TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BỘ MÔN THỰC HÀNH KIẾN TRÚC MÁY TÍNH

---o0o---

BÁO CÁO FINAL PROJECT

THỰC HÀNH KIẾN TRÚC MÁY TÍNH

Giáo viên hướng dẫn : ThS. Lê Bá Vui

Lớp : 147789

Nhóm sinh viên thực hiện :

Đặng Quang Huy 20225853

Lê Quang Khải 20225638

Hà Nội, 06-2024

Mục lục

I. Project 5: Biểu thức trung tố hậu tố - Đặng Quang Huy

YÊU CẦU BÀI TOÁN

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 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 các phép toán cộng, trừ, nhân, chia lấy thương (/), chia lấy dư (%), đóng mở ngoặc.

1 Cách thực hiện

Nhập và hiển thị biểu thức trung tố.

- Chuyển đổi biểu thức trung tố sang hậu tố.
- In biểu thức hậu tố.
- Tính toán giá trị của biểu thức hậu tố.
- Hiển thị kết quả và hỏi người dùng có muốn tiếp tục hay không.

2 Ý tưởng và thuật toán

Lưu biểu thức trung tố:

- Ý tưởng: Đọc kí tự đầu vào nếu nhận được chữ số thì sẽ lưu trạng thái lại và đọc tiếp chữ số tiếp theo nếu đọc được toán tử thì đẩy số đó vào stack. Nếu đọc được tiếp chữ số thì đó sẽ là số có 2 chữ số . Còn nếu đọc được thêm 1 chữ số thì sẽ báo lỗi .

Thuật toán đổi biểu thức trung tố sang hậu tố:

- Để đổi biểu thức trung tố sang hậu tố, ta sẽ dùng ngăn xếp và xâu.
  Bước 1: Đưa 1 biểu thức trung tố vào 1 xâu kí tự và đặt tên là infix.
  Bước 2: Tạo ra 1 xâu mới để lưu biểu thức hậu tố, đặt tên là postfix.
  Bước 3: Thực hiện theo các yêu cầu sau
  Nếu kí tự là số thì lưu vào postfix.
  Nếu kí tự là toán tử, nếu ngăn xếp trống thì đẩy vào ngăn xếp .
  Nếu kí tự là dấu ‘(‘ thì cho vào ngăn xếp.

Nếu gặp kí tự ‘)’ thì sẽ lấy hết kí tự sau dấu ‘(‘ cho vào postfix.
Nếu toán tử đang xét có bậc cao hơn toán tử ở đỉnh ngăn xếp thì đẩy toán tử vào ngăn xếp .
Nếu toán tử đang xét có bậc bằng toán tử ở đỉnh ngăn xếp thì lấy toán tử đỉnh ngăn xếp ra, xếp vào postfix và đẩy toán tử đang xét vào ngăn xếp.
Nếu toán tử đang xét có bậc nhỏ hơn toán tử ở đỉnh ngăn xếp thì lấy toán tử đang xét và xếp vào postfix.
Bước 4: Thực hiện bước 3 cho đến khi kết thúc biểu thức và tất cả các toán tử toán hạng được xếp vào postfix, khi đó ta có biểu thức hậu tố.

Tính giá trị biểu thức bằng biểu thức hậu tố

Bước 1: Quét toàn bộ biểu thức từ trái sang phải.
Bước 2: Tạo 1 ngăn xếp mới.
Bước 3: Nếu phần tử được quét là toán hạng thì đưa vào ngăn xếp.
Bước 4: Nếu phần tử được quét là toán tử thì lấy 2 toán hạng trong ngăn xếp ra, sau đó tính toán giá trị của chúng dựa vào toán tử này, sau đố đẩy lại vào ngăn xếp.
Bước 5: Thực hiện bước 3 và bước 4 cho đến khi kết thúc biểu thức và trong ngăn xếp còn 1 giá trị duy nhất. Đó chính là giá trị của biểu thức.

3 Stack

- **infix**: `.space 256`
  - Lưu trữ biểu thức trung tố mà người dùng nhập vào. Kích thước 256 byte để đảm bảo đủ không gian cho chuỗi đầu vào dài.
- **postfix**: `.space 256`
  - Lưu trữ biểu thức hậu tố được chuyển đổi từ biểu thức trung tố. Kích thước 256 byte.
- **operator**: `.space 256`
  - Lưu trữ các toán tử trong quá trình chuyển đổi từ biểu thức trung tố sang hậu tố. Kích thước 256 byte.
- **stack**: `.space 256`
  - Lưu trữ ngăn xếp trong quá trình tính toán biểu thức hậu tố. Kích thước 256 byte.
- **endMsg**: `.asciiz "Ban co muon tiep tuc?"`
  - Thông báo hỏi người dùng có muốn tiếp tục không.
- **errorMsg**: `.asciiz "Input sai cu phap"`

- Thông báo lỗi khi nhập không đúng cú pháp.

- **startMsg**: `.asciiz "Hay nhap chuoi infix\nNote: chi chua + - * / % ()\ncac so tu 00-99"`
    - Thông báo hướng dẫn người dùng nhập biểu thức trung tố.
- **prompt_postfix**: `.asciiz "Bieu thuc hau to: "`
    - Thông báo để in ra biểu thức hậu tố.
- **prompt_result**: `.asciiz "Ket qua: "`
    - Thông báo để in ra kết quả tính toán của biểu thức hậu tố.
- **prompt_infix**: `.asciiz "Bieu thuc trung to: "`
    - Thông báo để in ra biểu thức trung tố mà người dùng đã nhập

4 Ý tưởng

a. Chuyển đổi Biểu thức Trung tố thành Hậu tố

Ý tưởng chính

Biểu thức trung tố là dạng biểu thức mà các toán tử nằm giữa các toán hạng (ví dụ: A + B). Biểu thức hậu tố là dạng biểu thức mà các toán tử nằm sau các toán hạng (ví dụ: A B +). Thuật toán này sử dụng ngăn xếp để chuyển đổi biểu thức từ trung tố sang hậu tố.

Các bước thực hiện

Quét từng ký tự trong biểu thức trung tố:

Nếu ký tự là một chữ số, đưa nó vào biểu thức hậu tố.

Nếu ký tự là một toán tử (+, -, *, /, %), so sánh độ ưu tiên của nó với toán tử trên đỉnh ngăn xếp:

Nếu độ ưu tiên của toán tử hiện tại cao hơn hoặc bằng, đưa toán tử trên đỉnh ngăn xếp vào biểu thức hậu tố, sau đó đưa toán tử hiện tại vào ngăn xếp.

Nếu độ ưu tiên thấp hơn, chỉ đưa toán tử hiện tại vào ngăn xếp.

Nếu ký tự là dấu ngoặc mở '(', đưa nó vào ngăn xếp.

Nếu ký tự là dấu ngoặc đóng ')', lấy tất cả toán tử trong ngăn xếp cho đến khi gặp dấu ngoặc mở và đưa vào biểu thức hậu tố.

Khi quét xong biểu thức trung tố, lấy tất cả các toán tử còn lại trong ngăn xếp và đưa vào biểu thức hậu tố.

b. Tính Toán Biểu thức Hậu tố

Ý tưởng chính

Biểu thức hậu tố rất dễ dàng để tính toán bằng cách sử dụng ngăn xếp. Khi gặp một toán hạng, đưa nó vào ngăn xếp. Khi gặp một toán tử, lấy hai toán hạng từ đỉnh ngăn xếp, thực hiện phép tính, và đưa kết quả trở lại ngăn xếp.
Các bước thực hiện
Quét từng ký tự trong biểu thức hậu tố:
Nếu ký tự là một chữ số, đưa nó vào ngăn xếp.
Nếu ký tự là một toán tử (+, -, *, /, %), lấy hai toán hạng từ đỉnh ngăn xếp, thực hiện phép tính, và đưa kết quả trở lại ngăn xếp.
Kết quả cuối cùng: Giá trị còn lại duy nhất trong ngăn xếp là kết quả của biểu thức.
Ví dụ minh họa
Giả sử bạn có biểu thức trung tố: 3 + 5 * (2 - 8)
Bước 1: Chuyển đổi sang hậu tố
Quét từ trái sang phải:
3 -> postfix: 3
+ -> stack: +
5 -> postfix: 3 5
* -> stack: + *
(-> stack: + * (
2 -> postfix: 3 5 2
- -> stack: + * (-
8 -> postfix: 3 5 2 8
) -> pop - to postfix: 3 5 2 8 - and pop (
Lấy các toán tử còn lại từ ngăn xếp:
postfix: 3 5 2 8 - * +
Bước 2: Tính toán biểu thức hậu tố
Quét từ trái sang phải:
3 -> stack: 3
5 -> stack: 3 5
2 -> stack: 3 5 2
8 -> stack: 3 5 2 8
- -> stack: 3 5 -6 (2 - 8 = -6)
* -> stack: 3 -30 (5 * -6 = -30)
+ -> stack: -27 (3 + -30 = -27)
Kết quả cuối cùng: -27

5 Source code

```
.data
    infix  .space 256
```

```
    postfix  .space 256
    operator  .space 256
    stack  .space 256
    endMsg  .asciiz "Ban co muon tiep tuc ?"
    errorMsg  .asciiz "Input sai cu phap"
    startMsg: .asciiz "Hay nhap chuoi infix\nNote: chi chua + - * / % ()\ncac so
tu00-99"
   prompt_postfix: .asciiz "Bieu thuc hau to:  "
    prompt_result  .asciiz "Ket qua: "
    prompt_infix  .asciiz "Bieu thuc trung to:  "
.text
start
# nhap vao bieu thuc trung to
    li      54
    la      $a0, startMsg
    la      $a1, infix
    la      $a2, 256
    syscall
    beq     $a1, 2, end          # neu an cancel thi dung lai
    beq     $a1, 3, start        # neu an enter thi bat dau nhap du lieu
# in bieu thuc trung to
    li      4
    la      $a0, prompt_infix
    syscall
    li      4
    la      $a0, infix
    syscall
    li      11
    li      '\n'
    syscall
# khoi tao cac trang thai
    li      0               # bien trang thai $s7

                                # trang thai "1" khi nhan vao so (0 -> 99)
                                # trang thai "2" khi nhan vao toan tu * / + - %
                                # trang thai "3" khi nhan vao dau "("
                                # trang thai "4" khi nhan vao dau ")"
    li      0               # dem so chu so?
    li      1               # luu dinh cua offset postfix
    li      1               # luu dinh cua offset toan tu
    la      $s3, offset      # load cac dia chi cua cac offset
    la      $s4, offsetPost
    la      $s5, offsetOpe
    addi    $s1, $zero, 1         # Set dia chi khoi tao infix la -1
# chuyen sang postfix
scanInfix               # For each moi ki tu trong postfix
# kiem tra dau vao
    addi $t1, $t1, 1                # tang vi tri con tro infix len 1 don vi i = i+
1
```

```
    lb      0                   # lay gia tri cua con tro infix hien tai
     beq     ''                  # neu la space tiep tuc scan
     beq $t4, '\n', EOF              # Scan ket thuc pop tat ca cac toan tu
sangpostfix
     beq     0                   # Neu trang thai la 0 => co 1 chu so
     beq     1                   # Neu trang thai la 1 => co 2 chu so
     beq     2                   # neu trang thai la 2 => co 3 chu so
     continueScan
     beq     '+'                 # kiem tra ki tu hien tai $t4
     beq     '-'
     beq $t4, '*', multiplyDivideModulo
     beq $t4, '/', multiplyDivideModulo
     beq     '%'
     beq     '('
     beq $t4, ')', closeBracket
wrongInput:                 # dau vao loi
     li $v0, 55
     la $a0, errorMsg
     li $a1, 2
     syscall
     j ask
finishScan:
# in bieu thuc infix
     # Print prompt:
     li      4
     la
     syscall
     li      1               # set gia tri infix hien tai la $s6= -1
printPostfix
     addi            1           # tang offset cua postfix hien tai
     add                         # load dia chi cua postfix hien tai
     lbu                         # Load gia tri cua postfix hien tai
     bgt                         # in ra postfix xong roi t nh ket qua
     bgt     99                  # neu postfix hien tai > 99 --> la mot toan tu
     # Neu khong thi la mot toan hang
     li $v0, 1
     add $a0,$t7,$zero
     syscall
     li      11
     li      ''
     syscall
     j                       # Loop
     printOperator
     li      11
     addi            100         # Decode toan tu
     add
     syscall
     li      11
     li      ''
```

```
    syscall
    j [illegible]          # Loop
finishPrint
    li [illegible] 11
    li [illegible] \n
    syscall
# tinh toan ket qua
    li [illegible] 4            # set offset cua dinh stack la -4
    la [illegible]              # Load dia chi dinh stack
    li [illegible] 1            # Dat offset cua Postfix hien tai la -1
CalculatorPost
    addi [illegible] 1          # tang offset hien tai cua Postfix
    add [illegible]             # Load dia chi cua postfix hien tai
    lbu [illegible]             # Load gia tri cua postfix hien tai
    bgt [illegible]             # tinh toan ket qua va in ra
    bgt $t7,99,calculate            # neu gia tri postfix hien tai > 99 --> toan tu--> lay ra 2 toan hang va tinh toan
# neu khong thi la toan hang
    addi [illegible] 4          # tang offset dinh stack len
    add [illegible]             # tang dia chi cua dinh stack
    sw [illegible]              # day so vao stack
    j [illegible]               # Loop
    calculate
        # Pop 1 so
        add [illegible]
      lw [illegible]
        # pop so tiep theo
        addi [illegible] 4
        add [illegible]
        lw [illegible]
        # Decode toan tu
        beq [illegible] 143
        beq [illegible] 145
        beq [illegible] 142
        beq [illegible] 147
        beq [illegible] 137
        plus
            add $t0,$t0,$t1     # tinh tong gia tri cua 2 con tro dang luu gia tritoan hang
          sw [illegible]        # luu gia tri cua con tro ra $t4
#           li $t0, 0           # Reset t0, t1
#           li $t1, 0
            j [illegible]
        minus
            sub [illegible]
            sw [illegible]
#           li $t0, 0           # Reset t0, t1
#           li $t1, 0
            j [illegible]
```

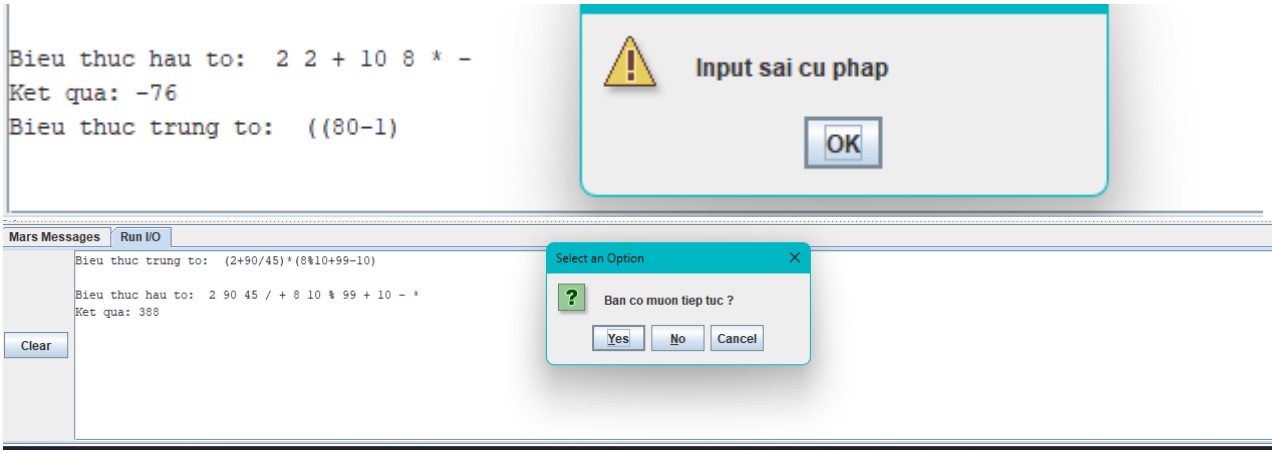
```
        multiply:
            mul $t0, $t1,$t0
            sw $t0,($t4)
#           li $t0, 0           # Reset t0, t1
#           li $t1, 0
            j CalculatorPost
        divide:
            div $t1, $t0
            mflo $t0
            sw $t0,($t4)
#           li $t0, 0           # Reset t0, t1
#           li $t1, 0
            j CalculatorPost
        modulo:
            div $t1, $t0
            mfhi $t0
            sw $t0,($t4)
#           li $t0, 0           # Reset t0, t1
#           li $t1, 0
          j CalculatorPost
printResult:
    li $v0, 4
    la $a0, prompt_result
    syscall
    li $v0, 1
    lw $a0,($t4)                # load gia tri cua $t4 ra con tro $t0
    syscall
    li $v0, 11
    li $a0, '\n'
    syscall
ask:                    # tiep tuc khong??
    li $v0, 50
    la $a0, endMsg
    syscall
    beq $a0,0,start
    beq $a0,2,ask
# End program
end:
    li $v0, 10
    syscall

# Sub program
EOF:
    beq $s7,2,wrongInput            # ket thuc khi gap toan tu hoac dau ngoac mo
    beq $s7,3,wrongInput
    beq $t5,-1,wrongInput           # -1 thi khong co dau vao
    j popAllOperatorInStack
digit1:
    beq $t4,'0',storeDigit1
```

```
    beq $t1,'1' storeDigit1
    beq $t1,'2' storeDigit1
    beq $t1,'3' storeDigit1
    beq $t1,'4' storeDigit1
    beq $t1,'5' storeDigit1
    beq $t1,'6' storeDigit1
    beq $t1,'7' storeDigit1
    beq $t1,'8' storeDigit1
    beq $t1,'9' storeDigit1
    j continueScan

digit2
    beq $t1,'0' storeDigit2
    beq $t1,'1' storeDigit2
    beq $t1,'2' storeDigit2
    beq $t1,'3' storeDigit2
    beq $t1,'4' storeDigit2
    beq $t1,'5' storeDigit2
    beq $t1,'6' storeDigit2
    beq $t1,'7' storeDigit2
    beq $t1,'8' storeDigit2
    beq $t1,'9' storeDigit2
    # neu khong nhap vao chu so thu 2
    jal undefinedError
    j continueScan
digit3
    # neu scan ra chu so thu 3 --> error
    beq $t1,'0' storeDigit3
    beq $t1,'1' storeDigit3
    beq $t1,'2' storeDigit3
    beq $t1,'3' storeDigit3
    beq $t1,'4' storeDigit3
    beq $t1,'5' storeDigit3
    beq $t1,'6' storeDigit3
    beq $t1,'7' storeDigit3
    beq $t1,'8' storeDigit3
    beq $t1,'9' storeDigit3
    # neu khong co chu so thu  3
    jal undefinedError
    j continueScan
plusMinus                   # Input is + -
    beq  2 plusMinus3           # Nhan toan tu sau toan tu hoac "("
    beq  3 plusMinus3
    beq  0 plusMinus3           # nhan toan tu truoc bat ki so nao
    li  2                   # Thay doi trang thai dau  vao thanh 2
continuePlusMinus
    beq $t5  1 undefinedError   # Khong co gi trong  stack -> day vao
    add $t8,$t6,$t3             # Load dia chi cua toan tu o dinh
    lb $t7,($t8)                # Load byte gia tri cua toan tu o dinh
```

```
    beq $t7,'(',inputOperatorToStack     # neu dinh la ( --> day vao
    beq $t7,'+',lowerPrecedence          # neu dinh la + - --> day vao
    beq $t7,'-',lowerPrecedence
    beq $t7,'*',lowerPrecedence          # neu dinh la * / % thi lay * / % ra roiday
vao
    beq $t7,'/',lowerPrecedence
    beq $t7,'%',lowerPrecedence
multiplyDivideModulo                     # dau vao la * / %
    beq $t6,2,inputOperatorToStack       # Nhan toan tu sau toan tu hoac "("
    beq $t6,3,inputOperatorToStack
    beq $t6,0,inputOperatorToStack       # Nhan toan tu truoc bat ki so nao
    li $t6,2                             # Thay doi trang thai dau vao thanh 2
    beq $t6,1,inputOperatorToStack       # Khong co gi trong stack -> day vao
    add $t8,$t6,$t3                      # Load dia chi cua toan tu o dinh
    lb $t7,($t8)                         # Load byte gia tri cua toan tu o dinh
    beq $t7,'(',inputOperatorToStack     # neu dinh la ( --> day vao
    beq $t7,'+',inputOperatorToStack     # neu dinh la + - --> day vao
    beq $t7,'-',inputOperatorToStack
    beq $t7,'*',equalPrecedence          # neu dinh la * / % day vao
    beq $t7,'/',equalPrecedence
    beq $t7,'%',equalPrecedence
openBracket                              # dau vap la (
    beq $t6,1,inputOperatorToStack       # Nhan "(" sau mot so hoac dau ")"
    beq $t6,4,inputOperatorToStack
    li $t6,3                             # Thay doi trang thai dau vao thanh 3
    j inputOperatorToStack
closeBracket                             # dau vao la  ")"
    beq $t6,2,inputOperatorToStack       # Nhan ")" sau mot toan tu hoac toan tu
    beq $t6,3,inputOperatorToStack
    li $t6,4                             # Thay doi trang thai dau vao thanh 4
    add $t8,$t6,$t3                      # Load dia chi toan tu dinh
    lb $t7,($t8)                         # Load gia tri cua toan tu o dinh
    beq $t7,'(',wrongInput               # Input bao gom () khong co gi o giua  --
>error
continueCloseBracket
    beq $t6,1,inputOperatorToStack       # khong tim duoc dau "(" --> error
    add $t8,$t6,$t3                      # Load dia chi cua toan tu o dinh
    lb $t7,($t8)                         # Load gia tri cua toan tu o dinh
    beq $t7,'(',inputOperatorToStack     # Tim ngoac phu hop
    jal PopOperatorToPostfix             # day toan tu o dinh vao postfix
    j continueCloseBracket               # tiep tuc vong lap cho den khi tim duoc
ngoacphu hop
equalPrecedence:                         #  nhan + - vao dinh stack la + - || nhan * / % vao
dinh stack la * / %
    jal PopOperatorToPostfix             # lay toan tu dinh stack ra Postfix
    j inputOperatorToStack               # day toan tu moi vao stack
lowerPrecedence                          # nhan + - vao dinh stack * / %
    jal PopOperatorToPostfix             # lay toan tu dinh stack ra va day
vaopostfix
```

```
    j [illegible]          # tiep tuc vong lap
inputOperatorToStack        # day dau vao cho toan tu
    add [illegible] 1       # tang offset cua toan tu o dinh len 1
    add [illegible]         # load dia chi cua  toan tu o dinh
    sb [illegible]          # luu toan tu nhap vao stack
    j scanInfix
PopOperatorToPostfix:           # lay toan tu o dinh va luu vao postfix
    addi [illegible] 1      # tang offet cua toan tu o dinh stack len 1
    add [illegible]         # load dia chi cua toan tu o dinh stack
    addi [illegible] 100    # ma hoa toan tu + 100 de tranh trung voi cac so
    sb $t7,($t8)                # luu toan tu vao postfix
    addi [illegible] 1      # giam offset cua toan tu o dinh stack di 1
    jr [illegible]
matchBracket            # xoa cap dau ngoac
    addi [illegible] 1      # giam offset cua toan tu o dinh stack di 1
    j [illegible]
popAllOperatorInStack       # lay het toan tu vao postfix
    jal [illegible]
    beq [illegible] 1 [illegible]       # stack rong --> ket thuc
    add [illegible]         # lay dia chi cua toan tu o dinh stack
    lb [illegible]          # lay gia tri cua toan tu o dinh stack
    beq [illegible] '(' [illegible]     # ngoac khong phu hop --> error
    beq [illegible] ')' [illegible]
    jal [illegible]
    j [illegible]               # lap cho den khi stack rong
storeDigit1
    beq [illegible] 4 [illegible]       # nhan vao so sau  ")"
    addi [illegible] 48     # luu chu so dau tien duoi dang so ma ascii cua chu so 0 la 48
    add [illegible] 1       # Thay doi trang thai thanh 1
    li $s7,1
    j scanInfix
storeDigit2:
    beq [illegible] 4 [illegible]       # nhan vao so sau  ")"
    addi [illegible] 48     # luu chu so thu hai duoi dang so
    mul [illegible] 10
    add [illegible]         # luu number = first digit * 10 + second digit
    add [illegible] 2       # thay doi trang thai thanh 2
    li [illegible] 1
    j [illegible]
numberToPostfix
    beq [illegible] 0 [illegible]
    addi [illegible] 1
    add [illegible]
    sb [illegible]          # luu so vao postfix
    li [illegible] 0        # thay doi trang thai ve 0
endnumberToPostfix
    jr [illegible]
```

6 Kết quả



II. Project 4: Postscripts CNC MarsBot – Lê Quang Khải

YÊU CẦU BÀI TOÁN

Thực hành Kiến trúc máy tính *Project cuối kỳ*

4. Postscript CNC Marsbot

Máy gia công cơ khí chính xác CNC Marsbot được dùng để cắt tấm kim loại theo các đường nét được qui định trước. CNC Marsbot có một lưỡi cắt dịch chuyển trên tấm kim loại, với giả định rằng:

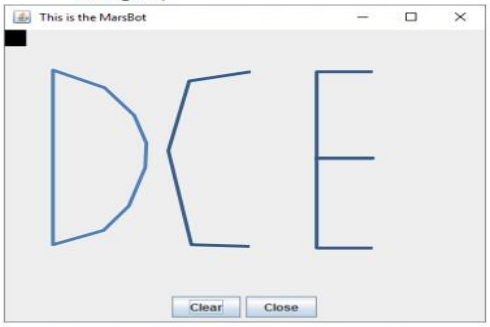
- Nếu lưỡi cắt dịch chuyển nhưng không cắt tấm kim loại, tức là Marsbot di chuyển nhưng không để lại vết (Track)
- Nếu lưỡi cắt dịch chuyển và cắt tấm kim loại, tức là Marsbot di chuyển và có để lại vết.

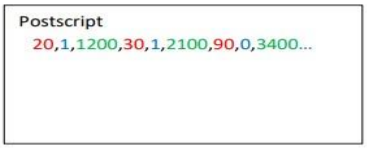
Để điều khiển Marsbot cắt đúng như hình dạng mong muốn, người ta nạp vào Marsbot một mảng cấu trúc gồm 3 phần tử:

- <Góc chuyển động>,<Cắt/Không cắt>,<Thời gian>
- Trong đó <Góc chuyển động> là góc của hàm HEADING của Marsbot
- <Cắt/Không cắt> thiết lập lưu vết/không lưu vết
- <Thời gian> là thời gian duy trì quá trình vận hành hiện tại

Hãy lập trình để CNC Marsbot có thể:

- Thực hiện cắt kim loại như đã mô tả
- Nội dung postscript được lưu trữ cố định bên trong mã nguồn
- Mã nguồn chứa 3 postscript và người dùng sử dụng 3 phím 0, 4, 8 trên bàn phím Key Matrix để chọn postscript nào sẽ được gia công.
- Một postscript chứa chữ DCE cần gia công. Hai script còn lại sinh viên tự đề xuất (tối thiểu 10 đường cắt)





0	1	2	3
4	5	6	7
8	9	a	b
c	d	e	f

1 Giới thiệu

Chương trình được viết bằng ngôn ngữ MIPS Assembly nhằm điều khiển một robot di chuyển theo các hướng dẫn được lưu trữ trong các đoạn mã (postscript). Mỗi postscript chứa thông tin về góc quay, trạng thái vẽ và thời gian di chuyển của robot. Chương trình cho phép người dùng chọn một trong ba postscript thông qua bàn phím và robot sẽ thực hiện các hướng dẫn tương ứng.

2 Cấu trúc dữ liệu

- **Postscript**: Mỗi postscript là một mảng các giá trị bao gồm ba phần tử: góc quay (degree), trạng thái vẽ (leave track), và thời gian di chuyển (duration).
- **postscript1, postscript2, postscript3**: Các mảng chứa các giá trị hướng dẫn cho robot.
- **postscript1_length, postscript2_length, postscript3_length**: Độ dài của mỗi mảng postscript.

3 Biến và hằng số

IN_ADDRESS_HEXA_KEYBOARD: Địa chỉ nhận đầu vào từ bàn phím

OUT_ADDRESS_HEXA_KEYBOARD: Địa chỉ của đầu ra bàn phím

HEADING: Địa chỉ để nhận góc quay của robot.

LEAVETRACK: Địa chỉ để điều khiển trạng thái vẽ của robot (0 hoặc 1)

WHEREX, WHEREY: Địa chỉ để lấy vị trí hiện tại của robot.

MOVING: Địa chỉ để điều khiển trạng thái di chuyển của robot.

4 Ý tưởng

Đầu tiên phải tạo mảng để chứa postscript, các postscript đã chọn ngoài DCE là

- KHAI (là 27 nét vẽ với K là 3 nét vẽ, H là 3 nét vẽ, A là 20 nét vẽ, I là 1 nét)
- HUY (là 26 nét vẽ với H là 3 nét vẽ U là 20 nét vẽ và Y là 3 nét vẽ)
- Xử lí Key Matrix – sử dụng Digital Lab Sim để có bộ 16 key tương ứng hình ảnh ở yêu cầu đề bài
- Do cơ chế cơ bản của MarsBot nên ta sẽ tạo 1 vòng lặp để xử lí từng đường cắt một được lấy từ postscript (xử lí 3 phần tử của mảng mỗi lần)
- Và khi vẽ xong cả 3 postscript đều được hoàn thành -> kết thúc chương trình.

5 Source code

```
.data
```

```
postscript1   .word     180 1 6000  90 1 1500  60 1 1000  0 1 5000  300 1 1000  270 1 1500  90 0 6500
270 1 1500  240 1 1000  180 1 5000  120 1 1000  90 1 1500  90 0 2500  270 1 1500  0 1 3000  90 1 1500
270 0 1500  0 1 3000  90 1 1500
postscript1_length .word  57
postscript2   .word     180 1 8600  0  0  4300  135 1 6080  315 0 6080  45 1 6080  225 0 6080  0 0 4300
90 0 6000  180 1 8600  90 0 4060  0 1 8600  180 0 4300  270 1 4060  180 0 4300  90 0 6000  0 1 6100  10 1 470
20 1 470  30 1 470  40 1 470  50 1 470  60 1 470  70 1 470  80 1 470  90 1 20  100 1 470  110 1 470  120 1 470
130 1 470  140 1 470  150 1 470  160 1 470  170 1 470  180 1 6100  0 0 4300  270 1 5000  180 0 4300  90 0 7000
0 1 8600
postscript2_length .word  117
postscript3:    .word       180,1,8600, 90,0,4600, 0,1,8600, 180,0,4300,
270,1,4600, 90,0,6000, 0,0,4310, 180,1,6100, 170,1,470, 160,1,470, 150,1,470,
140,1,470, 130,1,470, 120,1,470, 110,1,470, 100,1,470, 90,1,20, 80,1,470, 70,1,470,
60,1,470, 50,1,470, 40,1,470, 30,1,470, 20,1,470,10,1,470, 0,1,6100,
90,0,1500,150,1,5000,30,1,5000,210,0,5000,180,1,4300
postscript3_length: .word   93

Pleasechoose   .asciiz    "Press 0 to print DCE\nPress 4 to print KHAI\nPress 8 to print HUY\n\n"
Messenger1 .asciiz    "You choose to print DCE!\n"
enter  .asciiz    "\n"
Messenger2 .asciiz    "You choose to print KHAI!\n"
Messenger3    .asciiz    "You choose to print HUY!\n"
space  .asciiz
# declaring table
# postscript length = numberOfLines*3
.eqv    IN_ADDRESS_HEXA_KEYBOARD     0xFFFF0012
.eqv                                 0xFFFF0014
.eqv    HEADING       0xffff8010  # Integer: An angle between 0 and 359
                      # 0 : North (up)
                      # 90: East (right)
                      # 180: South (down)
                      # 270: West (left)
.eqv    LEAVETRACK    0xffff8020  # Boolean (0 or non-0):
                      # whether or not to leave a track
.eqv    WHEREX        0xffff8030  # Integer: Current x-location of MarsBot
.eqv    WHEREY        0xffff8040  # Integer: Current y-location of MarsBot
.eqv    MOVING        0xffff8050  # Boolean: whether or not to move

.text
    addi            0    # count number of successful postscript
    addi            0    # $t4 là biến check xem postscript1 đã được vẽ chưa
    addi            0    # $t5 là biến check xem postscript2 đã được vẽ chưa
    addi            0    # $s6 là biến check xem postscript3 đã được vẽ chưa
                  # 0 - chưa vẽ, 1 - đã vẽ => $s5 += 1
                    # $s5 === 3 => cả 3 postscript đều đã được vẽ => complete
    addi            3    # $s6 = 3 lần vẽ
polling
row1
    li
```

```
        li
        li      0x01            # check row 1 with key 0, 1, 2, 3
        sb      0
        lb      0               # read scan code of key button
        bne     0x00000011              # check trùng với 0 thì vẽ postscript1 nếu không thì check row2

draw0   li      4
        la
        syscall
        la
check0  bne     0                       # nếu poscript1 đã được vẽ 1 lần trước đó rồi thì $t5=1 và không
tăng $t4 nữa
        li      1               # postscript1 sẽ done
        addi            1       # done 1/3 postscript
postscript1_already_done
        la                      # gán $t7=length của mảng
        lw      0
        j
        nop

row2
        li
        li
        li      0x02            # check row 2 with key 4, 5, 6, 7
        sb      0
        lb      0               # read scan code of key button
        bne     0x00000012              # 4 - postscript2
draw4   li      4
        la
        syscall
        la
check4  bne     0                       # nếu poscript2 đã được vẽ 1 lần trước đó rồi thì $t6=1 và
không tăng $t4 nữa
        li      1               # postscript2 done
        addi            1       # done 1 postscript
postscript2_already_done
        la                      # gán $t7=length của mảng
        lw      0
        j
        nop

row3
        li
        li
        li      0x04            # check row 3 with key 8, 9, A, B
        sb      0
        lb      0               # read scan code of key button
        bne     0x00000014              # 8 - postscript3
draw8   li      4
```

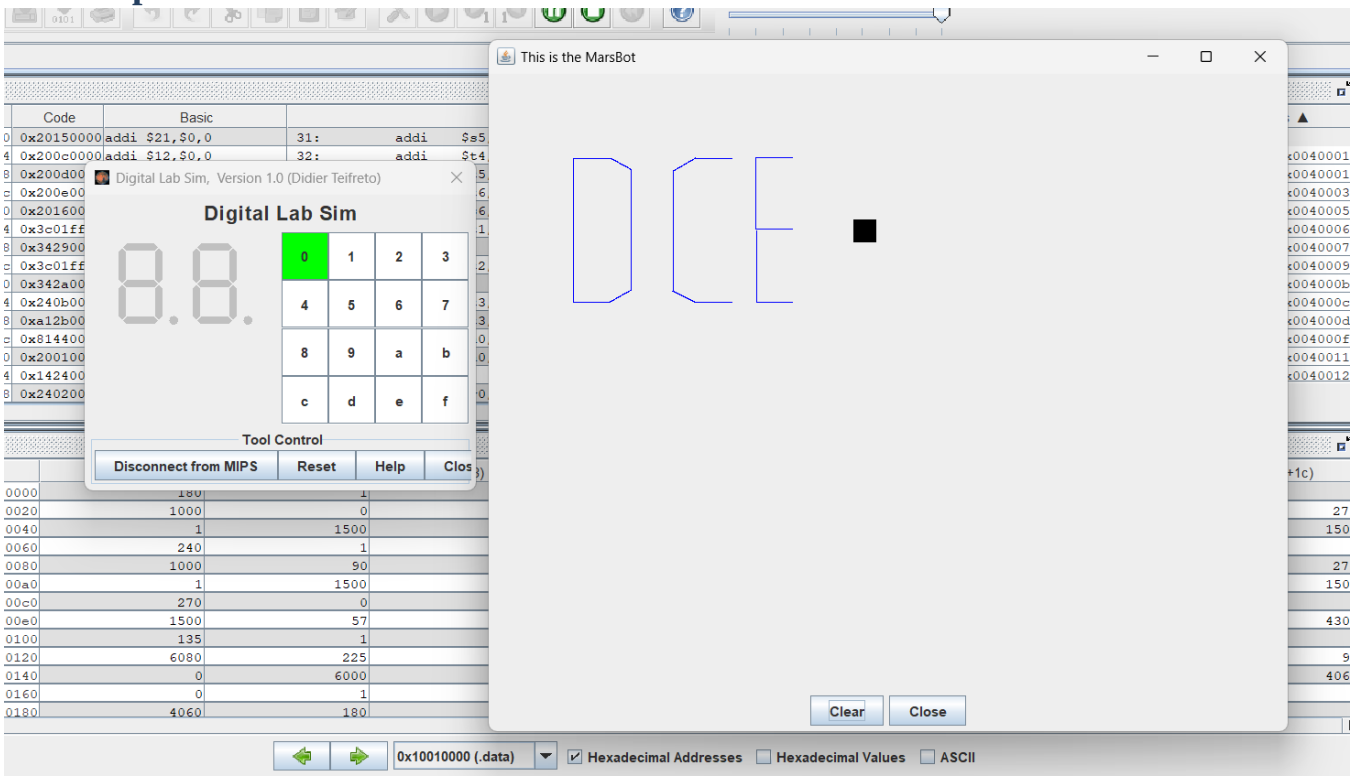
```
    la
    syscall
    la
check8: bne $t6, 0, postscript3_already_done     # nếu poscript3 đã được vẽ 1 lần
trước đó rồi thì $s5=1 và không tăng $t4 nữa
    li      1           # postscript3 done
    addi            1         # done 1 postscript
postscript3_already_done
    la                  # gán $t7=length của mảng
    lw      0
    j
    nop

invalid
    li      4
    la
    syscall
sleep_wait
    li      1000          # đợi 1000ms
    li      32
    syscall
    j

main
# Go to cut area
    jal                 # no draw track line
    addi            135    # Marsbot rotates given radius and start
start_running
    jal
    jal
start_sleep
    addi            32      # Keep running by sleeping in 5000 ms
    addi            5000
    syscall

    jal                 # keep old track

    li      0       # Set index counter for postscript array
loop
    beq                 # if i == numberOfLines*3 then quit
    sll             2    # s1 = 4i
    add                 # s1 = A[i]'s address
    lw      0         # s2 = A[i]'s value - <Góc chuyển động>
    addi            4
    lw      0         # s3 = A[i+1]'s value - <Cắt/Không cắt>
    addi            4
    lw      0         # s4 = A[i+2]'s value - <Thời gian chạy>

    jal                 # draw track line/ or not
```

```
running
    jal     [illegible]
    jal     [illegible]
sleep
    addi    [illegible]  32     # Keep running by sleeping bằng <Thời gian chạy>
   add      [illegible]
    syscall

    jal     [illegible]         # keep old track

    addi    [illegible]  3   # tăng lên 3 phần tử của mảng tiếp theo
    j       [illegible]
end_loop
    jal     [illegible]
    beq     [illegible]         # Nếu $s4 = $s6 thì dừng chương trình (với $s6 define bằng 3 ở ban đầu)
    j       [illegible]     # Nếu chưa vẽ đủ 3 postscript thì tiếp tục polling
end_main
    li      10
    syscall

#-------------------------------------------------------------
# GO procedure, to start running
# param[in] none
#-------------------------------------------------------------

GO
    li      [illegible]         # change MOVING port
    addi    [illegible]  1      # to logic 1,
    sb      0       [illegible]  # to start running
    jr      [illegible]

#-------------------------------------------------------------
# STOP procedure, to stop running
# param[in] none
#-------------------------------------------------------------

STOP
    li      [illegible]         # change MOVING port to 0
    sb      0       [illegible]  # to stop
   jr       [illegible]

#-------------------------------------------------------------
# TRACK procedure, to start drawing line
# param[in] none
#-------------------------------------------------------------

TRACK_UNTRACK
    li      [illegible]         # change LEAVETRACK port
    sb      0       [illegible]  # to start tracking/ or not
```

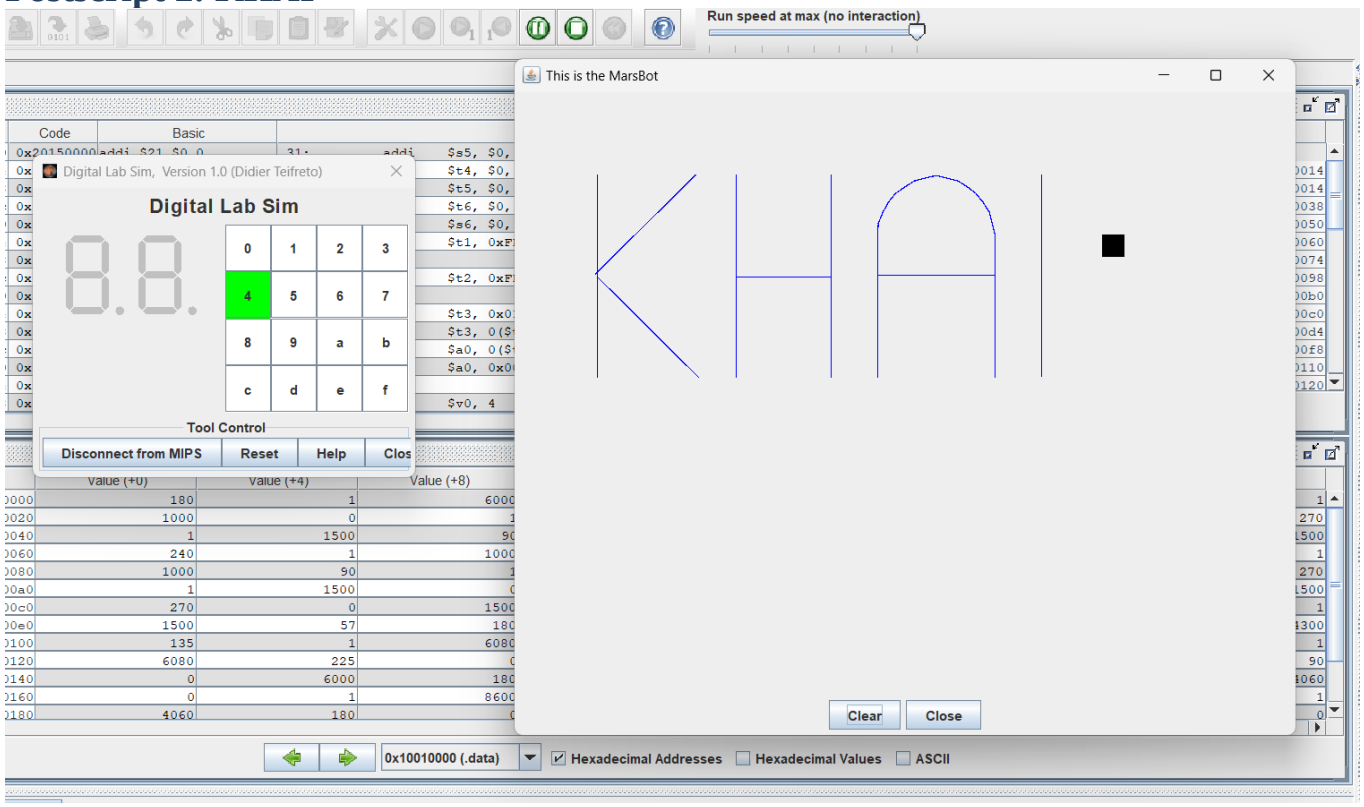
```
    jr $ra

#-----------------------------------------------------------
# UNTRACK procedure, to stop drawing line
# param[in] none
#-----------------------------------------------------------

TRACK
    li $t0, LEAVETRACK_port     # change LEAVETRACK port
    addi $t1, $zero, 1      # to logic 1,
    sb $t1, 0($t0)          # to start tracking
    jr $ra

UNTRACK
    li $t0, LEAVETRACK_port     # change LEAVETRACK port to 0
    sb $zero, 0($t0)        # to stop drawing tail
    jr $ra
#-----------------------------------------------------------
# ROTATE procedure, to rotate the robot
# param[in] $a0, An angle between 0 and 359
# 0 : North (up)
# 90: East (right)
# 180: South (down)
# 270: West (left)
#-----------------------------------------------------------
ROTATE
    li $t0, HEADING_port        # change HEADING port
    sw $a0, 0($t0)          # to rotate robot
    jr $ra
```

6 Kết quả

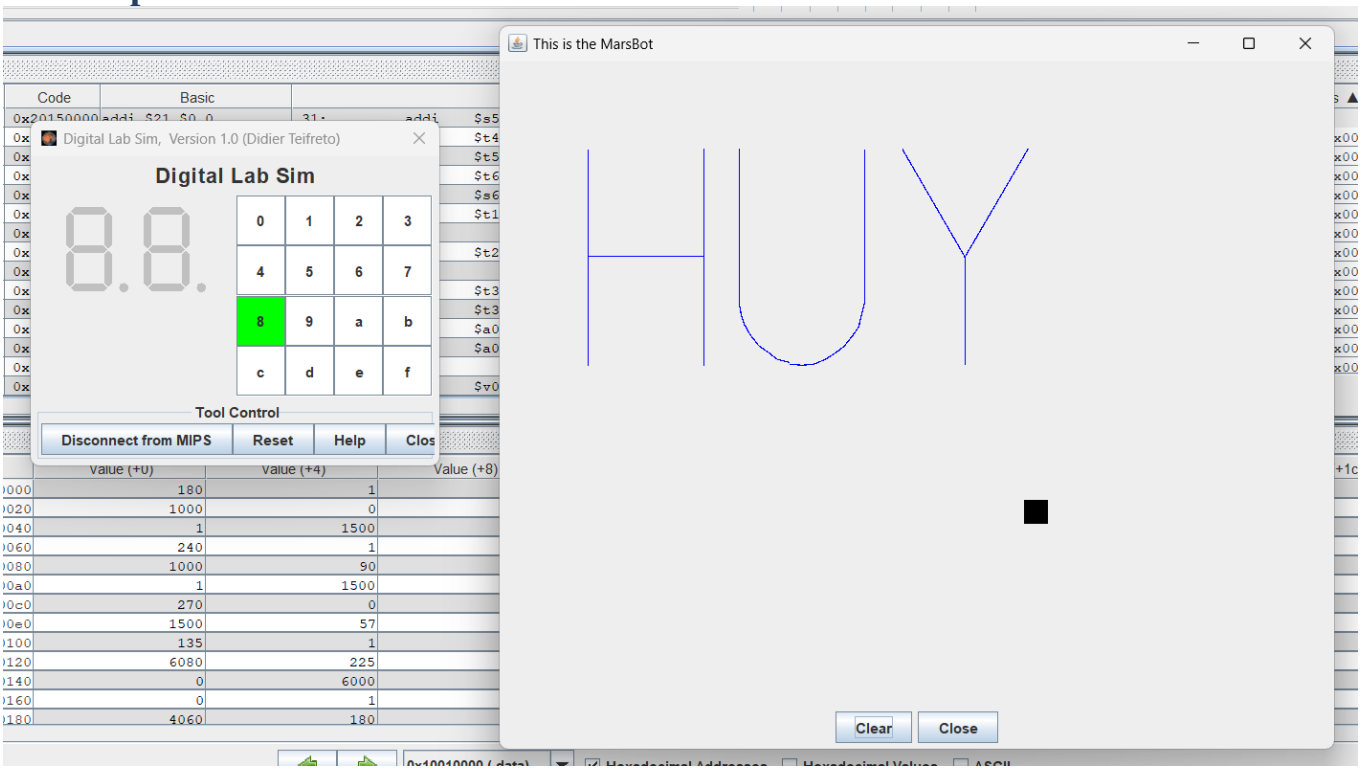
Postscript 1: DCE



Postscript 2: KHAI



Postscript 3: HUY



7 Giải thích code sơ bộ hay lưu đồ hoạt động

Khởi tạo các biến đếm:

- $s5: Đếm số postscript đã được thực hiện.
- $t4, $t5, $t6: Các biến để kiểm tra xem postscript1, postscript2, postscript3 đã được vẽ hay chưa.
- $s6: Tổng số postscript cần vẽ (3).

Polling (chờ phím):

- Chương trình liên tục kiểm tra đầu vào từ bàn phím để xác định người dùng đã nhấn phím nào.
- Dựa vào giá trị đầu vào, chương trình quyết định sẽ vẽ postscript nào.

Xử lý phím nhấn:

- Nếu phím 0 được nhấn: Thực hiện postscript1.
- Nếu phím 4 được nhấn: Thực hiện postscript2.
- Nếu phím 8 được nhấn: Thực hiện postscript3.
- Nếu phím khác: Hiển thị thông báo chọn phím hợp lệ và tiếp tục chờ phím.

Thực hiện postscript:

- Kiểm tra xem postscript tương ứng đã được thực hiện trước đó hay chưa.

- Nếu chưa, đánh dấu đã thực hiện và tăng biến đếm $s5 khi $s5=3 là khi đã vẽ đủ 3 postscript thì dừng chương trình
- Đọc độ dài của postscript và gọi hàm main để thực hiện.

Hàm main:

- Lặp qua từng hướng dẫn trong postscript và điều khiển robot theo từng bước duyệt mỗi lần 3 phần tử của mảng để
  - **ROTATE**: Đặt góc quay cho robot (góc vẽ từ 0-359^o)
  - **GO**: Bắt đầu di chuyển robot.
  - **TRACK_UNTRACK**: Thiết lập vẽ hoặc không vẽ (0 hoặc 1)
  - **SLEEP**: Tạm dừng trong khoảng thời gian được chỉ định.

Kiểm tra hoàn tất:

- Sau khi thực hiện xong postscript, kiểm tra xem tất cả các postscript đã được thực hiện hay chưa nếu $s5=3 thì dừng chương trình nếu không thì tiếp tục vẽ đến khi nào vẽ đủ 3 postscript thì dừng

Phân tích chi tiết

Polling: Phần polling là một vòng lặp vô hạn kiểm tra đầu vào từ bàn phím và quyết định hành động tương ứng. Điều này đảm bảo chương trình luôn sẵn sàng nhận lệnh từ người dùng.

Xử lý phím nhấn và kiểm tra postscript đã thực hiện: Các biến $t4, $t5, $t6 giúp theo dõi trạng thái của mỗi postscript để tránh thực hiện lại các postscript đã được vẽ.

Vòng lặp thực hiện postscript: Mỗi postscript được thực hiện bằng cách lặp qua từng hướng dẫn trong mảng và thực hiện các hành động tương ứng. Điều này cho phép linh hoạt trong việc thêm, sửa hoặc xóa các hướng dẫn mà không cần thay đổi cấu trúc chương trình.

8 Các vấn đề và hạn chế

Việc **điều kiện thoát hẳn chương trình** (đến end_main) sẽ chỉ đạt khi người dùng vẽ đủ 3 postscripts, vậy nên nếu người dùng cứ vẽ lại 1 postscript hoặc 2 postscripts thì chương trình sẽ không thoát mà phải chạy lại bằng tay –> hạn chế. Mặt khác người dùng có thể vẽ nhiều lần 1 postscript.

Mỗi lần vẽ xong 1 postscript người dùng sẽ phải **tự clear khu vực đã vẽ / cắt của MarsBot thủ công**, đồng thời **khi ấn 1 key** nào đó hợp lệ (Ví dụ: 0) thì phải **nhớ tắt**

key đó luôn bởi nếu để quên thì nó sẽ tiếp tục nhận key đó tiếp và vẽ lại postscript dẫn đến phải tắt chương trình thủ công và chạy lại.