Code Documentation

This ARM assembly code is designed to run on a Raspberry Pi and performs the following tasks:

Input Math Expression:

The program prompts the user to enter a mathematical expression involving arithmetic operations (+, -, *, /, ^), parentheses, and variables.

Validation:

The code performs several checks on the input expression, including:

- Ensuring the first and last characters are not operators.
- Checking for balanced parentheses.
- Verifying that the expression does not start or end with spaces.

Infix to Reverse Polish Notation (RPN) Conversion:

The program converts the infix expression (the standard algebraic notation) into Reverse Polish Notation (RPN) using the Shunting Yard Algorithm. This involves iterating through the input expression, pushing operands onto an output queue, and handling operators based on their precedence and associativity rules.

Variable Substitution:

The program prompts the user to enter values for any variables present in the expression. It then substitutes the variables with their corresponding values in the RPN expression.

Expression Evaluation:

The program evaluates the RPN expression by iterating through the RPN string. It uses a stack to perform arithmetic operations when encountering operators and pushes operands onto the stack when encountering numbers or variables.

Error Handling:

The code handles various error conditions, such as division by zero, invalid expressions, overflow errors, and invalid variable values.

Output: The final result of the evaluated expression is printed to the console.

Functions

The code contains several functions to handle various tasks, such as:

`cpyBuffer:` Copies the contents of one buffer to another.

`cleanBuffer:` Clears the contents of a buffer.

`overwriteVariable:` Replaces variables in the expression with their corresponding values.

`checkVariable:` Checks if a variable is present in the list of variables.

`readNum:` Converts a number from ASCII to decimal.

`cleaningExpression:` Removes unnecessary spaces and characters from the RPN expression.

`isOperator:` Checks if a character is an arithmetic operator and returns its precedence.

`isNumber:` Checks if a character is a digit.

`isNum:` Verifies if a string represents a valid number.

`readInput:` Reads user input into a buffer.

`writeString:` Writes a string to the console.

`strlen:` Calculates the length of a string.

`delReturn:` Removes the newline character from the end of a string.

`cleanDoubleSpace:` Removes consecutive spaces from a string.

`printEnter:` Prints a newline character to the console.

`int_to_ascii:` Converts an integer to its ASCII representation.

`reverse:` Reverses the order of characters in a string.

`pow:` Computes the power of a number.

`checkParenthesis:` Checks if the parentheses in an expression are balanced.

`NOTMinus:` Toggles the sign of the result (positive or negative).

`checkLastOperator:` Checks if the last character of the expression is an operator.

Important Code Blocks

Infix to RPN Conversion:

This is a crucial part of the program, where the infix expression (standard algebraic notation) is converted to Reverse Polish Notation (RPN) using the Shunting Yard Algorithm. The InfixToRPN label marks the beginning of this section. The algorithm works by iterating through the input expression character by character. If the character is an operand (number or variable), it is directly added to the output RPN string. If it's an operator, its precedence is compared with the operators already present in the stack. Based on the precedence rules, operators are either pushed onto the stack or popped from the stack and added to the output RPN string. This section involves several nested loops and branches to handle different cases, such as parentheses, operators with different precedence levels, and variables.

RPN Expression Evaluation:

After converting the infix expression to RPN, the program evaluates the RPN expression using a stack-based approach. The evaluate label marks the beginning of this section. The program iterates through the RPN string character by character. If the character is a number, it is pushed onto the stack. If the character is an operator, the top two operands are popped from the stack, the corresponding arithmetic operation is performed, and the result is pushed back onto the stack. This section includes several branches and subroutine calls to handle different arithmetic operations (suma, resta, multi, divi, exp).

Variable Substitution:

Before evaluating the RPN expression, the program needs to substitute any variables present in the expression with their corresponding values provided by the user. This section is marked by the askInput label. The program iterates through the list of variables and prompts the user to enter a value for each variable. It then calls the overwriteVariable function to replace occurrences of the variable in the new_operacion buffer with the provided value, storing the result in the new_rpn buffer. This section also includes error handling for invalid variable values entered by the user, marked by the invalidValueInput label.

Input Expression Validation:

The program performs several checks on the input expression to ensure its validity. These checks are scattered throughout the code:

- Ensuring the first and last characters are not operators: This is done by calling the isOperator function for the first and last characters of the input expression.
- Checking for balanced parentheses: The checkParenthesis function is called to verify that each opening parenthesis has a matching closing parenthesis in the expression.
- Checking for invalid starting and ending spaces: The program checks if the first character of the input expression is a space or a newline character, which would be considered invalid.

Error Handling:

The program has several error handling branches to catch and report various types of errors, such as:

- Division by zero: If the program encounters a division by zero operation while evaluating the RPN expression, it jumps to the divError label and prints an error message.
- Arithmetic overflow: If the result of an arithmetic operation exceeds the 32-bit signed integer range, it jumps to the overflowError label and prints an error message.
- Invalid expressions: If the input expression is deemed invalid (e.g., starting or ending with an operator, unbalanced parentheses), it jumps to the badExpression label and prints an error message.
- Too many variables: If the user tries to define more than the maximum allowed number of variables (10 in this case), it jumps to the overflowVariables label and prints an error message.

Helper Functions:

The program includes several helper functions to perform various tasks, such as:

- `cpyBuffer, cleanBuffer:` For copying and clearing buffers, respectively.
- `isOperator, isNumber, isNum:` For checking if a character is an operator, a digit, or a valid number, respectively.
- `readInput, writeString, strlen:` For reading input from the user, writing strings to the console, and calculating string lengths, respectively.
- `delReturn, cleanDoubleSpace:` For removing newline characters and consecutive spaces from strings, respectively.
- `int_to_ascii, reverse:` For converting integers to their ASCII representation and reversing strings, respectively.
- `pow:` For computing the power of a number.

These helper functions are called at various points throughout the program to perform their respective tasks.

Functions Parameters and Returns

---

cpyBuffer

Parameters:

- `r0` : Pointer to the buffer where the data will be copied.
- `r1` : Pointer to the buffer containing the data to be copied.

Return Value: None.

Description: Copies the contents of the buffer pointed to by `r1` into the buffer pointed to by `r0`.

---

cleanBuffer

Parameters:

- `r0`: Pointer to the buffer to be cleared.

Return Value: None.

Description: Clears the contents of the buffer pointed to by `r0` by setting all its elements to zero.

---

overwriteVariable

Parameters:

- `r0`: Pointer to the `new_operacion` buffer.
- `r1`: The variable character to be overwritten.
- `r2`: Pointer to the `num` buffer containing the variable's value.
- `r3`: Pointer to the `new_rpn` buffer.

Return Value: None.

Description: Overwrites occurrences of the specified variable character in the `new_operacion` buffer with the corresponding value from the `num` buffer, and stores the result in the `new_rpn` buffer.

---

checkVariable

Parameters:

- `r1`: The variable character to be checked.

Return Value:

- `r0`: 0 if the variable is not found in the list, 1 if it is found.

Description: Checks if the specified variable character is present in the list of variables.

---

readNum

Parameters:

- `r0`: Pointer to the buffer containing the numeric string.

Return Values:

- `r0`: Updated pointer to the buffer after reading the number.
- `r3`: The numeric value converted from the ASCII string.

Description: Converts a numeric string from ASCII to decimal representation.

---

cleaningExpression

Parameters: None.

Return Value: None.

Description: Cleans the RPN expression by removing unnecessary spaces and invalid characters, and stores the cleaned expression in the `new_operacion` buffer.

---

isOperator

Parameters:

- `r1`: The character to be checked.

Return Values:

- `r2`: 1 if the character is an operator, 0 otherwise.
- `r3`: The precedence value of the operator if it is an operator, undefined otherwise.

Description: Checks if the given character is an arithmetic operator and returns its precedence value if it is.

---

isNumber

Parameters:

- `r1` : The character to be checked.

Return Value:

- `r2` : 1 if the character is a digit, 0 otherwise.

Description: Checks if the given character is a digit (0-9).

---

isNum

Parameters:

- `r0` : Pointer to the buffer containing the numeric string.

Return Value:

- `r0` : 1 if the buffer contains a valid number, 0 otherwise.

Description: Verifies if the contents of the buffer represent a valid numeric string.

---

readInput

Parameters:

- `r1` : Pointer to the buffer where the input will be stored.
- `r2` : Size of the buffer.

Return Value: None.

Description: Reads user input from the console and stores it in the specified buffer.

---

writeString

Parameters:

- `r1` : Pointer to the buffer containing the string to be printed.
- `r2` : Length of the string.

Return Value: None.

Description: Prints the given string to the console.

---

strlen

Parameters:

- `r0` : Pointer to the string buffer.

Return Value:

- `r0` : Length of the string.

Description: Calculates the length of the given string.

---

delReturn

Parameters:

- `r0` : Pointer to the string buffer.

Return Value: None.

Description: Removes the newline character (`\n`) from the end of the string.

---

cleanDoubleSpace

Parameters:

- `r0` : Pointer to the string buffer.

Return Value: None.

Description: Removes consecutive spaces from the string by replacing them with the ASCII code `0x7f` .

---

printEnter

Parameters: None.

Return Value: None.

Description: Prints a newline character (`\n`) to the console.

---

int_to_ascii

Parameters:

- `r0` : The integer value to be converted.
- `r1` : Pointer to the buffer where the ASCII representation will be stored.

Return Value: None.

Description: Converts an integer to its ASCII representation and stores it in the specified buffer.

---

reverse

Parameters:

- `r0` : Pointer to the start of the string buffer.
- `r1` : Pointer to the end of the string buffer.

Return Value: None.

Description: Reverses the order of characters in the string.

---

pow

Parameters:

- `r3` : The base number.
- `r2` : The exponent.

Return Value:

- `r2` : The result of the power operation.

Description: Computes the power of a number by repeatedly multiplying the base by itself for the given exponent.

---

checkParenthesis

Parameters:

- `r0` : Pointer to the expression buffer.

Return Value:

- `r0` : 1 if the parentheses are balanced, 0 otherwise.

Description: Checks if the parentheses in the given expression are balanced (i.e., each opening parenthesis has a matching closing parenthesis).

NOTMinus

Parameters: None.

Return Value: None.

Description: Toggles the sign of the result by complementing the value stored in the `minus` variable, which is used to indicate whether the result should be negative or positive.

---

checkLastOperator

Parameters: None.

Return Values:

- `r1`: The last character of the expression.

Description: Checks if the last character of the expression is an operator and returns the character in `r1`.