BÁO CÁO THỰC HÀNH CUỐI KỲ 2022.2

MÔN KIẾN TRÚC MÁY TÍNH

ĐINH HUY DƯƠNG, NGUYỄN VĂN ĐĂNG

BÀI BÁO CÁO THỰC HÀNH PROJECT CUỐI KỲ KIẾN TRÚC MÁY TÍNH

Đinh Huy Dương 20215020

Nguyễn Văn Đăng 20215033

Bài 5: Chuyển biểu thức dạng trung tố sang hậu tố

Sinh viên thực hiện: Đinh Huy Dương 20215020

Chương trình nhập một xâu chứa biểu thức trung tố và sử dụng bộ nhớ Ngăn xếp (Stack) để lưu trữ các toán tử vào trong xâu đích (Xâu hậu tố - Postfix)

```
.data
    infix: .space 256
    postfix: .space 256
    startMsg: .asciiz "Enter infix expression\nNote: only allowed to use + - * / % ()\nNumber from 00-99"
    errorMsg: .asciiz "Error Input"
    endMsg: .asciiz "Do you want to continue?"
    prompt_postfix: .asciiz "Postfix expression: "
    prompt_infix: .asciiz "Infix expression: "
    prompt_result: .asciiz "Result: "
.text
#-----------------------------------------------------------------
main:
get_str: li $v0, 54
```

```
        la $a0, startMsg
        la $a1, infix
        la $a2, 256
        syscall
        beq $a1,-2,end
        beq $a1,-3,get_str
        li $v0,4
        la $a0, prompt_infix
        syscall
        li $v0, 4
        la $a0, infix
        syscall
        li $v0, 11
        li $a0, '\n'
        syscall
# Status
        li $s0,0                # 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           # Stack top offset

        la $t1, infix           # Infix current byte address +1 each loop
        la $t2, postfix
        addi $t1,$t1,-1         # Set initial address of infix to -1
scn_ifx: addi $t1,$t1,1                 # Increase infix position
        lb $t4, ($t1)                   # Load current infix input
```

```
        beq $t4, ' ', scn_ifx           # 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
scn_op: beq $t4, '+', plsMns
        beq $t4, '-', plsMns
        beq $t4, '*', mulDiv
        beq $t4, '/', mulDiv
        beq $t4, '%', mulDiv
        beq $t4, '(', openBracket
        beq $t4, ')', closeBracket
wrgipt:  li $v0, 55
        la $a0, errorMsg
        li $a1, 2
        syscall
        j again
doneScn:        li $v0, 4
        la $a0, prompt_postfix
        syscall
        li $t6,-1       # Load current of Postfix offset to -1
prn_pst: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,fin_pnt     # 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 prn_pst
printOp: li $v0, 11
        addi $t7,$t7,-100           # Decode operator
        add $a0,$t7,$zero
        syscall
        li $v0, 11
        li $a0, ' '
        syscall
        j prn_pst                   # Loop
fin_pnt: li $v0, 11
        li $a0, '\n'
        syscall
#-----------------------------------------------
# Calculate
        li $t6,0            # Current Postfix offset
loop:   add $t3,$t6,$t2             # Current byte of the Postfix string
        lbu $t4,0($t3)
        bgt $t4,100,cal             # If object is an operator, pop 2 latest num out and cal
        addi $sp,$sp,-4             # Else if object is a num, push into Stack
        sw $t4,0($sp)
        nop
        addi $t6,$t6,1
        bgt $t6,$t5,end_cal             # Loop until finish
        j loop
cal:    lw $t0,0($sp)
        lw $t1,4($sp)               # Pop 2 top num out the Stack into $t0, $t1
        addi $sp,$sp,8
        addi $t6,$t6,1
        beq $t4,143,plus
```

```
        beq $t4,145,minus
        beq $t4,142,multi
        beq $t4,147,divi
        beq $t4,137,mod
plus:   add $t7,$t1,$t0          # the result in the $t7
        addi $sp,$sp,-4          # push the result into the Stack
        sw $t7,0($sp)
        j loop
minus:  sub $t7,$t1,$t0
        addi $sp,$sp,-4
        sw $t7,0($sp)
        j loop
multi:  mul $t7,$t1,$t0
        addi $sp,$sp,-4
        sw $t7,0($sp)
        j loop
divi:   div $t7,$t1,$t0
        addi $sp,$sp,-4
        sw $t7,0($sp)
        j loop
mod:    div $t1,$t0
        mfhi $t7
        addi $sp,$sp,-4
        sw $t7,0($sp)
        j loop
end_cal: li $v0, 4
        la $a0, prompt_result
        syscall
        li $v0,1
        lw $a0,4($sp)
        syscall
```

```
        li $v0, 11
        li $a0, '\n'
        syscall
#---------------------------------------------
again:  li $v0, 50
        la $a0, endMsg
        syscall
        beq $a0,0,main
end:    li $v0, 10
        syscall
#-----------------------------------------------------------------
#Process the digit
EOF:    beq $s0,2,wrgipt                # End with an operator or open bracket
        beq $s0,3,wrgipt
        beq $t5,-1,wrgipt                       # Input nothing
        j popAll
digit1: beq $t4,'0',st1dg
        beq $t4,'1',st1dg
        beq $t4,'2',st1dg
        beq $t4,'3',st1dg
        beq $t4,'4',st1dg
        beq $t4,'5',st1dg
        beq $t4,'6',st1dg
        beq $t4,'7',st1dg
        beq $t4,'8',st1dg
        beq $t4,'9',st1dg
        j scn_op
digit2: beq $t4,'0',st2dg
        beq $t4,'1',st2dg
        beq $t4,'2',st2dg
        beq $t4,'3',st2dg
```

```
        beq $t4,'4',st2dg
        beq $t4,'5',st2dg
        beq $t4,'6',st2dg
        beq $t4,'7',st2dg
        beq $t4,'8',st2dg
        beq $t4,'9',st2dg
        # If do not receive second digit
        jal str_num
        j scn_op
        # If scan third digit --> error
digit3: beq $t4,'0',wrgipt
        beq $t4,'1',wrgipt
        beq $t4,'2',wrgipt
        beq $t4,'3',wrgipt
        beq $t4,'4',wrgipt
        beq $t4,'5',wrgipt
        beq $t4,'6',wrgipt
        beq $t4,'7',wrgipt
        beq $t4,'8',wrgipt
        beq $t4,'9',wrgipt
        # If do not receive third digit
        jal str_num
        j scn_op
#-----------------------------------------------------------------
# Process the operators in the Stack
plsMns:  beq $s0,2,wrgipt               # Receive operator after operator or open bracket
        beq $s0,3,wrgipt
        beq $s0,0,wrgipt        # Receive operator before any number
        li $s0,2                # Change input status to 1
cont_p: beq $t6,-1,push         # There is nothing in Stack --> push into
        lb $t7,($sp)                    # Load byte value of top Operator
```

```
        beq $t7,'(',push        # If top is ( --> push into
        beq $t7,'+',hi_prcd     # If top is + -
        beq $t7,'-',hi_prcd
        beq $t7,'*',lo_prcd     # If top is * /
        beq $t7,'/',lo_prcd
        beq $t7,'%',lo_prcd
mulDiv: beq $s0,2,wrgipt        # Receive operator after operator or open bracket
        beq $s0,3,wrgipt
        beq $s0,0,wrgipt        # Receive operator before any number
        li $s0,2                # Change input status to 1
        beq $t6,-1,push         # There is nothing in Stack --> push into
        lb $t7,($sp)                    # Load byte value of top Operator
        beq $t7,'(',push        # If top is ( --> push into
        beq $t7,'+',push        # If top is + - --> push into
        beq $t7,'-',push
        beq $t7,'*',hi_prcd     # If top is * /
        beq $t7,'/',hi_prcd
        beq $t7,'%',lo_prcd
openBracket:
        beq $s0,1,wrgipt        # Receive open bracket after a number or close bracket
        beq $s0,4,wrgipt
        li $s0,3                # Change input status to 1
        j push
closeBracket:
        beq $s0,2,wrgipt        # Receive close bracket after an operator or operator
        beq $s0,3,wrgipt
        li $s0,4
        lb $t7,($sp)                    # Load byte value of top Stack
        beq $t7,'(',wrgipt      # Input contain () without anything between --> error
continueCloseBracket:
        beq $t6,-1,wrgipt               # Can't find an open bracket --> error
```

```
        jal pop                 # Pop the top of Stack to Postfix
        lb $t7,($sp)            # Load byte value of top of Stack
        beq $t7,'(',match       # Find matched bracket
        j continueCloseBracket  # Then loop again till find a matched bracket or error
#-----------------------------------------------------------------
# Stack Processing
hi_prcd:jal pop             # Pop the top of Operator to Postfix
        j push                  # Push the new operator in
lo_prcd:jal pop             # Pop the top of Operator to Postfix
        j cont_p                # Loop again
push:   add $t6,$t6,1           # Increment top of the Stack offset
        addi $sp,$sp,-1         # Prepare the Stack
        sb $t4,($sp)            # Store the Operator or Brackets into Stack
        j scn_ifx
pop:    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 $sp,$sp,1          # Pop the Stack
        addi $t6,$t6,-1         # Decrement top of Stack offset
        jr $ra
match:  addi $t6,$t6,-1         # Decrement top of Stack offset
        addi $sp,$sp,1          # Pop the Bracket out the Stack
        j scn_ifx
popAll: jal str_num
        beq $t6,-1,doneScn      # Stack is empty --> finish
        lb $t7,($sp)            # Load byte value of top of Stack
        beq $t7,'(',wrgipt      # Unmatched bracket --> error
        beq $t7,')',wrgipt
        jal pop
        j popAll                # Loop until Stack empty
```

```
#-------------------------------------------------------------------------
# Second layer process
st1dg:  beq $s0,4,wrgipt        # Receive number after )
        addi $s4,$t4,-48        # Store first digit as number
        add $t9,$zero,1                 # Change status to 1 digit
        li $s0,1
        j scn_ifx

st2dg:  beq $s0,4,wrgipt        # 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 $s0,1
        j scn_ifx
str_num:        beq $t9,0,endstr
        addi $t5,$t5,1
        add $t8,$t5,$t2
        sb $s4,($t8)                    # Store number in Postfix
        add $t9,$zero,$zero             # Change status to 0 digit
endstr: jr $ra
```

PHẦN 1: Thuật toán chuyển từ biểu thức trung tố sang hậu tố được thể hiện ở đoạn mã giả sau:

```
If c = '('
        PUSH into S;
If c = Number
        postfix += c;
If c = ')'
        while  ( S not NULL and top(S) != '(' )
                postfix += top(S)
```

```
                    pop(S)
        pop(S)                  // Pop until '('
Else                                    // Add the operators
        If preced(c) > preced (top(S))
                push(c)  INTO S
        Else
                while(top(S) not NULL and preced(c) <= preced(top(S))
                        postfix += top(S)           //store and pop until found higher prece
                        pop(S)
                push(c)
while (S not NULL)              // Remaining operators
        postfix += top(S)
        pop(S)
```

Giải thích: Thuật toán sẽ duyệt qua biểu thức trung tố. Nếu gặp ngoặc mở "(" thì sẽ đẩy ngoặc vào trong Stack. Nếu gặp một số thì sẽ đưa thẳng ra biểu thức hậu tố (postfix).

Tiếp tục nếu gặp một toán tử có trình tự tính toán (precedence) lớn hơn đỉnh của Stack hoặc đỉnh của Stack chỉ có ngoặc mà không có toán tử nào khác, thì đẩy nó vào Stack. Trình tự tính toán như sau +- sẽ nhỏ hơn * / và %. Ngược lại, nếu toán tử đang đưa vào Stack gặp toán tử có cùng hoặc kém trình tự ở đỉnh Stack thì trước tiên sẽ đưa hết (pop) các toán tử ở đỉnh Stack ra postfix cho đến khi toán tử đang cho vào gặp toán tử trình tự lớn hơn hoặc không còn; lúc đấy sẽ đẩy (Push) vào trong Stack.

Trong trường hợp gặp ngoặc đóng ")", ta sẽ Pop và đẩy lần lượt ra Postfix đến khi nào gặp ngoặc mở "(".

Cuối cùng, nếu như Stack còn chưa trống, ta sẽ Pop hết các toán tử còn lại vào trong Postfix. Kết thúc thuật toán.

Kết quả mong đợi được như sau:

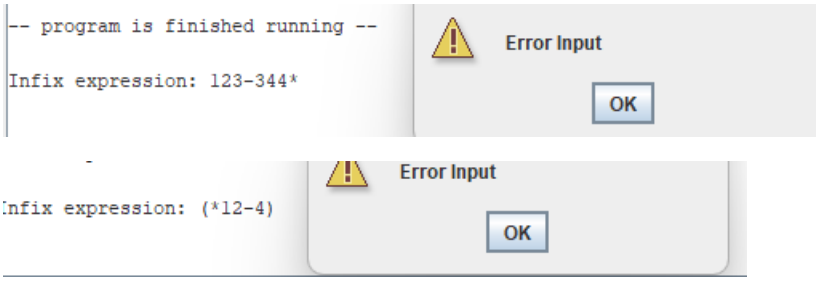
```
Infix expression: (1+22)*(35-4)-(4/2)%(6/3)

Postfix expression: 1 22 + 35 4 - * 4 2 / 6 3 / % -
```

Trong phần này, có 1 vài điểm cần đáng lưu tâm trong đoạn mã. Thứ nhất, đó là cách kiểm tra xem số là số có 2 chữ số hay 1. Bằng việc sử dụng thanh ghi $t9, ta có thể biết trước được đã lưu được bao nhiêu chữ số. Cụ thể khi ở trong nhãn scn_ifx để duyệt qua xâu, nếu gặp chữ số, ta sẽ kiểm tra trạng thái của $t9, giả sử chưa có số nào được nhận, $t9 =0, ta sẽ qua kiểm tra xem đó là số nào ở nhãn digit1, và tiếp tục nhảy đến strdg1 để tăng $t9 lên 1, quay lại scn_ifx duyệt lại xem liệu sau số này có số khác không. Lúc này $t9 = 1, ta nhảy qua digit2, giả sử chỉ có 1 chữ số nên đến cuối nhãn sẽ nhảy đến str_num để lưu số vào trong xâu Postfix

Để thuận lợi xác định các dấu toán tử, khi ta đẩy các dấu toán tử vào trong Stack, ta sẽ cộng các giá trị cần lưu thêm 100 để khi gặp các phần tử có giá trị lớn hơn 100, ta có thể trừ lại một trăm để in ra màn hình

Với các trường hợp sai Input, ta sử dụng thanh ghi $s0 làm biểu hiện trạng thái: Đã nhận số, toán tử, ngoặc mở/đóng để có thể biết được các phần tử phía trước vừa được duyệt qua là gì. Nếu nhập sai sẽ đưa người ta đến nhãn wrgipt để nhập lại từ đầu, ví dụ như nhập dấu ngay sau ngoặc, hay là nhập quá 2 chữ số,…



Ta sẽ duyệt xâu biểu thức trung tố (Infix) đến khi nào gặp ‘\n’ sẽ kết thúc duyệt và Popall để đẩy hết các toán tử còn lại trong Stack ra Postfix

PHẦN 2: Tính toán các giá trị trong biểu thức hậu tố

Thuật toán rất đơn giản, ta duyệt qua xâu Postfix, lần lượt đẩy các số vào trong Stack cho đến khi gặp một toán tử, ta sẽ Pop 2 số ở đỉnh ra, thực hiện phép tính và đẩy kết quả lại vào trong Stack. Khi duyệt hết xâu ta sẽ có kết quả của ta nằm ở gần đỉnh của Stack. (Do trong đoạn kiểm tra ta đặt điều kiện nếu duyệt quá số lượng của xâu sẽ ngắt nên 1 phần tử ngoài xâu vẫn lọt vào đỉnh)

```
Infix expression: (1+2)*(3-4)-(4/2)%(6/3)

Postfix expression: 1 2 + 3 4 - * 4 2 / 6 3 / % -
Result: -3
```

Bài 10: Tạo giả lập máy tính bỏ túi đơn giản sử dụng bàn phím keypad và led 7 thanh trên giả lập hợp ngữ MARS

Sinh viên thực hiện: Nguyễn Văn Đăng 20215033

a. Phương án thực hiện:

Sử dụng ngắt để thực hiện truy nhập dữ liệu từ keypad và tính toán.

Cụ thể: Chương trình sẽ chạy một vòng lặp vô hạn, cho tới khi người dùng ấn vào một phím trên keyboard, từ đó thực hiện ngắt vòng lặp và thực hiện xử lý ngoại lệ, đó chính là chương trình con thực hiện truy nhập và tính toán như một chiếc máy tính bỏ túi

b. Thuật toán

* Data:

- array: Chứa giá trị giải mã LED 7 thanh, với từng giá trị 0x3f, 0x6, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x7, 0x7f và 0x6f mà LED 7 thanh sẽ hiện các số từ 0 đến 9 tương ứng

- addition, subtract,…: Chứa các dòng thông báo hiển thị nhằm tiện cho việc theo dõi quá trình thực hiện chương trình

* Các thanh ghi được sử dụng:

- $s0: Chứa địa chỉ cơ sở của array giải mã LED 7 thanh

- $s1, $s2: Chứa giá trị nạp vào LED 7 thanh, với s1 là thanh ghi bên trái (hàng chục) và s2 là thanh ghi bên phải (hàng đơn vị)

- $s3, $s4: Giá trị quy đổi sang số từ địa chỉ thu nhận được từ keypad

- $t0: Thanh ghi tạm để tính toán địa chỉ truy nhập đến array chứa giá trị giải mã LED 7 thanh trong array, cũng như chứa địa chỉ truy nhập hiển thị LED 7 thanh trái và phải

- $t1: Thanh ghi tạm để chứa địa chỉ IN_ADDRESS_HEXA_KEYBOARD

- $t3: Chứa giá trị có bit thứ 7 là 1, nhằm cho phép ngắt từ keypad

- $t4, $t5: Chứa các giá trị của số được nhập từ keypad

- $t6: Giá trị xác thực, nếu giá trị chứa ở t8 lớn hơn 0 thì là 1, còn ngược lại thì là 0

- $t7: Chứa giá trị xác nhận đã có số được nhập vào từ keypad

- $t8: Chứa giá trị chức năng của các hàm cộng trừ nhân chia và lấy module, tương ứng từ 1 đến 5

* Giải thích thuật toán

Khi khởi tạo, chương trình lập tức thực hiện thiết lập s0, s1 và s2, với $s0 chứa địa chỉ cơ sở của array giải mã LED, $s1 và $s2 chứa giá trị đầu tiên của array, tức là giải mã LED ra số 0.

```
.text
        la $s0, array
        lw $s1, 0($s0)    # Defult starting digits: 0
        lw $s2, 0($s0)
        jal display
        nop
```

Ở hàm chính của chương trình, ta tạo một vòng lặp vô hạn bằng cách sử dụng lệnh jump trỏ pc về lệnh trước đó của nó. Trong vòng lặp có lệnh cho phép nhận exception bằng cách đặt t3 bằng 0x80 (bit thứ 7 bằng 1) và load giá trị đó vào IN_ADDRESS_HEXA_KEYBOARD hiện đã được load tại t1. Vòng lặp này sẽ chạy vô hạn cho tới khi người dùng thực hiện ấn một phím ở keypad.

Vào thời điểm một phím ở keypad được nhấn, chương trình sẽ thực hiện ngắt và chuyển sang chương trình handling exception. Tại đó, *get_code* sẽ thực hiện lấy địa chỉ của phím vừa được nhấn bằng cách xét từng hàng một, từ 0x1, 0x2, 0x4 đến 0x8.

```
get_code:               # Funtion to detect key pressed in all 4 rows
        li $t1, IN_ADDRESS_HEXA_KEYBOARD
        sb $t3, 0($t1)          # Only the 4 LSB in t3 matter
        li $t1, OUT_ADDRESS_HEXA_KEYBOARD
        lb $a0, 0($t1)
        andi $t3, $t3, 0x7f     # Omit the MSB in the half word so that shift left does not go out of
bound
        sll $t3, $t3, 1         # Traverse to the next row
        bgt $t3, 0x90, decode   # If scanning row out of bound, stop scanning
        # If t3 exceed scanning area, go to display (t3 take value more than 0x10 after shifting
        nop
        bnez $a0, decode        # If successfully fetched keynum, decode
        nop
        j get_code      # Haven't get input yet, continue scanning next row
        nop
```

Sau khi thành công nhận được giá trị địa chỉ của phím đã được ấn, chương trình thực hiện giải mã bằng cách so sánh nó với các giá trị địa chỉ định sẵn theo dạng switch case, chẳng hạn, với giá trị tại $a0 là 0x11, chương trình sẽ nhảy đến case 0, tức nhận dạng phím đã ấn là phím "0"

```
li $t3, 0
beq $a0, 0x11, case0
nop
```

Trong trường hợp đó, chương trình sẽ thực hiện chuyển giá trị giải mã hiện thể LED của chữ số đơn vị (tại $s2) sang chữ số hàng chục (tại $s1) bằng lệnh *addi $s1, $s2, $0*. Sau đó giá trị để đưa vào $s2 mới sẽ được tính theo công thức *s2 (mới) = s0 (cơ sở) + 4* t0*, với t0 là gía trị phím chữ số đã được ấn (trong case này là 0). Sau đó chương trình tiếp tục lưu lại giá trị thập phân cụ thể của nó, tương tự như trên nhưng với các giá trị số học từ 0 đến 9. Ở mỗi case, giá trị t7 sẽ được đặt thành 1, với ý nghĩa "Đã có số được ghi nhận từ phía người dùng"

```
case0:   add $s1, $s2, $0 # Shift display value from right to left
add $t0, $0, 0
sll $t0, $t0, 2
add $t0, $t0, $s0
lb $s2, ($t0)
move $s3, $s4            #Shift numeral value from right to left
addi $s4, $0, 0
addi $t7, $0, 1          # $t7: status of having input a number
jalr $t9
nop
j next_pc
nop
```

Với các case khác chương trình thực hiện tương tự, với giá trị t0 ứng với từng case.

Chương trình sẽ thực hiện điều này mỗi lần người dùng nhập một số mới từ keypad, việc này đảm bảo việc nhập sẽ diễn ra cho đến khi nào người dùng thực hiện thao tác tính toán khác.

Với case là các giá trị thuộc phím "+"; "-"; "*", "/", và "%", chương trình thực hiện tính toán dựa trên phím đã được ấn. \

Cụ thể, với case a0 bằng 0x44 (cộng), chương trình thực hiện lấy giá trị của số đã được người dùng nhập bằng công thức 10*s3+s4 và đưa vào thanh ghi t4, sau đó lưu lại ở stack. Sau đó, chương trình kiểm tra xem trước khi ấn phím "+", người dùng đã ấn một phép tính nào khác ở trước chưa, bằng cách so sánh xem t8 (giá trị chức năng của phép tính, với 1 là cộng, 2 là trừ, 3

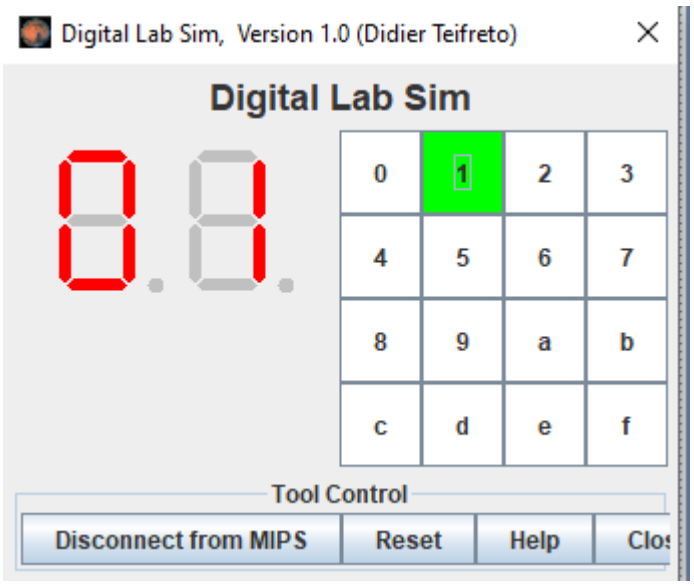
là nhân,…) hiện tại có lớn hơn 0 không, và giá trị t7 có khác 0 không (có số nào được ghi vào trước khi thực hiện tính toán không). Nếu cả hai điều này được thỏa mãn, chương trình sẽ nhảy sang hàm calculate để tính toán dựa trên số đang có sẵn trong ô nhớ, cùng với phép tính trước được quyết định bởi giá trị function tại t8.

- Nếu một trong hai không thỏa mãn, hàm calculate sẽ không thực hiện gì cả và trả về. Từ đó, chương trình tiếp tục thực hiện gán giá trị t8 mới bằng số hiệu phép tính tương ứng, trong case này là 1, sau đó in thông báo ra màn hình, gán giá trị t7 bằng 0, tức là không có số hạng mới nào được ghi nhận từ người dùng và quay trở lại main, chờ người dùng nhập số hạng thứ hai vào.
- Nếu cả hai điều kiện trên thỏa mãn, hàm calculate sẽ được thực hiện để tính toán phép tính cũ trước khi nhận toán tử mới, trong đó giá trị t8 cũ sẽ quyết định toán tử cũ là gì. Tại hàm này, giá trị t4 mới được ghi nhận từ lần gọi case này, đóng vai trò là số hạng thứ hai, cùng với giá trị t4 cũ được chứa trong stack được lấy ra để thực hiện phép tính. Kết quả phép tính đó sẽ tiếp tục được lưu trong stack để thực hiện những phép tính tiếp tục nếu cần. Sau khi thoát ra khỏi hàm calculate, chương trình lại tiếp tục thực hiện như trường hợp trên

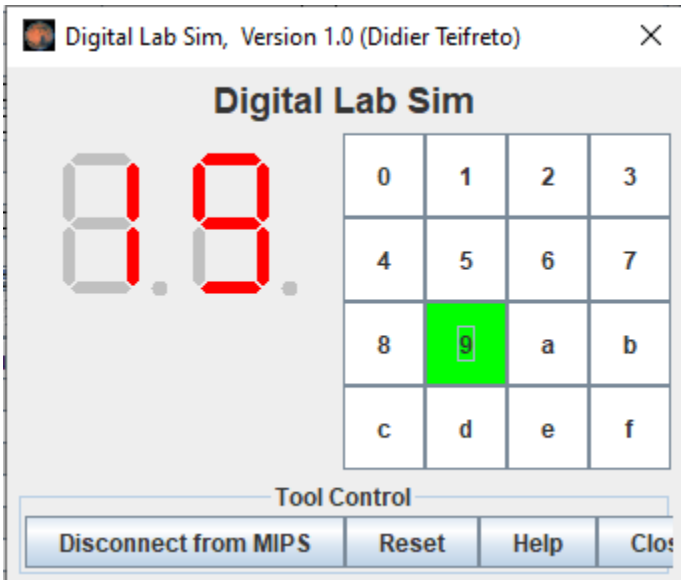
Với các case khác, chương trình thực hiện tương tự.

c. Chức năng, chạy minh họa

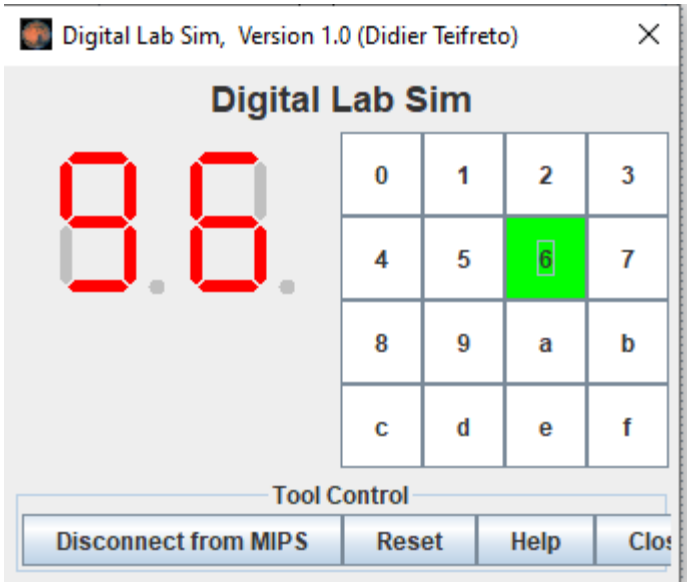
- Chương trình có thể nhận số vào từ bàn phím, với giới hạn là 2 chữ số tối đa. Khi một chữ số mới được đưa vào, nó sẽ đẩy các chữ số cũ sang bên phải. Ví dụ, khi ấn phím 1 từ khi khởi tạo chương trình, ta sẽ được kết quả:



Tại đây, tiếp tục ấn phím “9”, ta sẽ được kết quả:



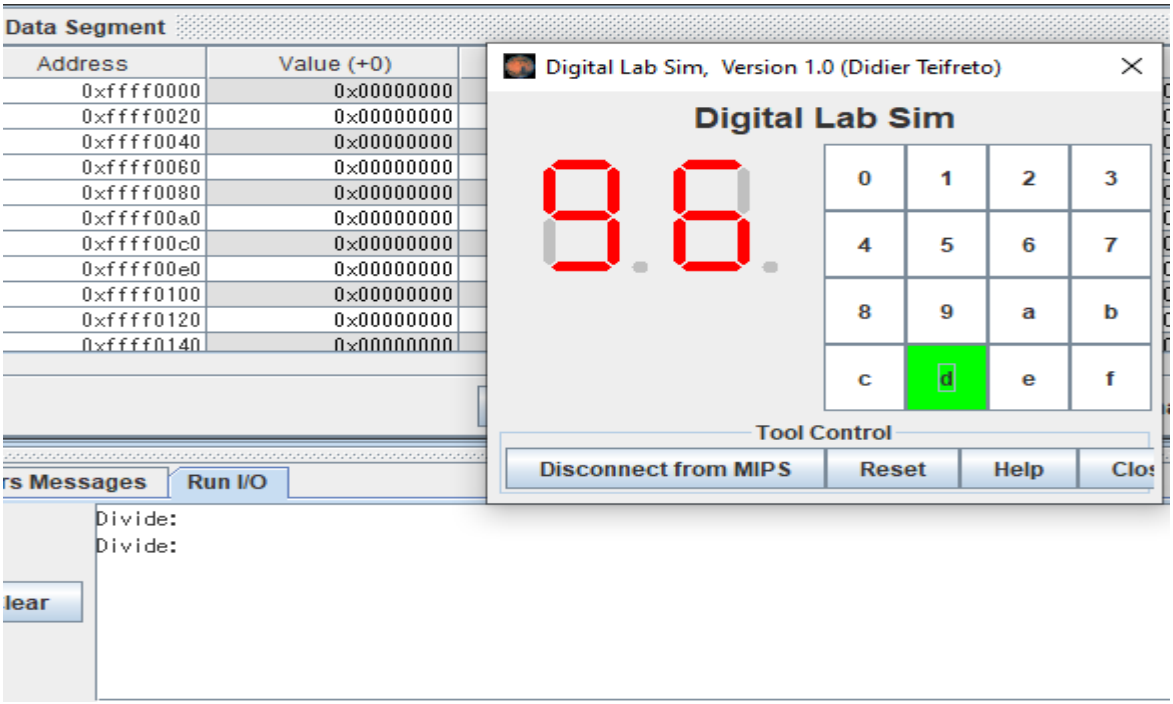
Tiếp tục ấn “6”, ta sẽ được:



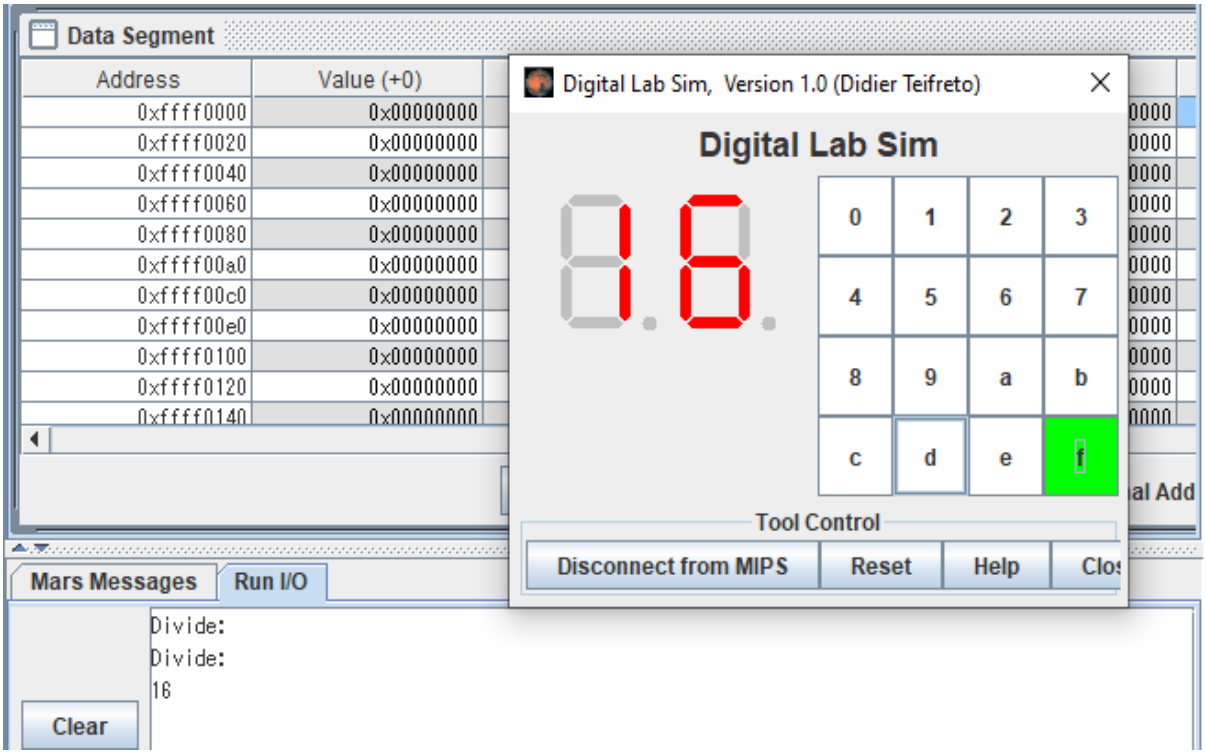
Đó là chức năng nhập số

- Chức năng tính toán:

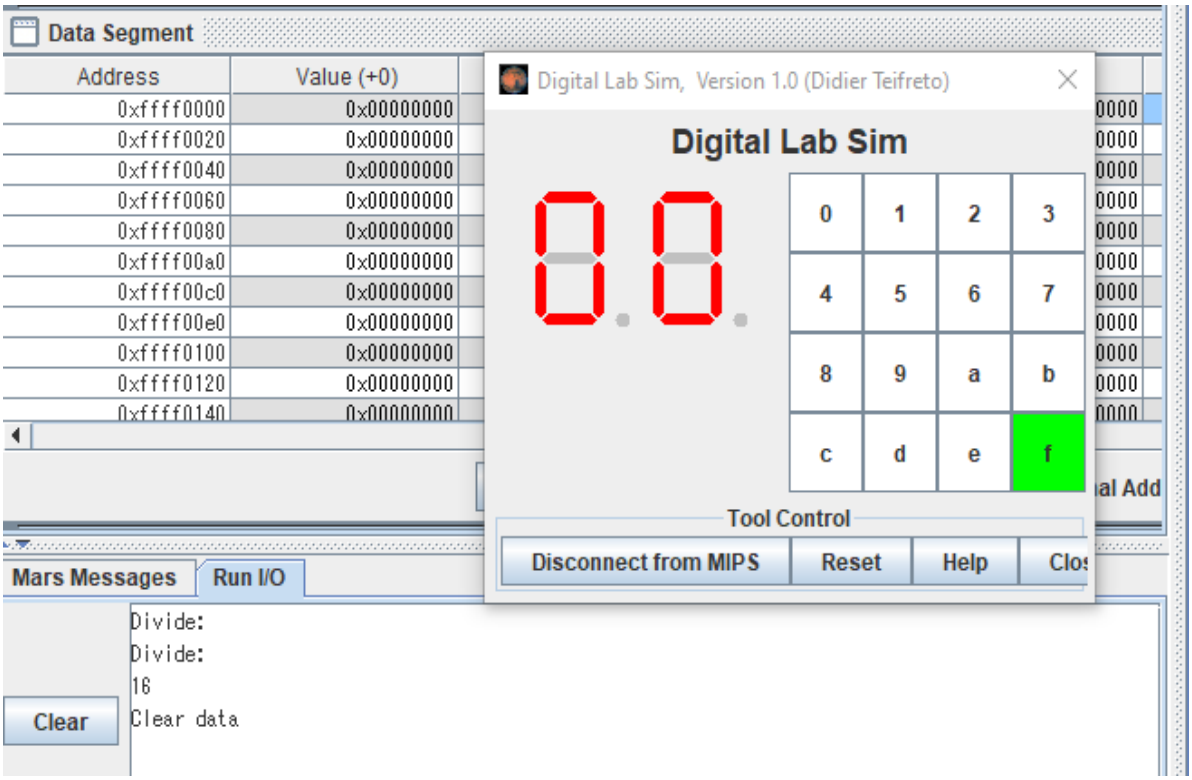
Như đã trình bày ở trên, với các phím từ “a” đến “e” là các toán tử cộng trừ nhân chia và lấy dư. Thực hiện ấn phím các phím này ta sẽ thu được thông báo với phép tính tương ứng (để minh họa), chẳng hạn khi nhấn phím “d”, ta sẽ được dòng thông báo divide:



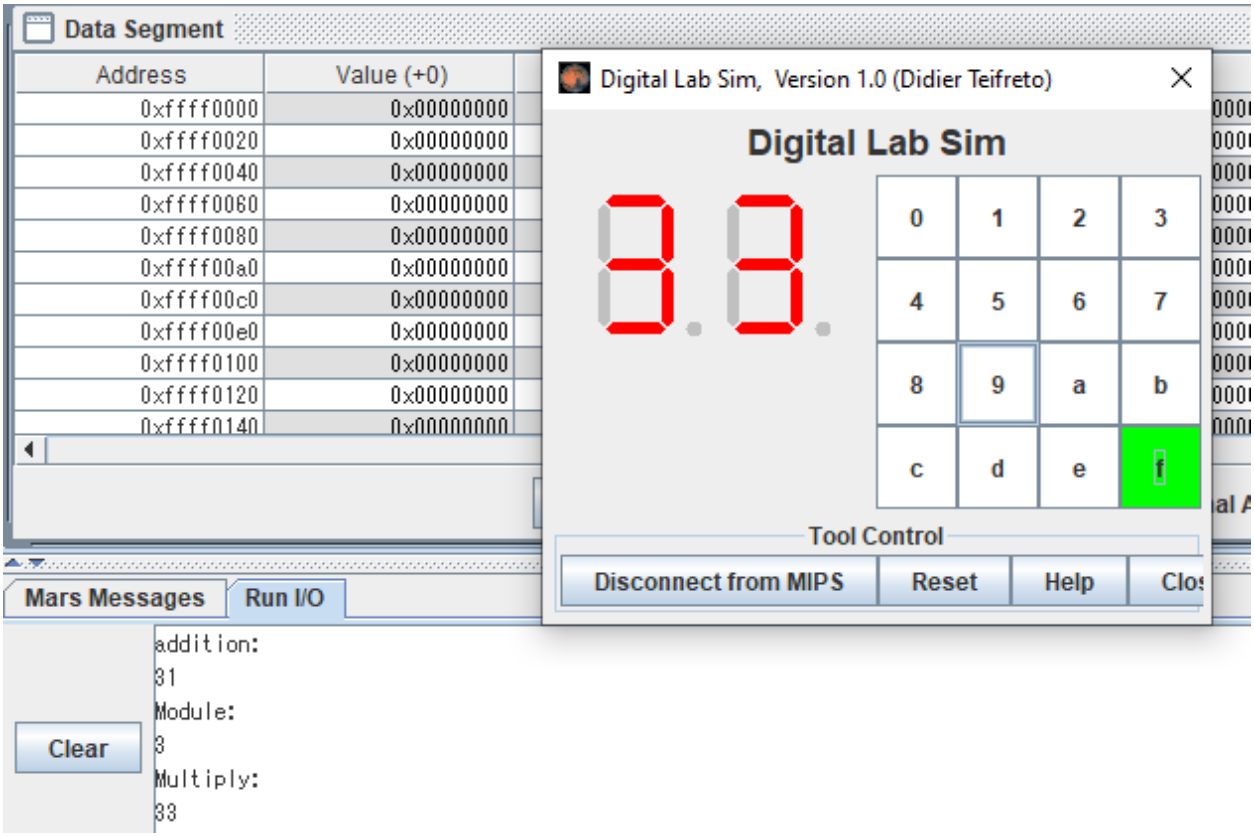
Tại đây tiếp tục nhập giá trị số tiếp theo, chẳng hạn nếu muốn chia cho 6, ta thực hiện ấn phím “0” và “6”, sau đó nhấn “f” để thu kết quả.



Khi ấn “f” một lần nữa, chương trình sẽ thực hiện xóa kết quả và quay lại trạng thái ban đầu

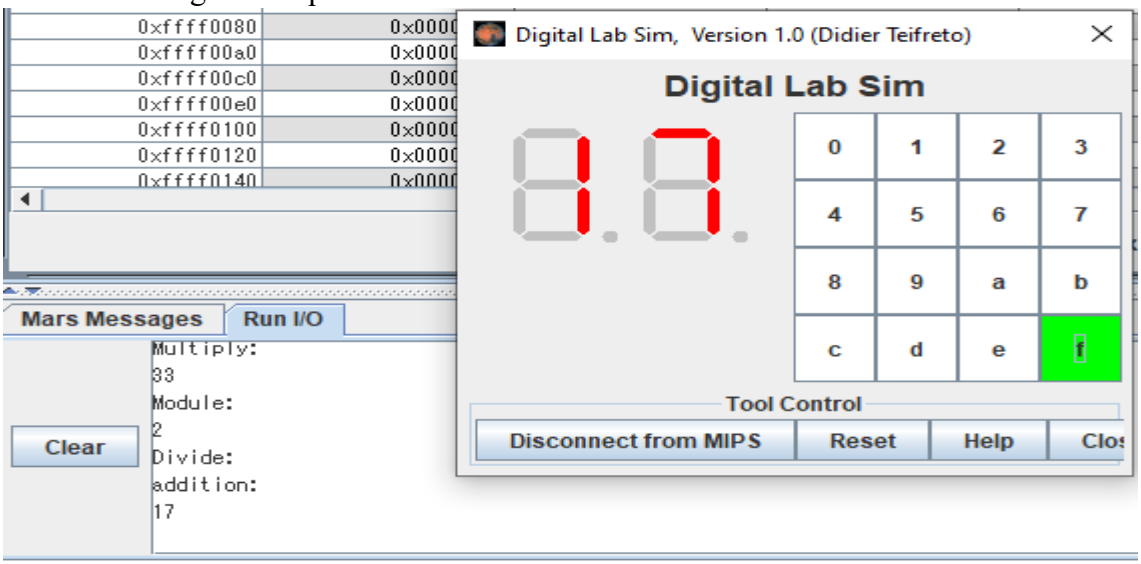


- Thực hiện liên tiếp các phép tính: Chương trình có thể thực hiện liên tiếp bằng cả hai cách, chẳng hạn, sau khi thực hiện phép cộng và có kết quả, ta có thể ấn phím toán tử từ “a” đến “e” để tiếp tục tính toán, với kết quả phép tính trước chính là số hạng thứ nhất của phép tính sau. Chẳng hạn:



Đây là kết quả của việc nhập: “12” “a” “19” “e” “04” “c” “11” “f”, ứng với phép tính ((12+19) mod 4) * 11.

- Chương trình cho phép người dùng thay đổi toán tử giữa chừng nếu chưa có số mới nào được nhập, chẳng hạn trong trường hợp dưới đây, người dùng đã chuyển từ toán tử "/" (phím "d") sang toán tử "+" (phím "a"). Điều này có thể quan sát được do thông báo "Divide" không có kết quả ở dưới.



d. Các hạn chế

- Chương trình không thể hiển thị các kết quả có số lớn hơn 99, chẳng hạn với phép tính cho đáp số 5033, chương trình chỉ hiển thị số "33" trên LED. Tuy nhiên, giá trị hiển thị ở Run I/O vẫn là 5033, và các phép tính nối tiếp đó cũng sẽ sử dụng 5033 để tính toán.

- Chương trình không thể hiển thị số âm. Tương tự như trên, trong trường hợp này LED sẽ không hiển thị gì cả, do khi đó các giá trị s3 và s4 tính được sẽ âm, dẫn đến truyền giá trị s1 và s2 vào t1 bị sai và bị NULL, dẫn đến không thể hiển thị gì cả.

- Thỉnh thoảng chương trình sẽ bị "kẹt nút", chẳng hạn, người dùng nhập một phím "2", nhưng chương trình chạy "2" 2 lần liên tiếp dẫn đến kết quả không mong muốn là 22. Điều này cũng xảy ra với các toán tử thông thường, nhưng không gây ảnh hưởng quá nhiều do hai lần ấn "+" liên tiếp không gây ảnh hưởng đến quá trình tính toán. Tuy nhiên hai lần "=" sẽ được hiểu là 1 lần reset dữ liệu, và kết quả vẫn hiển thị ở Run I/O. Vấn đề này chưa thể xử lý được, phỏng đoán nguyên nhân có thể là do giả lập của MARS hoặc thuật toán chưa tối ưu.

- Khi chạy chương trình trên trình MARS, cần phải đặt tốc độ từ 30 instance/giây trở xuống, nếu để tối đa chương trình dễ bị treo do thực hiện quá nhiều vòng lặp.

- Người dùng cần phải nhập chính xác số mình muốn ghi, nhất là với những số có 1 chữ số, phải nhập "06" thay vì "6" do chương trình sẽ lấy giá trị hàng đơn vị còn tồn đọng lại từ số đã được nhập từ trước mà chưa bị khử đi