**CASE STUDY ON CALCULATOR​**



**B-Tech/II Year CSE/IV Semester**

**19CSE214/Theory of Computation**

|  |  |
| --- | --- |
| Roll no | Name |
| CB.EN.U4CSE21103 | **AKASH V** |
| CB.EN.U4CSE21120 | **SANTOSH G** |
| CB.EN.U4CSE21121 | **NIKHIL G** |
| CB.EN.U4CSE21122 | **NARENDRA G** |
| CB.EN.U4CSE21128 | **MADHU K** |

**Amrita School of Engineering, Coimbatore**

**Department of Computer Science and Engineering**

**2022 -2023 Even Semester**

**PROBLEM STATEMENT​:**

Design and implement a finite state machine (FSM) that functions as a calculator. The calculator should be able to perform basic arithmetic operations such as addition, subtraction, multiplication, and division on positive integers.

​

**APPROACH:**

To design an FSM calculator with enter validation, follow these steps:

1. Define the states that represent different aspects of the calculator system, such as digit reading, performing operations, and end of the expression.
2. Determine the transitions between states based on the current input symbol and the current state. Transition to the next state when a valid input symbol is received.
3. Implement input validation by checking if the input symbols are digits from 0 to 9. If an invalid input is encountered, transition to an error state.
4. Include logic in the FSM to handle arithmetic operations like addition, subtraction, multiplication, and division. Perform the operations based on the input symbols, adhering to arithmetic rules.
5. Manage the end of the expression by validating that the final state of the FSM is a valid end state. This ensures that the entire expression has been processed correctly.

By following this approach, an FSM calculator can be designed that accepts inputs from 0 to 9, validates the inputs, performs arithmetic operations, and ensures accurate processing of the expression from start to finish.

**ANALYSIS:**

**Input:**

It accepts the numbers from ZERO-NINE as the inputs.

**Operations:**

It supports the Arithmetic Operations like Addition (‘+’), Subtraction (‘-’), Multiplication (‘\*’) and Division (‘/’) operations.

**Acceptance:**

1. The Calculator checks Whether the given input expression is in correct syntax and adheres to the given defined format.
2. It accepts only the (‘+’) / (‘-’) / None / numbers at the starting point and it does not accept the

(‘/’) and (‘\*’) at the starting of the expression.

3.In the given input expression it is mandatory to

give Equals to (‘=’) symbol to get the result.

1. Two Consecutive operators are not allowed.

ex: 6+- is not accepted by the calculator.

**Acceptance criteria:**

If the Given input expression is in Correct syntax form, then the Calculator accepts the expression. In all the other cases the calculator rejects the expression.

**LANGUAGE:**

No two operators are allowed side by side and the expression must end with '='

**GRAMMAR:**

S -> BA = / - BA =

A -> +B / -B / %B / \*B / λ

B -> (0-9) AC

C -> (0-9) AC / λ

**PUSHDOWN AUTOMATA:**

A Pushdown Automaton (PDA) is a type of finite-state machine that incorporates an additional stack memory. PDAs are used to recognize languages known as context-free languages, which have a more expressive power than regular languages recognized by finite automata.

It Consists of:

**An input alphabet:** A finite set of symbols that make up the input to the PDA.

Input symbols in this Calculator:

1. Arithmetic operators: {+, -, \*, /}.
2. Numbers {0 to 9}.
3. Equals to symbol {=}.

**A stack alphabet:** A finite set of symbols that make up the stack memory.

**A set of states:** A finite set of states that the PDA can be in.

Here we use totally 9 states in this Calculator:

Q0 is the initial state.

Q8 is the Final state.

Q1, Q2, Q3, Q4, Q5, Q6, Q7 are intermediate states.

**A transition function:** A function that maps the current state, input symbol, and the top symbol of the stack to a new state, a symbol to be pushed onto the stack, and a set of symbols to be popped from the stack.

**A start state:** The initial state where the PDA begins its computation.

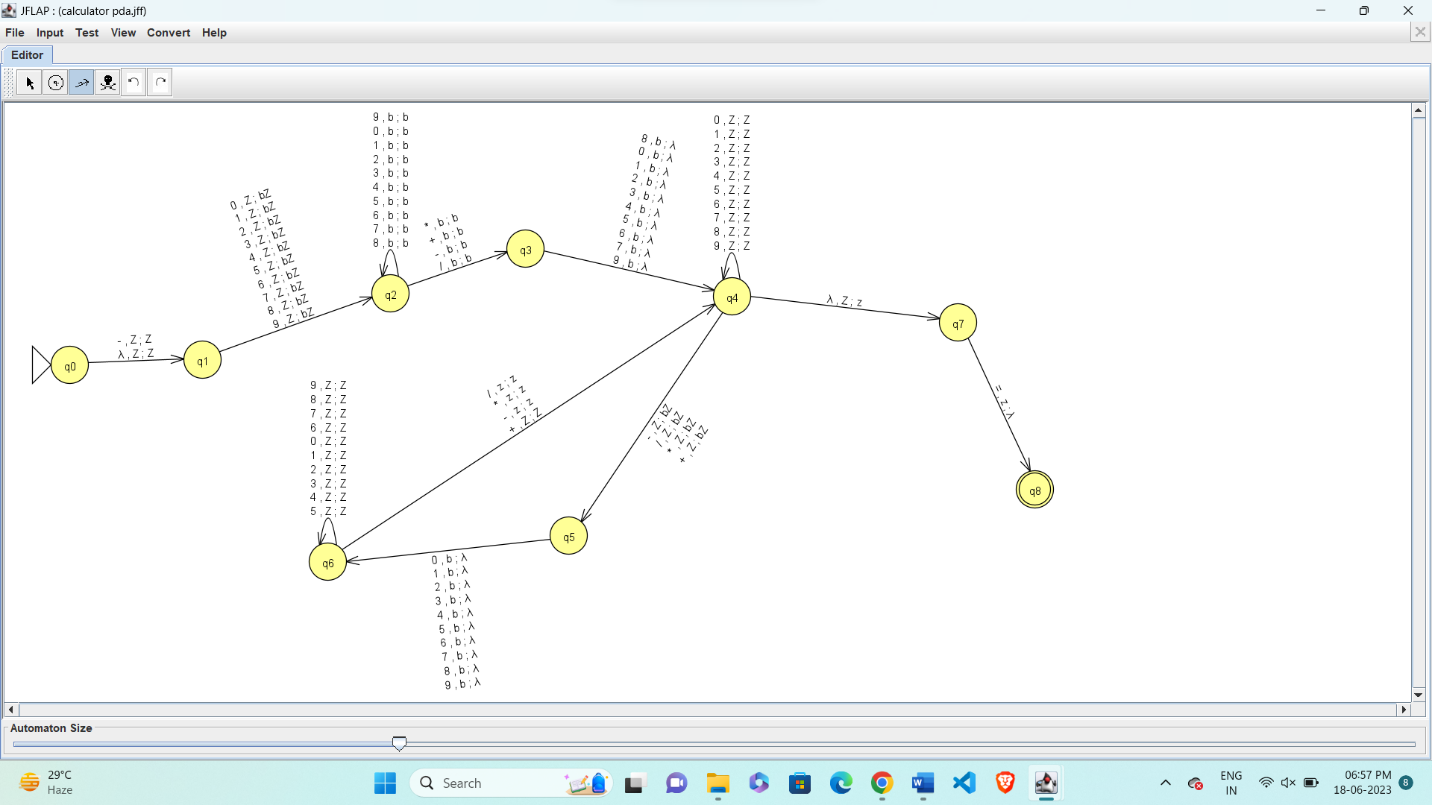
Here Q0 is the initial state.

**A set of accepting states:** The states in which the PDA accepts the input.

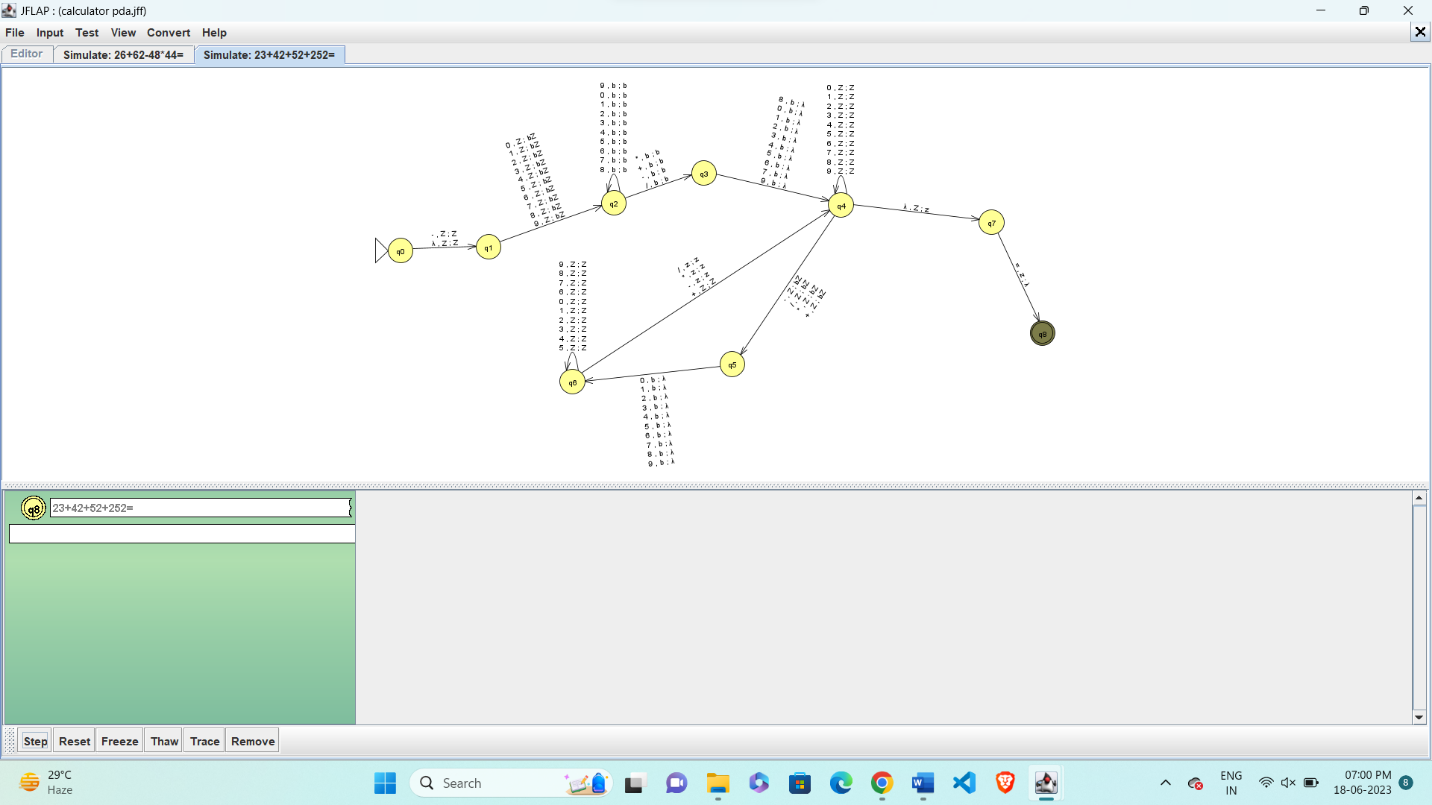
**Working of PDA:**

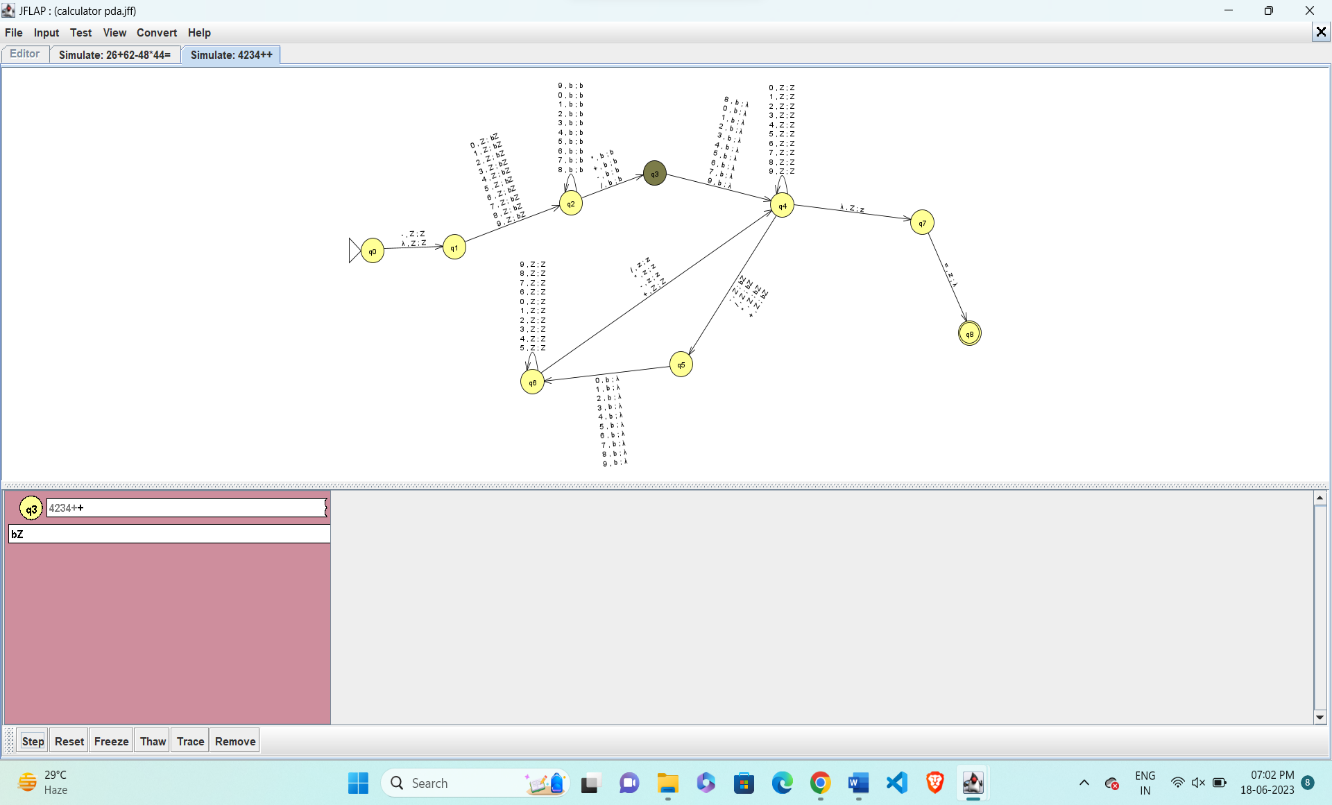
1. Initially, the PDA starts in the state q0 and accepts an input symbol of "+" or "-" or no input (lambda).
2. Upon receiving the input, the PDA transitions to state q1.
3. In state q2, the PDA checks for multi-digit inputs and transitions to itself while pushing the input digits onto the stack.
4. If an operator is encountered as input, the PDA transitions to state q3, where there is no stack pop or push operation.
5. Next, the PDA moves to a specific state for the second operand and performs a stack pop operation
6. In state q4, if the subsequent input is "=", the PDA transitions to the final state q7.
7. However, if another operator is given as input, the PDA proceeds to state q8.
8. In state q5, the PDA performs stack pop operations.
9. The PDA continues this loop, repeating the process.

**Construction of PDA:**



Accepting Case:



Non-accepting Case: Input = 4234++

**TURING MACHINE:**

A Turing Machine (TM) is a theoretical computational model that can simulate any algorithmic computation. It was introduced by Alan Turing in 1936 and serves as the foundation for the study of computability and complexity.

It consists of:

**An infinitely long tape divided into discrete cells:** Each cell can hold a symbol from a finite tape alphabet. The tape serves as the machine's memory and can be read from and written to by the machine.

**A tape head:** The tape head can move left or right along the tape, reading the symbol under it and writing new symbols.

**A finite set of states:** The Turing Machine can be in one of these states at any given time.

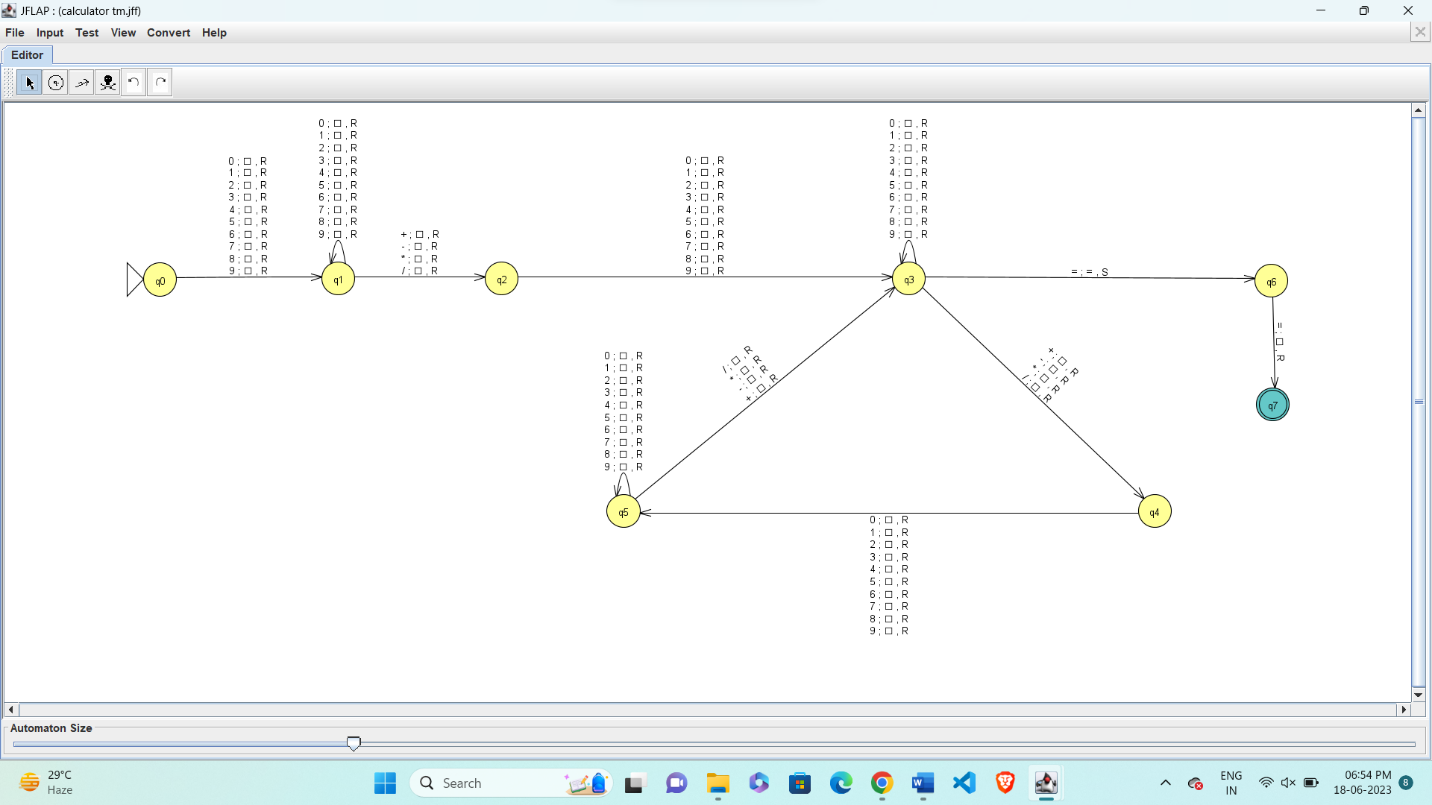
**A transition function**: This function determines the next state and action (symbol to write and direction to move the tape head) based on the current state and the symbol being read

* The Input symbols and set of sates for Turing machine are same as that of PDA

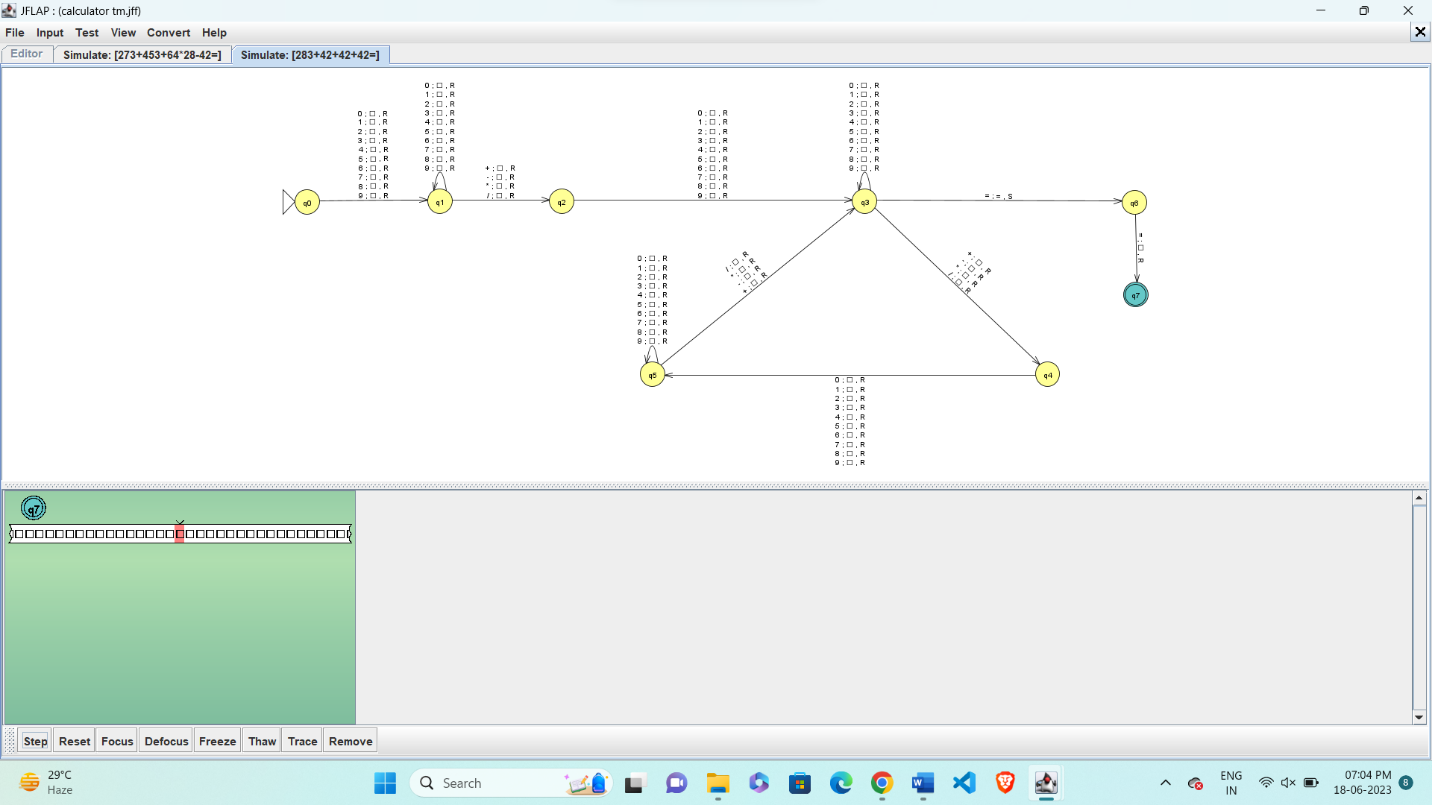
**Working of Turing Machine:**

1. In the designed Turing machine for a calculator, the initial state is q1. It accepts input symbols as integers from 0 to 9 and moves to the right while replacing the symbol with a blank.
2. After accepting a valid input, the Turing machine transitions to the next state, q2. In q2, it continues accepting input symbols as integers from 0 to 9. The self-loop in this state ensures the validity of n-digit numbers, such as 2574.
3. Once the first operand is accepted and validated, the Turing machine moves to the right and starts accepting operators like +, -, \*, and /. It replaces the operator symbol with a blank.
4. In state q4, the Turing machine validates the second operand.
5. If another operator is encountered before "=", the Turing machine validates it and proceeds to accept another operand in states q5 and q6 (e.g., 22 + 3 - 5).
6. Finally, if the input symbol is "=", the Turing machine halts as it reaches the final state, q7.
7. This Turing machine design ensures the correct handling of integers, operators, and operands in a calculator-like system, allowing for the validation and computation of arithmetic expressions.

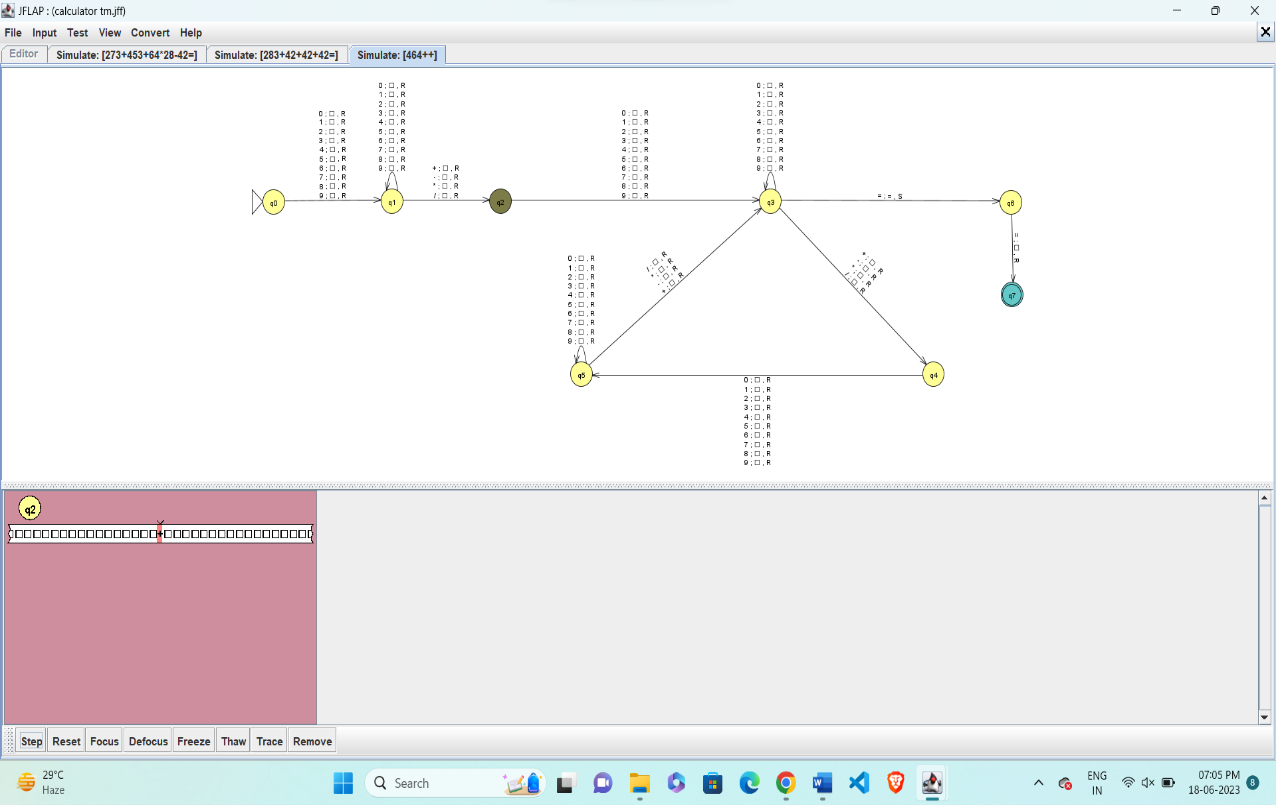
**Construction of Turing Machine:**



Accepting Case:



Non-accepting Case: Input = 464++



The above given input is rejected because there is no second operand.

**Conclusion:**

In conclusion, both Pushdown Automata (PDA) and Turing Machines are capable of performing input validation for calculators. However, they differ in their computational power and the complexity of the validation process.

PDA is a theoretical model of computation that extends the capabilities of finite automata by adding a stack. It is particularly suited for recognizing context-free languages, which allows it to handle nested structures like parentheses in calculator expressions. With a PDA, input validation for calculators can be implemented by maintaining a stack and checking the balance of opening and closing parentheses.

On the other hand, Turing Machines are more powerful and can recognize languages beyond the context-free level, including recursively enumerable and recursively decidable languages. Turing Machines can also perform input validation for calculators, but they can handle more complex validations beyond just parentheses. For example, they can check for syntax errors, invalid operators, and evaluate arithmetic expressions.

While both PDA and Turing Machines can perform input validation for calculators, the choice between the two depends on the complexity of the validation required. If the calculator's grammar is limited to parentheses and simple arithmetic operators, a PDA would suffice. However, if the calculator allows more complex expressions and error checking, a Turing Machine would be more suitable due to its greater computational power.

It's important to note that the implementation details and the specific programming language used can also influence the approach to calculator input validation. The conclusion presented here focuses on the theoretical aspects of PDA and Turing Machines in relation to input validation for calculators.