# Assembly Language and CA Lab: Final Project Report

*Tien Nguyen Duc*

**Sam Hoang Van - Hien Nguyen Quang**

# Contents

# 1 Problem 5

## 1.1 Project Analysis

### 1.1.1 Project Requirement

Make a mips assembly program caculate expression by using convert infix to postfix expression
Detail Requirement

- Input infix expression, eg: $9 + 2 + 8 * 6$
- Print postfix expression, eg: $9\ 2 + 8\ 6 * +$
- Caculate expression

Number is integer number in range $0 \rightarrow 99$
Operators are plus, minus, multiply, divide

### 1.1.2 Project Requirement

Make some subprogram or function that

- Convert infix expression to postfix expression
- Load postfix expression to stack and caculate

## 1.2 Project Algorithm and Solution

### 1.2.1 Convert infix to postfix expression

Inorder to convert infix to postfix expression, we use stack, string
First we need load infix expression to string i call it is str, create new string for store postfix expression then load character from string and i call it str2, which contain infix expression, by order in string then we consider:
if character is number the save it to str2 which use for store postfix expression
if character is operator, if stack is empty then push it to stack
if incoming character is operator has higher precedence than top of stack then push it to stack
if incoming character is operator has equal precedence with top of stack then pop top of stack and push to str2 and push incoming to stack
if the incoming character has lower precedence than the operator on the top of the stack, pop the stack and save the top operator to str2. Then test the incoming operator against the new top of stack. At the end of the expression, pop and save all operators on the stack to str2. then we have string has contain postfix expression

### 1.2.2 Load postfix expression to stack and caculate

Scan the Postfix string from left to right.
Initialise an empty stack.
If the scannned character is an operand, add it to the stack. If the scanned character is an operator, there will be atleast two operands in the stack.
If the scanned character is an Operator, then we store the top most element of the stack(topStack) in a variable temp. Pop the stack. Now evaluate topStack(Operator)temp. Let the result of this operation be retVal. Pop the stack and Push retVal into the stack.
Repeat this step till all the characters are scanned.
After all characters are scanned, we will have only one element in the stack. Return topStack.

## 1.3 Source code

```
1
2
3
4  .data
5  infix: .space 256
6  postfix: .space 256
7  stack: .space 256
8  prompt: .asciiz "Enter String contain infix expression :"
9  newLine: .asciiz "\n"
10 prompt_postfix: .asciiz "Postfix is: "
11 prompt_result: .asciiz "Result is: "
12 prompt_infix: .asciiz "Infix is: "
13
14 # get infix
15 .text
16  li $v0, 54
17  la $a0, prompt
18  la $a1, infix
19  la $a2, 256
20  syscall
21
22
23  la $a0, prompt_infix
24 li $v0, 4
25 syscall
26
27 la $a0, infix
28 li $v0, 4
29 syscall
30
31 # convert to postfix
32
33 li $s6, -1 # counter
34 li $s7, -1 # Scounter
35 li $t7, -1 # Pcounter
36 while:
37         la $s1, infix  #buffer = $s1
38         la $t5, postfix #postfix = $t5
39         la $t6, stack #stack = $t6
40         li $s2, '+'
41         li $s3, '-'
42         li $s4, '*'
43         li $s5, '/'
44     addi $s6, $s6, 1  # counter ++
```

```
45
46      # get buffer[counter]
47      add $s1, $s1, $s6
48      lb $t1, 0($s1)  # t1 = value of buffer[counter]
49
50
51
52      beq $t1, $s2, operator # '+'
53      nop
54      beq $t1, $s3, operator # '-'
55      nop
56      beq $t1, $s4, operator # '*'
57      nop
58      beq $t1, $s5, operator # '/'
59      nop
60      beq $t1, 10, n_operator # '\n'
61      nop
62      beq $t1, 32, n_operator # ' '
63      nop
64      beq $t1, $zero, endWhile
65      nop
66
67      # push number to postfix
68      addi $t7, $t7, 1
69      add $t5, $t5, $t7
70
71      sb $t1, 0($t5)
72
73
74      lb $a0, 1($s1)
75
76
77      jal check_number
78      beq $v0, 1, n_operator
79      nop
80
81      add_space:
82      add $t1, $zero, 32
83      sb $t1, 1($t5)
84      addi $t7, $t7, 1
85
86      j n_operator
87      nop
88
89      operator:
90      # add to stack ...
91
92      beq $s7, -1, pushToStack
93      nop
94
95      add $t6, $t6, $s7
96      lb $t2, 0($t6) # t2 = value of stack[counter]
97
98      # check t1 precedence
99      beq $t1, $s2, t1to1
100     nop
101     beq $t1, $s3, t1to1
102     nop
103
104     li $t3, 2
105
106     j check_t2
```

```
107     nop
108
109 t1to1:
110     li $t3, 1
111
112     # check t2 precedence
113 check_t2:
114
115     beq $t2, $s2, t2to1
116     nop
117     beq $t2, $s3, t2to1
118     nop
119
120     li $t4, 2
121
122     j compare_precedence
123     nop
124
125
126 t2to1:
127     li $t4, 1
128
129 compare_precedence:
130
131
132     beq $t3, $t4, equal_precedence
133     nop
134     slt $s1, $t3, $t4
135     beqz $s1, t3_large_t4
136     nop
137 ##################
138 # t3 < t4
139 # pop t2 from stack  and t2 ==> postfix
140 # get new top stack do again
141
142     sb $zero, 0($t6)
143     addi $s7, $s7, -1  # scounter ++
144     addi $t6, $t6, -1
145     la $t5, postfix #postfix = $t5
146     addi $t7, $t7, 1
147     add $t5, $t5, $t7
148     sb $t2, 0($t5)
149
150     #addi $s7, $s7, -1  # scounter = scounter - 1
151     j operator
152     nop
153
154 ##################
155 t3_large_t4:
156 # push t1 to stack
157     j pushToStack
158     nop
159 ##################
160 equal_precedence:
161 # pop t2  from stack  and t2 ==> postfix
162 # push to stack
163
164     sb $zero, 0($t6)
165     addi $s7, $s7, -1  # scounter ++
166     addi $t6, $t6, -1
167     la $t5, postfix #postfix = $t5
168     addi $t7, $t7, 1 # pcounter ++
```

```
169     add $t5, $t5, $t7
170
171     sb $t2, 0($t5)
172     j pushToStack
173     nop
174 ################
175 pushToStack:
176
177     la $t6, stack #stack = $t6
178     addi $s7, $s7, 1  # scounter ++
179     add $t6, $t6, $s7
180     sb $t1, 0($t6)
181
182     n_operator:
183     j while
184     nop
185
186 ########################
187 endWhile:
188
189     addi $s1, $zero, 32
190     add $t7, $t7, 1
191     add $t5, $t5, $t7
192     la $t6, stack
193     add $t6, $t6, $s7
194
195 popallstack:
196
197     lb $t2, 0($t6) # t2 = value of stack[counter]
198     beq $t2, 0, endPostfix
199     sb $zero, 0($t6)
200     addi $s7, $s7, -2
201     add $t6, $t6, $s7
202
203     sb $t2, 0($t5)
204     add $t5, $t5, 1
205
206
207     j popallstack
208     nop
209
210 endPostfix:
211 ############################################################################## END
        POSTFIX
212 # print postfix
213 la $a0, prompt_postfix
214 li $v0, 4
215 syscall
216
217 la $a0, postfix
218 li $v0, 4
219 syscall
220
221 la $a0, newLine
222 li $v0, 4
223 syscall
224
225
226 ##############################################################################
        Caculate
227
228 li $s3, 0 # counter
```

```
229 la $s2, stack #stack = $s2
230
231
232 # postfix to stack
233 while_p_s:
234     la $s1, postfix #postfix = $s1
235
236     add $s1, $s1, $s3
237     lb $t1, 0($s1)
238
239
240     # if null
241     beqz $t1 end_while_p_s
242     nop
243
244
245     add $a0, $zero, $t1
246     jal check_number
247     nop
248
249     beqz $v0, is_operator
250     nop
251
252     jal add_number_to_stack
253     nop
254
255     j continue
256     nop
257
258
259     is_operator:
260
261     jal pop
262     nop
263
264     add $a1, $zero, $v0 # b
265
266     jal pop
267     nop
268
269     add $a0, $zero, $v0 # a
270
271     add $a2, $zero, $t1 # op
272
273     jal caculate
274
275
276     continue:
277
278
279
280
281
282     add $s3, $s3, 1 # counter++
283
284     j while_p_s
285     nop
286
287
288 #-----------------------------------------------------------------
289 #Procedure caculate
290 # @brief caculate the number ("a op b")
```

```
# @param[int] a0 : (int) a
# @param[int] a1 : (int) b
# @param[int] a2 : operator(op) as character
#-------------------------------------------------------------------
caculate:
    sw $ra, 0($sp)
    li $v0, 0
    beq $t1, '*', cal_case_mul
    nop
    beq $t1, '/', cal_case_div
    nop
    beq $t1, '+', cal_case_plus
    nop
    beq $t1, '-', cal_case_sub

    cal_case_mul:
        mul $v0, $a0, $a1
        j cal_push
    cal_case_div:
        div $a0, $a1
        mflo $v0
        j cal_push
    cal_case_plus:
        add $v0, $a0, $a1
        j cal_push
    cal_case_sub:
        sub $v0, $a0, $a1
        j cal_push

    cal_push:
        add $a0, $v0, $zero
        jal push
        nop
        lw $ra, 0($sp)
        jr $ra
        nop



#-------------------------------------------------------------------
#Procedure add_number_to_stack
# @brief get the number and add number to stack at $s2
# @param[in] s3 : counter for postfix string
# @param[in] s1 : postfix string
# @param[in] t1 : current value
#-------------------------------------------------------------------
add_number_to_stack:
    # save $ra
    sw $ra, 0($sp)
    li $v0, 0

    while_ants:
        beq $t1, '0', ants_case_0
        nop
        beq $t1, '1', ants_case_1
        nop
        beq $t1, '2', ants_case_2
        nop
        beq $t1, '3', ants_case_3
        nop
        beq $t1, '4', ants_case_4
        nop
```

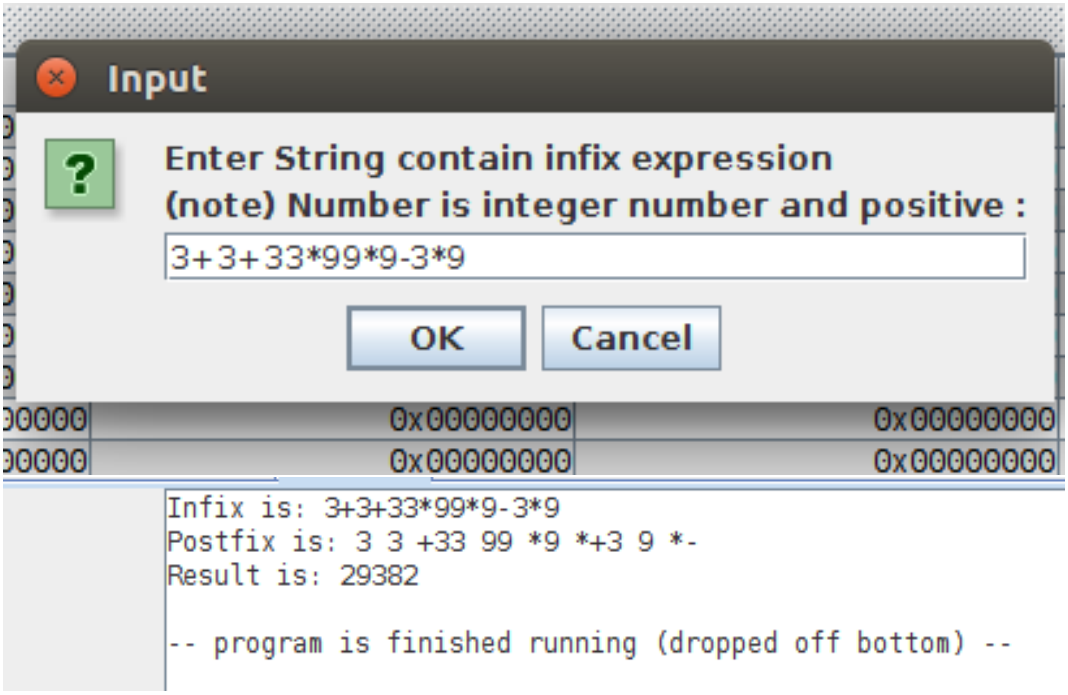
```
353         beq $t1, '5', ants_case_5
354         nop
355         beq $t1, '6', ants_case_6
356         nop
357         beq $t1, '7', ants_case_7
358         nop
359         beq $t1, '8', ants_case_8
360         nop
361         beq $t1, '9', ants_case_9
362         nop
363
364         ants_case_0:
365             j ants_end_sw_c
366         ants_case_1:
367             addi $v0, $v0, 1
368             j ants_end_sw_c
369             nop
370         ants_case_2:
371             addi $v0, $v0, 2
372             j ants_end_sw_c
373             nop
374         ants_case_3:
375             addi $v0, $v0, 3
376             j ants_end_sw_c
377             nop
378         ants_case_4:
379             addi $v0, $v0, 4
380             j ants_end_sw_c
381             nop
382         ants_case_5:
383             addi $v0, $v0, 5
384             j ants_end_sw_c
385             nop
386         ants_case_6:
387             addi $v0, $v0, 6
388             j ants_end_sw_c
389             nop
390         ants_case_7:
391             addi $v0, $v0, 7
392             j ants_end_sw_c
393             nop
394         ants_case_8:
395             addi $v0, $v0, 8
396             j ants_end_sw_c
397             nop
398         ants_case_9:
399             addi $v0, $v0, 9
400             j ants_end_sw_c
401             nop
402         ants_end_sw_c:
403
404             add $s3, $s3, 1 # counter++
405             la $s1, postfix #postfix = $s1
406
407             add $s1, $s1, $s3
408             lb $t1, 0($s1)
409
410             beq $t1, $zero, end_while_ants
411             beq $t1, ' ', end_while_ants
412
413             mul $v0, $v0, 10
414
```

```
415             j while_ants
416
417     end_while_ants:
418         add $a0, $zero, $v0
419         jal push
420         # get $ra
421         lw $ra, 0($sp)
422         jr $ra
423         nop
424
425
426 #-------------------------------------------------------------
427 #Procedure check_number
428 # @brief check character is number or not
429 # @param[int] a0 : character to check
430 # @param[out] v0 : 1 = true; 0 = false
431 #-------------------------------------------------------------
432 check_number:
433
434     li $t8, '0'
435     li $t9, '9'
436
437     beq $t8, $a0, check_number_true
438     beq $t9, $a0, check_number_true
439
440     slt $v0, $t8, $a0
441     beqz $v0, check_number_false
442
443     slt $v0, $a0, $t9
444     beqz $v0, check_number_false
445
446
447     check_number_true:
448
449     li $v0, 1
450     jr $ra
451     nop
452     check_number_false:
453
454     li $v0, 0
455
456     jr $ra
457     nop
458
459
460 #-------------------------------------------------------------
461 #Procedure pop
462 # @brief pop from stack at $s2
463 # @param[out] v0 : value to popped
464 #-------------------------------------------------------------
465 pop:
466     lw $v0, -4($s2)
467     sw $zero, -4($s2)
468     add $s2, $s2, -4
469     jr $ra
470     nop
471
472 #-------------------------------------------------------------
473 #Procedure push
474 # @brief push to stack at $s2
475 # @param[in] a0 : value to push
476 #-------------------------------------------------------------
```

```
push:
    sw $a0, 0($s2)
    add $s2, $s2, 4
    jr $ra
    nop


end_while_p_s:

# add null to end of stack


# print postfix
la $a0, prompt_result
li $v0, 4
syscall


jal pop
add $a0, $zero, $v0
li $v0, 1
syscall

la $a0, newLine
li $v0, 4
syscall
```

Listing 1: Infix - Postfix expression

## 1.4 Preview Image



# 2 Problem 7

## 2.1 Project Analysis

### 2.1.1 Project Requirement

### 2.1.2 Project Requirement

## 2.2 Project Algorithm and Solution

### 2.2.1 1111

### 2.2.2 2222

## 2.3 Source code

## 2.4 Preview Image