Assembly Postfix Expression Evaluater

The program takes hexadecimal expressions as input, evaluates expression and prints the result. Input must be valid expression and consists of terms that is seperated by a blank character . Also, terms and results of all subexpressions can be maximum 16 bit values. The program supports the following operations addition, multiplication, integer division, bitwise xor, bitwise or, and bitwise and.

Implementation

There are 22 labels in the program. The program jumps the start label firstly.

1 start:

Sets cx and bl to 0 and jumps the read_input label.

2 read_input:

Reads the first character of input, and checks if it is a whitspace, enter or a operation sign. If the character is one of the above, then jumps the corresponding label, else it is a number. If the ascii value of the character is bigger than ascii number of "9" then jumps the substract_zero label, else jumps to substract_a label.

3 add_last_digit_or_letter:

Shifts cx 1 bit and add temp value to cx. Then jumps read_input label.

4 addition:

Takes last 2 element of the stack and add them then push result to stack. After that jumps to read_input label.

5 multiplication:

Takes last 2 element of the stack and multiply them then push result to stack. After that jumps to read_input label.

6 division:

Takes last 2 element of the stack and make integer division then push result to stack. After that jumps to read_input label.

7 bitwise_xor:

Takes last 2 element of the stack and make bitwise xor then push result to stack. After that jumps to read_input label.

8 bitwise_or:

Takes last 2 element of the stack and make bitwise xor then push result to stack. After that jumps to read_input label.

9 bitwise_and:

Takes last 2 element of the stack and make bitwise xor then push result to stack. After that jumps to read_input label.

10 set_result:

If there is only one element in the expression jumps to setup_string, else push cx to stack then jumps to setup_string.

11 setup_string:

Makes 5th element of the buffer “$”, and jumps to convert_hexadecimal label.

12 number_finish:

If operation boolean is 0, then directly jumps to read_input label, else push cx to stack and then jumps to read_input label.

13 convert_hexadecimal:

Takes the last digit of the result and compare it with 10. If it is greater or equal to 10 then jumps to add_a else jumps to add_zero.

14 assign_char_to_output:

Checks if we convert all values to hexadecimal, if so it jumps to print_cr label. Otherwise jumps to convert_hexadecimal.

15 bitwise_and_starter:

It is just a dummy label to exceed to 128 bytes jumping limit.

16 print_cr:

Prints out a carriage return character.

17 print_out:

Gives the address of string to dx and prints the string (result).

18 digit_actual_value:

Subtracts ascii value of “0” from dx to convert string to integer the number.

19 letter_actual_value:

Subtracts ascii value of “A” from dx to convert string to integer the number.

20 get_digit_ascii:

Adds 48 to dx to convert integer number to string.

21 get_letter_ascii:

Adds 55 to dx to convert integer number to string.

22 exit:

Terminates the program.