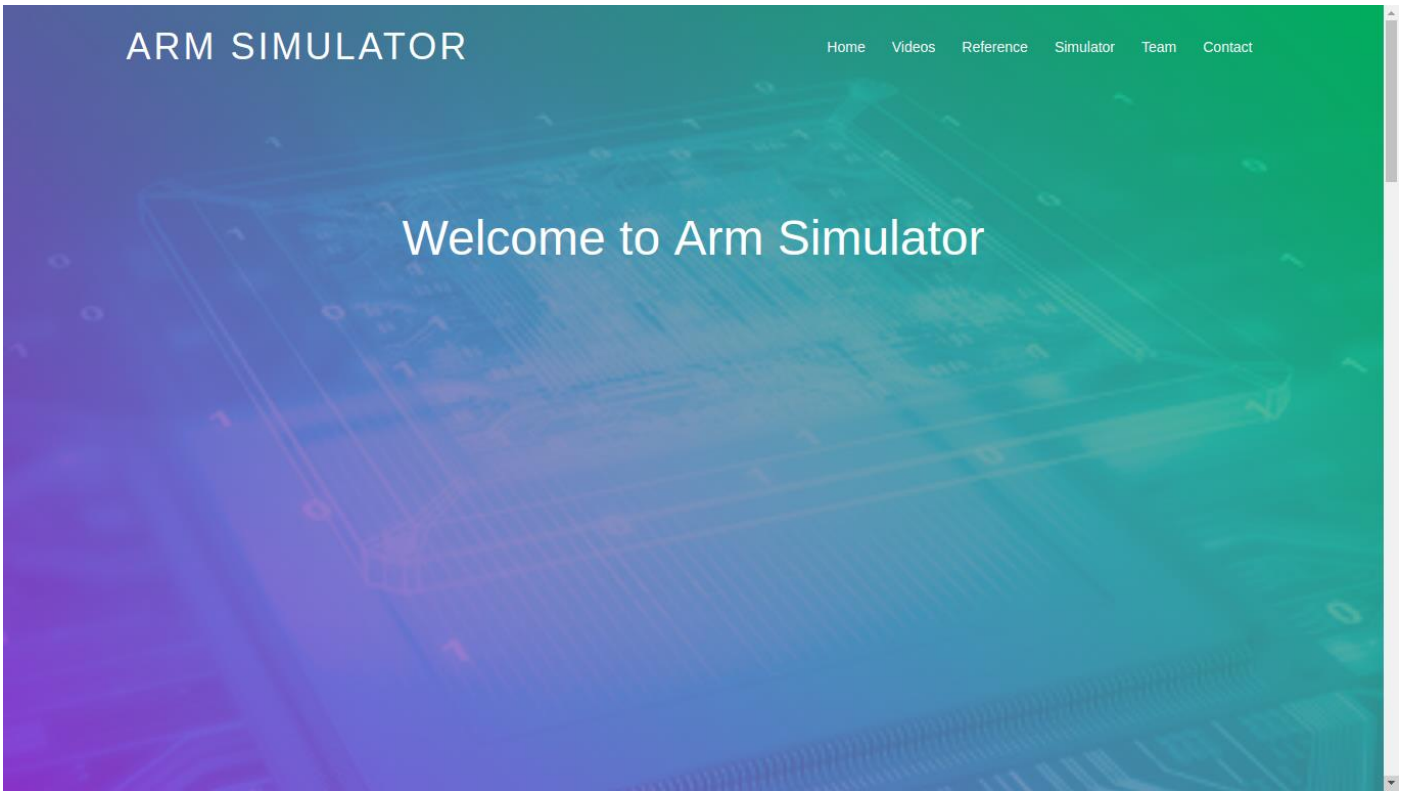
NAME OF THE FUNCTION	DESCRIPTION ABOUT THE FUNCTION
addIns(ins)	This functions takes instruction that's to be executed as its parameter. It writes the instruction in the instruction table and it is called by the direct() function.
addRow(ins)	This functions takes instruction that's to be executed as its parameter. It writes the instruction in the GPR table and creates a row of the general purpose registers.
ADC(n1, n2)	This function takes in n1 and n2 as its parameters which contains the values of register. This function performs ADDITION WITH CARRY operation where it adds the values of two operands and adds the value of carry flag to the result. According to the result the flag registers are updated.
ADD(n1, n2)	This function takes in n1 and n2 as its parameters which contains the values of register. This function performs ADDITION operation where it adds the values of two operands. According to the result the flag registers are updated.
AND(n1, n2)	This function takes in n1 and n2 as its parameters which contains the values of register. This function performs BITWISE AND operation where it does the AND operation for each bit. According to the result the flag registers are updated.
BIC(n1, n2)	This function takes in n1 and n2 as its parameters which contains the values of register. The second variable is passed

	to NOT () function and the BITWISE AND operation is performed with the modified value. According to the result the flag registers are updated.
EOR(n1, n2)	This function takes in n1 and n2 as its parameters which contains the values of register. EOR (Exclusive OR) operation is performed between these two operands. According to the result the flag registers are updated.
MOV(n1)	This function takes in n1 as its parameter which contains an immediate value or value of a register. This function performs a MOVE operation. According to the value of n1 flag registers are updated.
MVN(n1)	This function takes in n1 as its parameter which contains an immediate value or value of a register. Since the operation MOVE NOT the operand is passed to NOT () function. According to the value of n1 flag registers are updated.
ORR(n1, n2)	This function takes in n1 and n2 as its parameters which contains the values of register. This function performs BITWISE OR operation where it does the OR operation for each bit. According to the result the flag registers are updated.
RSB(n1, n2)	This function takes in n1 and n2 as its parameters which contains the values of register. This function performs REVERSE SUBTRACTION (n2-n1). According to the result the flag registers are updated.
RSC(n1, n2)	This function takes in n1 and n2 as its parameters which contains the values of register. This function performs REVERSE SUBTRACTION WITH CARRY. First (n2-n1) is performed, later the carry flag is subtracted from (n2-n1). According to the result the flag registers are updated.
SBC(n1, n2)	This function takes in n1 and n2 as its parameters which contains the values of register. This function performs SUBTRACTION WITH CARRY. First (n1-n2) is performed, later the carry flag is subtracted from the result of (n1-n2). According to the result the flag registers are updated.
SUB(n1, n2)	This function takes in n1 and n2 as its parameters which contains the values of register. This function performs SUBTRACTION of two operands (n1-n2). According to the result the flag registers are updated.
TEQ(n1, n2)	This function takes in n1 and n2 as its parameters which contains the values of register. This function performs BITWISE EOR (Exclusive OR). The difference between EOR() and TEQ() is the result is not written to the destination register but according to the result the flag registers are updated.

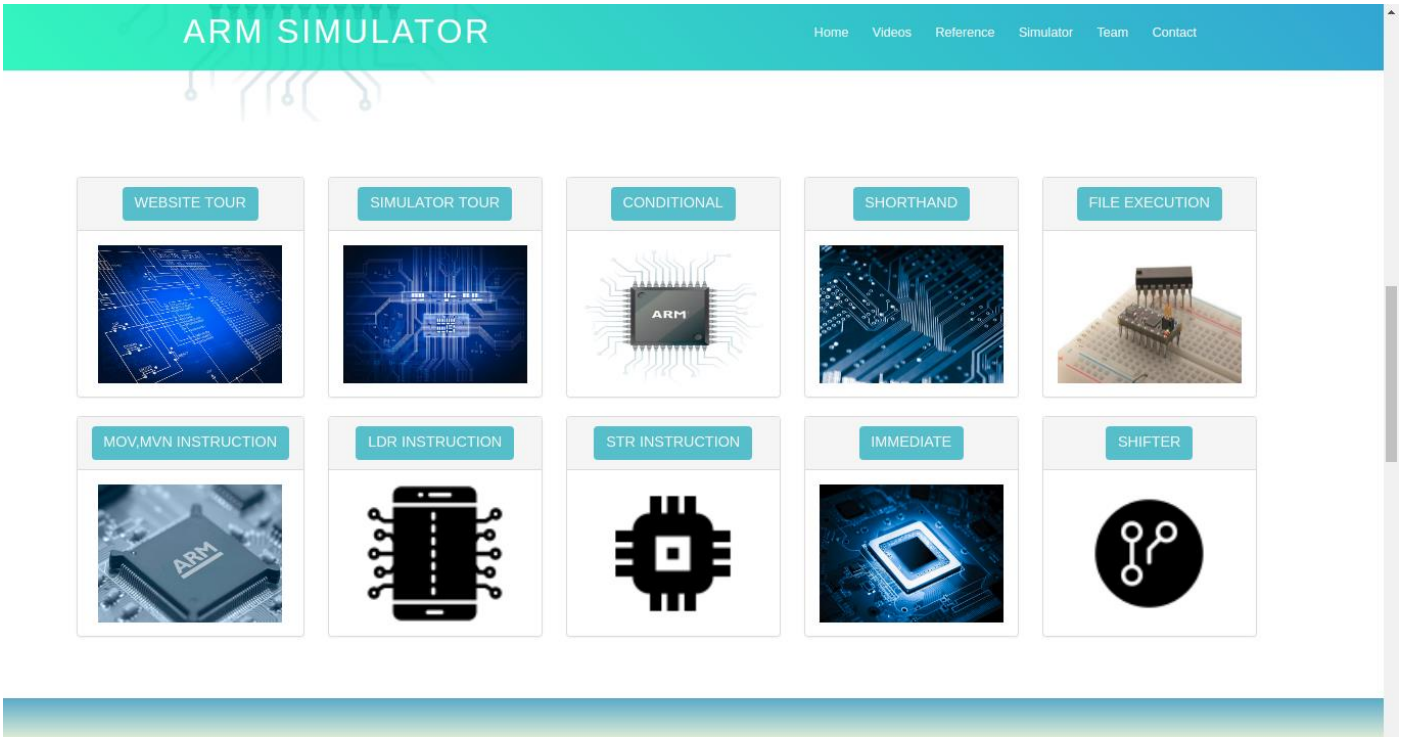
TST(n1, n2)	This function takes in n1 and n2 as its parameters which contains the values of register. This function performs BITWISE AND. The difference between AND () and TST () is the result is not written to the destination register but according to the result the flag registers are updated.
CMN(n1, n2)	This function takes in n1 and n2 as its parameters which contains the values of register. This function performs ADDITION. The difference between ADD() and CMN() is the result is not written to the destination register but according to the result the flag registers are updated.
CMP(n1, n2)	This function takes in n1 and n2 as its parameters which contains the values of register. This function performs SUBTRACTION. The difference between SUB() and CMP() is the result is not written to the destination register but according to the result the flag registers are updated.
MUL(n1, n2)	This function takes in n1 and n2 as its parameters which contains the values of register. This function performs multiplication of two numbers. According to the result the flag registers are updated.
LDR(reg, saddr)	This function takes in reg and saddr as its parameters which contains the values of register and address respectively. If the address mentioned is not present in STACK TABLE, then an ERROR alert pop's up by mentioning illegal access of memory. According to the value of the register the flag value is updated.
STR(reg, saddr)	This function takes in reg and saddr as its parameters which contains the values of register and address respectively. If the address mentioned is not present in STACK TABLE, then the address and its value is added to the table, else the value is rewritten on that address value. According to the value of the register the flag value is updated.
NOT(num)	This function takes a register value as it's parameter and does a NOT () operation on it. All the 1's are changed to 0 and 0's to 1.
Decbin(dec, length)	This function takes in dec and length as it's parameters which contains register value and the length of the binary representation of the register value. It converts a decimal number to binary of the length mentioned and returns the binary value.
binDec(binary)	This function takes in binary as it's parameters which contains a binary value. It converts the binary to its decimal equivalent.

SCREENSHOTS

1. FIRST PAGE OF THE PROJECT:

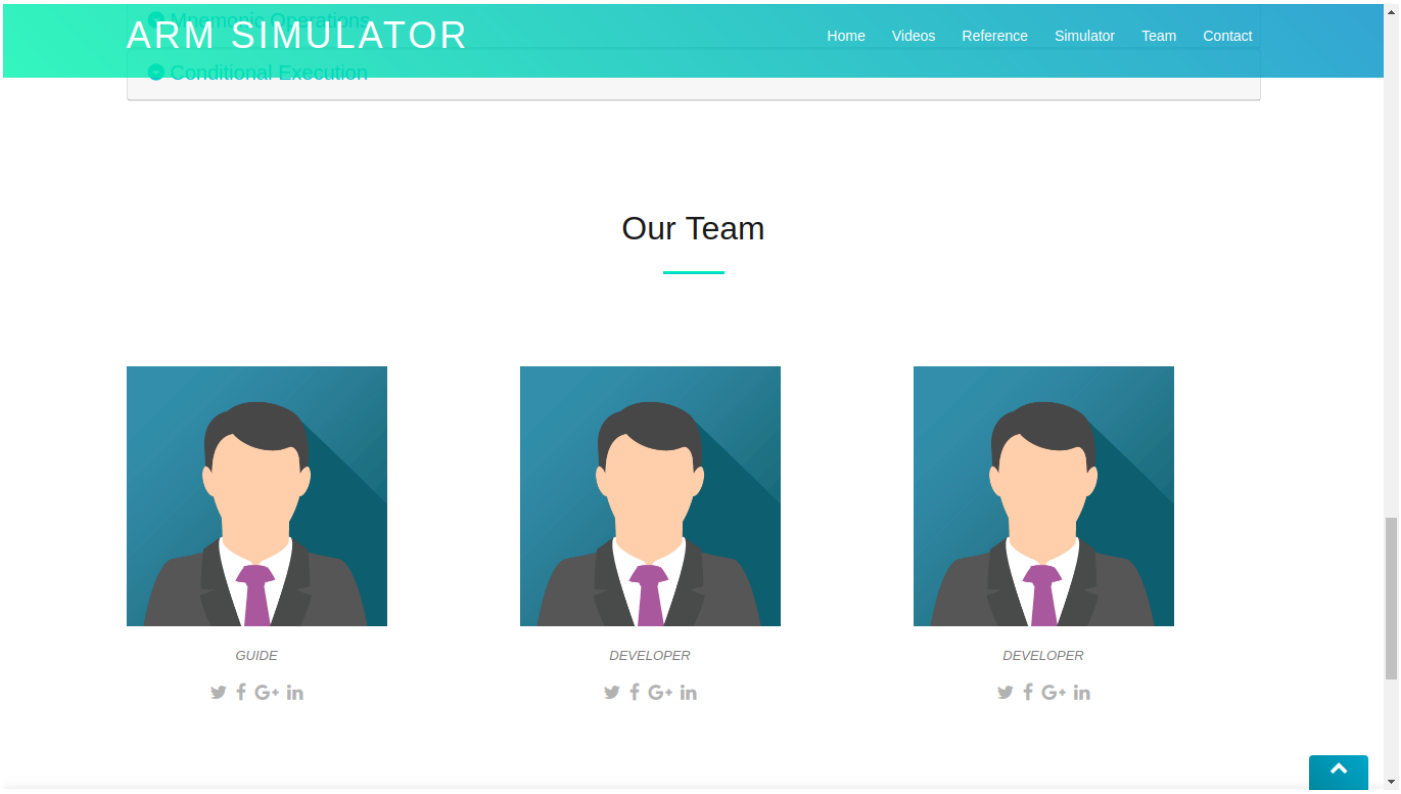


2. VIDEOS SECTION OF THE PROJECT:

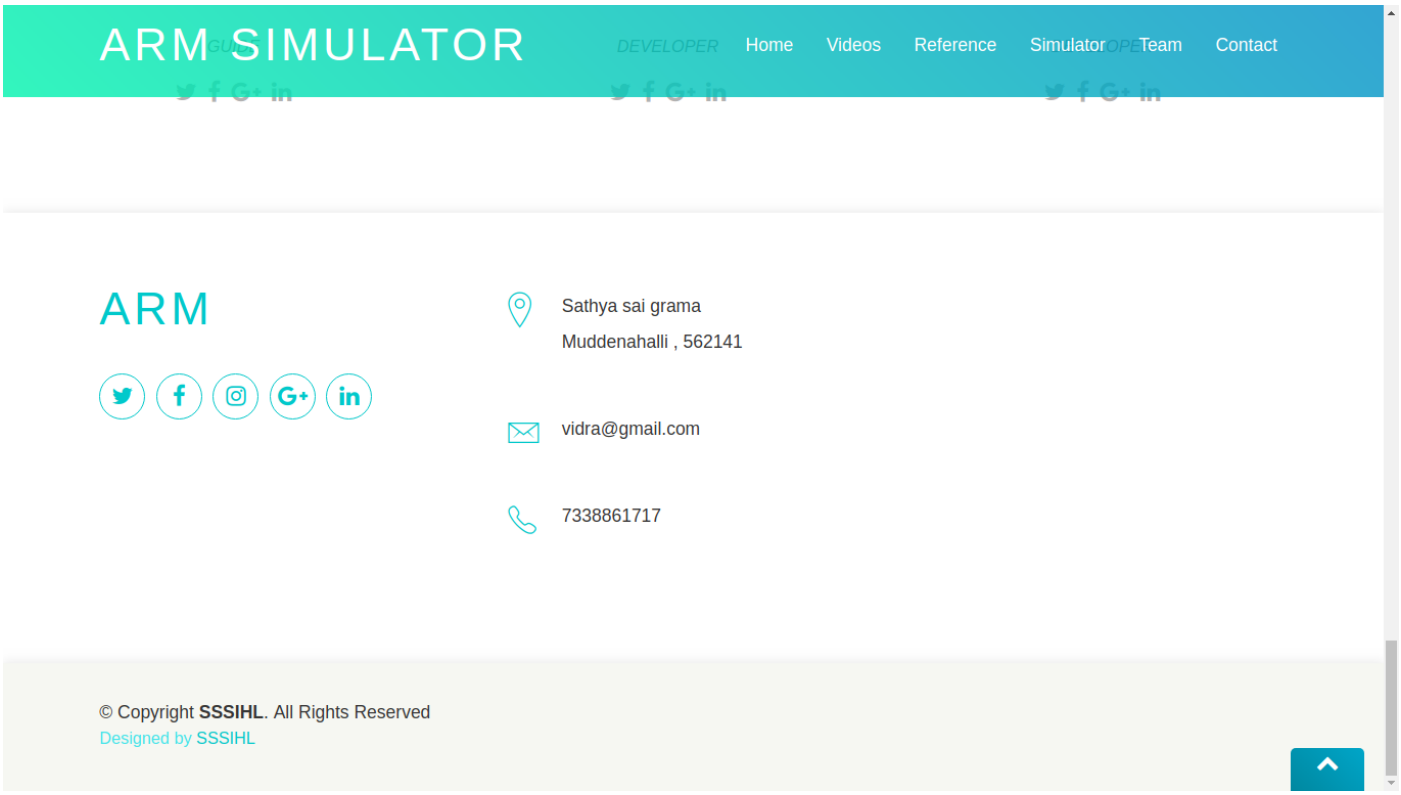


3. FREQUENTLY REFERED TOPICS:

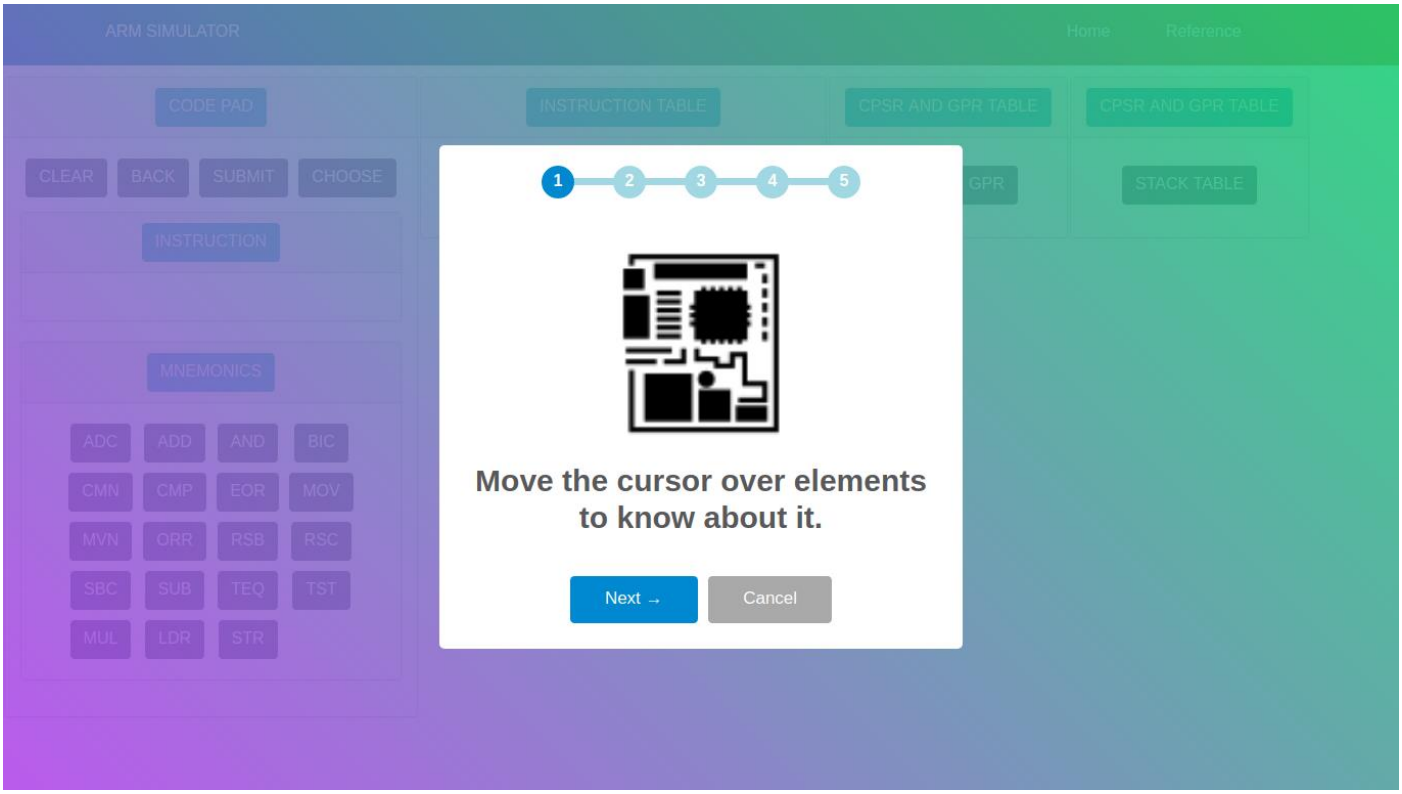
4. TEAM INVOLVED IN THIS PROJECT:



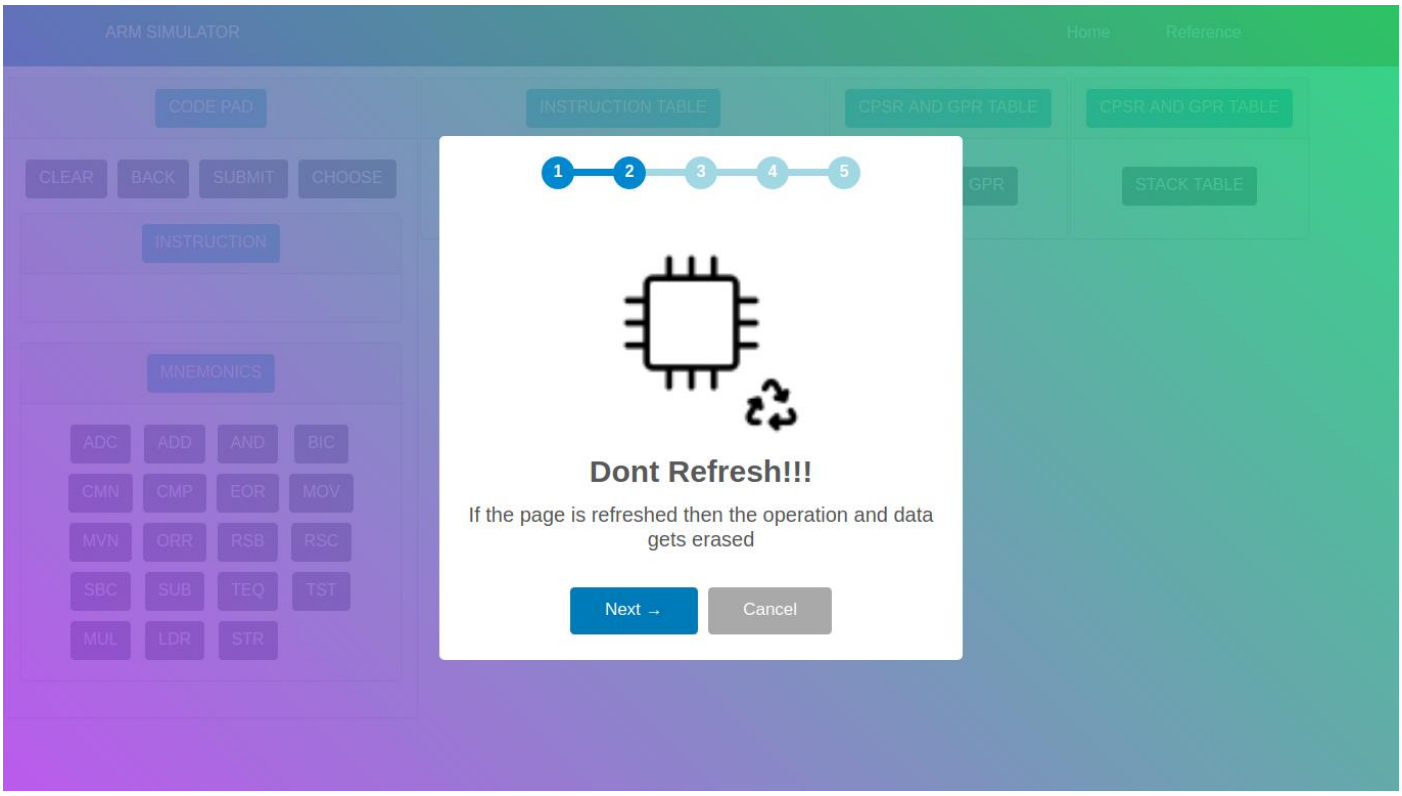
5. CONTACT PAGE:



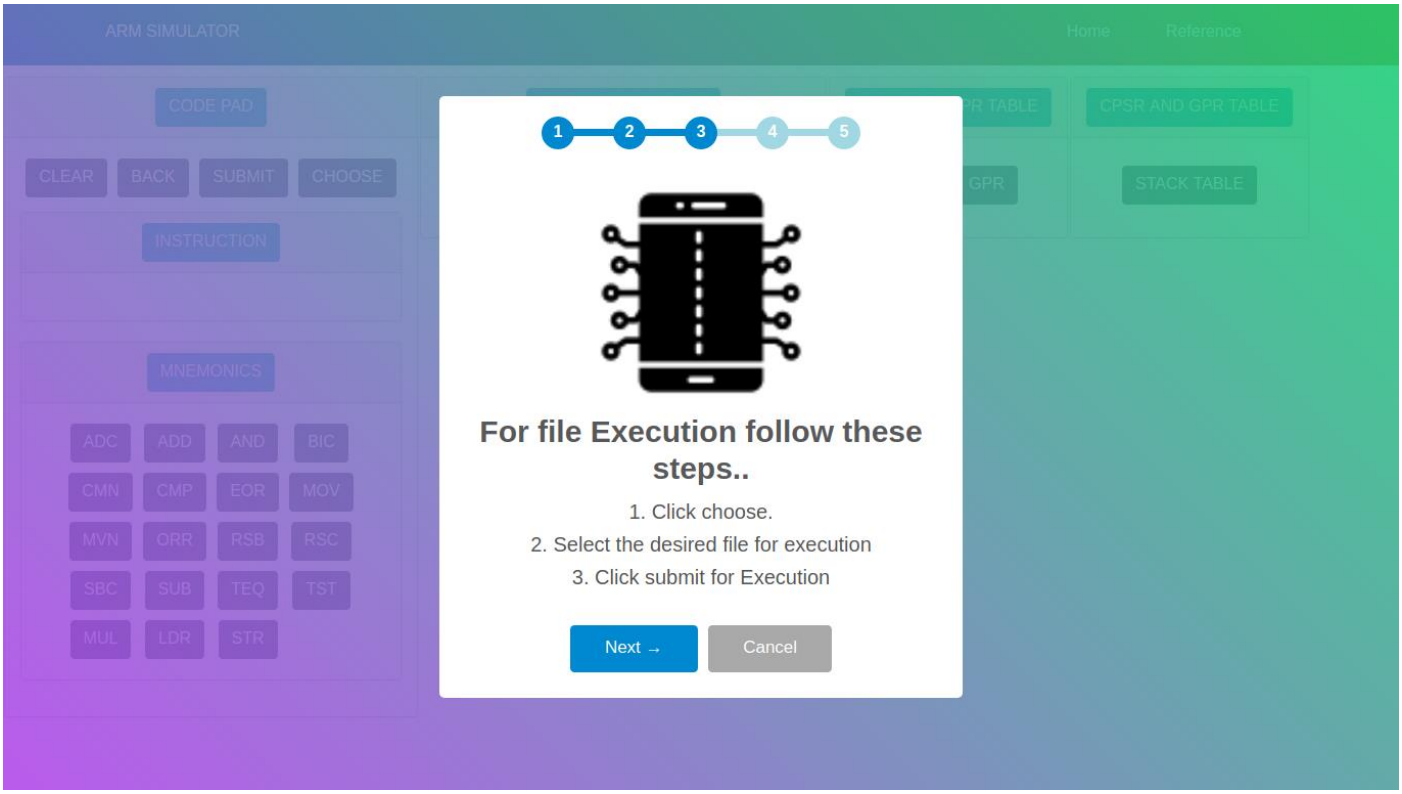
6. SIMULATOR PAGE (reminders):



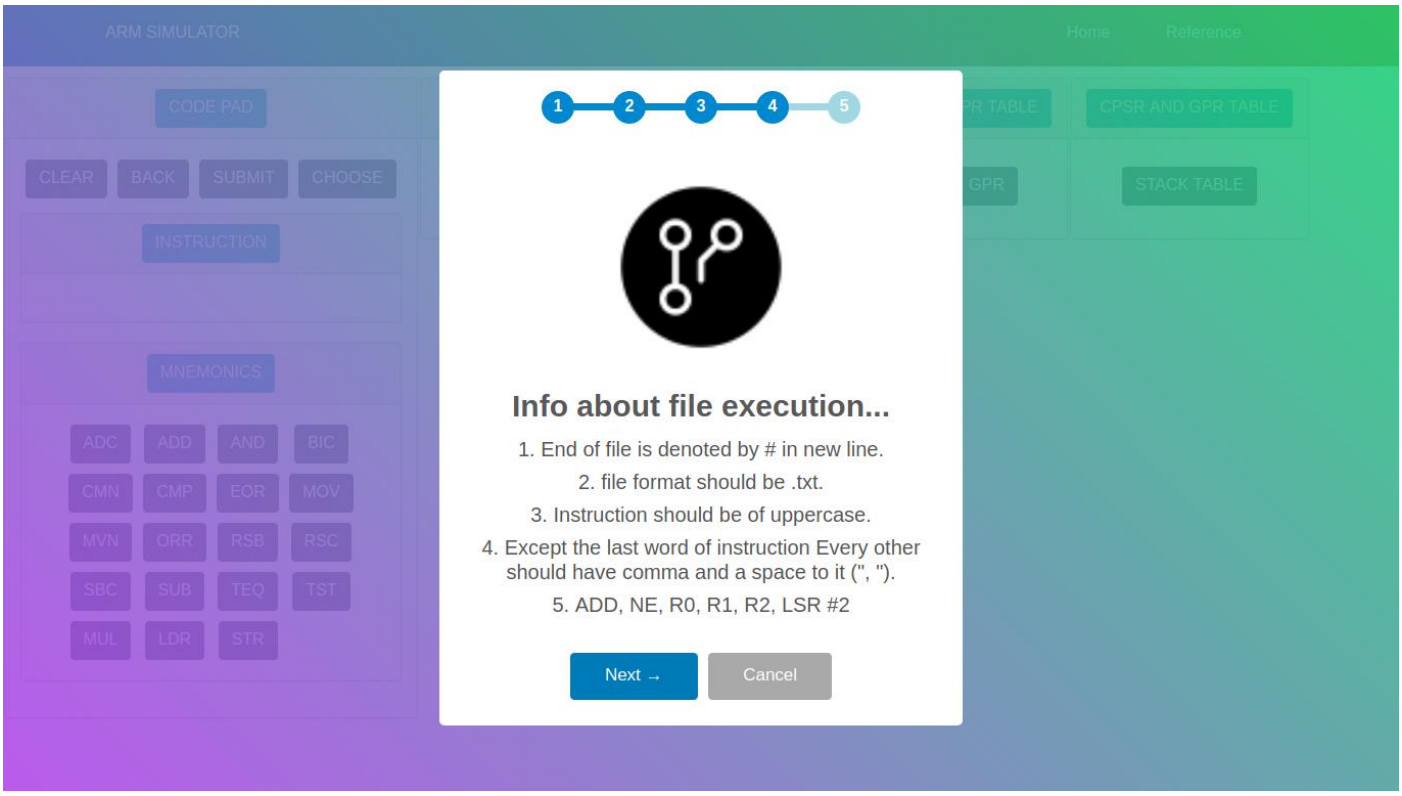
7. REMINDER 2:



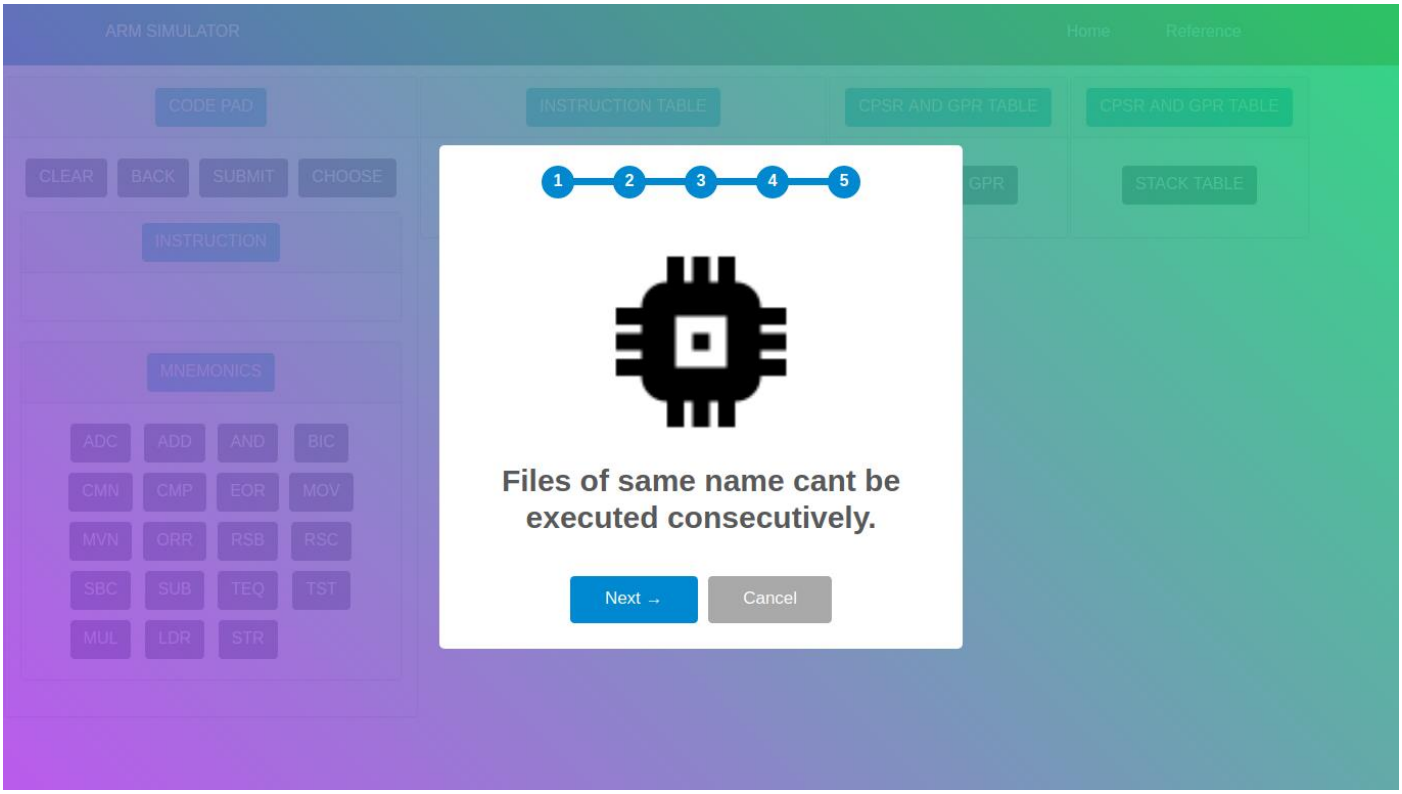
8. REMINDER 3:



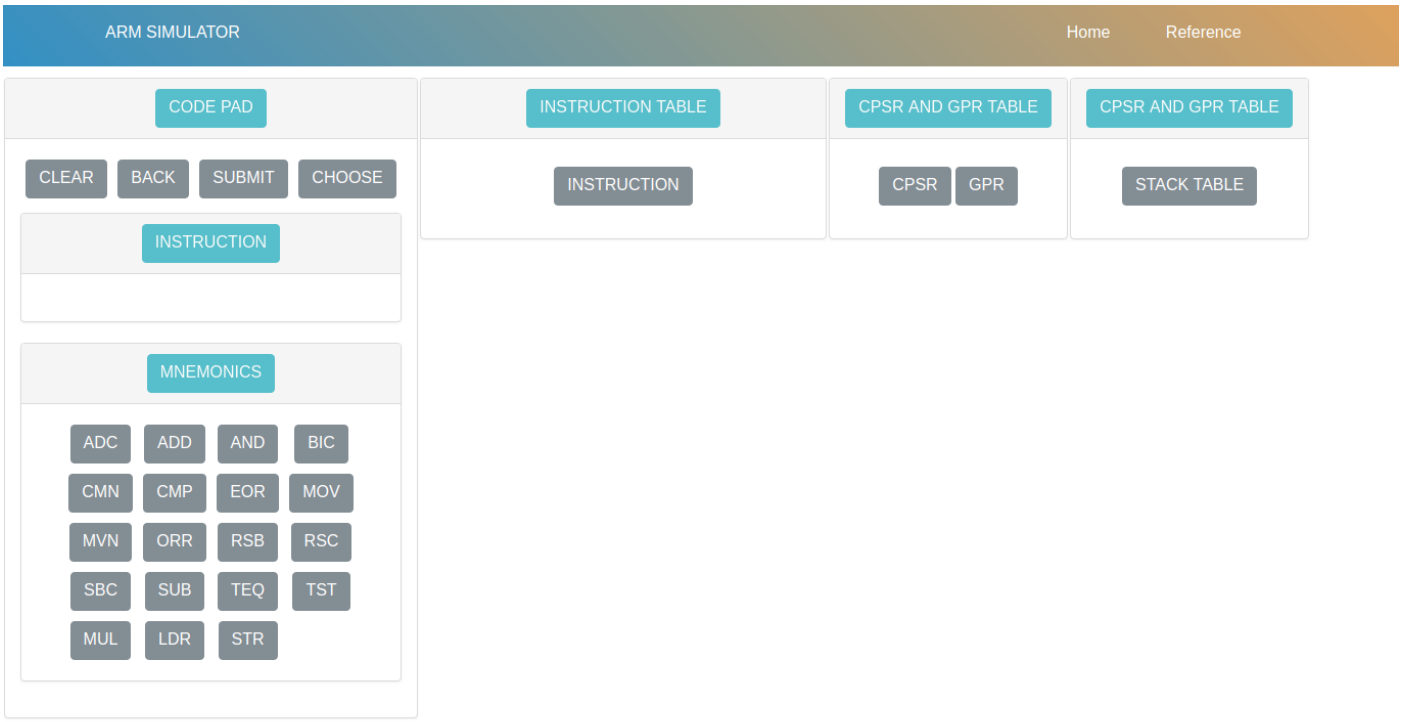
9. REMINDER 4:



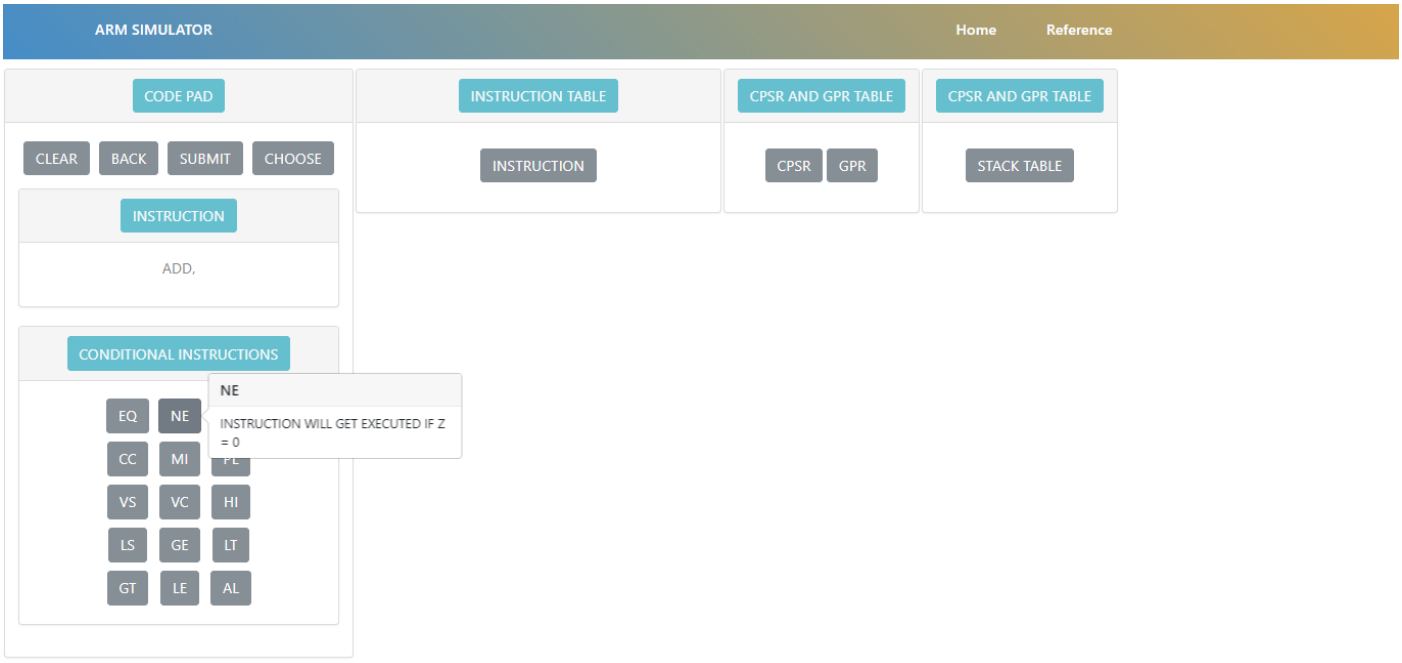
10. REMINDER 5:



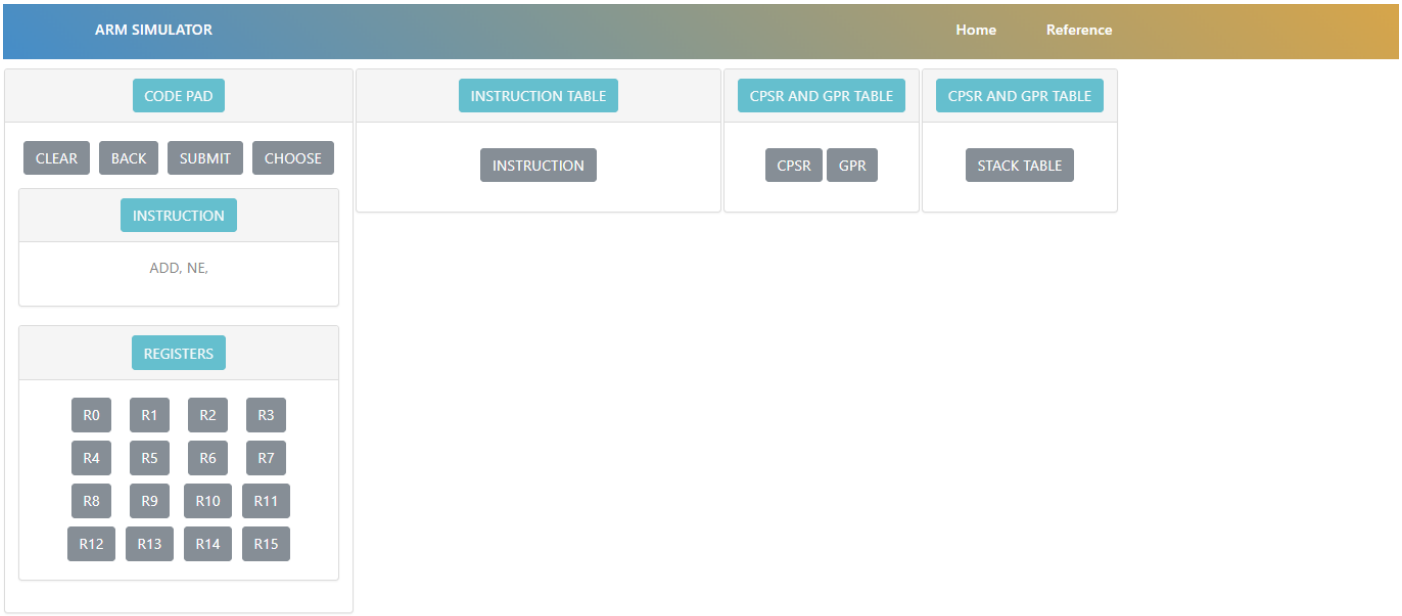
11. MNEMONIC OPERATION:



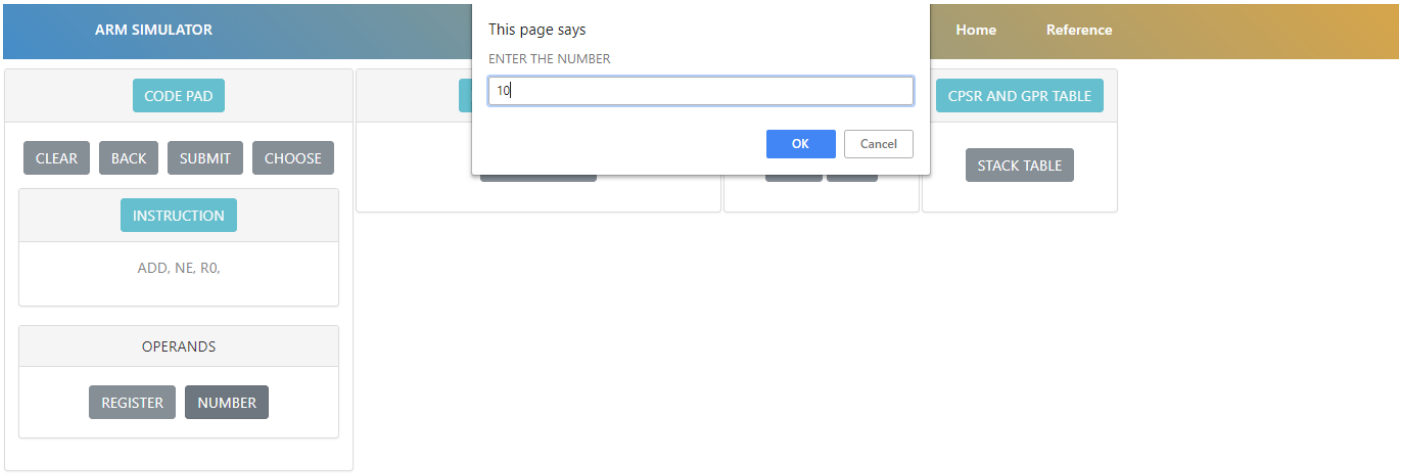
12. CONDITIONAL EXECUTION INSTRUCTION:



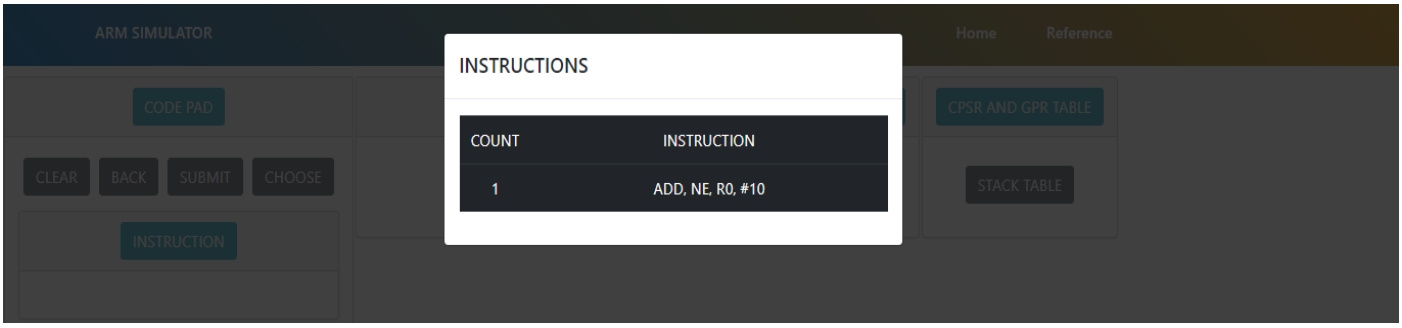
13. REGISTER SET:



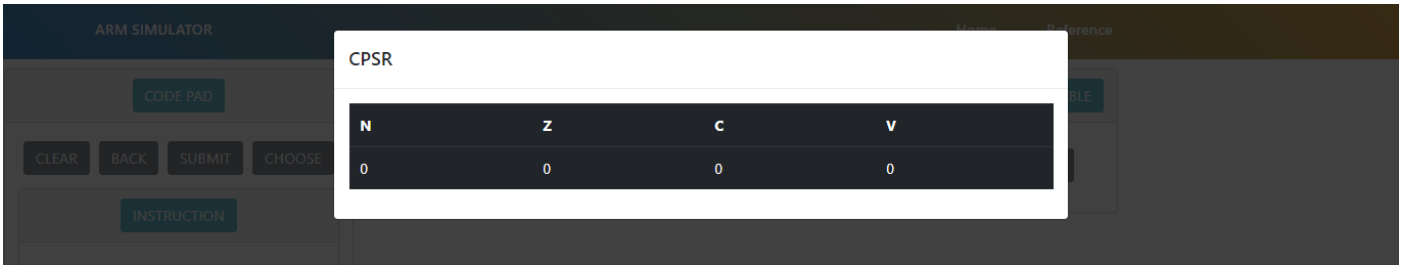
14. IMMEDIATE DATA:



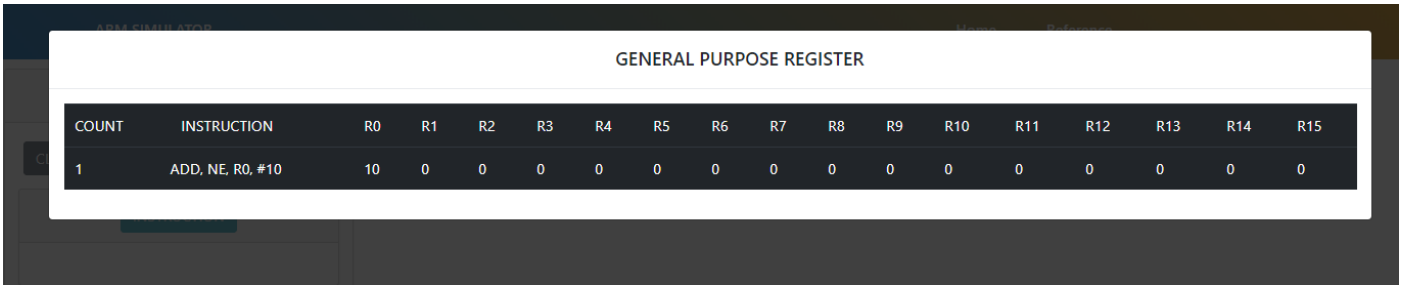
15. INSTRUCTION TABLE. (available onclick of instruction button in the simulator):



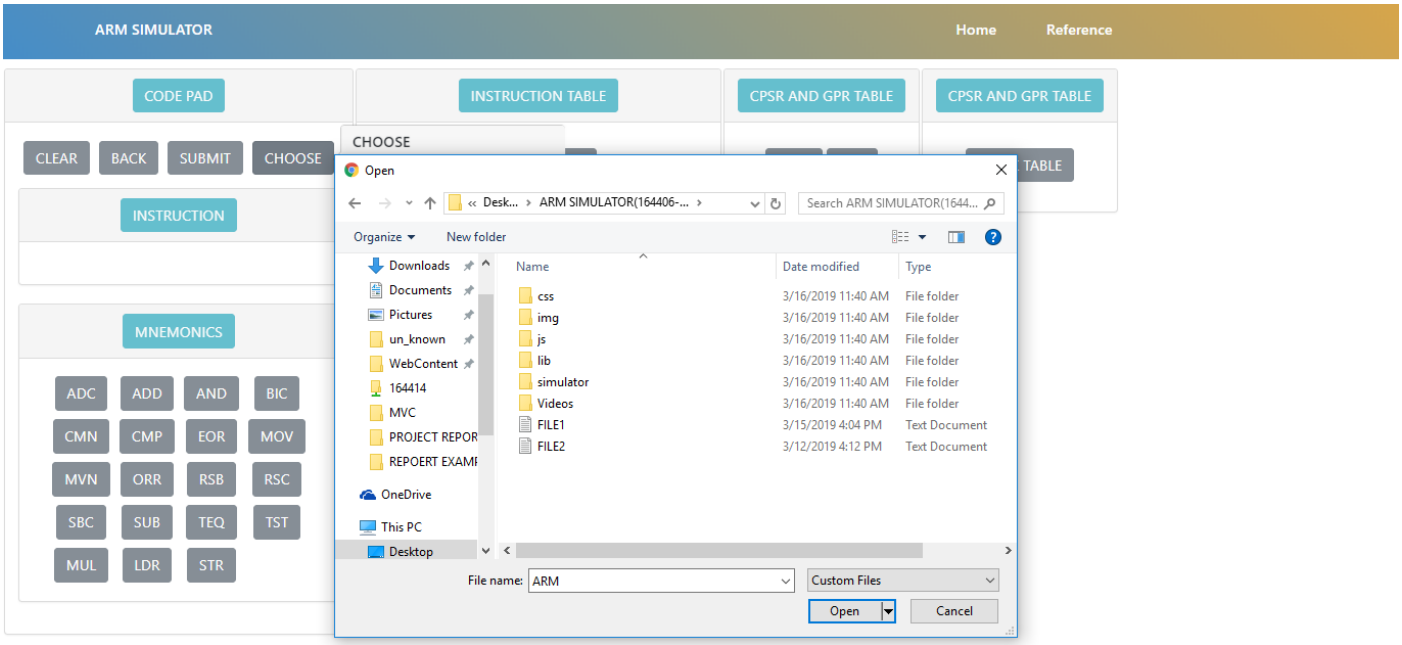
16. CPSR TABLE. (available onclick of CPSR button in the simulator):



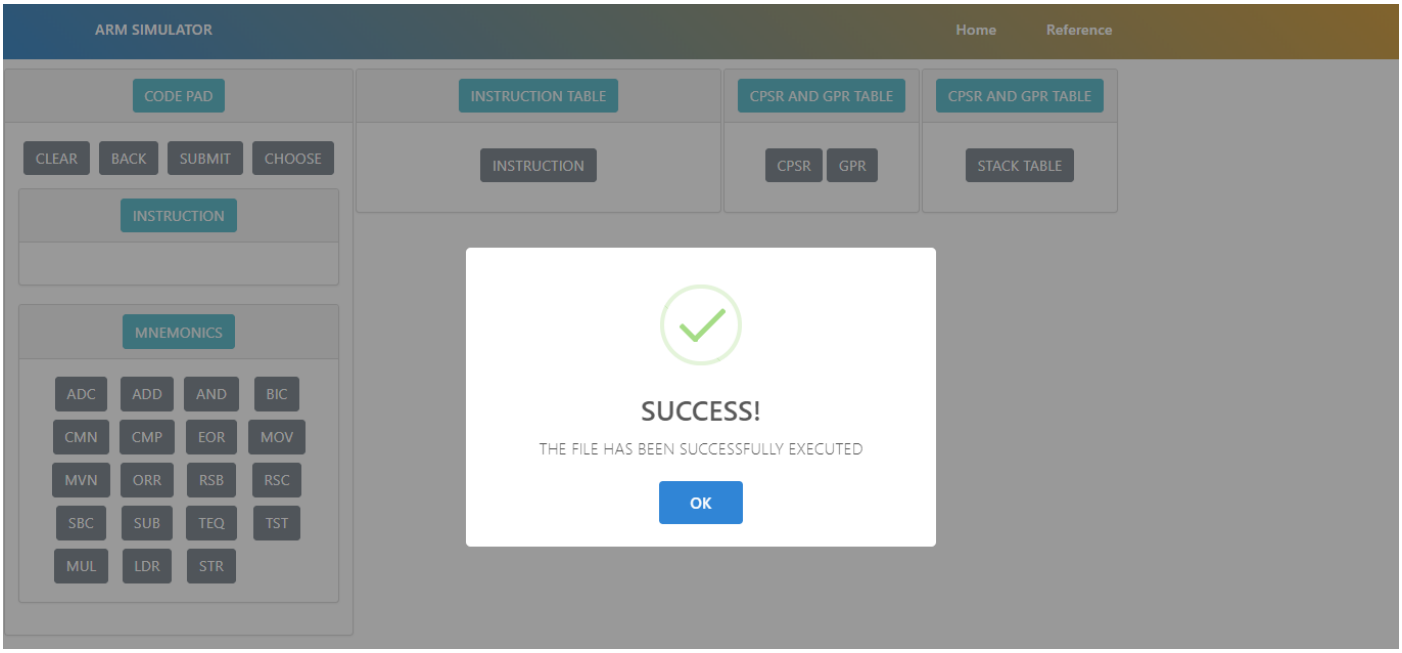
17. GPR TABLE. (available onclick of GPR button in the simulator):



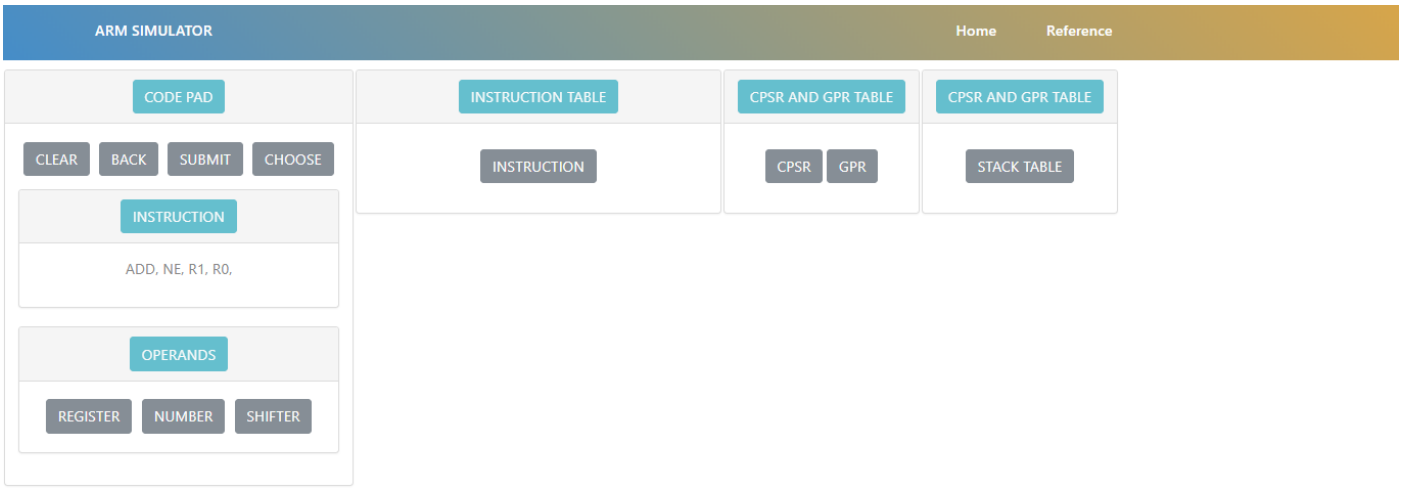
18. FILE EXECTUION. (browse option arises onclick of CHOOSE button in the simulator):



19. SUCCESS ALERT. (this alert appears on successful execution of instructions in the file):

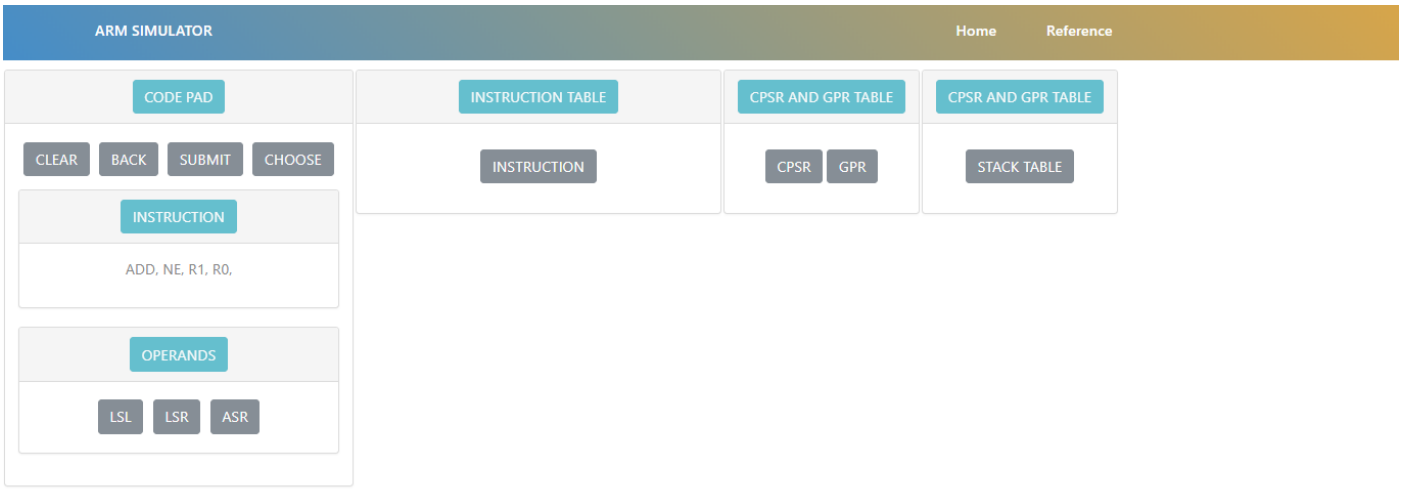


20. The instruction panel contains ‘ADD, NE, R1, R0 ‘. Onclick of BACK button it turns into:

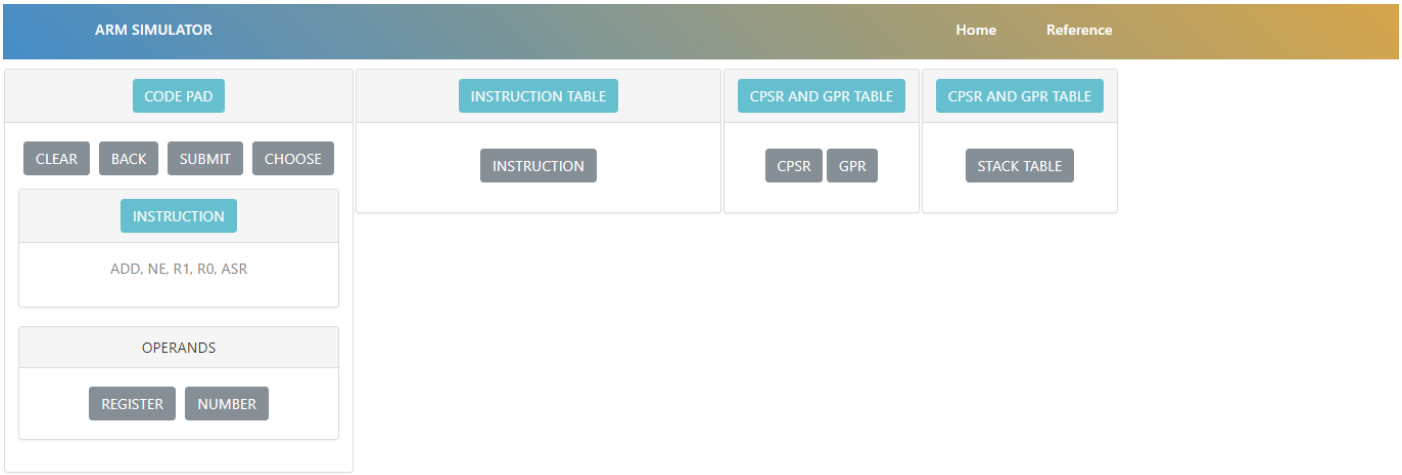


21. Onclick of BACK button ‘ADD, NE, R1, R0, ‘ changes into ‘ADD, NE, R1, ‘ :

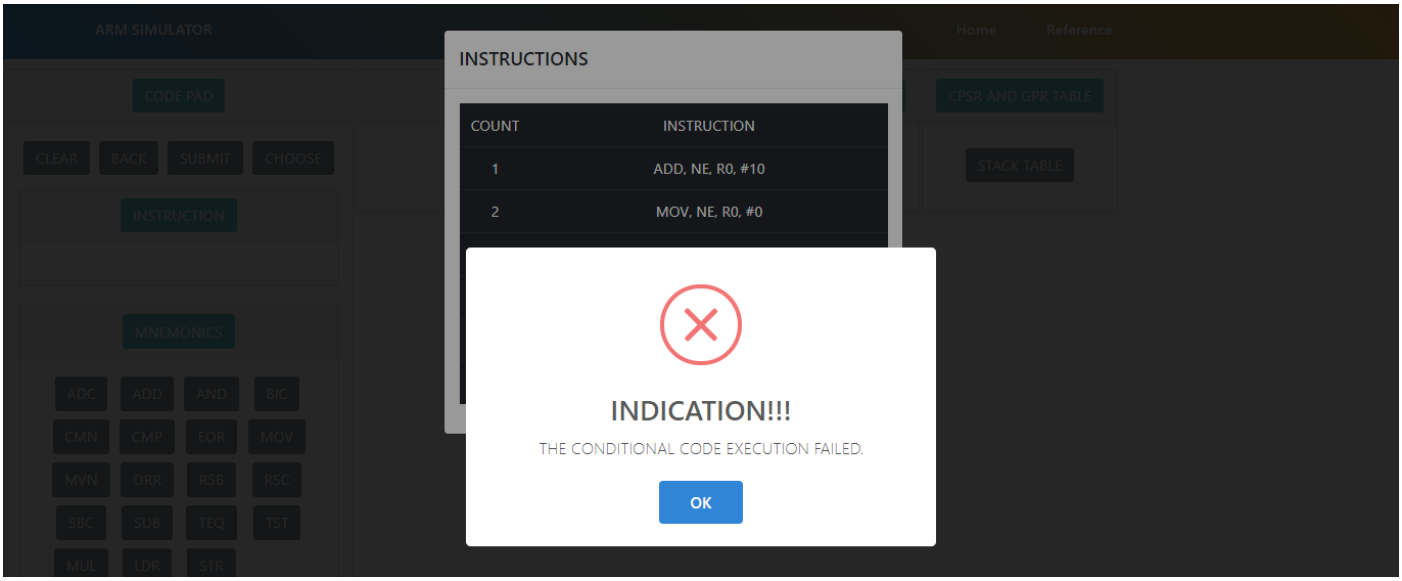
22. SHIFTERS AVAILABLE:



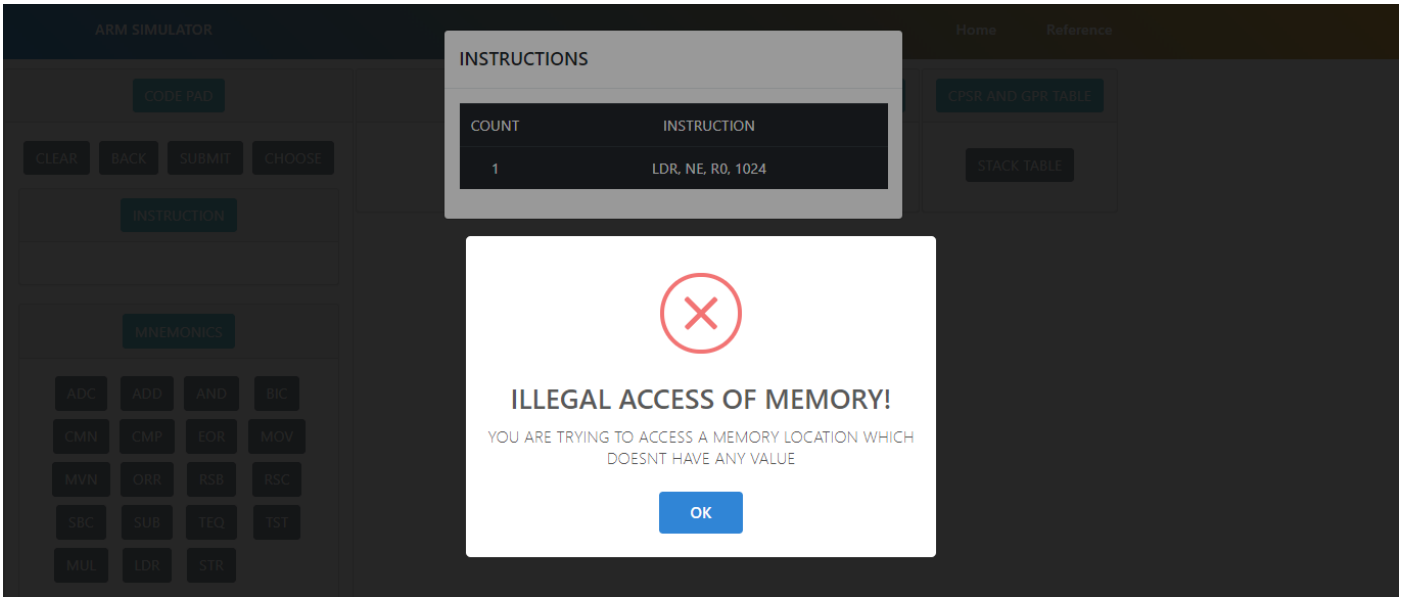
23. ONCLICK OF A SHIFTER, IT ASK'S WETHER THE OPERAND OF SHIFTER SHOULD BE EITHER A REGISTER OR A NUMBER:



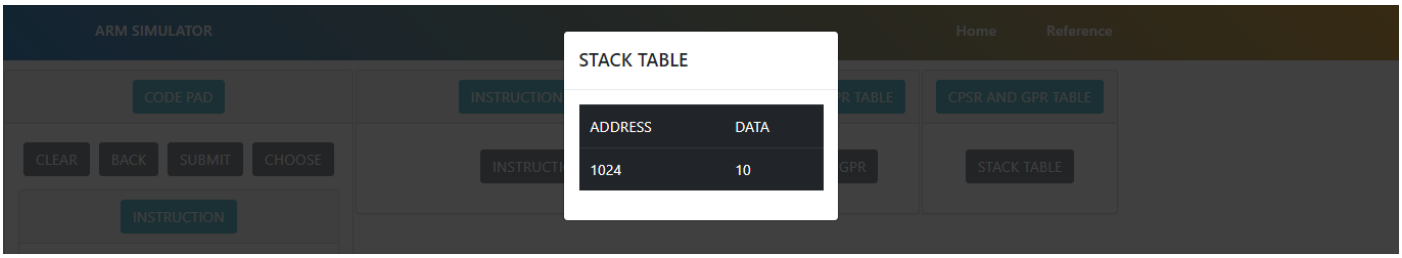
24. IF CONDITIONAL EXECUTION FAILS THEN THIS ALERT POP's UP ON THE SCREEN:



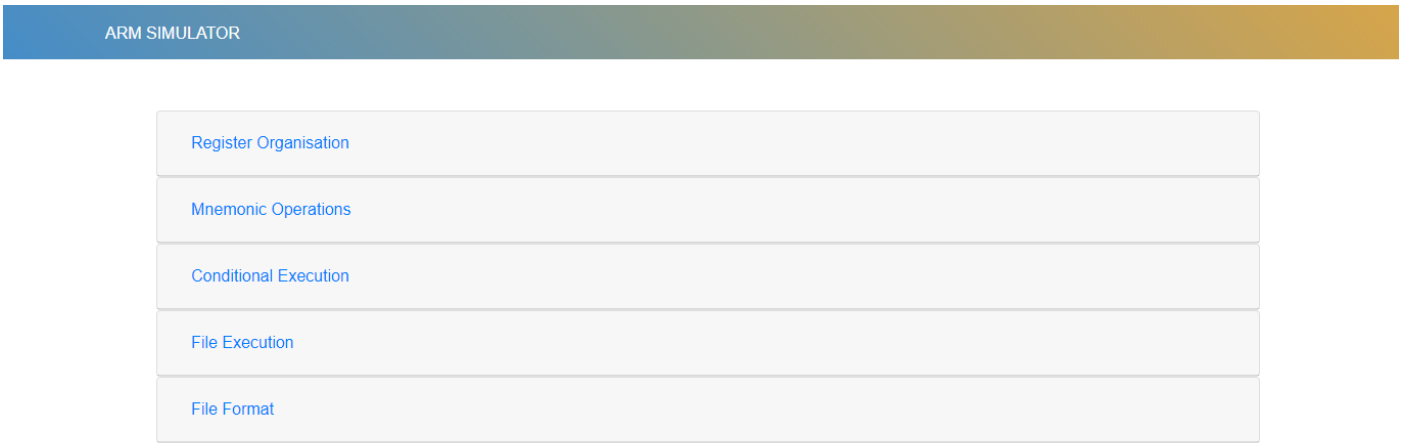
25. IF THE LOAD (LDR) INSTRUCTION GIVES AN ADDRESS THAT DOESN'T HAVE ANY VALUE THEN THIS ALERT POP's UP:



26.ON EXECUTION OF STR, NE, R0, 1024. THE STACK TABLE LOOKS LIKE THIS:



27. THE REFERNCE PAGE IN SIMULATOR:



LIMITATIONS

- Validation for file execution is not done.
- Since onchange() is given for file execution, files of same name can't be executed consecutively.
- Since the project is client side dependent the F5 button has been disabled but the browser refresh/reload button is not.
- The project can be used only after knowing some basics of Assembly language and basics of Microprocessor.
- Because of the browser the popup's sometime might not get cleared.

FUTURE SCOPE

- This simulator just performs basic arithmetic, load and store operations. In future some more operations can be included.
- Loops and subroutines can be added.
- Register pair operations can be performed.
- Machine cycle, Instruction cycle, and T-states can be added to the simulator.
- The simulator can be made more user friendly.
- Further Addressing Modes can be added.

REFERENCES

- Web Technology-Theory and practice by M. Srinivasan
- Webdeveloper.com
- Azeria Labs.com
- JavaScript Bible - Danny Goodman
- www.stackoverflow.com