Computer Architecture - Project III

Professor Billoo

Kevin Kerliu & Shyam Paidipati

Design Rationale

The following sections include an explanation and discussion of the fulfilled requirements of the project, the program architecture, and the challenges faced in completing the project.

Fulfillment of Requirements

Once the user compiles the source code using the Makefile, as explained in the README, the user can run the executable. The program is acts as a basic floating point PMDAS calculator, that is, it works with parentheses, multiplication & division, and addition & subtraction. The program reads the input command, an expression, from the command line and works with up to four operations. Inputs ought to be of the following form:
"<operand> <arithmetic operation> <operand>…<operand> <arithmetic operation> <operand>"
Operands may be integers or floating-point numbers.

Program Architecture

The source code begins with the data section, which is dedicated to 4-byte aligning all prompts, inputs, outputs, addresses, arrays, and any other predefined data.

The text section follows the data section and accounts for the bulk of the file. The code is divided into sections according to function as follows:

main:

main takes care of logistics; it accepts the input character array through argv, which is stored in r1 by default.

read_and_determine_input:

read_and_determine_input is a branch that reads a character from the input string and determines whether the character is a valid input, and if it is, whether it is an operator or a number. It then acts accordingly.

push_left_parenthesis:

push_left_parenthesis is a branch that pushes a right parenthesis in the input expression onto the storage_operator array.

push_right_parenthesis:

push_right_parenthesis is a branch that pushes a left parenthesis in the input expression onto the storage_operator array.

num:

num is a branch that catches poorly formatted inputs and throws an error if it sees one.

push_num:

push_num is a branch that pushes the next number in the input expression onto the storage_operator array.

decimal:

decimal is a branch that catches decimal errors, such as a decimal next to a parentheses or multiple decimals in one number input.

push_op:

push_op is a branch that pushes the next operation in the input expression onto the storage_operator array.

string_end:

string_end is a branch that deals with when we reach the end of the input char array. It also does final error checking.

input_to_func_evaluate:

input_to_func_evaluate is a branch that checks to see if all values and pointers are valid. It first starts checking if left parentheses were inputted.

parentheses_evaluate:

parentheses_evaluate is a branch that addresses if there are parentheses in the inputted char array. It makes calls to num_of_right_parentheses and evaluate to start addressing parentheses.

multiply_and_divide:

multiply_and_divide is a branch that addresses multiply and divide operations. With the null characters inputted into storage_operator.

multiply:

multiply is a branch that executes a multiply operation.

divide:

divide is a branch that executes a divide operation.

set_function_parameters:

set_function_parameters is a branch that allows for reindexing of both arrays.

addition_and_subtraction

addition_and_subtraction is a branch that addresses add and subtract operations.

add:

add is a branch that executes an add operation.

subtract:

subtract is a branch that executes a subtract operation.

end_expression:

end_expression is a branch that sets up for final_check_and_exit.

final_check_and_exit:

final_check_and_exit is a branch that checks if the number of left parentheses equals number of right parentheses. It also makes a call to iterate_storage_num to prepare to exit

loop:

loop is a branch that iterates through storage_operator until the null character at the end of the array is reached.

display_output:

display_output is a branch that displays the computed result with double precision.

error_print:

error_print is a branch that returns an error message in the case that the program has encountered and caught an error.

exit:

exit_program is a branch that restores the link register and exits the program.

evaluate:

evaluate is a function that solves a simple expression, that is, an expression without any parentheses.

num_of_right_parentheses:

num_of_right_parentheses is a function that determines the number of right parentheses in the input expression.

num_of_left_parentheses:

num_of_left_parentheses is a function that determines the number of left parentheses in the input expression, it was later determined that this function was unnecessary, but it shows our thought process in designing the program.

iterate_storage_num:

iterate_storage_num is a function that iterates through the operands, or numbers, to find the next operand to be operated on.

The end of the program is followed by the labels used to access all the 4-byte aligned data. Moreover, external C standard library functions used in the program are listed.

Challenges

The largest challenges we encountered were deciding on how our calculator would work and keeping track of the stack pointer once we decided on an approach. We considered multiple program architectures, with the main differences between architectures being whether the calculation would be done iteratively or concurrently. We finally decided on the iterative stack approach as it dealt with parentheses the best. Keeping track of our stack pointer was crucial for this approach as it allowed us to properly implement the order of operations. This allowed us to evaluate the input command string in the correct order, accounting for the order of operations. Moreover, this approach allowed us to be more efficient and reuse sections of our code.

A unique problem we had was that invalid inputs with the “$” character did not return an error if there was only one instance of the “$” or if it was not the first character in the string. We tried recognizing it with its ASCII value but because the terminal recognizes it specially, this did not work. We were able to catch each case except for “ ./calc “4$5” ”, which would return 4, everything before the “$”. Another challenge was displaying our result, for printing floating point numbers was tricky. A lack of resources and documentation added to this problem, but we were eventually successful.