

ASSIGNMENT DESIGN

---

**Problem Statement**

**Write a MIPS Assembly Program for evaluating an expression in postfix format.**

---

APAR AHUJA | ENTRY NO. 2019CS10465

ARNAV TULI | ENTRY NO. 2019CS10424

Course - COL216 | Prof. Preeti Ranjan Panda

# COL216 - Assignment 1

Arnav Tuli | Apar Ahuja

March 1, 2021

## 1 Approach and Design

### 1.1 Input - Output

- Source Code: **assignment\_2.asm** takes input (postfix string) from the **keyboard** and prints the output on **console**.
- Tester Code: **tester.asm** takes input from the **Test Input text file** and prints output on **console**.
- **Integer registers** are used to store numbers and carry out arithmetic. Suitable **overflow** detection is done, whenever the computation exceeds the range of 32-bit signed integer. (error is generated by the code)
- **Maximum size** of character buffer is upper bounded at **159** characters. This also limits the total number of integers to 159. Hence, size of **stack** is initialized to **640** bytes (160 integers) as a safe upper bound.

### 1.2 Algorithm

- Input user string and store it in expression character buffer. Max length of buffer is, **n = 159**
- Initialize **Buffer\_Counter** and **Array\_Counter** to 0.
- **loop:**
  - . - If **Buffer\_Counter**  $\geq$  **n** then **Display\_Result**, else load a character from `buffer[Buffer_Counter]`
  - . - If the character is a **Line Feed**, then **Display\_Result**, else check if it is a valid character.
  - . If the character is a number, then store it in `stack[4*Array_Counter]`, and add 1 to **Array\_Counter**.
  - . - If the character is an operator, then:
    - . - If the **Array\_Counter**  $< 2$ , then raise an error, as number of operators is more than or equal to the number of operands.
    - . - Else **pop** top two elements of the stack, and perform the required calculation. **Push** the result onto the stack and reduce **Array\_Counter** by 1.
  - . If the character is some other character, then raise **invalid character** error.
  - . - Increment **Buffer\_Counter** by 1.
  - . - **Jump** back to **loop**
- **Display\_Result:**
  - . Check if the stack is containing a **single element** or not.
  - . If the stack is **not** containing only one element, then **raise error** as number of operators and operands are not balanced in the given expression.
  - . Else access the **top** element (result of postfix expression) of the stack and display it on the **console**.
- **Loop Invariant:** Before iteration **i**: I have processed the first **i - 1** characters of the postfix expression.
  - .  $1 \leq i \leq \min(\text{length of expression} + 1, n + 1)$ .

### 1.3 Design

#### 1.3.1 Register

Integer Registers:

*v0*: used for making different syscalls

*a0*: used in making syscalls (outputting/inputting strings and integers)

*a1*: used in reading input string (buffer length)

*t0*: used to store the buffer counter (address of current character in buffer)

*t1*: used to store the number of elements in the array (stack)

*t2*: stores the integer counterpart of the character loaded from buffer (e.g. '0' - > 0)

*t3*: stores the character (byte) loaded from the expression buffer

*t4*: stores the array memory offset (to read/write to a particular address in array)

*t5*: stores the right operand while carrying out any computation

*t6*: stores the left operand while carrying out any computation

*t7*: temporarily stores the result of computation, and it is also used for making trivial comparisons

*t8*: used to detect overflow in case of multiplication

*s0*: stores ASCII code of 0 (48)

*s1*: stores ASCII code of 9 (57)

*s2*: stores ASCII code of \* (42)

*s3*: stores ASCII code of + (43)

*s4*: stores ASCII code of - (45)

*s5*: stores ASCII code of \n (10)

*s6*: stores the offset factor for integers (4)

### 1.3.2 Main Memory

#### ASCII:

# array used for implementing stack (max. 160 integers)

**array:** .space 640

# character buffer used to store the input string

**expression:** .space 160

# invalid character error

**error\_invalidChar:** .asciiiz "\nERROR: There is an invalid character present in the expression.\nMake sure that the operands are in the range 0-9, and only +, - and \* operators are used.\nProgram terminating!"

# illegal expression error

**error\_illegalExp:** .asciiiz "\nERROR: The number of operands and number of operators do not match.\nMake sure that the operands are in the range 0-9, and that for each operator exactly 2 operands are provided.\nProgram terminating!"

# overflow error (integer overflow, over 32 bits)

**error\_overflow:** .asciiiz "\nERROR: Arithmetic Overflow"

# input message

**msg\_input:** .asciiiz "Enter the postfix expression that needs to be evaluated: "

# output message

**msg\_output:** .asciiiz "\nThe value of the postfix expression is: "

# separator and newline string

**msg\_separator:** .asciiiz "\n \n-----\n\n"

**msg\_lf:** .asciiiz "\n"

### 1.4 Raising Errors

- if **number of operators and operands** don't match we raise *Error: The number of operands and number of operators do not match. Make sure that the operands are in the range 0-9, and that for each operator exactly 2 operands are provided.*

- if an **invalid character** is entered we raise *Error: There is an invalid character present in the expression. Make sure that the operands are in the range 0-9, and only +, - and \* operators are used.*

- if **no input** is provided then we raise *Error: The number of operands and number of operators do not*

*match. Make sure that the operands are in the range 0-9, and that for each operator exactly 2 operands are provided.*

- if there is **overflow in addition or subtraction** MIPS raises the *Error Arithmetic Overflow*.
- if there is **overflow in multiplication** we raise *Error: Arithmetic Overflow*.
- Multiplication overflow is detected by analysing the contents of **HI** and **LO** registers. Condition used is:
- At overflow, either **HI**  $\neq 0$ , or **LO**  $< 0$  (even when both numbers are same sign), or **LO**  $> 0$  (even when both numbers are opposite sign).

## 2 Testing Strategy

- Total of **243** test cases were generated and tested against as a part our extensive testing strategy
- use *TestCaseGenerator.py* to generate **randomized** test case files with correct output.
- *tester.asm* reads file and prints output on console. We then copy it into a text file and run *Checker.py*.
- *Checker.py* calculates the difference and stores it in "Difference.txt"
- We store count of cases with 0 difference and total number of cases to calculate **accuracy**.
  
- Types of Test Cases Used:
  - . - **Single:** 10 cases with only single digit is provided and no operator.
  - . - **Add:** only + operator is used. 10 test cases each of varying size- Small: 5, Medium: 25 and Large: 50
  - . - **Sub:** only - operator is used. 10 test cases each of varying size - Small: 5, Medium: 25 and Large: 50
  - . - **Mul:** only \* operator is used. 10 test cases each of varying size - Small: 5, Medium: 25 and Large: 50
  - . - **AddSub:** only + and - operator are used. 10 test cases of sizes - Small: 5, Medium: 25 and Large: 50
  - . - **SubMul:** only - and \* operator are used. 10 test cases of sizes - Small: 5, Medium: 25 and Large: 50
  - . - **AddMul** only + and \* operator are used. 10 test cases of sizes - Small: 5, Medium: 25 and Large: 50
  - . - **Mixed:** all operators used. 10 test cases of sizes - Small: 5, Medium: 25, Large: 50 and XLarge: 80
  - . - **Manual:** 8 manually generated cases for extensive testing
  - . - **Corner Cases:** 1. **No Input** 2. **Invalid Post-Order Input** 3. **Invalid Character** 4. **Overflow**

## 3 Result

We achieved a **100%** accuracy across all our test cases, with **0** difference in all outputs.

# Console Interface -

```

Enter the postfix expression that needs to be evaluated: 5
The value of the postfix expression is: 5

Enter the postfix expression that needs to be evaluated: 23+56*-
The value of the postfix expression is: -25

Enter the postfix expression that needs to be evaluated: 23&33*8++
ERROR: There is an invalid character present in the expression.
Make sure that the operands are in the range 0-9, and only +, - and * operators are used.
Program terminating!

Enter the postfix expression that needs to be evaluated: 33+33*
ERROR: The number of operands and number of operators do not match.
Make sure that the operands are in the range 0-9, and that for each operator exactly 2 operands are provided.
Program terminating!

Enter the postfix expression that needs to be evaluated: 34**
ERROR: The number of operands and number of operators do not match.
Make sure that the operands are in the range 0-9, and that for each operator exactly 2 operands are provided.
Program terminating!

Enter the postfix expression that needs to be evaluated:
ERROR: The number of operands and number of operators do not match.
Make sure that the operands are in the range 0-9, and that for each operator exactly 2 operands are provided.
Program terminating!
|
```