**FINAL PROJECT REPORT**

Assembly Mini Project Course 20192

Student Name: BÙI TÚ HOÀNG

Student ID: 20176761

**Problem number 5:**

Make a MIPS assembly program calculate expression by using convert infix to postfix expression

Detail Requirement

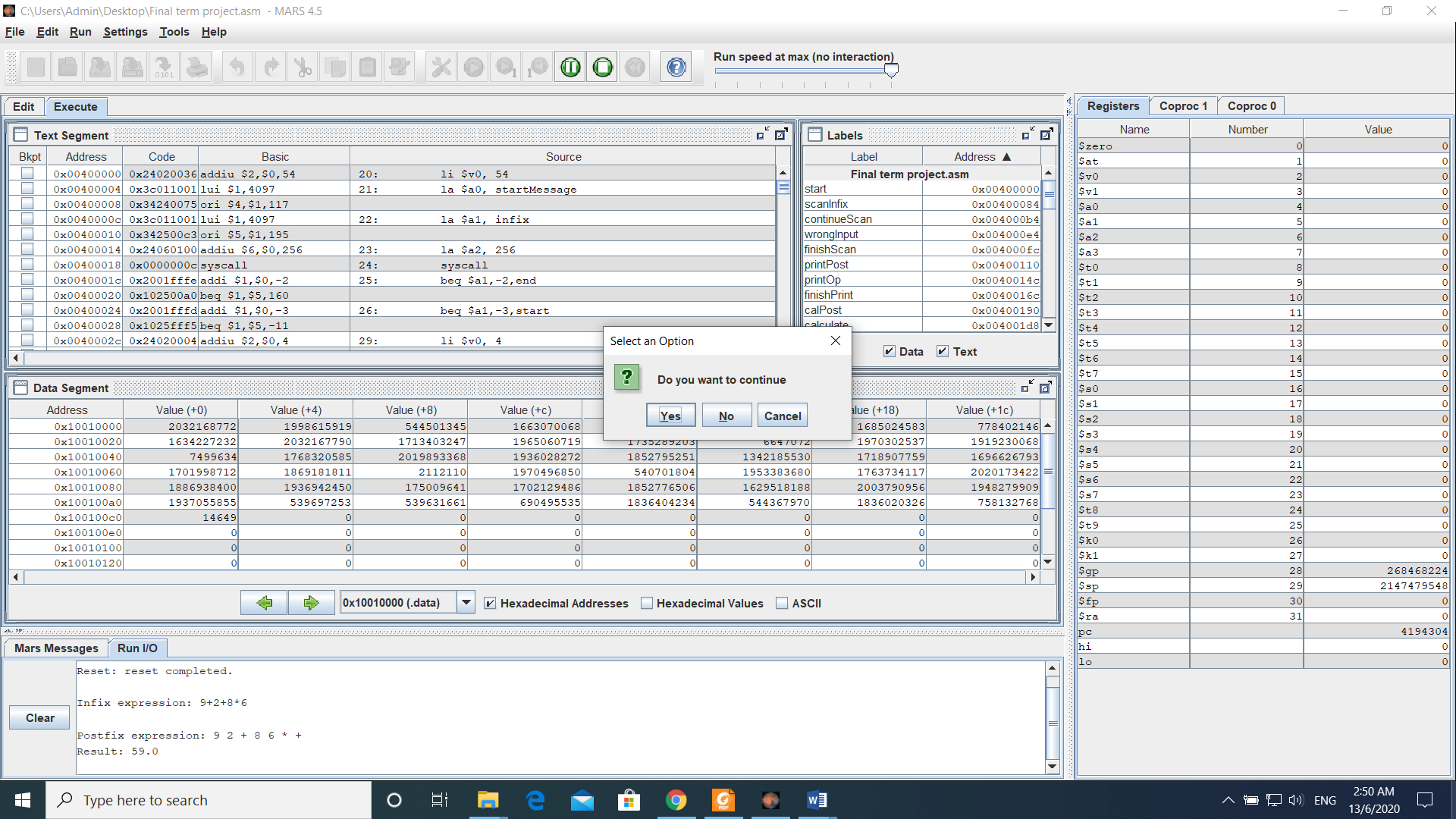
* Input infix expression, e.g. 9 + 2 + 8 *∗* 6
* Print postfix expression, e.g. 9 2 + 8 6 *∗* +
* Calculate expression

Number is integer number in range 0 🡪 99

Operators are plus, minus, multiply, divide

**How to use:**

* Open MARS, assemble the file and run the program
* Enter an expression into the input dialog and click OK
* The result will be shown at the Run I/O tab
* A confirm dialog will appear to ask user continue the program or not. Click Yes to continue the program, click No to exit the program.



**How the program work:**

In order to convert infix to postfix expression, we will use stack and string

First we need load infix expression to a string called “infix”, then create a new string for store postfix expression, called “postfix” and an operator stack. We will load character from string “infix” into “postfix”, by an order:

If the character is a number:

* If the number has one or two digits, we store and then save it to “postfix.
* If the number has three digits, an error occurs and warning dialog will appear.

In case the character is an operator:

* If the operator stack is empty, then we push the character to the stack.
* If the incoming character is an operator which has a higher precedence than the top of the operator stack, then push it to the stack.
* If the incoming character is an operator which has equal precedence with an operator on the top of the stack, we pop and push the top of stack to “postfix” and push the incoming character to stack.
* If the incoming character has lower precedence than the operator on the top of the stack, we pop and push the top of stack to “postfix” and then test the incoming operator against the new top of stack.

At the end of the expression, pop and save all operators in the stack to “postfix” and we have a string which contain postfix expression.

In order to calculate the expression, we will scan the “postfix” string from left to right.  
Firstly, we initialize an empty stack. If the scanned character is an operand, then add it to the stack. If the scanned character is an operator, there must be at least two operands in the stack, so we will pop them and calculate the operation. The result of this operation will be put into the stack. Repeat these steps until all the characters are scanned. Finally, we will have only one element in the stack which is the result of the expression.

**Source code:**

.data

askMessage: .asciiz "Do you want to continue"

endMessage: .asciiz "Goodbye. Thank you for using me"

errorMessage: .asciiz "Input error"

infixMessage: .asciiz "Infix expression: "

postfixMessage: .asciiz "Postfix expression: "

resultMessage: .asciiz "Result: "

startMessage: .asciiz "Enter infix expression\nNote: only allowed to use + - \* / ()\nNumber from 00-99"

infix: .space 256

postfix: .space 256

operator: .space 256

converter: .word 1

wordToConvert: .word 1

stack: .float

.text

start:

# Get infix expression

li $v0, 54

la $a0, startMessage

la $a1, infix

la $a2, 256

syscall

beq $a1,-2,end

beq $a1,-3,start

# Print infix

li $v0, 4

la $a0, infixMessage

syscall

li $v0, 4

la $a0, infix

syscall

li $v0, 11

li $a0, '\n'

syscall

# Status

li $s7,0 # 0 = initially receive nothing

# 1 = receive number

# 2 = receive operator

# 3 = receive (

# 4 = receive )

li $t9,0 # Count digit

li $t5,-1 # Postfix top offset

li $t6,-1 # Operator top offset

la $t1, infix # Infix current byte address +1 each loop

la $t2, postfix

la $t3, operator

addi $t1,$t1,-1 # Set initial address of infix to -1

# Convert to postfix

scanInfix: # Loop for each character in postfix

# Check all valid input option

addi $t1,$t1,1 # Increase infix position

lb $t4, ($t1) # Load current infix input

beq $t4, ' ', scanInfix # If scan spacebar ignore and scan again

beq $t4, '\n', EOF # Scan end of input --> pop all operator to postfix

beq $t9,0,digit1 # If state is 0 digit

beq $t9,1,digit2 # If state is 1 digit

beq $t9,2,digit3 # If state is 2 digit

continueScan:

beq $t4, '+', plusMinusOperator

beq $t4, '-', plusMinusOperator

beq $t4, '\*', multiplyDivideOperator

beq $t4, '/', multiplyDivideOperator

beq $t4, '(', openBracket

beq $t4, ')', closeBracket

wrongInput: # When detect wrong input situation

li $v0, 55

la $a0, errorMessage

li $a1, 2

syscall

j ask

finishScan:

# Print postfix expression

# Print prompt:

li $v0, 4

la $a0, postfixMessage

syscall

li $t6,-1 # Load current of Postfix offset to -1

printPost:

addi $t6,$t6,1 # Increment current of Postfix offset

add $t8,$t2,$t6 # Load address of current Postfix

lbu $t7,($t8) # Load value of current Postfix

bgt $t6,$t5,finishPrint # Print all postfix --> calculate

bgt $t7,99,printOp # If current Postfix > 99 --> an operator

# If not then current Postfix is a number

li $v0, 1

add $a0,$t7,$zero

syscall

li $v0, 11

li $a0, ' '

syscall

j printPost # Loop

printOp:

li $v0, 11

addi $t7,$t7,-100 # Decode operator

add $a0,$t7,$zero

syscall

li $v0, 11

li $a0, ' '

syscall

j printPost # Loop

finishPrint:

li $v0, 11

li $a0, '\n'

syscall

# Calculate

li $t9,-4 # Set top of stack offset to -4

la $t3,stack # Load stack address

li $t6,-1 # Load current of Postfix offset to -1

l.s $f0,converter # Load converter

calPost:

addi $t6,$t6,1 # Increment current of Postfix offset

add $t8,$t2,$t6 # Load address of current Postfix

lbu $t7,($t8) # Load value of current Postfix

bgt $t6,$t5,printResult # Calculate for all postfix --> print

bgt $t7,99,calculate # If current Postfix > 99 --> an operator

# Then popout 2 number to calculate

# If not then current Postfix is a number

addi $t9,$t9,4 # Current stack top offset

add $t4,$t3,$t9 # Current stack top address

sw $t7,wordToConvert

l.s $f10,wordToConvert # Load number to coproc1 to convert to float

div.s $f10,$f10,$f0

s.s $f10,($t4) # Push number into stack

sub.s $f10,$f10,$f10 # Reset f10

j calPost # Loop

calculate:

# Pop 1 number

add $t4,$t3,$t9

l.s $f3,($t4)

# Pop next number

addi $t9,$t9,-4

add $t4,$t3,$t9

l.s $f2,($t4)

# Decode operator

beq $t7,143,plus

beq $t7,145,minus

beq $t7,142,multiply

beq $t7,147,divide

plus:

add.s $f1,$f2,$f3

s.s $f1,($t4)

sub.s $f2,$f2,$f2 # Reset f2 f3

sub.s $f3,$f3,$f3

j calPost

minus:

sub.s $f1,$f2,$f3

s.s $f1,($t4)

sub.s $f2,$f2,$f2 # Reset f2 f3

sub.s $f3,$f3,$f3

j calPost

multiply:

mul.s $f1,$f2,$f3

s.s $f1,($t4)

sub.s $f2,$f2,$f2 # Reset f2 f3

sub.s $f3,$f3,$f3

j calPost

divide:

div.s $f1,$f2,$f3

s.s $f1,($t4)

sub.s $f2,$f2,$f2 # Reset f2 f3

sub.s $f3,$f3,$f3

j calPost

printResult:

li $v0, 4

la $a0, resultMessage

syscall

li $v0, 2

l.s $f12,($t4)

syscall

li $v0, 11

li $a0, '\n'

syscall

ask: # Ask user to continue or not

li $v0, 50

la $a0, askMessage

syscall

beq $a0,0,start

beq $a0,2,ask

# End program

end:

li $v0, 55

la $a0, endMessage

li $a1, 1

syscall

li $v0, 10

syscall

# Sub program

EOF:

beq $s7,2,wrongInput # End with an operator or open bracket

beq $s7,3,wrongInput

beq $t5,-1,wrongInput # Input nothing

j popAll

digit1:

beq $t4,'0',store1Digit

beq $t4,'1',store1Digit

beq $t4,'2',store1Digit

beq $t4,'3',store1Digit

beq $t4,'4',store1Digit

beq $t4,'5',store1Digit

beq $t4,'6',store1Digit

beq $t4,'7',store1Digit

beq $t4,'8',store1Digit

beq $t4,'9',store1Digit

j continueScan

digit2:

beq $t4,'0',store2Digit

beq $t4,'1',store2Digit

beq $t4,'2',store2Digit

beq $t4,'3',store2Digit

beq $t4,'4',store2Digit

beq $t4,'5',store2Digit

beq $t4,'6',store2Digit

beq $t4,'7',store2Digit

beq $t4,'8',store2Digit

beq $t4,'9',store2Digit

# If do not receive second digit

jal numberToPost

j continueScan

digit3:

# If scan third digit --> error

beq $t4,'0',wrongInput

beq $t4,'1',wrongInput

beq $t4,'2',wrongInput

beq $t4,'3',wrongInput

beq $t4,'4',wrongInput

beq $t4,'5',wrongInput

beq $t4,'6',wrongInput

beq $t4,'7',wrongInput

beq $t4,'8',wrongInput

beq $t4,'9',wrongInput

# If do not receive third digit

jal numberToPost

j continueScan

plusMinusOperator: # Input is + -

beq $s7,2,wrongInput # Receive operator after operator or open bracket

beq $s7,3,wrongInput

beq $s7,0,wrongInput # Receive operator before any number

li $s7,2 # Change input status to 2

continuePlusMinus:

beq $t6,-1,inputToOp # There is nothing in Operator stack --> push into

add $t8,$t6,$t3 # Load address of top Operator

lb $t7,($t8) # Load byte value of top Operator

beq $t7,'(',inputToOp # If top is ( --> push into

beq $t7,'+',equalPrecedence # If top is + -

beq $t7,'-',equalPrecedence

beq $t7,'\*',lowerPrecedence # If top is \* /

beq $t7,'/',lowerPrecedence

multiplyDivideOperator: # Input is \* /

beq $s7,2,wrongInput # Receive operator after operator or open bracket

beq $s7,3,wrongInput

beq $s7,0,wrongInput # Receive operator before any number

li $s7,2 # Change input status to 2

beq $t6,-1,inputToOp # There is nothing in Operator stack --> push into

add $t8,$t6,$t3 # Load address of top Operator

lb $t7,($t8) # Load byte value of top Operator

beq $t7,'(',inputToOp # If top is ( --> push into

beq $t7,'+',inputToOp # If top is + - --> push into

beq $t7,'-',inputToOp

beq $t7,'\*',equalPrecedence # If top is \* /

beq $t7,'/',equalPrecedence

openBracket: # Input is (

beq $s7,1,wrongInput # Receive open bracket after a number or close bracket

beq $s7,4,wrongInput

li $s7,3 # Change input status to 3

j inputToOp

closeBracket: # Input is )

beq $s7,2,wrongInput # Receive close bracket after an operator or operator

beq $s7,3,wrongInput

li $s7,4

add $t8,$t6,$t3 # Load address of top Operator

lb $t7,($t8) # Load byte value of top Operator

beq $t7,'(',wrongInput # Input contain () without anything between --> error

continueCloseBracket:

beq $t6,-1,wrongInput # Can't find an open bracket --> error

add $t8,$t6,$t3 # Load address of top Operator

lb $t7,($t8) # Load byte value of top Operator

beq $t7,'(',matchBracket # Find matched bracket

jal opToPostfix # Pop the top of Operator to Postfix

j continueCloseBracket # Then loop again till find a matched bracket or error

equalPrecedence: # Mean receive + - and top is + - || receive \* / and top is \* /

jal opToPostfix # Pop the top of Operator to Postfix

j inputToOp # Push the new operator in

lowerPrecedence: # Mean receive + - and top is \* /

jal opToPostfix # Pop the top of Operator to Postfix

j continuePlusMinus # Loop again

inputToOp: # Push input to Operator

add $t6,$t6,1 # Increment top of Operator offset

add $t8,$t6,$t3 # Load address of top Operator

sb $t4,($t8) # Store input in Operator

j scanInfix

opToPostfix: # Pop top of Operator in push into Postfix

addi $t5,$t5,1 # Increment top of Postfix offset

add $t8,$t5,$t2 # Load address of top Postfix

addi $t7,$t7,100 # Encode operator + 100

sb $t7,($t8) # Store operator into Postfix

addi $t6,$t6,-1 # Decrement top of Operator offset

jr $ra

matchBracket: # Discard a pair of matched brackets

addi $t6,$t6,-1 # Decrement top of Operator offset

j scanInfix

popAll: # Pop all Operator to Postfix

jal numberToPost

beq $t6,-1,finishScan # Operator empty --> finish

add $t8,$t6,$t3 # Load address of top Operator

lb $t7,($t8) # Load byte value of top Operator

beq $t7,'(',wrongInput # Unmatched bracket --> error

beq $t7,')',wrongInput

jal opToPostfix

j popAll # Loop till Operator empty

store1Digit:

beq $s7,4,wrongInput # Receive number after )

addi $s4,$t4,-48 # Store first digit as number

add $t9,$zero,1 # Change status to 1 digit

li $s7,1

j scanInfix

store2Digit:

beq $s7,4,wrongInput # Receive number after )

addi $s5,$t4,-48 # Store second digit as number

mul $s4,$s4,10

add $s4,$s4,$s5 # Stored number = first digit \* 10 + second digit

add $t9,$zero,2 # Change status to 2 digit

li $s7,1

j scanInfix

numberToPost:

beq $t9,0,endnumberToPost

addi $t5,$t5,1

add $t8,$t5,$t2

sb $s4,($t8) # Store number in Postfix

add $t9,$zero,$zero # Change status to 0 digit

endnumberToPost:

jr $ra