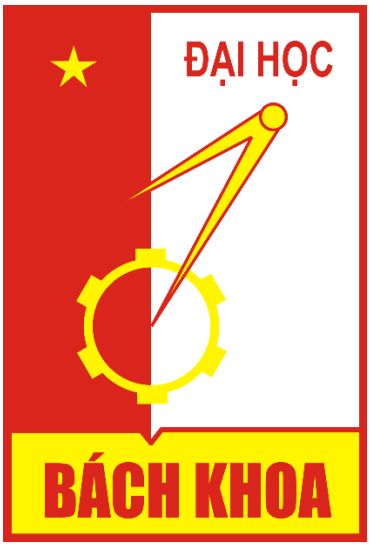
ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



BÁO CÁO PROJECT CUỐI KỲ HỌC PHẦN

THỰC HÀNH KIẾN TRÚC MÁY TÍNH

Giảng viên hướng dẫn : **ThS. Lê Bá Vui**
Lớp : **147789**
Nhóm sinh viên thực hiện : **Lê Bá Ngọc Hiểu - 20225627**
Phạm Quốc Minh - 20225743

Hà Nội, tháng 06 năm 2024

Mục Lục

1. Curiosity Marsbot – Lê Bá Ngọc Hiểu 20225627

Yêu cầu

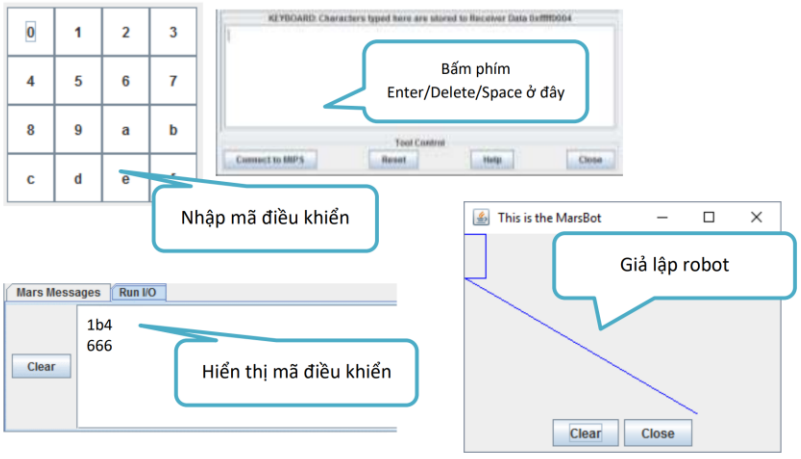
Xe tự hành Curiosity Marsbot chạy trên sao Hỏa, được vận hành từ xa bởi các lập trình viên trên Trái Đất. Bằng cách gửi đi các mã điều khiển từ một bàn phím ma trận, lập trình viên điều khiển quá trình di chuyển của Marbot như sau:

Mã điều khiển	**Ý nghĩa**
1b4	Marsbot bắt đầu chuyển động
c68	Marsbot đứng im
444	Rẽ trái 90 độ so với phương chuyển động gần nhất
666	Rẽ phải 90 độ so vơi phương chuyển động gần nhất
dad	Bắt đầu để lại vết trên đường
cbc	Chấm dứt để lại vết trên đường
999	Tự động đi theo lộ trình ngược lại. Không vẽ vết, không nhận mã khác cho tới khi kết thúc lộ trình ngược. Mô tả: Marsbot được lập trình để nhớ lại toàn bộ lịch sử các mã điều khiển và khoảng thời gian giữa các lần đổi mã. Vì vậy, nó có thể đảo ngược lại lộ trình để quay về điểm xuất phát.

Sau khi nhận mã điều khiển, Curiosity Marsbot sẽ không xử lý ngay, mà phải đợi lệnh kích hoạt mã từ bàn phím Keyboard & Display MMIO Simulator. Có 3 lệnh như vậy:

Kích hoạt mã	**Ý nghĩa**
Phím Enter	Kết thúc nhập mã và yêu cầu Marsbot thực thi.
Phím Delete	Xóa toàn bộ mã điều khiển đang nhập.
Phím Space	Lặp lại lệnh đã thực hiện trước đó.

Hãy lập trình để Marsbot có thể hoạt động như đã mô tả. Đồng thời bổ sung thêm tính năng: mỗi khi gửi một mã điều khiển cho Marsbot, hiển thị mã đó lên màn hình console để người xem có thể giám sát lộ trình của xe.



Phân tích hướng giải quyết

Chương trình Assembly này được viết để điều khiển một robot MarsBot với các chức năng chính như di chuyển, quay trái, quay phải, dừng lại, và để lại dấu vết.

Phân tích từng phần của mã nguồn và thuật toán

Thuật toán:

Ban đầu bật chế độ ngắt của bàn phím ma trận 4x4 của Digital Lab Sim.

Vì ban đầu khi người dung muốn bot di chuyển thì bot phải đi ngang sang phải nên phải xoay bot 90 độ ngay từ đầu và lưu vào lịch sử tọa độ và góc ban đầu là 0,0,90.

Sau đó sẽ chạy vòng lặp vô hạn để kiểm tra xem có phím vào được vào trong Keyboard and Display MMIO Similator hay không. Và khi nhấn 1 nút trong bàn phím của Digital Lab Sim thì ngắt sẽ xảy ra và chương trình sẽ lưu nút đó vào trong control_code.

Sau khi người dùng đã nhập đủ 3 kí tự để có 1 lệnh đúng. Hàm waitForKey chờ người ấn 1 trong 3 nút: Delete, Enter or Space chương trình sẽ thực thi như sau:

- **Phím Enter**: Kết thúc nhập mã và yêu cầu Marsbot thực thi: Lệnh thực thi này được thực hiện bằng cách: ban đầu sẽ kiểm tra xem lệnh nhập vào có phải là 1 lệnh hợp lệ hay không như đủ 3 kí tự hay không, có nằm trong những lệnh đã khai báo hay không(việc này được kiểm tra bởi hàm strcmp). Sau đó sẽ gọi hàm theo những hàm đã nhập vào và thực thi. Riêng hàm “999” sẽ dựa vào mảng lịch sử đã lưu tọa độ và góc. Khi quay lại chỉ cần góc đó cộng thêm 180 độ và khi chạy đến tọa độ đã lưu trước đó thì dừng lại và lại tiếp tục quay theo góc trong lịch sử. Tiếp tục cho đến khi tọa độ x, y đều bằng 0.
- **Phím Delete**: Xóa toàn bộ mã điều khiển đang nhập: Xóa bằng cách thực thi hàm con strClear bằng cách gán toàn bộ kí tự đã lưu trong control_code bằng giá trị của thanh ghi $zero.
- **Phím Space**: Lặp lại lệnh đã thực hiện trước đó: bằng cách khi chạy lệnh bằng Enter thì sẽ có 1 hàm là strCpy2 lưu lại các code đã chạy vào trong prev_control_code để khi ấn nút Space thì sẽ gọi hàm con strCpy1 để copy code từ prev_control_code và control_code sau đó sẽ gọi hàm checkCode để kiểm tra và thực hiện code của phần Enter.

Khai báo các giá trị hằng số và địa chỉ bộ nhớ

Các giá trị hằng số được khai báo để dễ dàng truy cập các phím và địa chỉ bộ nhớ. Sử dụng chỉ thị **.eqv** để gán giá trị.

```
# eqv for Digital Lab Sim
.eqv KEY_0 0x11
.eqv KEY_1 0x21
.eqv KEY_2 0x41
.eqv KEY_3 0x81
.eqv KEY_4 0x12
.eqv KEY_5 0x22
.eqv KEY_6 0x42
.eqv KEY_7 0x82
.eqv KEY_8 0x14
.eqv KEY_9 0x24
.eqv KEY_a 0x44
.eqv KEY_b 0x84
.eqv KEY_c 0x18
.eqv KEY_d 0x28
.eqv KEY_e 0x48
.eqv KEY_f 0x88

# eqv for Keyboard
.eqv IN_ADRESS_HEXA_KEYBOARD 0xFFFF0012
.eqv OUT_ADRESS_HEXA_KEYBOARD 0xFFFF0014
.eqv KEY_CODE 0xFFFF0004    # ASCII code from keyboard, 1 byte
.eqv KEY_READY 0xFFFF0000   # = 1 if has a new keycode ?
                    # Auto clear after lw

# eqv for Mars bot
.eqv HEADING 0xffff8010         # Integer: An angle between 0 and 359
                    # 0 : North (up)
                    # 90: East (right)
                    # 180: South (down)
                    # 270: West (left)
.eqv MOVING 0xffff8050      # Boolean: whether or not to move
.eqv LEAVETRACK 0xffff8020  # Boolean: 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
```

Các khai báo này giúp chương trình dễ dàng tham chiếu đến các địa chỉ bộ nhớ và mã phím khi thực hiện các thao tác.

Khai báo dữ liệu

Dữ liệu bao gồm các mã lệnh, thông báo lỗi và các biến lưu trữ lịch sử di chuyển của **MarsBot**.

```
.data
    MOVE_CODE: .asciiz "1b4"
    STOP_CODE: .asciiz "c68"
    TURN_LEFT_CODE: .asciiz "444"
    TURN_RIGHT_CODE: .asciiz "666"
    TRACK_CODE: .asciiz "dad"
    UNTRACK_CODE: .asciiz "cbc"
    GOBACKWARD_CODE: .asciiz "999"
    error_msg: .asciiz "Invalid command code: "

    x_history: .word 0 : 16
    y_history: .word 0 : 16
    a_history: .word 0 : 16
    l_history: .word 4
    a_current: .word 0
    is_going: .word 0
    is_tracking: .word 0
    control_code: .space 8
    code_length: .word 0
    prev_control_code: .space 8
```

- **MOVE_CODE, STOP_CODE, ...** : Chuỗi ASCII lưu các mã lệnh điều khiển MarsBot.
- **x_history, y_history, a_history**: Mảng lưu lịch sử vị trí và góc của MarsBot.
- **l_history**: Độ dài lịch sử.
- **a_current**: Góc hiện tại.
- **is_going, is_tracking**: Trạng thái di chuyển và để lại dấu vết.
- **control_code, code_length**: Lưu mã lệnh nhập vào và độ dài mã lệnh.
- **prev_control_code**: Lưu mã lệnh trước đó.

Phần mã chính

```
.text
main:
    li $k0, KEY_CODE
    li $k1, KEY_READY

    # Vòng lặp nhận mã lệnh từ bàn phím
inputLoop:
    lw $t0, 0($k1)
```

```
    beqz $t0, inputLoop
    lw $t0, 0($k0)
    # Xử lý mã lệnh
    j checkKeyCode
```

- **main**: Chương trình chính bắt đầu từ đây.
- **li $k0, KEY_CODE**: Tải địa chỉ chứa mã phím vào thanh ghi $k0.
- **li $k1, KEY_READY**: Tải địa chỉ kiểm tra phím sẵn sàng vào thanh ghi $k1.
- **inputLoop**: Vòng lặp liên tục kiểm tra xem có phím mới nhập vào không.
- **lw $t0, 0($k1)**: Đọc giá trị từ địa chỉ $k1 vào thanh ghi $t0.
- **beqz $t0, inputLoop**: Nếu $t0 bằng 0 (không có phím mới), tiếp tục vòng lặp.
- **lw $t0, 0($k0)**: Đọc mã phím từ địa chỉ $k0 vào thanh ghi $t0.
- **j checkKeyCode**: Chuyển đến nhãn checkKeyCode để xử lý mã phím.

Xử lý mã phím

```
checkKeyCode:
    li $t4, 3
    li $s0, '1'
    beq $t0, $s0, case_1
    li $s0, '2'
    beq $t0, $s0, case_2
    # (Các case khác tương tự)

case_1:
    li $s0, '1'
    j storeCode

# (Các case khác tương tự)

storeCode:
    la $s1, control_code
    la $s2, code_length
    lw $s3, 0($s2)
    addi $t4, $t4, -1

storeCodeLoop:
    addi $t4, $t4, 1
    bne $t4, $s3, storeCodeLoop
    add $s1, $s1, $t4
    sb $s0, 0($s1)
    addi $s0, $zero, '\n'
    addi $s1, $s1, 1
```

```
    sb $s0, 0($s1)
    addi $s3, $s3, 1
    sw $s3, 0($s2)
```

- **checkKeyCode**: Kiểm tra mã phím và chuyển đến case tương ứng.
- **case_1, case_2:** Nếu mã phím là '1', chuyển đến **case_1**. Tương tự với các phím khác.
- **storeCode**: Lưu mã phím vào control_code.

Thực hiện lệnh

```
executeCommand:
    # (Các lệnh điều khiển MarsBot tương ứng)
    # Ví dụ:
    li $t0, MOVE_CODE
    li $t1, 1
    sw $t1, 0($t0)

end:
    li $v0, 10
    syscall
```

- **executeCommand**: Thực hiện các lệnh điều khiển MarsBot dựa trên mã lệnh đã nhận.
- **end**: Kết thúc chương trình, gọi hệ thống để thoát.

Mã nguồn

```
# eqv for Digital Lab Sim
.eqv KEY_0 0x11
.eqv KEY_1 0x21
.eqv KEY_2 0x41
.eqv KEY_3 0x81
.eqv KEY_4 0x12
.eqv KEY_5 0x22
.eqv KEY_6 0x42
.eqv KEY_7 0x82
.eqv KEY_8 0x14
.eqv KEY_9 0x24
.eqv KEY_a 0x44
.eqv KEY_b 0x84
.eqv KEY_c 0x18
.eqv KEY_d 0x28
.eqv KEY_e 0x48
.eqv KEY_f 0x88

# eqv for Keyboard
.eqv IN_ADRESS_HEXA_KEYBOARD 0xFFFF0012
.eqv OUT_ADRESS_HEXA_KEYBOARD 0xFFFF0014
.eqv KEY_CODE 0xFFFF0004    # ASCII code from keyboard, 1 byte
.eqv KEY_READY 0xFFFF0000   # = 1 if has a new keycode ?
                    # Auto clear after lw

# eqv for Mars bot
.eqv HEADING 0xffff8010         # Integer: An angle between 0 and 359
                    # 0 : North (up)
                    # 90: East (right)
                    # 180: South (down)
                    # 270: West (left)
.eqv MOVING 0xffff8050      # Boolean: whether or not to move
.eqv LEAVETRACK 0xffff8020  # Boolean: 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

#-------------------------------------------------------------------------------
.data
    # CODE
    MOVE_CODE: .asciiz "1b4" # command code
    STOP_CODE: .asciiz "c68"
    TURN_LEFT_CODE: .asciiz "444"
    TURN_RIGHT_CODE: .asciiz "666"
```

```
    TRACK_CODE: .asciiz "dad"
    UNTRACK_CODE: .asciiz "cbc"
    GOBACKWARD_CODE: .asciiz "999"

    error_msg:  .asciiz "Invalid command code: "


    # HISTORY
    # save history before changing direction

    x_history: .word 0 : 16 # = 16 for easier debugging
    y_history: .word 0 : 16

    # For rotation
    a_history: .word 0 : 16
    l_history: .word 4          # history length

    a_current: .word 0          # current alpha

    is_going: .word 0
    is_tracking: .word 0

    # Code properties
    control_code: .space 8      # input command code
    code_length: .word 0        # input command length

    prev_control_code: .space 8     # store previous input code

.text

main:
    li $k0, KEY_CODE
    li $k1, KEY_READY

    li $t1, IN_ADRESS_HEXA_KEYBOARD # enable the interrupt of Digital Lab Sim
    li $t3, 0x80   # bit 7 = 1 to enable interrupt
    sb $t3, 0($t1)

# run at start of program
init:
    # increase length history by 4
    # (as saving current state: x = 0; y = 0; a = 90)

    lw $t7, l_history # l_history += 4
    addi $t7, $zero, 4
```

```
    sw $t7, l_history

    li $t7, 90
    sw $t7, a_current # a_current = 90 -> head to the right
    jal ROTATE
    nop

    sw $t7, a_history # a_history[0] = 90

    j waitForKey

# Function: print error to console
printError:
    li $v0, 4
    la $a0, error_msg
    syscall

printCode:
    li $v0, 4
    la $a0, control_code
    syscall
    j resetInput

repeatCode:
    # copy from the prev_control_code
    jal strCpy1
    j checkCode

resetInput:
    jal strClear
    nop

# Take input
waitForKey:
    lw $t5, 0($k1)   # $t5 = [$k1] = KEY_READY
    beq $t5, $zero, waitForKey  # if $t5 == 0 -> Polling
    nop
    beq $t5, $zero, waitForKey

readKey:
    lw $t6, 0($k0)   # $t6 = [$k0] = KEY_CODE
    # if $t6 == 'DEL' -> reset input
    beq $t6, 0x8, resetInput

    # if $t6 == 'SPACE' -> reset copy from previous input and
```

```
    # go to checkCode label
    beq $t6, 0x20, repeatCode

    # if $t6 != 'ENTER' kí tự xuống dòng -> Polling
    bne $t6, 0x0a, waitForKey
    nop
    bne $t6, 0x0a, waitForKey

checkCode:
    lw $s2, code_length   # code_length != 3 -> invalid code
    bne $s2, 3, printError

    la $s3, MOVE_CODE
    jal strcmp
    beq $t0, 1, go

    la $s3, STOP_CODE
    jal strcmp
    beq $t0, 1, stop

    la $s3, TURN_LEFT_CODE
    jal strcmp
    beq $t0, 1, turnLeft

    la $s3, TURN_RIGHT_CODE
    jal strcmp
    beq $t0, 1, turnRight

    la $s3, TRACK_CODE
    jal strcmp
    beq $t0, 1, track

    la $s3, UNTRACK_CODE
    jal strcmp
    beq $t0, 1, untrack

    la $s3, GOBACKWARD_CODE
    jal strcmp
    beq $t0, 1, goBackward
    nop

    j printError


# Perform function and print code
```

```
go:
    jal strCpy2
    jal GO
    j printCode

stop:
    jal strCpy2
    jal STOP
    j printCode

track:
    jal strCpy2
    jal TRACK
    j printCode

untrack:
    jal strCpy2
    jal UNTRACK
    j printCode


turnRight:
    jal strCpy2
    lw $t7, is_going
    lw $s0, is_tracking

    jal STOP
    nop
    jal UNTRACK
    nop

    la $s5, a_current
    lw $s6, 0($s5)  # $s6 is heading at now
    addi $s6, $s6, 90 # increase alpha by 90*
    sw $s6, 0($s5)  # update a_current

    jal saveHistory
    jal ROTATE

    beqz $s0, noTrack1
    nop
    jal TRACK
    noTrack1: nop

    beqz $t7, noGo1
```

```
    nop
    jal GO
noGo1:
    nop
    j printCode


turnLeft:
    jal strCpy2
    lw $t7, is_going
    lw $s0, is_tracking

    jal STOP
    nop
    jal UNTRACK
    nop

    la $s5, a_current
    lw $s6, 0($s5)  # $s6 is heading at now
    addi $s6, $s6, -90 # decrease alpha by 90*
    sw $s6, 0($s5)  # update a_current

    jal saveHistory
    jal ROTATE

    beqz $s0, noTrack2
    nop
    jal TRACK
    noTrack2: nop

    beqz $t7, noGo2
    nop
    jal GO
noGo2:
    nop
    j printCode


goBackward:
    jal strCpy2
    li $t7, IN_ADRESS_HEXA_KEYBOARD # Disable interrupts when going backward
        sb $zero, 0($t7)

    lw $s5, l_history  # $s5 = code_length
    jal UNTRACK
```

```
    jal GO

goBackward_turn:
    addi $s5, $s5, -4   # code_length--
    lw $s6, a_history($s5)  # $s6 = a_history[code_length]
    addi $s6, $s6, 180  # $s6 = the reverse direction of alpha
    sw $s6, a_current
    jal ROTATE
    nop

goBackward_toTurningPoint:
    lw $t9, x_history($s5)  # $t9 = x_history[i]
    lw $t7, y_history($s5)  # $t9 = y_history[i]

get_x:
    li $t8, WHEREX   # $t8 = x_current
    lw $t8, 0($t8)

    bne $t8, $t9, get_x  # x_current == x_history[i]
    nop
    bne $t8, $t9, get_x

get_Y:
    li $t8, WHEREY   # $t8 = y_current
    lw $t8, 0($t8)

    bne $t8, $t7, get_Y  # y_current == y_history[i]
    nop
    bne $t8, $t7, get_Y  # y_current == y_history[i]

    beq $s5, 0, goBackward_end  # l_history == 0
    nop    # -> end

    j goBackward_turn   # else -> turn

goBackward_end:
    jal STOP
    sw $zero, a_current  # update heading
    jal ROTATE

    addi $s5, $zero, 0
    sw $s5, l_history  # reset l_history = 0


    #bật lại chế độ ngắt và reset bot quay 90 độ khi bắt đầu.
```

```
    #ban đầu backup dữ liệu
    addi $sp, $sp, 4
    sw $t1, 0($sp)
    addi $sp, $sp, 4
    sw $t3, 0($sp)
    addi $sp, $sp, 4
    sw $t7, 0($sp)


    #bật lại chế độ ngắt
    li $t1, IN_ADRESS_HEXA_KEYBOARD # enable the interrupt of Digital Lab Sim
    li $t3, 0x80   # bit 7 = 1 to enable interrupt
    sb $t3, 0($t1)

    #xoay 90 độ khi bắt đầu chạy bot
    lw $t7, l_history # l_history += 4
    addi $t7, $zero, 4
    sw $t7, l_history

    li $t7, 90
    sw $t7, a_current # a_current = 90 -> head to the right
    jal ROTATE
    nop

    sw $t7, a_history # a_history[0] = 90
    #restore lại dữ liệu
    lw $t7, 0($sp)
    addi $sp, $sp, -4
    lw $t3, 0($sp)
    addi $sp, $sp, -4
    lw $t1, 0($sp)
    addi $sp, $sp, -4


    j printCode

#-----------------------------------------------------------
# saveHistory()
#-----------------------------------------------------------

saveHistory:
    addi $sp, $sp, 4   # backup
    sw $t1, 0($sp)
    addi $sp, $sp, 4
    sw $t2, 0($sp)
```

```
    addi $sp, $sp, 4
    sw $t3, 0($sp)
    addi $sp, $sp, 4
    sw $t4, 0($sp)
    addi $sp, $sp, 4
    sw $s1, 0($sp)
    addi $sp, $sp, 4
    sw $s2, 0($sp)
    addi $sp, $sp, 4
    sw $s3, 0($sp)
    addi $sp, $sp, 4
    sw $s4, 0($sp)

    lw $s1, WHEREX   # s1 = x
    lw $s2, WHEREY   # s2 = y
    lw $s4, a_current  # s4 = a_current

    lw $t3, l_history  # $t3 = l_history
    sw $s1, x_history($t3)  # store: x, y, alpha
    sw $s2, y_history($t3)
    sw $s4, a_history($t3)

    addi $t3, $t3, 4   # update lengthPath
    sw $t3, l_history

    lw $s4, 0($sp)   # restore backup
    addi $sp, $sp, -4
    lw $s3, 0($sp)
    addi $sp, $sp, -4
    lw $s2, 0($sp)
    addi $sp, $sp, -4
    lw $s1, 0($sp)
    addi $sp, $sp, -4
    lw $t4, 0($sp)
    addi $sp, $sp, -4
    lw $t3, 0($sp)
    addi $sp, $sp, -4
    lw $t2, 0($sp)
    addi $sp, $sp, -4
    lw $t1, 0($sp)
    addi $sp, $sp, -4

saveHistory_end:
    jr $ra
```

```
#################################################################################
# Procedure for Mars bot
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# GO()
#---------------------------------------------------------------
GO:
    addi $sp, $sp, 4   # backup
    sw $at, 0($sp)
    addi $sp, $sp, 4
    sw $k0, 0($sp)

    li $at, MOVING   # change MOVING port
    addi $k0, $zero, 1  # to logic 1,
    sb $k0, 0($at)   # to start running

    li $t7, 1   # is_going = 0
    sw $t7, is_going

    lw $k0, 0($sp)   # restore back up
    addi $sp, $sp, -4
    lw $at, 0($sp)
    addi $sp, $sp, -4

GO_end:
    jr $ra

#---------------------------------------------------------------
# STOP()
#---------------------------------------------------------------
STOP:
    addi $sp, $sp, 4   # backup
    sw $at, 0($sp)

    li $at, MOVING   # change MOVING port to 0
    sb $zero, 0($at)  # to stop

    sw $zero, is_going  # is_going = 0

    lw $at, 0($sp)   # restore back up
    addi $sp, $sp, -4

STOP_end:
    jr $ra

#---------------------------------------------------------------
```

```
# TRACK()
#-------------------------------------------------------------
TRACK:
    addi $sp, $sp, 4   # backup
    sw $at, 0($sp)
    addi $sp, $sp, 4
    sw $k0, 0($sp)

    li $at, LEAVETRACK  # change LEAVETRACK port
    addi $k0, $zero,1  # to logic 1,
    sb $k0, 0($at)   # to start tracking

    addi $s0, $zero, 1
    sw $s0, is_tracking

    lw $k0, 0($sp)   # restore back up
    addi $sp, $sp, -4
    lw $at, 0($sp)
    addi $sp, $sp, -4

TRACK_end:
    jr $ra

#-------------------------------------------------------------
# UNTRACK()
#-------------------------------------------------------------
UNTRACK:
    addi $sp, $sp, 4  # backup
    sw $at, 0($sp)

    li $at, LEAVETRACK # change LEAVETRACK port to 0
    sb $zero, 0($at) # to stop drawing tail

    sw $zero, is_tracking

    lw $at, 0($sp)  # restore back up
    addi $sp, $sp, -4

UNTRACK_end:
    jr $ra

#-------------------------------------------------------------
# ROTATE()
#-------------------------------------------------------------
ROTATE:
```

```
    addi $sp, $sp, 4  # backup
    sw $t1, 0($sp)
    addi $sp, $sp, 4
    sw $t2, 0($sp)
    addi $sp, $sp, 4
    sw $t3, 0($sp)

    li $t1, HEADING # change HEADING port
    la $t2, a_current
    lw $t3, 0($t2)  # $t3 is heading at now
    sw $t3, 0($t1)  # to rotate robot

    lw $t3, 0($sp)  # restore back up
    addi $sp, $sp, -4
    lw $t2, 0($sp)
    addi $sp, $sp, -4
    lw $t1, 0($sp)
    addi $sp, $sp, -4

ROTATE_end:
    jr $ra

#============================================================================
# Procedure for string
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# strcmp()
# - input: $s3 = string to compare with control_code
# - output: $t0 = 0 if not equal, 1 if equal
#----------------------------------------------------------
strcmp:
    addi $sp, $sp, 4   # back up
    sw $t1, 0($sp)
    addi $sp, $sp, 4
    sw $s1, 0($sp)
    addi $sp,$sp,4
    sw $t2, 0($sp)
    addi $sp, $sp, 4
    sw $t3, 0($sp)

    xor $t0, $zero, $zero  # $t1 = return value = 0
    xor $t1, $zero, $zero  # $t1 = i = 0

strcmp_loop:
    beq $t1, 3, strcmp_equal  # if i = 3 -> end loop -> equal
    nop
```

```
    lb $t2, control_code($t1)  # $t2 = control_code[i]

    add $t3, $s3, $t1  # $t3 = s + i
    lb $t3, 0($t3)   # $t3 = s[i]

    beq $t2, $t3, strcmp_next  # if $t2 == $t3 -> continue the loop
    nop

    j strcmp_end

strcmp_next:
    addi $t1, $t1, 1
    j strcmp_loop

strcmp_equal:
    add $t0, $zero, 1  # i++

strcmp_end:
    lw $t3, 0($sp)   # restore the backup
    addi $sp, $sp, -4
    lw $t2, 0($sp)
    addi $sp, $sp, -4
    lw $s1, 0($sp)
    addi $sp, $sp, -4
    lw $t1, 0($sp)
    addi $sp, $sp, -4

    jr $ra

#-------------------------------------------------------------
# strClear()
#-------------------------------------------------------------
strClear:
    addi $sp, $sp, 4   # backup
    sw $t1, 0($sp)
    addi $sp, $sp, 4
    sw $t2, 0($sp)
    addi $sp, $sp, 4
    sw $s1, 0($sp)
    addi $sp, $sp, 4
    sw $t3, 0($sp)
    addi $sp, $sp, 4
    sw $s2, 0($sp)
```

```
    lw $t3, code_length   # $t3 = code_length
    addi $t1, $zero, -1  # $t1 = -1 = i

strClear_loop:
    addi $t1, $t1, 1   # i++
    sb $zero, control_code  # control_code[i] = '\0'

    bne $t1, $t3, strClear_loop # if $t1 <=3 resetInput loop
    nop

    sw $zero, code_length  # reset code_length = 0

strClear_end:
    lw $s2, 0($sp)   # restore backup
    addi $sp, $sp, -4
    lw $t3, 0($sp)
    addi $sp, $sp, -4
    lw $s1, 0($sp)
    addi $sp, $sp, -4
    lw $t2, 0($sp)
    addi $sp, $sp, -4
    lw $t1, 0($sp)
    addi $sp, $sp, -4

    jr $ra

#-------------------------------------------------------------
# strCpy1(): copy value from prev to current code
#-------------------------------------------------------------
strCpy1:
    addi $sp, $sp, 4   # backup
    sw $t1, 0($sp)
    addi $sp, $sp, 4
    sw $t2, 0($sp)
    addi $sp, $sp, 4
    sw $s1, 0($sp)
    addi $sp, $sp, 4
    sw $t3, 0($sp)
    addi $sp, $sp, 4
    sw $s2, 0($sp)

    li $t2, 0
    # load address of control_code
    la $s1, control_code
```

```
    # load address of prev_control_code
    la $s2, prev_control_code

strCpy1_loop:
    beq $t2, 3, strCpy1_end

    # $t1 as control_code[i]
    lb $t1, 0($s2)
    sb $t1, 0($s1)

    addi $s1, $s1, 1
    addi $s2, $s2, 1
    addi $t2, $t2, 1

    j strCpy1_loop

strCpy1_end:
    # reset code length
    li $t3, 3
    sw $t3, code_length

    lw $s2, 0($sp)   # restore backup
    addi $sp, $sp, -4
    lw $t3, 0($sp)
    addi $sp, $sp, -4
    lw $s1, 0($sp)
    addi $sp, $sp, -4
    lw $t2, 0($sp)
    addi $sp, $sp, -4
    lw $t1, 0($sp)
    addi $sp, $sp, -4

    jr $ra


#-------------------------------------------------------------
# strCpy2(): copy value from current code to prev code
#-------------------------------------------------------------
strCpy2:
    addi $sp, $sp, 4   # backup
    sw $t1, 0($sp)
    addi $sp, $sp, 4
    sw $t2, 0($sp)
    addi $sp, $sp, 4
    sw $s1, 0($sp)
```

```
    addi $sp, $sp, 4
    sw $t3, 0($sp)
    addi $sp, $sp, 4
    sw $s2, 0($sp)

    li $t2, 0
    # load address of prev_control_code
    la $s1, prev_control_code

    # load address of control_code
    la $s2, control_code

strCpy2_loop:
    beq $t2, 3, strCpy2_end

    # $t1 as control_code[i]
    lb $t1, 0($s2)
    sb $t1, 0($s1)

    addi $s1, $s1, 1
    addi $s2, $s2, 1
    addi $t2, $t2, 1

    j strCpy2_loop

strCpy2_end:
    lw $s2, 0($sp)   # restore backup
    addi $sp, $sp, -4
    lw $t3, 0($sp)
    addi $sp, $sp, -4
    lw $s1, 0($sp)
    addi $sp, $sp, -4
    lw $t2, 0($sp)
    addi $sp, $sp, -4
    lw $t1, 0($sp)
    addi $sp, $sp, -4

    jr $ra

#=============================================================================
# GENERAL INTERRUPT SERVED ROUTINE for all interrupts
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
.ktext 0x80000180
#-----------------------------------------------------
# SAVE the current REG FILE to stack
```

```
#-----------------------------------------------------
backup:
    addi $sp, $sp, 4
    sw $ra, 0($sp)

    addi $sp, $sp, 4
    sw $t1, 0($sp)

    addi $sp, $sp, 4
    sw $t2, 0($sp)

    addi $sp, $sp, 4
    sw $t3, 0($sp)

    addi $sp, $sp, 4
    sw $a0, 0($sp)

    addi $sp, $sp, 4
    sw $at, 0($sp)
    addi $sp, $sp, 4
    sw $s0, 0($sp)
    addi $sp, $sp, 4
    sw $s1, 0($sp)
    addi $sp, $sp, 4
    sw $s2, 0($sp)
    addi $sp, $sp, 4
    sw $t4, 0($sp)
    addi $sp, $sp, 4
    sw $s3, 0($sp)
    #-------------------------------------------------------
    # Processing
    #-------------------------------------------------------
getCode:
    li $t1, IN_ADRESS_HEXA_KEYBOARD
    li $t2, OUT_ADRESS_HEXA_KEYBOARD

    # scan row 1
    li $t3, 0x81
    sb $t3, 0($t1)
    lbu $a0, 0($t2)
    bnez $a0, getCodeInChar

    # scan row 2
    li $t3, 0x82
    sb $t3, 0($t1)
```

```
    lbu $a0, 0($t2)
    bnez $a0, getCodeInChar

    # scan row 3
    li $t3, 0x84
    sb $t3, 0($t1)
    lbu $a0, 0($t2)
    bnez $a0, getCodeInChar

    # scan row 4
    li $t3, 0x88
    sb $t3, 0($t1)
    lbu $a0, 0($t2)
    bnez $a0, getCodeInChar

getCodeInChar:
    beq $a0, KEY_0, case_0
    beq $a0, KEY_1, case_1
    beq $a0, KEY_2, case_2
    beq $a0, KEY_3, case_3
    beq $a0, KEY_4, case_4
    beq $a0, KEY_5, case_5
    beq $a0, KEY_6, case_6
    beq $a0, KEY_7, case_7
    beq $a0, KEY_8, case_8
    beq $a0, KEY_9, case_9
    beq $a0, KEY_a, case_a
    beq $a0, KEY_b, case_b
    beq $a0, KEY_c, case_c
    beq $a0, KEY_d, case_d
    beq $a0, KEY_e, case_e
    beq $a0, KEY_f, case_f

case_0:
    li $s0, '0'  # $s0 store code in char type
    j storeCode
case_1:
    li $s0, '1'
    j storeCode
case_2:
    li $s0, '2'
    j storeCode
case_3:
    li $s0, '3'
    j storeCode
```

```
case_4:
    li $s0, '4'
    j storeCode
case_5:
    li $s0, '5'
    j storeCode
case_6:
    li $s0, '6'
    j storeCode
case_7:
    li $s0, '7'
    j storeCode
case_8:
    li $s0, '8'
    j storeCode
case_9:
    li $s0, '9'
    j storeCode
case_a:
    li $s0, 'a'
    j storeCode
case_b:
    li $s0, 'b'
    j storeCode
case_c:
    li $s0, 'c'
    j storeCode
case_d:
    li $s0, 'd'
    j storeCode
case_e:
    li $s0, 'e'
    j storeCode
case_f:
    li $s0, 'f'
    j storeCode

storeCode:
    la  $s1, control_code
    la $s2, code_length
    lw $s3, 0($s2)   # $s3 = strlen(control_code)
    addi $t4, $t4, -1   # $t4 = i

storeCodeLoop:
    addi $t4, $t4, 1
```

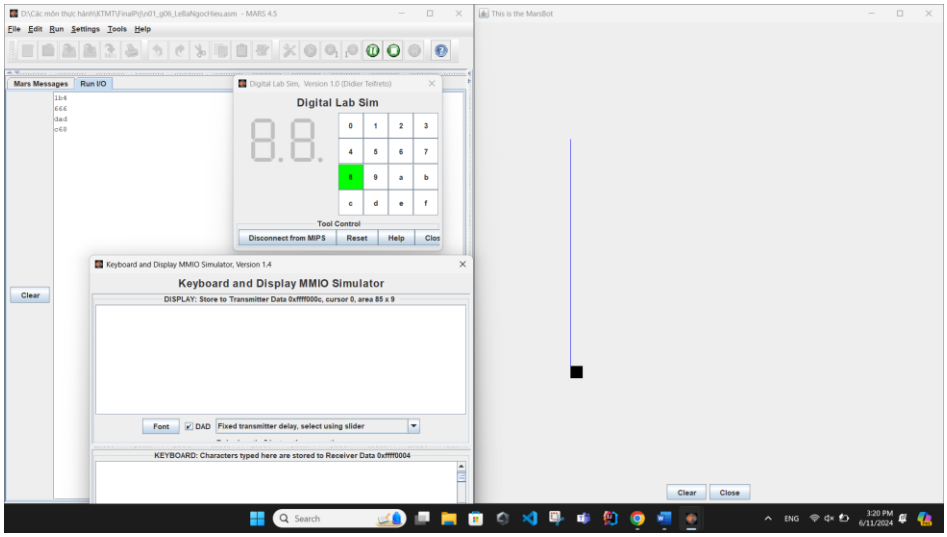
```
    bne $t4, $s3, storeCodeLoop
    add $s1, $s1, $t4  # $s1 = control_code + i
    sb $s0, 0($s1)   # control_code[i] = $s0

    #Luc nao cũng có dấu xuống dòng ở cuối
    addi $s0, $zero, '\n'  # add '\n' character to end of string
    addi $s1, $s1, 1
    sb $s0, 0($s1)

    addi $s3, $s3, 1
    sw $s3, 0($s2)   # update code_length

#-----------------------------------------------------------
# Evaluate the return address of main routine
# epc <= epc + 4
#-----------------------------------------------------------
next_pc:
    mfc0 $at, $14  # $at <= Coproc0.$14 = Coproc0.epc
    addi $at, $at, 4  # $at = $at + 4 (next instruction)
    mtc0 $at, $14  # Coproc0.$14 = Coproc0.epc <= $at
#-----------------------------------------------------------
# RESTORE the REG FILE from STACK
#-----------------------------------------------------------
restore:
    lw $s3, 0($sp)
    addi $sp, $sp, -4
    lw $t4, 0($sp)
    addi $sp, $sp, -4
    lw $s2, 0($sp)
    addi $sp, $sp, -4
    lw $s1, 0($sp)
    addi $sp, $sp, -4
    lw $s0, 0($sp)
    addi $sp, $sp, -4
    lw $at, 0($sp)
    addi $sp, $sp, -4
    lw $a0, 0($sp)
    addi $sp, $sp, -4
    lw $t3, 0($sp)
    addi $sp, $sp, -4
    lw $t2, 0($sp)
    addi $sp, $sp, -4
    lw $t1, 0($sp)
    addi $sp, $sp, -4
    lw $ra, 0($sp)
```

```
    addi $sp, $sp, -4
return: eret # Return from exception
```

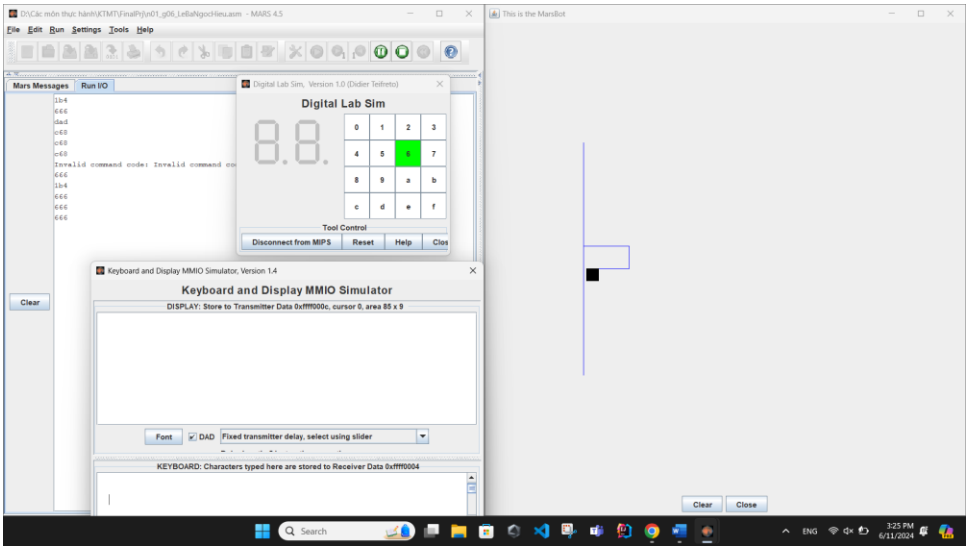
Kết quả chạy và mô phỏng

Khi chương trình chạy, nó sẽ liên tục đọc mã phím từ bàn phím và thực hiện các hành động tương ứng trên MarsBot. Các hành động bao gồm di chuyển, quay trái/phải, dừng lại, và để lại dấu vết.

Ví dụ, khi nhận được mã lệnh “1b4”, MarsBot sẽ di chuyển về phía trước. Khi nhận được mã lệnh “666”, MarsBot sẽ quay phải, sau đó sẽ gõ lệnh “dad” để để lại vết khi chạy và lệnh “c68” để dừng chương trình lại.



Sau đó gọi thêm lệnh “666” và sử dụng space 3 lần để gọi lại lệnh:



7. Chương trình kiểm tra cú pháp lệnh MIPS – Phạm Quốc Minh 20225743

Yêu cầu

Trình biên dịch của bộ xử lý MIPS sẽ tiến hành kiểm tra cú pháp các lệnh hợp ngữ trong mã nguồn, xem có phù hợp về cú pháp hay không, rồi mới tiến hành dịch các lệnh ra mã máy.

Hãy viết một chương trình kiểm tra cú pháp của 1 lệnh hợp ngữ MIPS bất kì (không làm với giả lệnh) như sau:

- Nhập vào từ bàn phím một dòng lệnh hợp ngữ. Ví dụ *beq s1,31,t4*
- Kiểm tra xem mã opcode có đúng hay không? Trong ví dụ trên, opcode là beq là hợp lệ thì hiện thị thông báo *"opcode: beq, hợp lệ"*
- Kiểm tra xem tên các toán hạng phía sau có hợp lệ hay không? Trong ví dụ trên, *toán hạng s1 là hợp lệ, 31 là không hợp lệ, t4 thì khỏi phải kiểm tra nữa vì toán hạng trước đã bị sai rồi*.

Gợi ý: nên xây dựng một cấu trúc chứa khuôn dạng của từng lệnh với tên lệnh, kiểu của toán hạng 1, toán hạng 2, toán hạng 3

Phân tích hướng giải quyết

Xây dựng một cấu trúc chứa khuôn dạng của từng lệnh với tên lệnh (opcode), kiểu toán hạng 1, kiểu toán hạng 2, kiểu toán hạng 3, các cấu trúc chứa tên thanh ghi và các ký tự in hoa và in thường.

Sau đó lần lượt kiểm tra xâu nhập vào theo từng toán hạng, kiểm tra từ opcode -> toán hạng 1 -> toán hạng 2 -> toán hạng 3 có tuân thủ theo các cấu trúc mà mình đặt ra hay không. Sau đó in kết quả ra màn hình.

Phân tích thuật toán

Data

```
.data
    inputMsg:   .asciiz "Hay nhap lenh hop ngu can kiem tra: "
    hopleOCMsg: .asciiz "Opcode: "
    hopleTHMsg: .asciiz "Toan hang: "
    hopleMsg:   .asciiz "Hop le.\n"
    restartMsg: .asciiz "Kiem tra tiep?(0 : Co | 1 : Khong) "
    errorMsg:   .asciiz "Lenh khong ton tai hoac sai cu phap!\n"
    endMsg:     .asciiz "\nLenh hop ngu da dung cu phap!\n"
    command:    .space  150
    opcode:     .space  15
    token:      .space  25
    number:     .space  20
    ident:      .space  50
    # Quy luat cua libray: opcode co do dai = 5 byte
    # moi lenh co 3 toan hang va chi co 4 loai la: thanh ghi = 1, hang so nguyen
=2, dinh danh = 3 hoac khong co = 0.
    khuondang:  .asciiz
"beq**123;or***111;xor**111;lui**120;jr***100;jal**300;addi*112;add**111;sub**111
;ori**112;and**111;beq**113;bne**113;j****300;nop**000;"
    NhomKiTu:   .asciiz
"qwertyuiopasdfghjklmnbvcxzQWERTYUIOPASDFGHJKLZXCVBNM0123456789_"
    thanhghi:   .asciiz "$zero
$at   $v0   $v1   $a0   $a1   $a2   $a3   $t0   $t1   $t2   $t3   $t4   $t5   $t6
  $t7   $s0   $s1   $s2   $s3   $s4   $s5   $s6   $s7   $t8   $t9   $k0   $k1   $
gp   $sp   $fp   $ra   $0    $1    $2    $3    $4    $5    $7    $8    $9    $10
  $11   $12   $13   $14   $15   $16   $17   $18   $19   $20   $21   $22   $21   $
22   $23   $24   $25   $26   $27   $28   $29   $30   $31   "
```

- Tạo một xâu **khuondang** gồm các chuỗi 8 bit được ngăn cách nhau bằng dấu phẩy với 5 bit đầu là **opcode** và 3 bit sau tương ứng với dạng của 3 toán hạng với (0 là không có, 1 là thanh ghi, 2 là số nguyên và 3 là nhãn).
- Xâu **NhomKiTu** là các chữ cái từ a đến z viết hoa và viết thường để kiểm tra nhãn.
- Xâu **thanhghi** là chuỗi các thanh ghi được ngăn cách với nhau bằng dấu space để kiểm tra tên thanh ghi có đúng hay không.

Chương trình con

inputData

```
inputData:  # Doc lenh nhap vao tu ban phim
    li      $v0, 4
    la      $a0, inputMsg
    syscall
    li      $v0, 8
    la      $a0, command        # chua dia chi cua lenh nhap vao
    li      $a1, 100
    syscall
    li      $t2, 0              # i
```

Giải thích: Đây là hàm hiển thị yêu cầu nhập lệnh để kiểm tra và lưu vào **command**.

saveOpcode

```
saveOpcode:
    la      $a1, opcode         # luu cac ki tu doc duoc vao opcode
    add     $t3, $a0, $t2       # dich bit
    add     $t4, $a1, $t2
    lb      $t1, 0($t3)         # doc tung ki tu cua command
    sb      $t1, 0($t4)
    beq     $t1, 32, done       # gap ki tu ' ' -> luu ki tu nay vao opcode de xu
ly
    beq     $t1, 0, done        # ket thuc chuoi command
    addi    $t2, $t2, 1
    j       saveOpcode
```

Giải thích: Đây là hàm đọc từng kí tự của Opcode từ **command** vừa được lưu cho tới khi gặp kí tự khoảng cách ‘ ’ thì dừng lại và lưu vào **opcode** sau đó chuyển qua phần xử lý Opcode.

Xử lý Opcode

```
done:
    li      $t7,-9
    la      $a2, khuondang
CheckOpcode:
    li      $t1, 0 # i
    li      $t2, 0 # j
    addi    $t7, $t7, 9         # buoc nhay = 9 de den vi tri opcode trong
khuondang
    add     $t1, $t1, $t7       # cong buoc nhay
```

```
    compare:
        add     $t3, $a2, $t1   # t3 tro thanh con tro cua khuondang
        lb      $s0, 0($t3)
        beq     $s0, 0, error   # khong tim thay opcode nao trong khuondang
        beq     $s0, 42, check  # gap ki tu '*' -> check xem opcode co giong nhau
tiep ko?.
        add     $t4, $a1, $t2
        lb      $s1, 0($t4)
        bne     $s0, $s1, CheckOpcode   # so sanh 2 ki tu. dung thi so sanh tiep,
sai thi nhay den phan tu chua khuon danh lenh tiep theo.
        addi    $t1, $t1, 1         # i+=1
        addi    $t2, $t2, 1         # j+=1
        j       compare
    check:
        add     $t4, $a1, $t2
        lb      $s1, 0($t4)
        bne     $s1, 32, check2     # neu ki tu tiep theo khong phai ' ' => lenh
khong hop le. chi co doan dau giong.
    checkContinue:
        add     $t9,$t9,$t2         # t9 = luu vi tri de xu ly token trong
command
        li      $v0, 4
        la      $a0, hopleOCMsg     # opcode hop le
        syscall
        li      $v0, 4
        la      $a0, opcode
        syscall
        li      $v0, 4
        la      $a0, hopleMsg
        syscall
        j       readToanHang1
    check2:     # neu ki tu tiep theo khong phai '\n' => lenh khong hop le. chi
co doan dau giong.
        bne     $s1, 10, error
        j       checkContinue
```

Giải thích:

- Sau khi lưu opcode ta bắt đầu lưu vị trí của thanh ghi t7 thành -9 để khi vào hàm **CheckOpcode** sẽ được cộng thêm 9 để bắt đầu check opcode đầu tiên. Các opcode cách nhau 9 byte nên phải cộng thêm 9 để có thể nhảy tới opcode tiếp theo để so sánh.
- Sau đó gán vị trí bước nhảy cho t3 để thành con trỏ trỏ tới opcode tiếp theo

- Các biến i và j được sử dụng để đếm vị trí hiện tại trong chuỗi opcode đang được kiểm tra và dùng để đếm vị trí hiện tại trong library của.

Xử lý toán hạng

```
readToanHang1:
    # xac dinh kieu toan hang trong khuondang
    # t7 dang chua vi tri khuon dang lenh trong khuondang
    li      $t1, 0
    addi    $t7, $t7, 5         # chuyen den vi tri toan hang 1 trong khuondang
    add     $t1, $a2, $t7       # a2 chua dia chi khuondang
    lb      $s0, 0($t1)
    addi    $s0, $s0, -48       # chuyen tu char -> int
    li      $t8, 1              # thanh ghi = 1
    beq     $s0, $t8, checkTokenReg
    li      $t8, 2              # hang so nguyen = 2
    beq     $s0, $t8, checkInt
    li      $t8, 3              # dinh danh = 3
    beq     $s0, $t8, checkIdent
    li      $t8, 0              # khong co toan hang = 0
    beq     $s0, $t8, checkNT
    j       end
checkTokenReg:
    la      $a0, command
    la      $a1, token          # luu ten thanh ghi vao token de so sanh
    li      $t1, 0
    li      $t2, -1
    addi    $t1, $t9, 0

    readToken:
        addi    $t1, $t1, 1         # i
        addi    $t2, $t2, 1         # j
        add     $t3, $a0, $t1
        add     $t4, $a1, $t2
        lb      $s0, 0($t3)
        add     $t9, $zero, $t1     # vi tri toan hang tiep theo trong command
        beq     $s0, 35, compare1   # gap dau '# '
        beq     $s0, 32, checkTokenReg  # gap dau ' '
        beq     $s0, 44, readTokenDone  # gap dau ','
        beq     $s0, 0, readTokenDone   # gap ki tu ket thuc
        sb      $s0, 0($t4)
        j       readToken
    readTokenDone:
        sb      $s0, 0($t4)         # luu them ',' vao de compare
        li      $t1, -1 # i
        li      $t2, -1 # j
```

```
        li      $t4, 0
        li      $t5, 0
        add     $t2, $t2, $k1
        la      $a1, token
        la      $a2, thanhghi
        j       compareToken
compareToken:
    addi    $t1, $t1, 1
    addi    $t2, $t2, 1
    add     $t4, $a1, $t1
    lb      $s0, 0($t4)
    beq     $s0, 0, end
    add     $t5, $a2, $t2
    lb      $s1, 0($t5)
    beq     $s1, 0, error
    beq     $s1, 32, checkLengthToken
    bne     $s0, $s1, jump
    j       compareToken

    checkLengthToken:
        beq     $s0, 44, compare1
        beq     $s0, 10, compare1
        j       compare2
    jump:
        addi    $k1,$k1,6
        j       readTokenDone
    compare1:
        la      $a0, hopleTHMsg         # opcode hop le
        syscall
        li      $v0, 4
        la      $a0, token
        syscall
        li      $v0, 4
        la      $a0, hopleMsg
        syscall
        addi    $v1, $v1, 1         # dem so toan hang da doc.
        li      $k1, 0              # reset buoc nhay
        beq     $v1, 1, readToanHang2
        beq     $v1, 2, readToanHang3
        j       end
    compare2:
        j       error
checkInt: # kiem tra co phai hang so nguyen hay ko
    la      $a0, command
```

```
    la      $a1, number         # luu day chu so vao number de so sanh tung chu
so co thuoc vao numberGroup hay khong.
    li      $t1, 0
    li      $t2, -1
    addi    $t1, $t9, 0

    readNumber:
        addi    $t1, $t1, 1     # i
        addi    $t2, $t2, 1     # j
        add     $t3, $a0, $t1
        add     $t4, $a1, $t2
        lb      $s0, 0($t3)
        add     $t9, $zero, $t1         # vi tri toan hang tiep theo trong
command
        beq     $s0, 35, compareNum1    # gap dau '# '
        beq     $s0, 32, checkInt       # gap dau ' '
        beq     $s0, 44, readNumberDone # gap dau ','
        beq     $s0, 0, readNumberDone  # gap ki tu ket thuc
        sb      $s0, 0($t4)
        j       readNumber
    readNumberDone:
        sb      $s0, 0($t4)         # luu them ',' vao de compare
        li      $t1, -1             # i
        li      $t4, 0
        la      $a1, number
        j       compareNumber
compareNumber:
    addi    $t1, $t1, 1
    add     $t4, $a1, $t1
    lb      $s0, 0($t4)
    beq     $s0, 0, end
    beq     $s0, 45, compareNumber  # bo dau '-'
    beq     $s0, 10, compareNum1
    beq     $s0, 44, compareNum1
    li      $t2, 48
    li      $t3, 57
    slt     $t5, $s0, $t2
    bne     $t5, $zero, compareNum2
    slt     $t5, $t3, $s0
    bne     $t5, $zero, compareNum2
    j       compareNumber

    compareNum1:
        la      $a0, hopleTHMsg
        syscall
```

```
        li      $v0, 4
        la      $a0, number
        syscall
        li      $v0, 4
        la      $a0, hopleMsg
        syscall
        addi    $v1, $v1, 1         # dem so toan hang da doc.
        li      $k1, 0              # reset buoc nhay
        beq     $v1, 1, readToanHang2
        beq     $v1, 2, readToanHang3
        j       end
    compareNum2:
        j       error
checkIdent:
    la      $a0, command
    la      $a1, ident      # luu ten thanh ghi vao indent de so sanh
    li      $t1, 0
    li      $t2, -1
    addi    $t1, $t9, 0

    readIndent:
        addi    $t1, $t1, 1     # i
        addi    $t2, $t2, 1     # j
        add     $t3, $a0, $t1
        add     $t4, $a1, $t2
        lb      $s0, 0($t3)
        add     $t9, $zero, $t1 # vi tri toan hang tiep theo trong command
        beq     $s0, 35, compareIdent1  # gap dau '# '
        beq     $s0, 32, checkIdent     # gap dau ' '
        beq     $s0, 44, readIdentDone  # gap dau ','
        beq     $s0, 0, readIdentDone   # gap ki tu ket thuc
        sb      $s0, 0($t4)
        j readIndent
    readIdentDone:
        sb      $s0, 0($t4)             # luu them ',' vao de compare

        loopj:
            li      $t1, -1             # i
            li      $t2, -1             # j
            li      $t4, 0
            li      $t5, 0
            add     $t1, $t1, $k1
            la      $a1, ident
            la      $a2, NhomKiTu
            j       compareIdent
```

```
compareIdent:
    addi    $t1,$t1,1
    add     $t4, $a1, $t1
    lb      $s0, 0($t4)
    beq     $s0, 0, end
    beq     $s0, 10, compareIdent1
    beq     $s0, 44, compareIdent1

    loop:
        addi    $t2,$t2,1
        add     $t5, $a2, $t2
        lb      $s1, 0($t5)
        beq     $s1, 0, compareIdent2
        beq     $s0, $s1, jumpIdent     # so sanh ki tu tiep theo trong ident
        j       loop                    # tiep tuc so sanh ki tu tiep theo trong
NhomKiTu
    jumpIdent:
        addi    $k1,$k1,1
        j       loopj

    compareIdent1:
        la      $a0, hopleTHMsg             # opcode hop le
        syscall
        li      $v0, 4
        la      $a0, ident
        syscall
        li      $v0, 4
        la      $a0, hopleMsg
        syscall
        addi    $v1, $v1, 1             # dem so toan hang da doc.
        li      $k1, 0                  # reset buoc nhay
        beq     $v1, 1, readToanHang2
        beq     $v1, 2, readToanHang3
        j end
    compareIdent2:
        j error
checkNT:
    la      $a0, command
    li      $t1, 0
    li      $t2, 0
    addi    $t1, $t9, 0
    add     $t2, $a0, $t1
    lb      $s0, 0($t2)
    addi    $v1, $v1, 1         # dem so toan hang da doc.
    li      $k1, 0              # reset buoc nhay
```

```
    beq     $v1, 1, readToanHang2
    beq     $v1, 2, readToanHang3
```

- **readToanHang1**: kiểm tra bit số 5 xem là 1,2,3 sau đó tiến hành kiếm tra tương ứng Nếu là 0 thì nhảy đến nhãn **checkNT**:
    - 1 thì nhảy đến nhãn **checkTokenReg**:
    - 2 thì nhảy đến nhãn **checkHSN**:
    - 3 thì nhảy đến nhãn **checkIdent**:
- **checkTokenReg**: lưu xâu vào mảng token để so sánh lần lượt với các thanh ghi ở trong mảng thanh ghi.vẫn áp dụng thuật toán so sánh xâu nếu gặp dấu "," ngừng so sánh nếu gặp dấu space bỏ qua.
- **checkInt**:lưu xâu vào mảng number xem nó có giống trong numberGroup hay không nếu số không nằm trong đoạn từ 0 đến 9 thì nhảy đến nhãn kết thúc và số không hợp lệ.
- **checkIdent**:lưu nhãn vào để so sánh xem nó có giống trong charGroup hay không nếu không trùng với các kí tự trong **NhomKiTu** kèm theo một biến $v1 để kiểm tra xem đang là toán hạng thứ mấy để nhảy đến các nhãn **readToanHang2**: hoặc **readToanHang3**:

Xử lý toán hạng 2,3

- Các chương trình con còn lại đều có cơ chế hoạt động giống như khi xử lý toán hạng 1. Các chương trình cũng đã có chức năng bỏ qua kí tự khoảng cách để kiểm tra toán hạng tiếp theo.

Chương trình restart

```
continue: # lap lai chuong trinh.
    li      $v0, 4
    la      $a0, restartMsg
    syscall
    li      $v0, 5
    syscall
    add     $t0, $v0, $zero
    beq     $t0, $zero, resetAll
    j       endMain
```

- Chương trình xử lý reset chương trình về trạng thái ban đầu để có thể kiểm tra lệnh hợp ngữ mới

Mã nguồn

```
.data
    inputMsg:   .asciiz "Hay nhap lenh hop ngu can kiem tra: "
    hopleOCMsg: .asciiz "Opcode: "
    hopleTHMsg: .asciiz "Toan hang: "
    hopleMsg:   .asciiz "Hop le.\n"
    restartMsg: .asciiz "Kiem tra tiep?(0 : Co | 1 : Khong) "
    errorMsg:   .asciiz "Lenh khong ton tai hoac sai cu phap!\n"
    endMsg:     .asciiz "\nLenh hop ngu da dung cu phap!\n"
    command:    .space  150
    opcode:     .space  15
    token:      .space  25
    number:     .space  20
    ident:      .space  50
    # Quy luat cua libray: opcode co do dai = 5 byte
    # moi lenh co 3 toan hang va chi co 4 loai la: thanh ghi = 1, hang so nguyen
=2, dinh danh = 3 hoac khong co = 0.
    khuondang:  .asciiz
"beq**123;or***111;xor**111;lui**120;jr***100;jal**300;addi*112;add**111;sub**111
;ori**112;and**111;beq**113;bne**113;j****300;nop**000;"
    NhomKiTu:   .asciiz
"qwertyuiopasdfghjklmnbvcxzQWERTYUIOPASDFGHJKLZXCVBNM0123456789_"
    thanhghi:   .asciiz "$zero
$at   $v0   $v1   $a0   $a1   $a2   $a3   $t0   $t1   $t2   $t3   $t4   $t5   $t6
  $t7   $s0   $s1   $s2   $s3   $s4   $s5   $s6   $s7   $t8   $t9   $k0   $k1   $
gp   $sp   $fp   $ra   $0    $1    $2    $3    $4    $5    $7    $8    $9    $10
  $11   $12   $13   $14   $15   $16   $17   $18   $19   $20   $21   $22   $21   $
22   $23   $24   $25   $26   $27   $28   $29   $30   $31   "

.text

inputData:  # Doc lenh nhap vao tu ban phim
    li      $v0, 4
    la      $a0, inputMsg
    syscall
    li      $v0, 8
    la      $a0, command        # chua dia chi cua lenh nhap vao
    li      $a1, 100
    syscall
    li      $t2, 0              # i
saveOpcode:
    la      $a1, opcode         # luu cac ki tu doc duoc vao opcode
    add     $t3, $a0, $t2       # dich bit
    add     $t4, $a1, $t2
```

```
    lb      $t1, 0($t3)         # doc tung ki tu cua command
    sb      $t1, 0($t4)
    beq     $t1, 32, done       # gap ki tu ' ' -> luu ki tu nay vao opcode de xu
ly
    beq     $t1, 0, done        # ket thuc chuoi command
    addi    $t2, $t2, 1
    j       saveOpcode

#<--xu ly opcode-->
done:
    li      $t7,-9
    la      $a2, khuondang
CheckOpcode:
    li      $t1, 0 # i
    li      $t2, 0 # j
    addi    $t7, $t7, 9         # buoc nhay = 9 de den vi tri opcode trong
khuondang
    add     $t1, $t1, $t7       # cong buoc nhay

    compare:
        add     $t3, $a2, $t1   # t3 tro thanh con tro cua khuondang
        lb      $s0, 0($t3)
        beq     $s0, 0, error   # khong tim thay opcode nao trong khuondang
        beq     $s0, 42, check  # gap ki tu '*' -> check xem opcode co giong nhau
tiep ko?.
        add     $t4, $a1, $t2
        lb      $s1, 0($t4)
        bne     $s0, $s1, CheckOpcode   # so sanh 2 ki tu. dung thi so sanh tiep,
sai thi nhay den phan tu chua khuon danh lenh tiep theo.
        addi    $t1, $t1, 1         # i+=1
        addi    $t2, $t2, 1         # j+=1
        j       compare
    check:
        add     $t4, $a1, $t2
        lb      $s1, 0($t4)
        bne     $s1, 32, check2     # neu ki tu tiep theo khong phai ' ' => lenh
khong hop le. chi co doan dau giong.
    checkContinue:
        add     $t9,$t9,$t2         # t9 = luu vi tri de xu ly token trong
command
        li      $v0, 4
        la      $a0, hopleOCMsg     # opcode hop le
        syscall
        li      $v0, 4
        la      $a0, opcode
```

```
        syscall
        li      $v0, 4
        la      $a0, hopleMsg
        syscall
        j       readToanHang1
    check2:     # neu ki tu tiep theo khong phai '\n' => lenh khong hop le. chi
co doan dau giong.
        bne     $s1, 10, error
        j       checkContinue
# <!--ket thuc xu ly opcode -->

#<--xu li toan hang-->
readToanHang1:
    # xac dinh kieu toan hang trong khuondang
    # t7 dang chua vi tri khuon dang lenh trong khuondang
    li      $t1, 0
    addi    $t7, $t7, 5         # chuyen den vi tri toan hang 1 trong khuondang
    add     $t1, $a2, $t7       # a2 chua dia chi khuondang
    lb      $s0, 0($t1)
    addi    $s0, $s0, -48       # chuyen tu char -> int
    li      $t8, 1              # thanh ghi = 1
    beq     $s0, $t8, checkTokenReg
    li      $t8, 2              # hang so nguyen = 2
    beq     $s0, $t8, checkInt
    li      $t8, 3              # dinh danh = 3
    beq     $s0, $t8, checkIdent
    li      $t8, 0              # khong co toan hang = 0
    beq     $s0, $t8, checkNT
    j       end

#<--check Token Register-->
checkTokenReg:
    la      $a0, command
    la      $a1, token          # luu ten thanh ghi vao token de so sanh
    li      $t1, 0
    li      $t2, -1
    addi    $t1, $t9, 0

    readToken:
        addi    $t1, $t1, 1         # i
        addi    $t2, $t2, 1         # j
        add     $t3, $a0, $t1
        add     $t4, $a1, $t2
        lb      $s0, 0($t3)
        add     $t9, $zero, $t1     # vi tri toan hang tiep theo trong command
```

```
        beq     $s0, 35, compare1   # gap dau '# '
        beq     $s0, 32, checkTokenReg  # gap dau ' '
        beq     $s0, 44, readTokenDone  # gap dau ','
        beq     $s0, 0, readTokenDone   # gap ki tu ket thuc
        sb      $s0, 0($t4)
        j       readToken
    readTokenDone:
        sb      $s0, 0($t4)         # luu them ',' vao de compare
        li      $t1, -1 # i
        li      $t2, -1 # j
        li      $t4, 0
        li      $t5, 0
        add     $t2, $t2, $k1
        la      $a1, token
        la      $a2, thanhghi
        j       compareToken
compareToken:
    addi    $t1, $t1, 1
    addi    $t2, $t2, 1
    add     $t4, $a1, $t1
    lb      $s0, 0($t4)
    beq     $s0, 0, end
    add     $t5, $a2, $t2
    lb      $s1, 0($t5)
    beq     $s1, 0, error
    beq     $s1, 32, checkLengthToken
    bne     $s0, $s1, jump
    j       compareToken

    checkLengthToken:
        beq     $s0, 44, compare1
        beq     $s0, 10, compare1
        j       compare2
    jump:
        addi    $k1,$k1,6
        j       readTokenDone
    compare1:
        la      $a0, hopleTHMsg         # opcode hop le
        syscall
        li      $v0, 4
        la      $a0, token
        syscall
        li      $v0, 4
        la      $a0, hopleMsg
        syscall
```

```
        addi    $v1, $v1, 1         # dem so toan hang da doc.
        li      $k1, 0              # reset buoc nhay
        beq     $v1, 1, readToanHang2
        beq     $v1, 2, readToanHang3
        j       end
    compare2:
        j       error
#<!--ket thuc check Token Register-->

#<--check toan hang la hang so nguyen-->
checkInt: # kiem tra co phai hang so nguyen hay ko
    la      $a0, command
    la      $a1, number         # luu day chu so vao number de so sanh tung chu
so co thuoc vao numberGroup hay khong.
    li      $t1, 0
    li      $t2, -1
    addi    $t1, $t9, 0

    readNumber:
        addi    $t1, $t1, 1     # i
        addi    $t2, $t2, 1     # j
        add     $t3, $a0, $t1
        add     $t4, $a1, $t2
        lb      $s0, 0($t3)
        add     $t9, $zero, $t1         # vi tri toan hang tiep theo trong
command
        beq     $s0, 35, compareNum1    # gap dau '# '
        beq     $s0, 32, checkInt       # gap dau ' '
        beq     $s0, 44, readNumberDone # gap dau ','
        beq     $s0, 0, readNumberDone  # gap ki tu ket thuc
        sb      $s0, 0($t4)
        j       readNumber
    readNumberDone:
        sb      $s0, 0($t4)         # luu them ',' vao de compare
        li      $t1, -1             # i
        li      $t4, 0
        la      $a1, number
        j       compareNumber
compareNumber:
    addi    $t1, $t1, 1
    add     $t4, $a1, $t1
    lb      $s0, 0($t4)
    beq     $s0, 0, end
    beq     $s0, 45, compareNumber  # bo dau '-'
    beq     $s0, 10, compareNum1
```

```
    beq     $s0, 44, compareNum1
    li      $t2, 48
    li      $t3, 57
    slt     $t5, $s0, $t2
    bne     $t5, $zero, compareNum2
    slt     $t5, $t3, $s0
    bne     $t5, $zero, compareNum2
    j       compareNumber

    compareNum1:
        la      $a0, hopleTHMsg
        syscall
        li      $v0, 4
        la      $a0, number
        syscall
        li      $v0, 4
        la      $a0, hopleMsg
        syscall
        addi    $v1, $v1, 1         # dem so toan hang da doc.
        li      $k1, 0              # reset buoc nhay
        beq     $v1, 1, readToanHang2
        beq     $v1, 2, readToanHang3
        j       end
    compareNum2:
        j       error
#<!--ket thuc check toan hang la hang so nguyen-->

#<--check Indent-->
checkIdent:
    la      $a0, command
    la      $a1, ident      # luu ten thanh ghi vao indent de so sanh
    li      $t1, 0
    li      $t2, -1
    addi    $t1, $t9, 0

    readIndent:
        addi    $t1, $t1, 1     # i
        addi    $t2, $t2, 1     # j
        add     $t3, $a0, $t1
        add     $t4, $a1, $t2
        lb      $s0, 0($t3)
        add     $t9, $zero, $t1 # vi tri toan hang tiep theo trong command
        beq     $s0, 35, compareIdent1  # gap dau '# '
        beq     $s0, 32, checkIdent     # gap dau ' '
        beq     $s0, 44, readIdentDone  # gap dau ','
```

```
        beq     $s0, 0, readIdentDone   # gap ki tu ket thuc
        sb      $s0, 0($t4)
        j readIndent
    readIdentDone:
        sb      $s0, 0($t4)             # luu them ',' vao de compare

        loopj:
            li      $t1, -1             # i
            li      $t2, -1             # j
            li      $t4, 0
            li      $t5, 0
            add     $t1, $t1, $k1
            la      $a1, ident
            la      $a2, NhomKiTu
            j       compareIdent
compareIdent:
    addi    $t1,$t1,1
    add     $t4, $a1, $t1
    lb      $s0, 0($t4)
    beq     $s0, 0, end
    beq     $s0, 10, compareIdent1
    beq     $s0, 44, compareIdent1

    loop:
        addi    $t2,$t2,1
        add     $t5, $a2, $t2
        lb      $s1, 0($t5)
        beq     $s1, 0, compareIdent2
        beq     $s0, $s1, jumpIdent     # so sanh ki tu tiep theo trong ident
        j       loop                    # tiep tuc so sanh ki tu tiep theo trong
NhomKiTu
    jumpIdent:
        addi    $k1,$k1,1
        j       loopj

    compareIdent1:
        la      $a0, hopleTHMsg             # opcode hop le
        syscall
        li      $v0, 4
        la      $a0, ident
        syscall
        li      $v0, 4
        la      $a0, hopleMsg
        syscall
        addi    $v1, $v1, 1             # dem so toan hang da doc.
```

```
        li      $k1, 0                  # reset buoc nhay
        beq     $v1, 1, readToanHang2
        beq     $v1, 2, readToanHang3
        j end
    compareIdent2:
        j error
#<!--ket thuc check Indent-->

#<--kiem tra khong co toan hang-->
checkNT:
    la      $a0, command
    li      $t1, 0
    li      $t2, 0
    addi    $t1, $t9, 0
    add     $t2, $a0, $t1
    lb      $s0, 0($t2)
    addi    $v1, $v1, 1         # dem so toan hang da doc.
    li      $k1, 0              # reset buoc nhay
    beq     $v1, 1, readToanHang2
    beq     $v1, 2, readToanHang3
#<!--ket thuc kiem tra khong co toan hang-->

#<--check Token Register 2-->
readToanHang2:
    # xac dinh kieu toan hang trong khuondang
    # t7 dang chua vi tri khuon dang lenh trong khuondang
    li      $t1, 0
    la      $a2,khuondang
    addi    $t7, $t7, 1         # chuyen den vi tri toan hang 2 trong kuondang
    add     $t1, $a2, $t7       # a2 chua dia chi khuondang
    lb      $s0, 0($t1)
    addi    $s0,$s0,-48         # chuyen tu char -> int
    li      $t8, 1              # thanh ghi = 1
    beq     $s0, $t8, checkTokenReg
    li      $t8, 2              # hang so nguyen = 2
    beq     $s0, $t8, checkInt
    li      $t8, 3              # dinh danh = 3
    beq     $s0, $t8, checkIdent
    li      $t8, 0              # khong co toan hang = 0
    beq     $s0, $t8, checkNT
    j       end
#<!--ket thuc check Token Register 2-->

#<--check Token Register 3-->
readToanHang3:
```

```
    # xac dinh kieu toan hang trong khuondang
    # t7 dang chua vi tri khuon dang lenh trong khuondang
    li      $t1, 0
    la      $a2, khuondang
    addi    $t7, $t7, 1         # chuyen den vi tri toan hang 3 trong khuondang
    add     $t1, $a2, $t7       # a2 chua dia chi khuondang
    lb      $s0, 0($t1)
    addi    $s0,$s0,-48         # chuyen tu char -> int
    li      $t8, 1              # thanh ghi = 1
    beq     $s0, $t8, checkTokenReg
    li      $t8, 2              # hang so nguyen = 2
    beq     $s0, $t8, checkInt
    li      $t8, 3              # dinh danh = 3
    beq     $s0, $t8, checkIdent
    li      $t8, 0              # khong co toan hang = 0
    beq     $s0, $t8, checkNT
    j       end
#<!--ket thuc check Token Register 3-->

#<--check Token Register 3-->
continue: # lap lai chuong trinh.
    li      $v0, 4
    la      $a0, restartMsg
    syscall
    li      $v0, 5
    syscall
    add     $t0, $v0, $zero
    beq     $t0, $zero, resetAll
    j       endMain
resetAll:
    li  $v0, 0
    li  $v1, 0
    li  $a0, 0
    li  $a1, 0
    li  $a2, 0
    li  $a3, 0
    li  $t0, 0
    li  $t1, 0
    li  $t2, 0
    li  $t3, 0
    li  $t4, 0
    li  $t5, 0
    li  $t6, 0
    li  $t7, 0
    li  $t8, 0
```

```
    li  $t9, 0
    li  $s0, 0
    li  $s1, 0
    li  $s2, 0
    li  $s3, 0
    li  $s4, 0
    li  $s5, 0
    li  $s6, 0
    li  $s7, 0
    li  $k0, 0
    li  $k1, 0
    j   inputData
error:
    li  $v0, 4
    la  $a0, errorMsg
    syscall
    j   continue
end:
    li  $v0, 4
    la  $a0, endMsg
    syscall
    j   continue
endMain:
```

Kết quả chạy

Trường hợp đúng lệnh hợp ngữ:

```
Hay nhap lenh hop ngu can kiem tra: addi $t1,$t2,5
Opcode: addi Hop le.
Toan hang: $t1,Hop le.
Toan hang: $t2,Hop le.
Toan hang: 5
Hop le.

Lenh hop ngu da dung cu phap!
Kiem tra tiep?(0 : Co | 1 : Khong)
```

Lệnh jump với nhãn

```
Hay nhap lenh hop ngu can kiem tra: j exit
Opcode: j  o Hop le.
Toan hang: exit
Hop le.

Lenh hop ngu da dung cu phap!
Kiem tra tiep?(0 : Co | 1 : Khong)
```

Trường hợp có ký tự khoảng trắng/không có ký tự khoảng trắng:

```
Hay nhap lenh hop ngu can kiem tra: addi $t2,      $t7,   5
Opcode: addi Hop le.
Toan hang: $t2,Hop le.
Toan hang: $t7,Hop le.
Toan hang: 5
Hop le.

Lenh hop ngu da dung cu phap!
Kiem tra tiep?(0 : Co | 1 : Khong)
```

Các trường hợp lệnh sai:

Lệnh sai opcode

```
Hay nhap lenh hop ngu can kiem tra: addo $t1,$t2,$zero
Lenh khong ton tai hoac sai cu phap!
Kiem tra tiep?(0 : Co | 1 : Khong)
```

Lệnh có các ký tự đặc biệt

```
Hay nhap lenh hop ngu can kiem tra: j :{}{}""++___
Opcode: j do Hop le.
Lenh khong ton tai hoac sai cu phap!
Kiem tra tiep?(0 : Co | 1 : Khong)
```

Lệnh sai kiểu

```
Hay nhap lenh hop ngu can kiem tra: li $s1, $s2, $s3
Lenh khong ton tai hoac sai cu phap!
Kiem tra tiep?(0 : Co | 1 : Khong)
```

Lệnh thiếu dấu ,

```
Hay nhap lenh hop ngu can kiem tra: li $t1 1
Lenh khong ton tai hoac sai cu phap!
Kiem tra tiep?(0 : Co | 1 : Khong)
```