Design for COL216 Assignment 2

Shrey J. Patel, 2019CS10400
Aayush Goyal 2019CS10452

March 2021

1 Aim

To design a MIPS Assembly Program for evaluating an expression given in post-fix format.

2 I/O specifications

Input: A string of post-fix expression and a Line Feed to mark its end. The characters of the string should be between '0' to '9' or one of the operators: '*', '+' or '-'

Output: An integer which denotes the value of post-fix expression.

Constraints: The value of expression should be such it could be expressed as a 32 bit signed integer, otherwise there will be overflows and the results will not be correct.

3 Approach

We noted that, for calculating the value of the post fix expression we have to follow a general procedure. We have to keep pushing the digits (characters from '0' to '9') into the stack and whenever we encounter one of the '*', '+', '-' operators we have to apply this operator on the top 2 digits present in the stack.

For example if the post-fix expression is 345*+, the corresponding in-fix expression would be 3+4*5 and it's value should be 23. Now according to our approach we will push 3 then 4 and then 5. Now when we encounter '*' operator we pop 4 and 5 from the stack and multiply them and the product (i.e 20) is pushed onto the stack. Now when we encounter '+' we pop 3 and 20 and we push 3+20 into the stack. Now we encounter a line feed and thus we stop iterating in string and the only value left in the stack (i.e. 23) is our final answer. Note that, for the subtraction operation, we subtract the first

element(i.e. the element currently at the top of the stack) from the second element(i.e. the element which is just below the top element. This is also the same order in which they appear in the input string.

4 Code Design and Implementation in MIPS assembly language

Included in code.asm with every step explained with proper comments. For implementing a stack we have used a register '$sp' which points to the address of the top element. And the stack pointer i.e. '$sp' functions in a reverse sense, so we can traverse up the stack by decreasing the pointer while we can go down by increasing the pointer(which is reverse compared to normal). Hence for a push operation we have reduced the address (i.e. value of $sp) by 4 and for a pop operation we have increased the value of address by 4.

4.1 Algorithm

Using the above approach, we designed an iterative algorithm which iterates over all characters of the string, and in each iteration, the size of stack increases by 1 if it's a digit (b/w 0 to 9) or decreases by 1 if it's an operator ('*', '+' or '-').

Iteration of string is terminated when we encounter a line feed. If the expression was a valid post-fix expression, then only one element should be left in the stack. That value represents the value of the post-fix expression.

4.2 Analysis

n = No. of characters in the string

1) **Space Complexity**: $O(n)$
We have used a constant number of registers and a stack. The maximum size of the stack can go to $O(n)$, so total space complexity is $O(n)$.

2) **Time Complexity**: $O(n)$
There are n characters and so n iterations of the loop. In each iteration we first take a character of string as an input. Then we do either addition, subtraction or multiplication and all of them take constant time, so total time taken across all iterations is $O(n)$

4.3 Testing & Exception Handling

1. We manually created our own test cases, both containing correct and incorrect(logically) in-fix expressions, and provided them as input to the C++ file *in_to_post_convertor.cpp* and obtained the corresponding post-fix

expressions which we then provided as input expressions to our MIPS code file *code.asm*. In case of correct expressions, we obtained the correct value of that expression as an output. While for incorrect post-fix expressions, we have raised an exception named *Invalid Expression*.

Examples:

(a) Correct post-fix expression: 89*3+3-92*+ , which simplifies to 8*9+3-3+9*2, whose value is 90

(b) Incorrect post-fix expression: 33568*+, which simplifies to 3+356*8, which is an invalid input, as only single digit integers are allowed, so our code raises an exception.

2. We checked whether it is throwing exception for the case when invalid characters are given as input by the user and threw the exception *Invalid Character*.

   Example: 3t6++ is an invalid post-fix expression as the character 't' is not allowed inside an arithmetic expression, so our code raises an exception.

3. We have also provided an exception called *Empty Expression*, in case the input post-fix expression is an empty string.

Hence all the test cases have helped us to ensure that our code.asm is correct. Getting correct outputs on some test cases can never be the right criteria to judge if an algorithm is correct or not but extensive checking can still ensure us that we have done the things correctly.