TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
——————— * ———————

Báo Cáo KTMT

Sinh viên thực hiện: Hà Công Tuấn (Nhóm trưởng
Mã số sinh viên : **20194704**
Lớp : **IT-E6**
Giáo viên hướng dẫn : **Nguyễn Đình Thuận**

Hà Nội, tháng 07 năm 2021

Đề Tài 5

Biểu thức trung tố hậu tố:

Viết chương trình tính giá trị biểu thức bất kỳ bằng phương pháp duyệt biểu thức hậu tố.Các yêu cầu cụ thể:

1.Nhập vào biểu thức trung tố, ví dụ: 9 + 2 + 8 * 6

2.In ra biểu thức ởdạng hậu tố, ví dụ: 9 2 + 8 6* +

3.Tính ra giá trịcủa biểu thức vừa nhập .

Các hằng sốlà số nguyên, trong phạm vi từ 0 -> 99.

Toán tử bao gồm phép cộng, trừ, nhân, chia lấy thương

Ý Tưởng :

- Chuyển từ trung tố sang hậu tố : nếu là một số thì ta đưa vào hậu tố
  Nếu là một toán tử thì ta đưa vào stack , khi một toán tử khác được đưa vào stack thì toán tử nào có mức độ ưu tiên cao hơn hoặc bằng thì được pop ra khỏi stack đưa vào hậu tố . Lặp đến khi hết biểu thức trung tố thì ta pop toàn bộ các toán tử có trong stack vào hậu tố . Ta được biểu thức hậu tố
- Tính biểu thức hậu tố : nếu là một số thì ta đưa vào stack , khi gặp toán tử thì ta lấy 2 toán hạng trong stack ra và tính dựa trên toán tử đó và push kết quả vào stack . Thực hiện như vậy cho đến khi trong ngăn xếp chỉ còn giá trị duy nhất

CODE:

Khởi tạo :

Cấp phát bộ nhớ cho trung tố, hậu tố, stack :

```
trung_to: .space 256
hau_to: .space 256
stack: .space 256
```

khởi tạo các thanh ghi $s6,$s7,$t7 là các biến đếm

```
li $s6, -1 # counter
li $s7, -1 # Scounter
li $t7, -1 # Pcounter
```

khởi tạo thanh ghi với địa chỉ của nhãn :

```
la $s1, trung_to  #buffer = $s1
la $t5, hau_to #hau_to = $t5
 la $t6, stack #stack = $t6
```

gán giá trị cho các thanh ghi s2,s3,s4,s5 :

```
li $s2, '+'
 li $s3, '-'
li $s4, '*'
li $s5, '/'
```

1. Chuyển từ trung tố sang hậu tố

Vòng while: Load byte từ biểu thức trung tố vào thanh ghi $t1 .
Kiểm tra thanh ghi $t1 nếu là các toàn tử nhẩy đến nhãn operator
nếu $t1 không phải là toán tử (t1 là một số) thì đọc vào biểu thức hậu tố . Sau đó kiểm tra byte tiếp theo xem có phải 1 toán tử hay là một số với hàm Check_number : _ nếu phần tử tiếp theo là toàn tử thì thực hiện vòng lặp while mới
với nhãn n_operator

- Nếu là toán tử thì tiếp tục đưa vào biểu thức hậu tố

Xét hết phần tử trong biểu thức trung tố thì ta nhẩy đến endWhile

```
while:
        la $s1, trung_to  #buffer = $s1
        la $t5, hau_to #hau_to = $t5
        la $t6, stack #stack = $t6
        li $s2, '+'
        li $s3, '-'
        li $s4, '*'
        li $s5, '/'
        addi $s6, $s6, 1  # counter ++ s6 = s6

        # get buffer[counter]
        add $s1, $s1, $s6
        lb $t1, 0($s1)  # t1 = value of buffer[counter]



        beq $t1, $s2, operator # '+'
        nop
        beq $t1, $s3, operator # '-'
        nop
        beq $t1, $s4, operator # '*'
        nop
        beq $t1, $s5, operator # '/'
        nop
        beq $t1, 10, n_operator # '\n'
        nop
        beq $t1, 32, n_operator # ' '
```

```
beq $t1, $zero, endWhile
nop

# push number to hau_to( ??y s? ??n h?u
addi $t7, $t7, 1 # t7 ++
add $t5, $t5, $t7

sb $t1, 0($t5)#doc t1 neu là toan hang vao hau to


lb $a0, 1($s1)# doc ky tu tiep theo trong trung to vao a0


jal check_number # kiem tra ky tu do co phai la 1 so hay khong
beq $v0, 1, n_operator # v0 la gia tri tra ve neu v0 = 1 thi no là 1 so con v0 = 0 thi no la toan
nop

add_space: # them khoang cach vao bieu thuc hau to
add $t1, $zero, 32
sb $t1, 1($t5)
addi $t7, $t7, 1

j n_operator
nop
```

Hàm toán_tu:

Nếu ký tự tiếp theo là 1 toán tử thì ta thêm 1 dấu cách vào Hau_to qua hàm add_space Sau khi lặp vòng while thì ký tự tiếp theo là toán tử được so sánh với các dấu cộng, trừ , nhân , chia , cách , xuống dong (\n)

Nếu là cộng, trừ,nhân,chia thì nhẩy đến nhã là toan_tu

Biến đếm s7 mà bằng -1 thì đây là toán tử đầu tiên được thêm vào stack (pushtostack) Nếu trường hợp đây không phải toán tử đầu tiên thì ta phải xét đến mức độ ưu tiên của toán tử vào : s1 = '+' hoặc s1 = '-' nhảy đến nhãn t1to1 thì thanh ghi $s3 được gán giá trị là 1. Nếu không thì s1 xẽ là nhân hoặc chia thì thanh ghi $s3 được gán là 2

```
toan_tu:
# add to stack ...

beq $s7, -1, pushToStack # toan tu dau tien vao stack
nop

add $t6, $t6, $s7
lb $t2, 0($t6) # t2 = value of stack[counter]

# check t1 precedence
beq $t1, $s2, t1to1
nop
beq $t1, $s3, t1to1
nop

li $t3, 2

j check_t2
nop
```

$t2 lưu giá trị của toán tử ban đầu vào stack . check_t2 tượng tự như check t1 nếu là cộng hoặc trừ thì gán giá trị cho thanh ghi $t4 là bằng 1 còn không thì bằng 2
So sánh mức ưu tiên bằng hàm compare_precedence

```
t1to1:
        li $t3, 1

        # check t2 precedence
check_t2:

        beq $t2, $s2, t2to1
        nop
        beq $t2, $s3, t2to1
        nop

        li $t4, 2

        j compare_precedence
        nop


t2to1:
        li $t4, 1
```

compare_precedence:
nếu giá trị t3 = t4 thì 2 toán tử có mức độ ưu tiên ngang nhau ta pop toán tử hiện có trong stack ra và thêm toán tử mới vào
nếu t3 < t4 thì ta pop toán tử trong stack ra và push toán mới vào stack
nếu t3>t4 thì ta thêm toán tử mới vào stack

```
compare_precedence:


        beq $t3, $t4, equal_precedence
        nop
        slt $s1, $t3, $t4
        beqz $s1, t3_large_t4
        nop
################
# t3 < t4
# pop t2 from stack  and t2 ==> hau_to
# get new top stack do again

        sb $zero, 0($t6)
        addi $s7, $s7, -1  # scounter ++
        addi $t6, $t6, -1
        la $t5, hau_to #hau_to = $t5
        addi $t7, $t7, 1
        add $t5, $t5, $t7
        sb $t2, 0($t5)

        #addi $s7, $s7, -1  # scounter = scounter - 1
        j toan_tu
        nop
```

```
t3_large_t4:
# push t1 to stack
        j pushToStack
        nop
#################
equal_precedence:
# pop t2  from stack  and t2 ==> hau_to
# push to stack

        sb $zero, 0($t6)
        addi $s7, $s7, -1  # scounter ++
        addi $t6, $t6, -1
        la $t5, hau_to #hau_to = $t5
        addi $t7, $t7, 1 # pcounter ++
        add $t5, $t5, $t7

        sb $t2, 0($t5)
        j pushToStack
        nop
```

Pushtostack:

Gán lại giá trị cho thanh ghi $t6 là giá trị địa chỉ của stack

Thêm toán tử mới vào stack

```
pushToStack:

        la $t6, stack #stack = $t6
        addi $s7, $s7, 1  # scounter ++
        add $t6, $t6, $s7
        sb $t1, 0($t6)  # them t1 vao stack

        n_toan_tu:
        j while
        nop
```

Khi đưa hết giá trị trong biểu thức hậu tố ta nhảy đến nhã endwhile trong vòng lặp while ban đầu

Ta đưa hết toán tử trong stack vào biểu thức hậu tố bằng hàm popallstack

```
endWhile:

        #addi $s1, $zero, 32
        add $t7, $t7, 1
        add $t5, $t5, $t7
        la $t6, stack
        add $t6, $t6, $s7

popallstack:

        lb $t2, 0($t6) # t2 = value of stack[counter]
        beq $t2, 0, endPostfix
        sb $zero, 0($t6)
        addi $s7, $s7, -2
        add $t6, $t6, $s7

        sb $t2, 0($t5)
        add $t5, $t5, 1


        j popallstack
        nop
```

Endposfix: in ra màn hình biểu thức hậu tố

```
endPostfix:
#########################################
# print hau_to
la $a0, string_hau_to
li $v0, 4
syscall

la $a0, hau_to
li $v0, 4
syscall

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

2. Tính biểu thức hậu tố

Khởi tạo lại

```
li $s3, 0 # counter
la $s2, stack #stack = $s2
```

Vòng lặp while:

Đọc ký tự từ biểu thức hậu tố vào thanh ghi $t1 kiểm tra các điều kiện có thể sảy ra

```
while_p_s:
        la $s1, hau_to #hau_to = $s1

        add $s1, $s1, $s3
        lb $t1, 0($s1) # load byte vào thanh ghi t1


        # if null
        beqz $t1 end_while_p_s # neu t1 bang 0 thi end
        nop


        add $a0, $zero, $t1 # a0 = t1
        jal check_number     # kiêm tra xem do co phai la toan tu hay so
        nop

        beqz $v0, is_toan_tu # neu V0 = 0 thi la mot toan tu va nhay den opertor
        nop

        jal add_number_to_stack # them so vao stack
        nop

        j continue
        nop
```

Ký load byte ra t1 là 1 số thì ta thêm số đó vào stack

Vòng lặp while_ants: t1 là số từ 0->9 nhảy đến nhãn ants_end_sw_c

```
add_number_to_stack:
        # save $ra
        sw $ra, 0($sp)# luu giá trị đỉnh stack
        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
                beq $t1, '5', ants_case_5
                nop
                beq $t1, '6', ants_case_6
                nop
                beq $t1, '7', ants_case_7
                nop
                beq $t1, '8', ants_case_8
                nop
                beq $t1, '9', ants_case_9
                nop

        ants_case_0:
                j ants_end_sw_c
        ants_case_1:
                addi $v0, $v0, 1
                j ants_end_sw_c
                nop
        ants_case_2:
                addi $v0, $v0, 2
                j ants_end_sw_c
                nop
        ants_case_3:
                addi $v0, $v0, 3
                j ants_end_sw_c
                nop
        ants_case_4:
                addi $v0, $v0, 4
                j ants_end_sw_c
                nop
        ants_case_5:
                addi $v0, $v0, 5
                j ants_end_sw_c
                nop
        ants_case_6:
                addi $v0, $v0, 6
                j ants_end_sw_c
                nop
```

```
ants_case_7:
        addi $v0, $v0, 7
        j ants_end_sw_c
        nop
ants_case_8:
        addi $v0, $v0, 8
        j ants_end_sw_c
        nop
ants_case_9:
        addi $v0, $v0, 9
        j ants_end_sw_c
        nop
```

Ants_end_sw_c:

Tăng biến đếm lên , load byte vào t1 . Trường hợp t1 bằng 0 hoặc bằng space thì kết thúc vòng lặp while_ants nếu không thì v0 trước đó được lưu giá trị tương ứng nhân lên 10 và quay lại vòng lặp . Nhiệp vụ là đọc 1 số có hơn 1 chữ số vào v0

```
ants_end_sw_c:

        add $s3, $s3, 1 # counter++
        la $s1, hau_to #hau_to = $s1

        add $s1, $s1, $s3
        lb $t1, 0($s1)

        beq $t1, $zero, end_while_ants
        beq $t1, ' ', end_while_ants

        mul $v0, $v0, 10 #v0 = 10*v0

        j while_ants
```

End_while_ants:

Gán giá trị v0 có được từ vòng lặp while_ants cho a0

Nhảy đến nhã push bằng jal được lưu địa chỉ lại tại thanh ghi $ra

Sau khi quay lại từ hàm push thì ta load địa chỉ của con trỏ mảng từ thanh ghi $sp

Vào $ra và nhẩy về add_number_to_stack

```
end_while_ants:
        add $a0, $zero, $v0# a0 = v0
        jal push
        # get $ra
        lw $ra, 0($sp) #doc con tro ngan xep
        jr $ra
        nop
```

Push:
load toàn bộ giá trị tại thanh ghi a0 vào s2 .Tăng địa chỉ s2 lên 4 để nhẩy đến stack tiếp theo. Quay lại push ở hàm end_while_ants

```
push:
        sw $a0, 0($s2)
        add $s2, $s2, 4 #s2= s2+4 tăng địa chỉ lên 4
        jr $ra
        nop
```

Jr $ra tại end_while_ants quay về add_number_to_stack thì nhẩy đến nhãn continue
Và thực hiện vòng lặp while_p_s

```
        continue:

        add $s3, $s3, 1 # counter++

        j while_p_s
        nop
```

Khi vòng lặp while_p_s nhận được 1 toán tử thì nhẩy đến is_toan_tu
Jal pop quay lại 1 thì gán a1 = giá trị v0
Jal pop quay lại lần 2 thì gán a0 = giá trị v0
Và gán toán tử bằng a2. Nhảy tơi caculate

```
        is_toan_tu:

        jal pop # nhay den pop
        nop

        add $a1, $zero, $v0 # b

        jal pop
        nop

        add $a0, $zero, $v0 # a

        add $a2, $zero, $t1 # op

        jal caculate
```

Jal pop lần đầu để pop phần tử từ stack ra thanh ghi v0
Khởi tạo lại vùng địa chỉ đấy quay lại
Jal pop lần 2 cũng tượng tự

```
pop:
        lw $v0, -4($s2)#doc ra v0
        sw $zero, -4($s2)# xoa ô day
        add $s2, $s2, -4 #  ??a v? vi tri ?ó
        jr $ra
        nop
```

Calculate: kết quả tính toàn được lưu vào thanh ghi $v0 và nhảy đến cal_push

```
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: thanh ghi $a0 được gán giá trị biểu thức tính lại push vào stack

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

End_while_p_s: khi đọc hết biểu thức hậu tố thí load word tại pop để đọc hết giá trị trong stack vào v0 và in ra màn hình

```
end_while_p_s:

# add null to end of stack


# print hau_to
la $a0, ket_qua
li $v0, 4
syscall


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

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

Đề tài 3

Kiểm tra tốc độ và độ chính xác khi gõ văn bản:

Chương trình sau sẽ đo tốc độ gõ bàn phím và hiển thị kết quả bằng 2 đèn led7 đoạn.

Nguyên tắc:

-Cho một đoạn văn bản mẫu, cố định sẵn trong mã nguồn. Ví dụ“bo mon ky thuat may tinh”

-Sử dụng bộ định thời Timer (trong bộ giả lập Digi Lab Sim) để tạo ra khoảng thời gian để đo. Đây là thời gian giữa 2 lần ngắt, chu kì ngắt.

-Trong thời khoảng đó, người dùng nhập các kí tự từ bàn phím. Ví dụ nhập “bo mOn ky 5huat may tinh”. Chương trình cần phải đếm số kí tự đúng(trong ví dụ trên thì người dùng gõ sai chữ O và 5) mà người dùng đã gõ và hiển thị lên các đèn led.

Ý Tưởng:

- Sử dụng vào lặp để lưu ký tự người dùng gõ vào 1 mảng cố định
- Sử dụng Sycll sleep để đo thời gian từ lúc bắt đầu đến khi kết thúc chương trình
- So sánh chuỗi người dùng nhập vào với chuỗi mẫu để tìm ký tự đúng
- Tốc độ gõ = số phím gõ/thời gian chương trình chạy
- Hiển thị kết quả lên trên đèn Led