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

BÁO CÁO FINAL PROJECT
THỰC HÀNH KIẾN TRÚC MÁY TÍNH

Họ và tên sinh viên: Trần Huy Hoàng

Mã số sinh viên: 20210386

Giảng viên hướng dẫn: TS. Hoàng Văn Hiệp

Mã lớp: 139362

Hà Nội, 07-2023

BÀI TẬP 5 FINAL PROJECT

Đề bài: 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 ở dạng hậu tố, ví dụ: 9 2 + 8 6 * + 3.

Tính ra giá trị của biểu thức vừa nhập

Các hằng số là số nguyên, trong phạm vi từ 0 → 99.

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

I. Các thuật toán sử dụng

Thuật toán 1:

Chuyển biểu thức trung tố (Infix) sang biểu thức hậu tố (Postfix):

- Duyệt biểu thức trung tố từ trái sang phải
- Nếu gặp toán hạng (Operand): đưa ra tức thì
- Nếu gặp dấu mở ngoặc: push nó vào Stack
- Nếu gặp toán tử (Operator): pop khỏi Stack tất cả các toán tử cho đến khi gặp toán tử có độ ưu tiên thấp hơn toán tử đang xét, sau đó nạp toán tử đang xét vào Stack (* / % có cùng độ ưu tiên, + - có cùng độ ưu tiên; * / % có độ ưu tiên cao hơn + -)
- Khi duyệt hết Infix: pop tất cả các toán tử còn lại ra khỏi Stack

Thuật toán 2:

Tính giá trị của biểu thức hậu tố (Postfix):

- Duyệt biểu thức Postfix từ trái sang phải
- Nếu gặp toán hạng thì Push nó vào Stack
- Nếu gặp phép toán (toán tử) thì thực hiện phép toán này với 2 toán hạng được pop ra từ Stack
- Push giá trị vừa tính được vào Stack
- Tiếp tục cho đến khi trong Stack chỉ còn lại 1 giá trị duy nhất - chính là kết quả của biểu thức.

II. Ý tưởng thực hiện

- Chuyển biểu thức trung tố sang biểu thức hậu tố bằng ***Thuật toán 1***
- Tính giá trị biểu thức hậu tố vừa tìm được bằng ***Thuật toán 2***

III. Mã nguồn MIPS

```
.data
    Infix:                  .space 256      # Khai báo vùng nhớ 256 byte chứa biểu thức trung tố
    Postfix:        .space 256      # Khai báo vùng nhớ 256 byte chứa biểu thức hậu tố
    Stack:                  .space 256      # Khai báo vùng nhớ 256 byte cho Stack
    Msg_Input_Infix:        .asciiz  "Nhap bieu thuc trung to \n(Chi su dung cac toan tu: + - * / % va cac toan hang tu 0->99): "
    Prompt_Infix:           .asciiz  "\nBieu thuc trung to: "
    Prompt_Postfix:         .asciiz  "Bieu thuc hau to: "
    Prompt_Value:           .asciiz  "Gia tri bieu thuc trung to la: "
    Error_Msg:              .asciiz  "Bieu thuc ban vua nhap khong hop le!"
    Ask_Msg:                .asciiz  "Ban co muon tiep tuc khong?"
    End_Msg:                .asciiz  "Chuong trinh ket thuc!"
.text
Start:
# Yêu cầu nhập biểu thức trung tố
    li $v0, 54
    la $a0, Msg_Input_Infix  # Hiện ra Input Dialog String và thông báo Msg_Input_Infix
    la $a1, Infix                   # Lưu giá trị người dùng nhập vào Infix
    la $a2, 256                     # Tối đa 256 ký tự
    syscall
    beq $a1, -2, End_Program        # Nếu status là -2 thì rẽ nhánh đến End_Program
    beq $a1, -3, Start              # Nếu status là -3 thì rẽ nhánh đến Start
# In ra biểu thức trung tố
    li $v0, 4
```

```
    la $a0, Prompt_Infix            # In ra Prompt_Infix
    syscall
    li $v0, 4
    la $a0, Infix                   # In ra Infix
    syscall

    # $t0: Thanh ghi biểu diễn trạng thái:
    # Trạng thái 0 ($t0 = 0): Chưa nhập gì
    # Trạng thái 1 ($t0 = 1): Nhập vào toán hạng
    # Trạng thái 2 ($t0 = 2): Nhập vào toán tử
    # Trạng thái 3 ($t0 = 3): Nhập vào dấu '('
    # Trạng thái 4 ($t0 = 4): Nhập vào dấu ')'

    li $t0, 0           # Ban đầu khởi tạo trạng thái 0
    li $t1, 0           # $t1 là biến điếm số chữ số
    li $t2, -1                  # $t2 là offset của Infix
    li $t3, -1                  # $t3 là offset của Postfix
    li $t4, -1                  # $t4 là offset của Stack
    la $s1, Infix               # $s1 lưu địa chỉ của Infix
    la $s2, Postfix   # $s1 lưu địa chỉ của Postfix
    la $s3, Stack               # $s1 lưu địa chỉ của Stack

#Duyệt qua các phần tử của Infix
Scan_Infix:
    addi $t2, $t2, 1            # Mỗi lần scan thì tăng offset của Infix lên 1
    add $t6, $s1, $t2                   # $t6 lưu địa chỉ của ký tự hiện tại đang scan
trong Infix
    lb $t7, 0($t6)                      # Load giá trị của ký tự hiện tại đang scan trong
Infix vào $t7
    beq $t7, ' ', Scan_Infix    # Nếu scan dấu cách ' ', thì quay lại Scan_Infix
    beq $t7, '\n', EOF                  # Nếu đã scan đến cuối Infix thì rẽ nhánh đến
EOF
    beq $t1, 0, One_Digit               # Nếu $t1 = 0 (0 chữ số) thì rẽ nhánh đến
One_Digit
    beq $t1, 1, Two_Digits              # Nếu $t1 = 1 (1 chữ số) thì rẽ nhánh đến
Two_Digits
    beq $t1, 2, Three_Digits    # Nếu $t1 = 2 (2 chữ số) thì rẽ nhánh đến Three_Digits
Continue_Scan_Infix:
    beq $t7, '+', Plus_Minus    # Nếu scan toán tử '+' thì rẽ nhánh đến Plus_Minus
    beq $t7, '-', Plus_Minus    # Nếu scan toán tử '-' thì rẽ nhánh đến Plus_Minus
    beq $t7, '*', Mul_Div_Mod           # Nếu scan toán tử '*' thì rẽ nhánh đến
Mul_Div_Mod
    beq $t7, '/', Mul_Div_Mod           # Nếu scan toán tử '/' thì rẽ nhánh đến
Mul_Div_Mod
    beq $t7, '%', Mul_Div_Mod           # Nếu scan toán tử '%' thì rẽ nhánh đến
Mul_Div_Mod
    beq $t7, '(', Open_Parenthesis      # Nếu scan dấu '(' thì rẽ nhánh đến
Open_Parenthesis
    beq $t7, ')', Close_Parenthesis     # Nếu scan dấu ')' thì rẽ nhánh đến
Close_Parenthesis

# Khi phát hiện nhập sai
```

```
Error_Input:
    li $v0, 55
    la $a0, Error_Msg                   # In ra Error_Msg
    li $a1, 2
    syscall
    j Ask
Finish_Scan_Infix:

# In ra Postfix
    li $v0, 4
    la $a0, Prompt_Postfix              # In ra Prompt_Postfix:
    syscall
    li $t5,-1                           # $t5 dùng để lưu trữ số lần lặp qua các phân tử của
Postfix (offset Postfix hiện tại)
Print_Postfix:
    addi $t5, $t5, 1                    # Mỗi lần lặp thì tăng $t5 lên 1
    add $t9, $s2, $t5                   # Gán $t9 bằng địa chỉ của phần tử Postfix hiện
tại
    lbu $t8, ($t9)                      # Load giá trị của phần tử Postfix hiện tại vào
$t8
    bgt $t5, $t3, Finish_Print_Postfix# Nếu $t5 > $t3 tức là đã duyệt qua hết các phần tử
trong Postfix thì rẽ nhánh đến Finish_Print_Postfix
    bgt $t8, 99, Print_Operator         # So sánh giá trị của phần tử hiện tại của
Postfix: Nếu phần tử đó lơn hơn 99 thì đó là 1 toán tử (vì toan hạng chỉ từ 0->99)
    # Nếu phần tử đó không là toán tử thì sẽ là toán hạng
    li $v0, 1
    add $a0,$t8,$0                      # In ra $t8 vừa load được từ Postfix
    syscall
    li $v0, 11
    li $a0, ' '                         # In ra dấu cách ' '
    syscall
    j Print_Postfix                     # Nhảy đến Print_Postfix để tiếp tục vòng lặp
    Print_Operator:
    li $v0, 11
    addi $t8, $t8, -100                 # Giải mã toán tử bằng cách trừ đi 100(vì mã
hóa đã cộng toán tử thêm 100)
    add $a0, $t8, $0                    # In toán tử
    syscall
    li $v0, 11
    li $a0, ' '                         # In ra dấu cách ' '
    syscall
    j Print_Postfix                     # Vòng lặp Print_Postfix
Finish_Print_Postfix:
    li $v0, 11
    li $a0, '\n'                        # Xuống dòng
    syscall

# Tính giá trị biểu thức Postfix
    li $t9, -4                          # $t9 dùng để lưu offset của Stack
    li $t5, -1                          # $t5 dùng để lưu trữ số lần lặp qua các phân tử
của Postfix (offset Postfix hiện tại)
Calculate_Postfix:
```

```
    addi $t5, $t5, 1                    # Mỗi lần lặp thì tăng $t5 lên 1
    add $t6, $s2, $t5                   # Gán $t6 bằng địa chỉ của phần tử Postfix hiện
tại
    lbu $t7, 0($t6)                     # Load giá trị của phần tử Postfix hiện tại vào
$t7
    bgt $t5, $t3, Print_Calculated_Value    # Nếu $t5 > $t3 tức là đã duyệt qua hết các
phần tử trong Postfix thì rẽ nhánh đến Finish_Print_Postfix
    bgt $t7, 99, Calculate              # Nếu giá trị của phần tử Postfix hiện tại > 99 thì
rẽ nhánh đến Calculate
    addi $t9, $t9, 4                    # Mỗi lần lặp thì tăng $t9 lên 4
    add $t8, $s3, $t9                   # Load địa chỉ top Stack vào $t8
    sw $t7, 0($t8)                      # Store giá trị ở $t7 vào 0($t8)
    j Calculate_Postfix                 # Nhảy đến Calculate_Postfix

Calculate:
    # Pop 1 số
    add $t8, $s3, $t9                   # Load địa chỉ top Stack
    lw $t0, 0($t8)                      # Load giá trị của phần tử có địa chỉ ở $t8 vào
$t0
    # Pop số tiếp theo
    addi $t9, $t9, -4                   # Giảm $t9 đi 4
    add $t8,$s3,$t9                     # Load địa chỉ của phần tử tiếp theo trong Stack
vào $t8
    lw $t1, 0($t8)                      # Load giá trị của phần tử có địa chỉ ở $t8 vào
$t1
    # Giải mã
    beq $t7, 143, Plus                  # Nếu $t7 = 143 thì rẽ nhánh đến Plus (mã
ASCII của '+' là 43)
    beq $t7, 145, Minus                 # Nếu $t7 = 145 thì rẽ nhánh đến Minus (mã
ASCII của '-' là 45)
    beq $t7, 142, Mul                   # Nếu $t7 = 142 thì rẽ nhánh đến Mul (mã ASCII
của '*' là 42)
    beq $t7, 147, Div                   # Nếu $t7 = 147 thì rẽ nhánh đến Div (mã ASCII
của '/' là 47)
    beq $t7, 137, Mod                   # Nếu $t7 = 137 thì rẽ nhánh đến Mod (mã
ASCII của '%' là 37)
    Plus:
    add $t0, $t0, $t1                   # Gán $t0 = $t0 + $t1 ( $t0 bằng tổng 2 phần tử
vào Pop ra từ Stack)
    sw $t0, 0($t8)                      # Push giá trị của $t0 vào Stack (0($t8) là địa chỉ
của top Stack hiện tại)
    j Calculate_Postfix                 # Nhảy đến Calculate_Postfix
    Minus:
    sub $t0, $t1, $t0               # Gán $t0 = $t1 - $t0 ( $t0 bằng hiệu 2 phần tử vào Pop
ra từ Stack)
    sw $t0, 0($t8)                      # Push giá trị của $t0 vào Stack (0($t8) là địa chỉ
của top Stack hiện tại)
    j Calculate_Postfix                 # Nhảy đến Calculate_Postfix
    Mul:
    mul $t0, $t1, $t0               # Gán $t0 = $t1 * $t0 ( $t0 bằng tích 2 phần tử vào Pop
ra từ Stack)
```

```
    sw $t0, 0($t8)                  # Push giá trị của $t0 vào Stack (0($t8) là địa chỉ
của top Stack hiện tại)
    j Calculate_Postfix             # Nhảy đến Calculate_Postfix
    Div:
    div $t1, $t0                    # Thực hiện $t1/$t0 và thương của phép chia
được lưu ở thanh ghi LO
    mflo $t0                # Gán $t0 = giá trị thanh ghi LO
    sw $t0, 0($t8)                  # Push giá trị của $t0 vào Stack (0($t8) là địa chỉ
của top Stack hiện tại)
    j Calculate_Postfix             # Nhảy đến Calculate_Postfix
    Mod:
    div $t1, $t0                    # Thực hiện $t1/$t0 và dư của phép chia được
lưu ở thanh ghi HI
    mfhi $t0                # Gán $t0 = giá trị thanh ghi HI
    sw $t0, 0($t8)                  # Push giá trị của $t0 vào Stack (0($t8) là địa chỉ
của top Stack hiện tại)
    j Calculate_Postfix             # Nhảy đến Calculate_Postfix

#In ra giá trị sau khi tính toán
Print_Calculated_Value:
    li $v0, 4
    la $a0, Prompt_Value            # In ra Prompt_Value
    syscall
    li $v0, 1
    lw $a0, 0($t8)                  # Load giá trị ở địa chỉ 0($t8) vào $a0 để in ra
(sau khi tính toán thì giá trị ở 0($t8) chính là kết quả của biểu thức hậu tố)
    syscall
    li $v0, 11
    li $a0, '\n'                    # Xuống dòng
    syscall

# Hỏi người dùng có muốn tiếp tục chương trình hay không
Ask:        li $v0, 50
    la $a0, Ask_Msg                         # In ra Ask_Msg
    syscall
    beq $a0, 0, Start               # Nếu người dùng chọn Yes thì rẽ nhánh đến
Start
    beq $a0, 2, Ask                 # Nếu người dùng chọn Cancel thì rẽ nhánh đến
Ask

# Nếu người dùng không chọn Yes hoặc Cancel => Người dùng chọn No => Kết thúc
chương trình
End_Program:
    li $v0, 55
    la $a0, End_Msg                         # In ra End_Msg
    li $a1, 1
    syscall
    li $v0, 10                      # Exit
    syscall
#
================================================================
================================================
```

```
# Các chương trình con
EOF:
    beq $t0, 2, Error_Input                 # Nếu kết thúc scan Infix với trạng thái 2 (toán tử ở cuối) thì rẽ nhánh đến Error_Input để thông báo lỗi
    beq $t0, 3, Error_Input                 # Nếu kết thúc scan Infix với trạng thái 3 (dấu '(' ở cuối) thì rẽ nhánh đến Error_Input để thông báo lỗi
    j Pop_All                               # Nhảy đến Pop_All

One_Digit:                                  # Khi $t1 = 0, One_Digit thực hiện kiểm tra xem có nhận vào chữ số nào hay không
    beq $t7, '0', Store_1_Digit             # Nếu $t7 là 1 trong các chữ số 0->9 thì rẽ nhánh đến Store_1_Digit
    beq $t7, '1', Store_1_Digit
    beq $t7, '2', Store_1_Digit
    beq $t7, '3', Store_1_Digit
    beq $t7, '4', Store_1_Digit
    beq $t7, '5', Store_1_Digit
    beq $t7, '6', Store_1_Digit
    beq $t7, '7', Store_1_Digit
    beq $t7, '8', Store_1_Digit
    beq $t7, '9', Store_1_Digit
    j Continue_Scan_Infix                   # Nếu không phải là chữ số thì nhảy Continue_Scan_Infix

Two_Digits:                                 # Khi $t1 = 1, Two_Digits thực hiện kiểm tra xem có nhận vào chữ số nào hay không
    beq $t7, '0', Store_2_Digit             # Nếu $t7 là 1 trong các chữ số 0->9 thì rẽ nhánh đến Store_2_Digit
    beq $t7, '1', Store_2_Digit
    beq $t7, '2', Store_2_Digit
    beq $t7, '3', Store_2_Digit
    beq $t7, '4', Store_2_Digit
    beq $t7, '5', Store_2_Digit
    beq $t7, '6', Store_2_Digit
    beq $t7, '7', Store_2_Digit
    beq $t7, '8', Store_2_Digit
    beq $t7, '9', Store_2_Digit
    # Nếu không nhận chữ số thứ 2:
    jal Number_To_Postfix                   # Jump and link đến Number_To_Postfix
    j Continue_Scan_Infix                   # Nhảy đến Continue_Scan_Infix

Three_Digits:                               # Khi $t1 = 2, Three_Digits thực hiện kiểm tra xem có nhận vào chữ số nào hay không, nếu nhận và chữ số thứ 3 --> Rẽ nhánh đến Error_Input để thông báo lỗi
    beq $t7, '0', Error_Input
    beq $t7, '1', Error_Input
    beq $t7, '2', Error_Input
    beq $t7, '3', Error_Input
    beq $t7, '4', Error_Input
    beq $t7, '5', Error_Input
    beq $t7, '6', Error_Input
    beq $t7, '7', Error_Input
```

```
    beq $t7, '8', Error_Input
    beq $t7, '9', Error_Input
    # Nêu không nhận chữ số thứ 3:
    jal Number_To_Postfix                   # Jump and link đến Number_To_Postfix
    j Continue_Scan_Infix                   # Nhảy đến Continue_Scan_Infix

Store_1_Digit:                                  # Gặp toán hạng là số có 1 chữ số
    beq $t0, 4, Error_Input                 # Nếu nhận vào 1 số  sau dấu ')' thì rẽ nhánh
đến Error_Input ( $t0 = 4 là đang ở trạng thái '(' )
    addi $s7, $t7, -48                      # $s7 lưu chữ số thứ nhất
    li $t1, 1                           # Gán $t1 = 1 ($t1 đếm số chữ số)
    li $t0, 1                           # Gán $t0 = 1 để chuyển sang trạng thái sang 1
    j Scan_Infix                            # Nhảy đến Scan_Infix

Store_2_Digit:                                  # Gặp toán hạng là số có 2 chữ số
    beq $t0, 4, Error_Input                 # Nếu nhận vào 1 số  sau dấu ')' thì rẽ nhánh
đến Error_Input ( $t0 = 4 là đang ở trạng thái '(' )
    addi $s6, $t7, -48                      # $s6 lưu chữ số thứ 2
    mul $s7, $s7, 10                        # Gán $s7 = $s7*10
    add $s7, $s7, $s6                       # $s7 =  $s7 + $s6 = (chứ số thứ nhất)*10 +
(chữ số thứ 2)
    li $t1, 2                           # Gán $t1 = 2 ($t1 đếm số chữ số)
    li $t0, 1                           # Gán $t0 = 1 để chuyển sang trạng thái sang 1
    j Scan_Infix                            # Nhảy đến Scan_Infix

Number_To_Postfix:                              # Lưu toán hạng (số) vào Postfix
    beq $t1, 0, End_Number_To_Postfix       # Nếu $t1 = 0 thì End_Number_To_Postfix
    addi $t3, $t3, 1                    # Tăng offset của Postfix lên 1 ($t3 là offset của Postfix)
    add $t9, $s2, $t3                       # Load địa chỉ của phần tử hiện tại của Postfix
vào $t9
    sb $s7,($t9)                            # Store số đang lưu ở $s7 vào Postfix
    li $t1, 0                           # Gán $t1 = 0 để trở về trạng thái 0 chữ số
    End_Number_To_Postfix:
    jr $ra

Plus_Minus:                                     # Input là + -
    beq $t0, 2, Error_Input                 # Thông báo lỗi nếu nhận vào 1 toán tử ngay
sau 1 toán tử
    beq $t0, 3, Error_Input                 # Thông báo lỗi nếu nhận vào 1 toán tử ngay
sau dấu mở ngoặc
    beq $t0, 0, Error_Input                 # Thông báo lỗi nếu nhận vào 1 toán tử ngay
đầu tiên
    li $t0, 2                           # Chuyển trạng thái input sang 2
    Continue_Plus_Minus:
    beq $t4, -1, Push_Stack                 # Nếu không có gì trong Stack thì đẩy toán tử
vào
    add $t9, $s3, $t4                       # Load địa chỉ của top Stack vào $t9
    lb $t8, ($t9)                           # Load giá trị tại top Stack vào $t8
    beq $t8, '(', Push_Stack            # Nếu top Stack đang là '(' thì rẽ nhánh đến Push_Stack
    beq $t8, '+', Equal_Precedence          # Nếu top Stack hiện tại là + thì rẽ nhánh đến
Equal_Precedence
```

```
    beq $t8, '-', Equal_Precedence          # Nếu top Stack hiện tại là - thì rẽ nhánh đến
Equal_Precedence
    beq $t8, '*', Lower_Precedence          # Nếu top Stack hiện tại là * thì rẽ nhánh đến
Lower_Precedence
    beq $t8, '/', Lower_Precedence          # Nếu top Stack hiện tại là / thì rẽ nhánh đến
Lower_Precedence
    beq $t8, '%', Lower_Precedence          # Nếu top Stack hiện tại là % thì rẽ nhánh đến
Lower_Precedence

Mul_Div_Mod:                                    # Input là * / %
    beq $t0, 2, Error_Input                 # Thông báo lỗi nếu nhận vào 1 toán tử ngay
sau 1 toán tử
    beq $t0, 3, Error_Input                 # Thông báo lỗi nếu nhận vào 1 toán tử ngay
sau dấu mở ngoặc
    beq $t0, 0, Error_Input                 # Thông báo lỗi nếu nhận vào 1 toán tử ngay
đầu tiên
    li $t0, 2                           # Chuyển trạng thái input sang 2
    beq $t4, -1, Push_Stack                 # Nếu không có gì trong Stack thì đẩy toán tử
vào Stack
    add $t9, $s3, $t4                       # Load địa chỉ của top Stack vào $t9
    lb $t8, ($t9)                           # Load giá trị tại top Stack vào $t8
    beq $t8, '(', Push_Stack            # Nếu top Stack là '(' thì đẩy toán tử hiện tại vào Stack
    beq $t8, '+', Push_Stack            # Nếu top Stack là '+' thì đẩy toán tử hiện tại vào Stack
    beq $t8, '-', Push_Stack            # Nếu top stack hiện tại là '-' thì đẩy toán tử hiện tại vào
stack
    beq $t8, '*', Equal_Precedence          # Nếu top stack hiện tại là * thì rẽ nhánh đến
Equal_Precedence
    beq $t8, '/', Equal_Precedence          # Nếu top stack hiện tại là / thì rẽ nhánh đến
Equal_Precedence
    beq $t8, '%', Equal_Precedence          # Nếu top stack hiện tại là % thì rẽ nhánh đến
Equal_Precedence

Open_Parenthesis:                               # Input là '('
    beq $t0, 1, Error_Input                 # Thông báo lỗi nếu nhận vào dấu '(' ngay sau 1
toán hạng
    beq $t0, 4, Error_Input                 # Thông báo lỗi nếu nhận vào dấu '(' ngay sau
dấu ')'
    li $t0, 3                           # Gán $t0 = 3 để chuyển trạng thái input sang 3
    j Push_Stack                            # Nhảy đến Push_Stack để đẩy toán tử hiện tại
vào Stack

Close_Parenthesis:                              # Input là ')'
    beq $t0, 2, Error_Input                 # Thông báo lỗi nếu nhận vào dấu ')' ngay sau 1
toán hạng
    beq $t0, 3, Error_Input                 # Thông báo lỗi nếu nhận vào dấu ')' ngay dấu '('
    li $t0, 4                           # Gán $t0 = 4 để chuyển trạng thái input sang 4
    add $t9, $s3, $t4                       # Load địa chỉ của top Stack vào $t9
    lb $t8, ($t9)                           # Load giá trị tại top Stack vào $t8
    Continue_Close_Parenthesis:
    beq $t4, -1, Error_Input            # Nếu không có gì trong stack thì thông báo lỗi
    add $t9, $s3, $t4                       # Load địa chỉ của top Stack vào $t9
    lb $t8, ($t9)                           # Load giá trị tại top Stack vào $t8
```

```
    beq $t8, '(', Match_Parenthesis          # Nếu tìm thấy dấu '(' tương ứng thì rẽ nhánh
đến Match_Parathesis
    jal Pop_Operator_To_Postfix              # Pop các toán tử trong Stack sang Postfix
    j Continue_Close_Parenthesis             # Nhảy đến Continue_Close_Parenthesis để
tiếp tục vòng lặp

Equal_Precedence:                            # Nạp vào + - va top Stack đang là + - hoặc
nhập vào * / % va top Stack đang là * / % (độ ưu tiên của toán tử hiện tại thấp = độ ưu tiên
của toán tử ở top Stack)
    jal Pop_Operator_To_Postfix              # Jump and link đến Pop_Operator_To_Postfix
để pop các toán tử trong Stack sang Postfix
    j Push_Stack                             # Nhảy đến Push_Stack để đẩy toán tử hiện tại
vào Stack

Lower_Precedence:                            # Nạp vào + - và top Stack đang là * / % (độ ưu
tiên của toán tử hiện tại thấp < độ ưu tiên của toán tử ở top Stack)
    jal Pop_Operator_To_Postfix              # Jump and link đến Pop_Operator_To_Postfix
để pop các toán tử trong Stack sang Postfix
    j Continue_Plus_Minus                    # Nhảy đến Continue_Plus_Minus để tiếp tục
thực hiện Plus_Minus

Push_Stack:                                          # Đẩy toán tử hoặc dấu ngoặc vào
Stack
    add $t4, $t4, 1                          # Tăng offset của Postfix lên 1
    add $t9, $s3, $t4                        # Load địa chỉ của top Stack vào $t9
    sb $t7, ($t9)                            # Lưu toán tử hiện tại vào top Stack
    j Scan_Infix                             # Nhảy đến Scan_Infix

Pop_Operator_To_Postfix:                     # Pop toán tử trong Stack rồi đưa vào Postfix
    addi $t3, $t3, 1                 # Tăng offset của Postfix
    add $t9, $t3, $s2                        # Load địa chỉ của Postfix hiện tại
    addi $t8, $t8, 100                       # Mã hóa toán tử bằng cách cộng thêm 100
    sb $t8, ($t9)                            # Store toán tử vào vị trí hiện tại Postfix
    addi $t4, $t4, -1                # Giảm offset ở đỉnh Stack đi 1
    jr $ra

Match_Parenthesis:                           # Loại bỏ 1 cặp dấu ngoặc nếu chúng tương
ứng với nhau
    addi $t4, $t4, -1                # Giảm offset của Stack đi 1
    j Scan_Infix                             # Nhảy đến Scan_Infix

Pop_All:                                     # Pop mọi toán tử đang còn trong Stack sang
Postfix
    jal Number_To_Postfix                    # Jump and link Number_To_Postfix
    beq $t4, -1, Finish_Scan_Infix           # Nếu Stack trống thì Finish_Scan_Infix
    add $t9, $s3, $t4                        # Load địa chỉ của top Stack vào $t9
    lb $t8, ($t9)                            # Load giá trị tại top Stack vào $t8
    beq $t8, '(', Error_Input        # Nếu trong Stack vẫn còn '(' hoặc ')'thì rẽ nhánh đến
Error_Input để thông báo lỗi
    beq $t8, ')', Error_Input
    jal Pop_Operator_To_Postfix              # Jump and link đến Pop_Operator_To_Postfix
    j Pop_All                                # Nhảy đến Pop_All để tiếp tục lặp
```