ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
KHOA KỸ THUẬT MÁY TÍNH

IT3280

THỰC HÀNH KIẾN TRÚC MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN CUỐI KÌ

MÔN THỰC HÀNH KIẾN TRÚC MÁY TÍNH

MÃ LỚP: 147789 - KỲ 2023.2 - NĂM HỌC 2023 - 2024

NHÓM 8

Trần Quang Huy	20226109	Huy.TQ226109@sis.hust.edu.vn
Ngô Mạnh Hùng	20226083	Hung.NM226083@sis.hust.edu.vn

GIẢNG VIÊN HƯỚNG DẪN: **ThS. Lê Bá Vui**

Hà Nội — 6.2024

Mục lục

DANH SÁCH HÌNH VẼ

Chương 1

CURIOSITY MARSBOT

1.1 ĐỀ BÀI

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:

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 với 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 đến khi kết thúc lộ trình ngược.

Đặc điểm:

- Marsbot được lập trình để nhớ lại toàn bộ lịch sử các mã điều khiển và khoảng thời gian giữa các lần đổi mã.
- Nhờ đó, Marsbot 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:

Enter Kết thúc nhập mã và yêu cầu Marsbot thực thi.
Delete Xóa toàn bộ mã điều khiển đang nhập.
Space Lặp lại lệnh đã thực hiện trước đó. Tiếp tục với yêu cầu này.

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.

1.2 ĐỊNH HƯỚNG CÁCH LÀM

1. Khi người dùng nhập 1 ký tự vào Digital Lab Sim, tạo ra 1 ngắt để lưu ký tự vào bộ nhớ, tạo nên mã điều khiển.
2. Kiểm tra xem ký tự được nhập trong KeyBoard and Display MMIO. Khi nhập ký tự enter, nếu có lỗi, chương trình sẽ in ra thông báo lỗi.
3. Lần lượt kiểm tra xem mã điều khiển được nhập vào có trùng khớp với các đoạn mã đã quy định sẵn không. Nếu không, thông báo đoạn mã đó bị lỗi. Ngược lại, thực hiện theo thao tác đã quy định sẵn.
4. Nếu mã được nhập vào là mã để rẽ trái (phải), lưu tọa độ x, y và góc trước khi rẽ vào lần lượt 03 mảng số nguyên (x_history, y_history, a_history)
5. In ra màn hình mã điều khiển đã nhập và xóa khỏi bộ nhớ. Nếu mã sai, in ra thông báo lỗi.

MIPS CODE EXPLANATION

1.2.1 Dữ liệu

Giải thích

Khai báo các giá trị cần sử dụng. 3 mảng số nguyên x_history, y_history, a_history lần lượt lưu trữ các thông tin về x,y và góc alpha

```
.data
    # mang luu tru thong tin vi tri x,y va goc a
    x_history:      .word 0 : 16
    y_history:      .word 0 : 16
    a_history:      .word 0 : 16
    l_history:      .word 4     # history length

    a_current:      .word 0     # gia tri goc a khoi tao

    isGoing:        .word 0
    isTracking:     .word 0

    inputCode:      .space 8    # input command code
    code_history:       .space 64   # history code
    inputLength:            .word 0     # input command length

    GO_CODE:        .asciiz "1b4"
    STOP_CODE:      .asciiz "c68"
    GOLEFT_CODE:        .asciiz "444"
    GORIGHT_CODE:   .asciiz "666"
    TRACK_CODE:     .asciiz "dad"
    UNTRACK_CODE:       .asciiz "cbc"
    REVERT: .asciiz "999"
    WRONG_CODE:     .asciiz "Wrong control code"
```

Listing 1.1: MIPS Assembly Example

1.2.2 SetStartHeading

Giải thích

Đây là hàm khởi tạo và cập nhật lịch sử các biến x, y và a. Cụ thể, nó đặt a thành 90, thực hiện một phép xoay thông qua hàm `ROTATE`, và cập nhật lịch sử

```
    lw   $t7, l_history       # l_history += 4
    addi     $t7, $zero, 4        # luu thong tin x = 0; y = 0; a
       = 90
    sw   $t7, l_history

    li   $t7, 90
    sw   $t7, a_current   # goc a = 90 do
    jal ROTATE
    nop

    sw   $t7, a_history + 4   # a_history[1] = 90
                    # x_history[1] =0
                    # y_history[1] = 0
    j    waitForKey
```

1.2.3 Phần Loại Bỏ Mã

Giải thích

Mục đích tổng thể của hàm này là để đặt lại bộ đệm `inputCode` bằng cách đặt mỗi byte thành `'\0'` và đặt lại `inputLength` về 0,đảm bảo rằng bộ đệm được xóa sạch để nhận đầu vào mới.

```
resetInput:
        jal removeCode
        nop
removeCode:
    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, inputLength             # $t3 = inputLength
    addi     $t1, $zero, -1       # $t1 = -1 = i

removeCode_loop:
    addi     $t1, $t1, 1          # i++
    sb   $zero, inputCode($t1)        # inputCode[i] = '\0'

    bne $t1, $t3, removeCode_loop    # if $t1 <=3 resetInput loop
```

```
    nop

    sw  $zero, inputLength      # reset inputLength = 0

removeCode_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
```

1.2.4 Phần lặp lại mã

Giải thích

Mục đích tổng thể của hàm này là để lặp lại lệnh cuối cùng được nhập bởi người dùng khi phát hiện được phím cách. Nó thực hiện điều này bằng cách sao chép lệnh trước đó từ code_history sang inputCode và cập nhật độ dài đầu vào tương ứng.

```
repeat:#khi nguoi dung nhap Space se in ra cau lenh truoc do
    addi    $sp, $sp, 4 # back up
        sw  $t0, 0($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  $s0, 0($sp)
        addi    $sp, $sp, 4
        sw  $s1, 0($sp)


        la $s0, code_history
        lb $t0,0($s0)
        beq $t0,$0,waitForKey
        nop
        beq $t0,$0,waitForKey
        li $t0,3
        sb $t0, inputLength
        la $s0, code_history
        la $s1, inputCode
        li $t1,0

```

```
repeat_loop:
        add $t2, $s0,$t1
        lb $t2,0($t2)
        add $t3, $s1,$t1
        sb $t2,0($t3)
        beq $t1,$t0,end_repeat_loop
        add $t1,$t1,1
        j repeat_loop
end_repeat_loop:
    lw  $s1, 0($sp)          # restore backup
    addi    $sp, $sp, -4
    lw  $s0, 0($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  $t0, 0($sp)
    addi    $sp, $sp, -4
    j CheckKeyCode
```

1.2.5 Phần so sánh chuỗi

Giải thích

Mục đích của hàm strcmp là để so sánh ba ký tự đầu tiên của chuỗi inputCode với ba ký tự đầu tiên của một chuỗi khác được trỏ tới bởi $s3$. Nếu chúng bằng nhau, hàm sẽ đặt $t0$ thành 1. Nếu chúng không bằng nhau, $t0$ sẽ vẫn giữ giá trị 0

```
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)

    li  $t0,0       # $t0  = 0
    li  $t1,0       # $t1 = i = 0

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

    lb  $t2, inputCode($t1)     # $t2 = inputCode[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          # i++
    j   strcmp_loop

strcmp_equal: add   $t0, $zero, 1       # $t0 = 1

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
```

1.2.6 Lưu trữ đường đi

Giải thích

Mục đích của hàm là lưu trữ thông tin vị trí x,y và góc alpha mỗi lần robot di chuyển

```
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
```

```
22
23      lw  $t3, l_history      # $t3 = l_history
24      addi    $t3,$t3,4
25      sw  $s1, x_history($t3)     # store: x, y, alpha
26      sw  $s2, y_history($t3)
27      sw  $s4, a_history($t3)
28
29      sw  $t3, l_history
30
31      lw  $s4, 0($sp)         # restore backup
32      addi    $sp, $sp, -4
33      lw  $s3, 0($sp)
34      addi    $sp, $sp, -4
35      lw  $s2, 0($sp)
36      addi    $sp, $sp, -4
37      lw  $s1, 0($sp)
38      addi    $sp, $sp, -4
39      lw  $t4, 0($sp)
40      addi    $sp, $sp, -4
41      lw  $t3, 0($sp)
42      addi    $sp, $sp, -4
43      lw  $t2, 0($sp)
44      addi    $sp, $sp, -4
45      lw  $t1, 0($sp)
46      addi    $sp, $sp, -4
47
48  saveHistory_end: jr $ra
```

1.2.7 Save Code

Giải thích

Lưu trữ thông tin của các input nhập vào, phù hợp để sau này thực hiện các hàm như repeat hay revert

```
1   save_code:  addi    $sp, $sp, 4 # back up
2           sw  $t0, 0($sp)
3           addi    $sp, $sp, 4
4           sw  $t1, 0($sp)
5           addi    $sp, $sp, 4
6           sw  $t2, 0($sp)
7           addi    $sp, $sp, 4
8           sw  $t3, 0($sp)
9           addi    $sp, $sp, 4
10          sw  $s0, 0($sp)
11          addi    $sp, $sp, 4
12          sw  $s1, 0($sp)
13
14
15
16          la $t0,inputLength
17          lw $t0,inputLength
```

```
        la $s0, inputCode
        la $s1, code_history
        li $t1,0        # i =0
save_code_loop:
        add $t2, $s0,$t1
        lb $t2,0($t2)       # $t2 = inputCode[i]
        add $t3, $s1,$t1
        sb $t2,0($t3)       # code_history[i] = inputCode[i]
        beq $t1,$t0,end_save_code_loop
        add $t1,$t1,1
        j save_code_loop
end_save_code_loop:



    lw  $s1, 0($sp)         # restore backup
    addi    $sp, $sp, -4
    lw  $s0, 0($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  $t0, 0($sp)
    addi    $sp, $sp, -4
    jr $ra
```

1.2.8 Check Key Code

Giải thích

Kiểm tra mã điều khiển có hợp lệ về độ dài và khớp với một trong các mã đã được quy ước

```
CheckKeyCode:
    lw  $s2, inputLength            # inputLength != 3 ->
       invalid code
    bne $s2, 3, printErrorMsg

    la  $s3, GO_CODE
    jal strcmp
    beq $t0, 1, case_go

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

    la  $s3, GOLEFT_CODE
    jal strcmp
    beq $t0, 1, case_turnLeft

```

```
    la   $s3, GORIGHT_CODE
    jal strcmp
    beq $t0, 1, case_turnRight

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

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

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

    j    printErrorMsg

    switch:
    case_go:     jal save_code
             j    go
    case_stop:   jal save_code
             j    stop
    case_turnLeft:  jal save_code
             j    turnLeft
    case_turnRight: jal save_code
             j    turnRight
    case_track: jal save_code
             j    track
    case_untrack:    jal save_code
             j    untrack
    case_goBackWard:jal save_code
             j    goBackward
    default:
```

1.3 GIẢI THÍCH CÁC HÀM CHỨC NĂNG ĐIỀU KHIỂN

GO, STOP

Giải thích

Điều khiển robot chuyển động hoặc dừng lại bằng cách thay đổi giá trị logic cổng MOVING, set isGoing bằng 0 hoặc 1 tương ứng

```
    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          # isGoing = 0
    sw  $t7, isGoing

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

    jr  $ra

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, isGoing      # isGoing = 0

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

    jr $ra
```

turnRight, turnLeft

Giải thích

- Cả hai hàm đều đảm bảo rằng robot đã dừng lại và không được theo dõi.
- Sau đó, cập nhật hướng của robot dựa trên hướng quay.
- Sau khi cập nhật hướng, ghi lại vị trí và hướng mới vào lịch sử.
- Cuối cùng, chúng tiếp tục theo dõi và di chuyển nếu chúng đã được kích hoạt trước đó

```
turnRight:
    lw  $t7, isGoing
    lw  $s0, isTracking

    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:
    lw  $t7, isGoing
    lw  $s0, isTracking

    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
```

TRACK, UNTRACK

Giải thích

Để lại dấu hoặc tắt để lại dấu bằng cách thay đổi logic cổng LEAVETRACK 0 hoặc 1, set isTracking bằng 0 hoặc 1 tương ứng

```
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, isTracking

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

    jr $ra

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, isTracking

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

    jr  $ra
```

ROTATE

1.3.1 Giải thích

Hàm ROTATE được dùng để cập nhật hướng của robot bằng cách lấy hướng hiện tại từ a_current và lưu trữ nó vào cổng HEADING, cho phép robot thay đổi hướng di chuyển của mình

```
    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

    jr  $ra
```

GoBackward

Giải thích

- Hàm goBackward điều phối việc di chuyển ngược lại của robot đến một điểm quay trước đó được lưu trữ trong lịch sử.
- Hàm goBackward_turn: Sau khi đi ngược lại, hàm tính toán hướng ngược lại của hướng của robot (a_current) bằng cách lấy hướng cuối cùng từ lịch sử, cộng thêm 180 độ vào đó và cập nhật a_current tương ứng. Sau đó, nó gọi hàm ROTATE để cập nhật hướng.
- goBackward_toTurningPoint: Lấy các tọa độ x và y từ lịch sử để xác định điểm quay trước đó, và kiểm tra các tọa độ x và y hiện tại cho đến khi chúng khớp với các tọa độ của điểm quay trong lịch sử
- Nếu điểm quay đã được đạt tới (l_history == 4), hàm sẽ kết thúc. Nếu không, nó giảm l_history và gọi đệ quy goBackward_turn để tiếp tục đi ngược lại cho đến khi đạt đến điểm quay.
- Một khi robot đạt đến điểm quay, nó sẽ gọi hàm STOP, đặt lại hướng hiện tại (a_current) và gọi hàm ROTATE để căn chỉnh với hướng mới. Sau đó, nó đặt lại l_history thành 4, cho biết rằng robot đã hoàn thành quá trình đi ngược lại đến điểm quay

```
goBackward:
    li  $t7, IN_ADDRESS_HEXA_KEYBOARD   # Disable interrupts
       when going backward
        sb  $zero, 0($t7)

    lw  $s5, l_history          # $s5 = length history
    jal UNTRACK
    jal GO

goBackward_turn:

    lw  $s6, a_history($s5)     # $s6 = a_history[l_history]
    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]
    get_x:
```

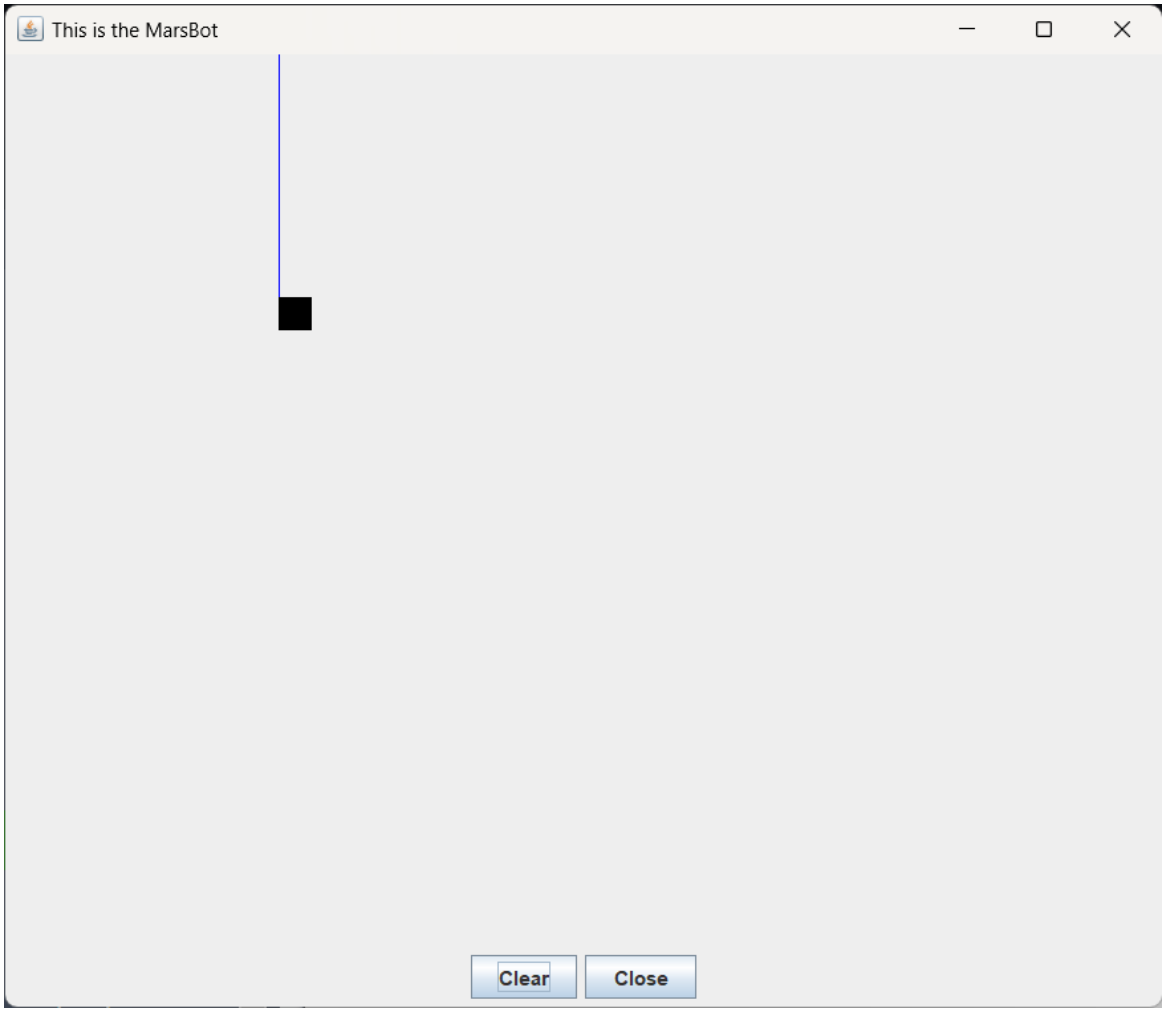
```
    li   $t8, WHEREX      # $t8 = x_current
    lw   $t8, 0($t8)
    bne $t8, $t9, get_x      # x_current == x_history[i]
    nop              # -> get y

    lw   $t9, y_history($s5)      # $t9 = y_history[i]
    get_Y:
    li   $t8, WHEREY      # $t8 = y_current
    lw   $t8, 0($t8)
    bne $t8, $t9, get_Y      # y_current == y_history[i]
    nop              # -> turn or end

    beq $s5, 4, goBackward_end  # l_history == 4
    nop              # -> end
    addi $s5, $s5, -4        # l_history--
    j goBackward_turn        # else -> turn

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

    addi    $s5, $zero, 4
    sw   $s5, l_history       # reset l_history = 4

    j    printCode
```

1.4 KẾT QUẢ

Kết quả thu được sau khi chạy các dòng lệnh như sau:

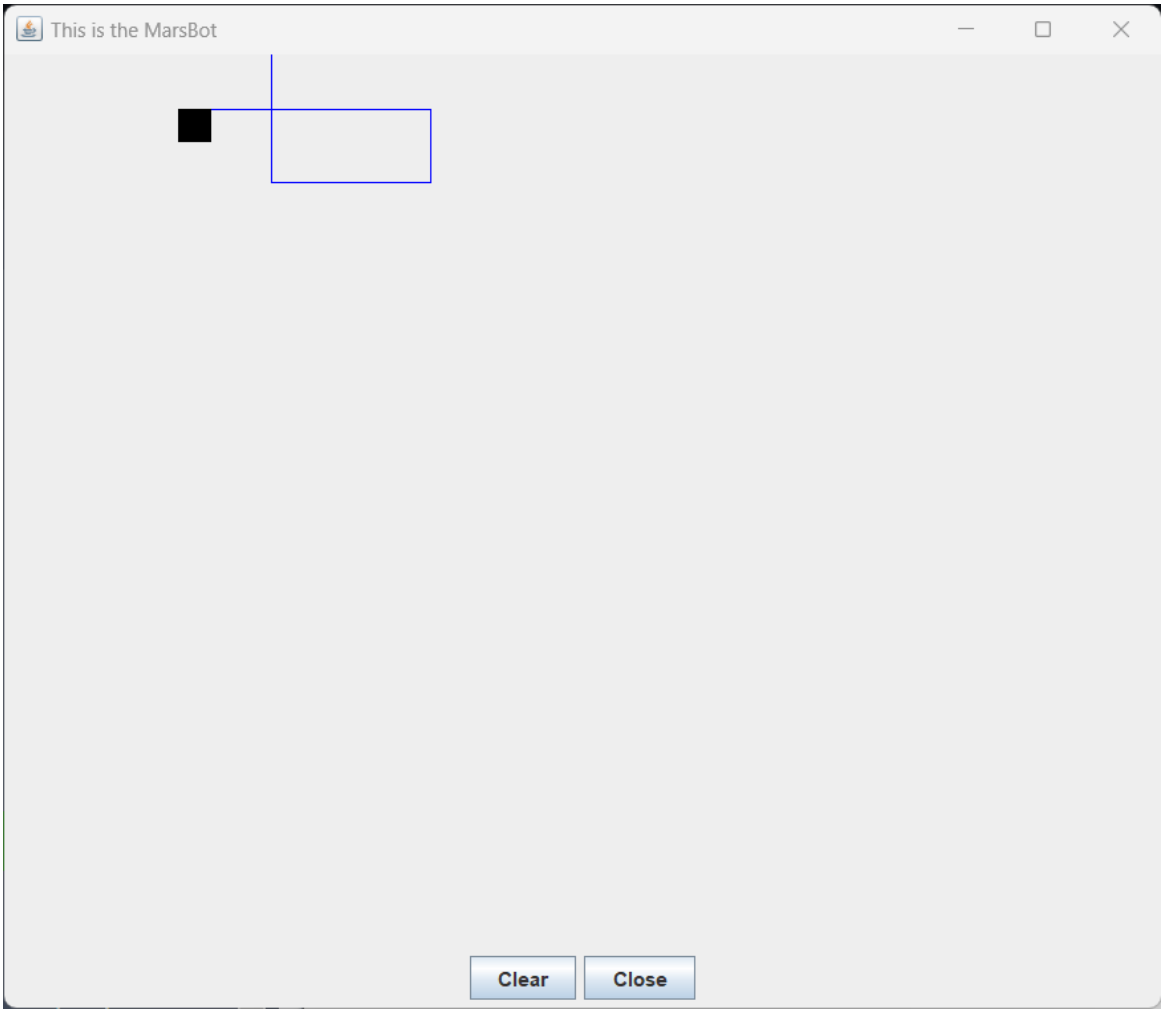
```
dad
1b4
666
c68
```

Hình 1.1: Dòng lệnh 1

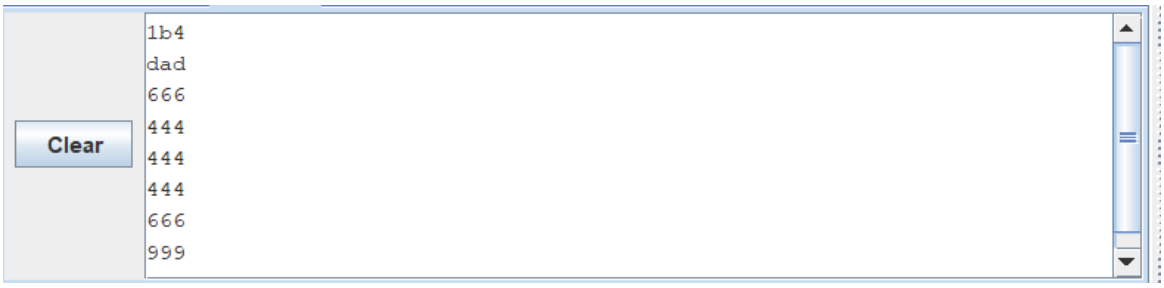


Hình 1.2: Kết quả TH1

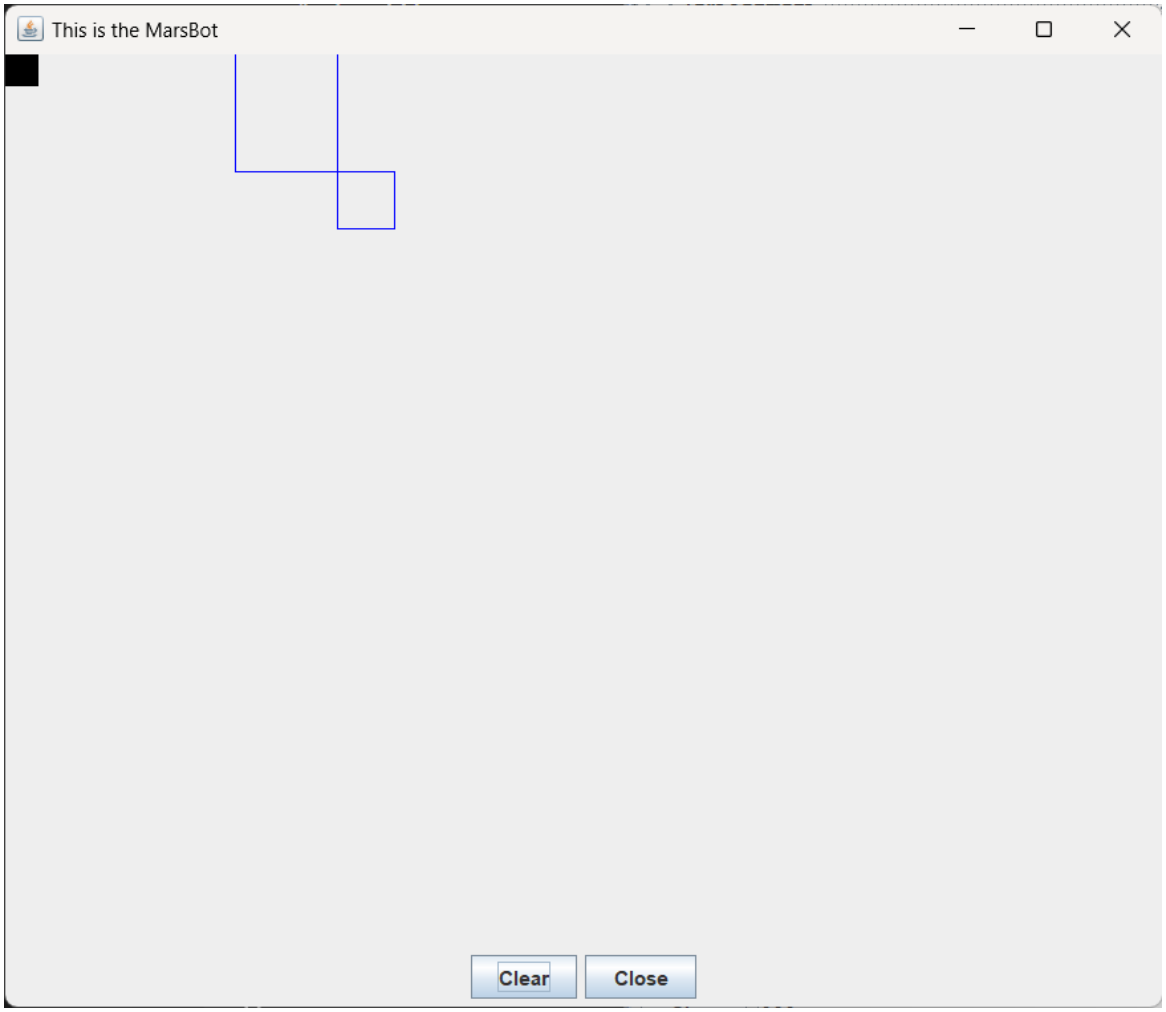
Hình 1.3: Dòng lệnh 2



Hình 1.4: Kết quả TH2



Hình 1.5: Dòng lệnh 3



Hình 1.6: Kết quả TH3

Chương 2

CHƯƠNG TRÌNH KIỂM TRA CÚ PHÁP LỆNH MIPS

2.1 ĐỀ BÀI

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 *s*1 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.

2.2 ĐỊNH HƯỚNG CÁCH LÀM

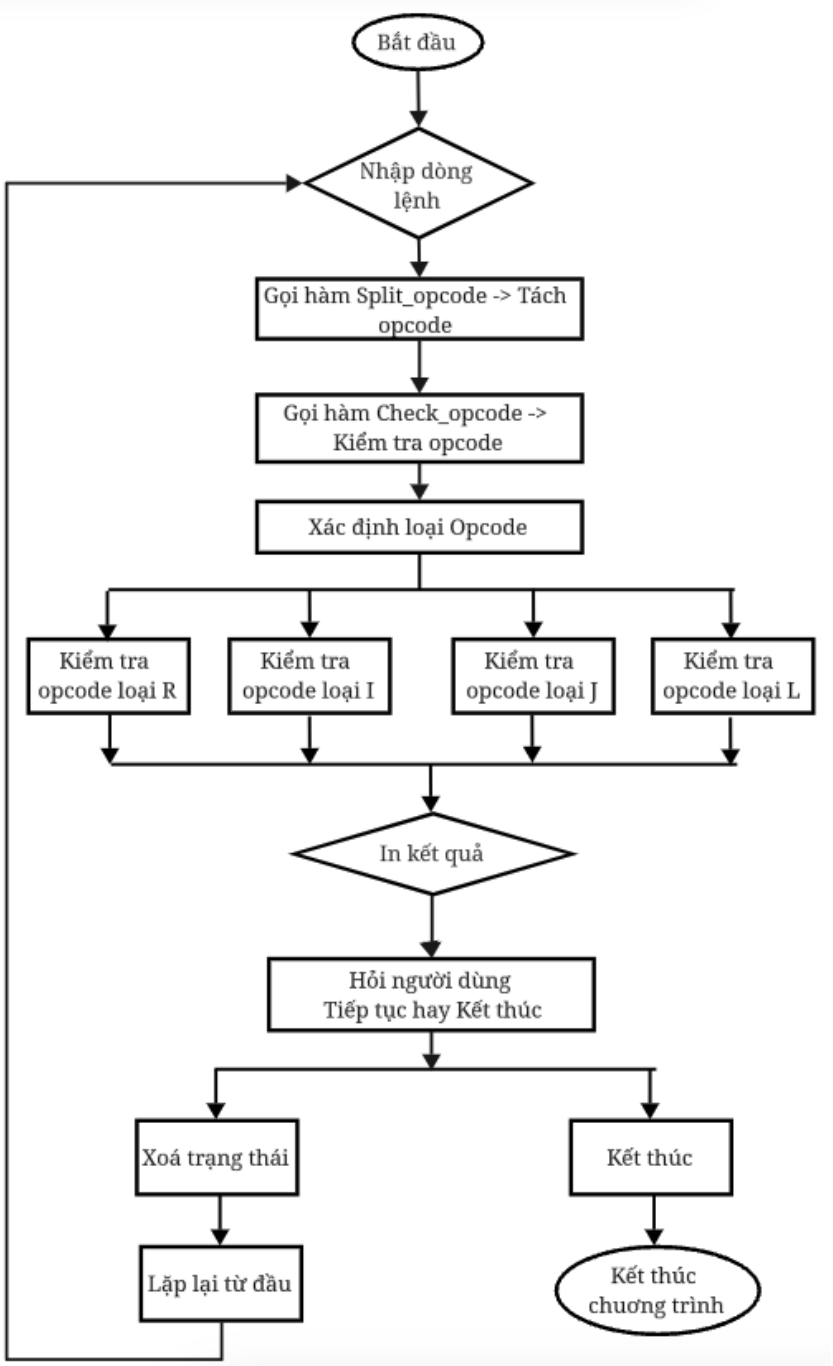
2.2.1 Quy trình tổng thể

1. **Khởi tạo**
   - **Lưu trữ opcode và thanh ghi hợp lệ** Chúng ta cần lưu các opcode và thanh ghi hợp lệ vào một vị trí cụ thể. Điều này cho phép kiểm tra tính hợp lệ của các lệnh nhập vào. Sử dụng mảng ký tự để lưu trữ các opcode mẫu và các thanh ghi mẫu, các giá trị này được phân cách bằng dấu '/' và kết thúc bằng dấu cách.
   - **Chia opcode thành các nhóm:** Việc chia opcode thành các nhóm không chỉ dựa trên việc chúng thuộc khuôn dạng R, I hay J mà còn dựa trên khuôn dạng lệnh khi code.
     Ví dụ: `addi` và `beq` thuộc loại I nhưng khuôn dạng code là khác hẳn nhau:

  - `addi $s1, $s2, 2`
  - `beq $s1, $s2, LoopA`

2. **Logic chính**
   - Tách mã opcode từ dòng lệnh đầu vào.
   - Kiểm tra tính hợp lệ của opcode so với các tập lệnh đã được định trước.
   - Dựa trên loại opcode (R, I, J, L), tiếp tục tách và xác thực các thanh ghi và số trong lệnh
3. **Gọi hàm và xác thực**
   - **Gọi hàm chính**
     - `Split_opcode`: Tách phần opcode và lưu vào hàng đợi.
     - `Split_Register_and_Number`: Tách phần thanh ghi hoặc số và lưu vào hàng đợi.
     - `Split_Sign_ExtImm`: Tách ký tự dùng để tách khuôn dạng `0($s2)` và lưu vào hàng đợi.
     - `Check_opcode`: Kiểm tra opcode có hợp lệ hay không và xác định khuôn dạng của nó, sau đó gán giá trị tương ứng cho `$s4`:
       * R (`$s4 = 1`)
       * R_1 (`$s4 = 2`)
       * R_2 (`$s4 = 6`)
       * I (`$s4 = 3`)
       * I_1 (`$s4 = 7`)
       * J (`$s4 = 4`)
       * J_1 (`$s4 = 8`)
       * L (`$s4 = 9`)
       * L_1 (`$s4 = 10`)
       * Đặc biệt (syscall, nop $\rightarrow$ `$s4 = 5`)
       * Opcode sai (`$s4 = 0`)
   - **Kiểm tra và nhảy tới hàm phù hợp** Ứng với từng giá trị của `$s4`, chúng ta sẽ nhảy tới hàm kiểm tra tương ứng với khuôn dạng đó `(R, R1, I, ...)`.
4. **Lặp lại và thoát**
   - **Hiển thị kết quả** Sau khi kiểm tra, chúng ta sẽ in ra kết quả rằng câu lệnh nhập vào có cấu trúc đúng hay sai.
   - **Lặp lại kiểm tra** Sau khi hiển thị kết quả, hỏi người dùng có muốn kiểm tra tiếp không. Nếu có, lặp lại quy trình từ bước tách và kiểm tra câu lệnh mới. Nếu không, thoát khỏi chương trình.

2.2.2 Lưu đồ thuật toán



Hình 2.1: Lưu đồ thuật toán

2.3 THUẬT TOÁN VÀ MÃ NGUỒN TỪNG BƯỚC

2.3.1 Khai báo dữ liệu

```
.data
Message1: .asciiz "Nhap dong lenh can check: "
Message2: .asciiz "\nOpcode: "
Message3: .asciiz ", hop le!"
Message4: .asciiz " khong hop le!"
Message5: .asciiz " \nCau lenh dung!\n--------------------\n"
Message6: .asciiz " \nCau lenh sai!\n--------------------\n"
Message7: .asciiz " \n"
Message8: .asciiz "Thanh ghi "
Message9: .asciiz "So "
Message10: .asciiz "Nhan "
Message11: .asciiz "Ban muon kiem tra tiep khong?"
string: .space 100
#Luu cac opcode can check vao mang
Opcode_R_Check: .asciiz "/add/sub/addu/subu/and/or/slt/sltu/nor/
   srav/srlv/movn/movz/mul/ "
Opcode_R_Check_1: .asciiz "/beq/bne/ "
Opcode_R_Check_2: .asciiz "/div/divu/mfc0/mult/multu/clo/clz/
   move/negu/not/madd/maddu/msub/msubu/ "
Opcode_I_Check: .asciiz "/addi/addiu/andi/ori/slti/sltiu/sll/srl
   /sra/ "
Opcode_I_Check_1: .asciiz "/li/lui/ "
Opcode_J_Check: .asciiz "/j/jal/ "
Opcode_J_Check_1: .asciiz "/jr/mfhi/mthi/mflo/mtlo/ "
Opcode_L_Check: .asciiz "/lb/lbu/lh/lhu/ll/lw/sb/sc/sh/sw/lwc1/
   ldc1/swc1/sdc1/ "
Opcode_L_Check_1: .asciiz "/la/ "
Special_command: .asciiz "/syscall/nop/ "
Register_Check: .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/$sp/$fp/$ra/$0/$1/$2/$3/$4/$5/$6/$7/
   $8/$9/$10/$11/$12/$13/$14/$15/$16/$17/$18/$19/$20/$21/$22/$23
   /$24/$25/$26/$27/$28/$29/$30/$31/ "
chain\_check: .word     # Chua xau ki tu  ang  xet
.text
start:
    la  $s2, chain_check #Dia chi chua chain_check
    li  $s6, 32     #s6=space
    li  $s7, 47     #s7 = '/'
#Nhap dong lenh can check
    li  $v0, 54
    la  $a0, Message1
    la  $a1, string
    la  $a2, 100
    syscall
    la  $s1, string
```

1. **Khai báo thông báo dạng chuỗi**

`Message1, Message2, ..., Message11`: Đây là các thông báo dạng chuỗi sử dụng .asciiz để khai báo. Các thông báo này sẽ được sử dụng trong chương trình để hiển thị thông tin cho người dùng.

2. **Khai báo danh sách opcode để kiểm tra**
   `Opcode_R_Check, Opcode_R_Check_1, ..., Opcode_L_Check_1, Special_command`: Đây là các chuỗi chứa các opcode hợp lệ để chương trình sử dụng để kiểm tra tính hợp lệ của các lệnh người dùng nhập vào.

2.3.2 Chương trình chính

```
#main
    jal     Split_opcode
    jal     Check_opcode
    beq     $s4, $zero, False_opcode #Opcode false
    addi    $t0, $zero, 5         #Syscall, nop->Right code
    beq     $s4, $t0, Right_code
    addi    $t5, $zero, 1
    beq     $s4, $t5, R_Check_Register_and_Number
    addi    $t5, $zero, 2
    beq     $s4, $t5, R_1_Check_Register_and_Number
    addi    $t5, $zero, 3
    beq     $s4, $t5, I_Check_Register_and_Number
    addi    $t5, $zero, 4
    beq     $s4, $t5, J_Check_Register_and_Number
    addi    $t5, $zero, 6
    beq     $s4, $t5, R_2_Check_Register_and_Number
    addi    $t5, $zero, 7
    beq     $s4, $t5, I_1_Check_Register_and_Number
    addi    $t5, $zero, 8
    beq     $s4, $t5, J_1_Check_Register_and_Number
    addi    $t5, $zero, 9
    beq     $s4, $t5, L_Check_Register_and_Number
    addi    $t5, $zero, 10
    beq     $s4, $t5, L_1_Check_Register_and_Number
    j   End_main
```

Giải thích chương trình chính

- Tách opcode từ dòng lệnh nhập vào, kiểm tra xem opcode nhập vào là dạng nào (R,I,J,L, đặc biệt).
- Kiểm tra xem `$s4` đúng hay sai, nếu sai chạy tới hàm `False_opcode` và in ra màn hình opcode sai. Nếu `$s4=0`, đó là `syscall`, nhảy tới `Right_opcode`.
- Sau đó là câu lệnh so sánh `$s4` với lần lượt với tham số 1,2,3,4. Với mỗi opcode, nhảy tới hàm check Thanh ghi và Số hạng tương ứng.
- Trong trường hợp không khớp với bất kì loại opcode, nhảy tới `End main`.

2.3.3 Tách Opcode và Kiểm tra, phân loại Opcode

a. Chức năng một số thanh ghi được sử dụng

Thanh ghi	Ý nghĩa
$s0	Vị trí phần tử cuối của hàng đợi
$s1	Địa chỉ của lệnh đầu vào
$s2	Địa chỉ của hàng đợi (phần tử đầu tiên)
$s4	Lưu giá trị tương ứng với opcode của lệnh
$s5	Vị trí đang load từ $s1 (ví dụ: "beq $s1, $s2, ABC" thì $s5 = 4)
$t9	Kí tự cuối cùng được load
$s6	Giá trị 32, tương ứng với kí tự khoảng trắng (space)
$s7	Giá trị 47, tương ứng với kí tự '/'
$s3	Địa chỉ của chuỗi opcode mẫu (đã lưu từ trước)
$a2	Địa chỉ của kí tự opcode đang được load từ hàng đợi cần kiểm tra
$a3	Địa chỉ của kí tự đang được load của opcode mẫu
$s4	Chứa giá trị của khuôn dạng lệnh (mặc định ban đầu là 0)
$t1	Biến đếm i (mặc định ban đầu là 0)
$t2	Kí tự được load từ $a2
$t3	Kí tự được load từ $a3
$t0	Số kí tự của opcode mẫu

Bảng 2.1: Các thanh ghi và ý nghĩa của chúng

b. Hàm tách opcode

```
#-----------------------------------------------------
#Tach ma opcode
Split_opcode:
#Khoi tao:
    li  $s5, 0      #Vi tri load ban dau cua lenh nap vao
    li  $s0, 0      #Vi tri phan tu cuoi cua mang chain_check
    li  $t1, 0      #i=0
#Tach ky tu dau tien cua opcode, loai bo ki tu space thua
Loop1:
    add $a2, $s1, $t1   #a2 = Dia chi cua ky tu dang load
    add $a3, $s2, $s0   #a3 = Dia chi dang nap vao hang doi
    lb  $t0, 0($a2)
    beq $t0, $zero, EndLoop #Gap null => ket thuc vong lap 1
    beq $t0, $s6, Loop1_them    #s6=space -> Xu li loai bo space
    sb  $t0, 0($a3)         #Nap ky tu vao hang doi
    addi    $s0, $s0, 1     #Dich chuyen vi tri cuoi cua hang
       doi sang phai
    addi    $t1, $t1, 1     #Tang vi tri cuoi hang doi
    addi    $s5, $s5, 1     #Tang vi tri load ban dau lenh nhap
       vao.
#tach cac ki tu tiep theo cua opcode luu vao queue, gapnewline/
   space-> end
Loop2:
    add $a2, $s1, $t1   #a2 = Dia chi cua ky tu dang load
    add $a3, $s2, $s0   #a3 = Dia chi dang nap vao hang doi
```

```
    lb  $t0, 0($a2)
    beq $t0, $zero, EndLoop #Gap null => ket thuc vong lap 1
    beq $t0, $s6, EndLoop   #Gap space => ket thuc vong lap 1
    li  $t5, 10         #t5=newline
    beq $t0, $t5, EndLoop   #Gap newline => ket thuc vong lap 1
    sb  $t0, 0($a3)         #Nap ky tu vao hang doi
    addi    $s0, $s0, 1     #Dich chuyen vi tri cuoi cua hang
       doi sang phai
    addi    $t1, $t1, 1
    addi    $s5, $s5, 1
    j   Loop2
EndLoop:
    #Chen ky tu NULL cho hang doi
    sb  $zero, 0($a3)
    add $s5, $s0, $zero     #Luu vi tri ki tu dang doc vao s5
    addi    $s0, $s0, -1
    jr  $ra
```

Giải thích

- **Vòng lặp** `Loop1`: Hàm giúp load kí tự đầu tiên vào các thanh ghi đồng thời để loại bỏ các kí tự space thừa ở trước opcode. Chương trình sẽ được quay lại lặp tiếp cho đến khi gặp kí tự `NULL` hoặc tất cả các ký tự đầu tiên của opcode đã được tách và lưu vào hàng đợi.
- **Vòng lặp** `Loop2`: Vòng lặp thứ hai nhằm load các kí tự tiếp theo của opcode và lưu vào hàng đợi. Tương tự như vòng lặp đầu tiên, nhưng vòng lặp này kiểm tra thêm các ký tự dấu cách (space) và xuống dòng (newline) để kết thúc vòng lặp nếu gặp. `Loop2` kết thúc khi gặp space hoặc newline hoặc null.
- **Kết thúc** `(EndLoop)`: Thêm ký tự NULL vào cuối hàng đợi để đánh dấu kết thúc chuỗi opcode. Sau đó thực hiện các thao tác lưu vị trí kí tự và trở lại hàm main.

c. Hàm check opcode

```
#-----------------------------------------------------
#Check Opcode
Check_opcode:
    li  $s4, 0  #s4 bieu thi cho khuon dang lenh: Saiopcode: 0,
       R: 1, R_1: 2, I: 3, J: 4, Dac biet: 5

#Check_R
    la  $s3, Opcode_R_Check #chua list opcode mau
    li  $t1, 0      #i=0
Loop1_R:
    add $a3, $s3, $t1   #load byte cua opcode mau
    lb  $t3, 0($a3)
    addi    $t1, $t1, 1
    bne $t3, $s7, Loop1_R   #S7='/'
    li  $t0, 0      #So ki tu cua opcode mau
Loop2_R:
    add $a3, $s3, $t1   #load byte cua opcode mau
    lb  $t3, 0($a3)
    add $a2, $s2, $t0   #Load byte cua opcode can check
```

```
19      lb  $t2, 0($a2)
20      beq $t3, $s7, Check_R   #'s7'='/'
21      beq $t3, $s6, End_Loop_R    #s6 = space
22      bne $t2, $t3, Loop1_R_them  #Kiem tra xem opcode check va
           opcode mau co giong nhau khong
23      #khong giong nhay toi Loop1_R_them de bat dau ki tu tiep
           theo
24      beq $t2, $t3, Loop2_R_them  #Quay tro lai kiem tra next ki
           tu
25  End_Loop_R:
```

Đoạn mã này có chức năng kiểm tra xem opcode cần kiểm tra có khớp với opcode mẫu hay không. Cụ thể:

- **Khởi tạo giá trị ban đầu:**
  - `$s3`: Địa chỉ chứa danh sách các opcode mẫu.
  - `$t1`: Biến đếm, khởi tạo bằng 0 (i=0).
- **Vòng lặp Loop1_R:**
  - Lấy từng byte từ opcode mẫu và lưu vào `$t3`.
  - Tăng `$t1` để dịch chuyển đến byte tiếp theo trong opcode mẫu.
  - Nếu ký tự hiện tại là `'/'`, tiếp tục vòng lặp `Loop1_R`.
- **Khởi tạo giá trị `$t0`:**
  - Đặt `$t0` là số ký tự của opcode mẫu (khởi tạo bằng 0).
- **Vòng lặp Loop2_R:**
  - Lấy từng byte từ opcode mẫu và opcode cần kiểm tra, lưu vào `$t3` và `$t2` tương ứng.
  - Nếu ký tự hiện tại trong opcode mẫu là `'/'`, nhảy đến `Check_R`.
  - Nếu ký tự hiện tại trong opcode mẫu là khoảng trắng (space), kết thúc vòng lặp `Loop2_R`.
  - So sánh ký tự hiện tại của opcode mẫu và opcode cần kiểm tra:
    - Nếu không khớp, nhảy đến `Loop1_R_them` để bắt đầu lại vòng lặp `Loop1_R` từ ký tự tiếp theo.
    - Nếu khớp, tiếp tục vòng lặp `Loop2_R_them` để kiểm tra ký tự tiếp theo.
- **Kết thúc vòng lặp End_Loop_R:**
  - Kết thúc quá trình kiểm tra và nhảy đến nhãn `End_Loop_R`.

Chức năng tổng quan: Đoạn mã này kiểm tra từng ký tự của opcode cần kiểm tra với opcode mẫu để xác định xem chúng có khớp nhau hay không. Quá trình này lặp đi lặp lại cho đến khi toàn bộ các ký tự đã được so sánh hoặc phát hiện ký tự không khớp.

2.3.4 Kiểm tra lệnh theo từng khuôn dạng lệnh

Sau khi xác định khuôn dạng lệnh thông qua opcode, giá trị của thanh ghi `s4` sẽ được gán tương ứng với loại lệnh. Dựa trên giá trị của `s4`, chương trình sẽ nhảy đến các hàm kiểm tra tương ứng với từng khuôn dạng lệnh cụ thể.
a. Ví dụ hàm kiểm tra khuôn lệnh R (bao gồm các lệnh: `/add/sub/addu/subu/and/or/slt/sltu/nor/s`

```
#-----------------------------------------------------
#Check cac thanh ghi va so
R_Check_Register_and_Number:
    jal Right_opcode
    jal Split_Register_and_Number
    jal Check_Register
    jal Split_Register_and_Number
    jal Check_Register
    jal Split_Register_and_Number
    jal Check_Register
    addi    $t5, $zero, 10      #s10= newline
    beq $t9, $t5, Right_code
    addi    $t5, $zero, 0       #t5=NULL
    beq $t9, $t5, Right_code
    j   False_code
```

- **Gọi hàm** `Right_opcode`:
  - Đầu tiên, hàm `Right_opcode` được gọi để in ra thông báo rằng opcode trong câu lệnh là hợp lệ. Việc này đảm bảo rằng opcode đã được kiểm tra và xác nhận trước đó.
- **Gọi hàm** `Split_Register_and_Number` **và** `Check_Register`:
  - Tiếp theo, chúng ta cần kiểm tra 3 thanh ghi trong câu lệnh để đảm bảo chúng đúng.
  - Đầu tiên, hàm `Split_Register_and_Number` được gọi để tách thanh ghi ra khỏi chuỗi lệnh.
  - Sau đó, hàm `Check_Register` được gọi để kiểm tra xem thanh ghi đó có hợp lệ hay không.
  - Vì câu lệnh dạng R có 3 thanh ghi, nên quá trình tách và kiểm tra này được lặp lại ba lần để kiểm tra cả ba thanh ghi.
- **Kiểm tra ký tự cuối cùng (**`$t9`**)**:
  - Sau khi kiểm tra xong các thanh ghi, chúng ta cần xác định xem ký tự cuối cùng của lệnh có hợp lệ không.
  - Biến `$t9` chứa ký tự cuối cùng của chuỗi lệnh.
  - So sánh ký tự cuối cùng này với ký tự xuống dòng (newline, mã ASCII là 10) và ký tự NULL (mã ASCII là 0).
  - Nếu ký tự cuối cùng là newline hoặc NULL, lệnh được xác nhận là đúng (`Right_code`).
  - Nếu không, lệnh được xác nhận là sai (`False_code`).

b. Hàm kiểm tra khuôn lệnh I (bao gồm các lệnh `/addi/addiu/andi/ori/slti/sltiu/sll/srl/sra/`

```
I_Check_Register_and_Number:
    jal Right_opcode
    jal Split_Register_and_Number
    jal Check_Register
    jal Split_Register_and_Number
    jal Check_Register
    jal Split_Register_and_Number
    jal Check_Number
```

```
    addi    $t5, $zero, 10
    beq $t9, $t5, Right_code
    addi    $t5, $zero, 0
    beq $t9, $t5, Right_code
    j   False_code
```

Giải thích

- **Gọi hàm** `Right_opcode`:
- **Kiểm tra hai thanh ghi đầu tiên**:
  - Tiếp theo, chúng ta cần kiểm tra hai thanh ghi đầu tiên trong câu lệnh. Để làm điều này, chúng ta sử dụng hàm `Split_Register_and_Number` để tách thanh ghi ra khỏi chuỗi lệnh.
  - Sau đó, hàm `Check_Register` được gọi để kiểm tra xem thanh ghi đó có hợp lệ hay không.
  - Quá trình này được lặp lại hai lần để kiểm tra cả hai thanh ghi đầu tiên.
- **Kiểm tra phần tử cuối cùng (số)**:
  - Đối với phần tử cuối cùng của lệnh dạng I, chúng ta cần kiểm tra xem nó có phải là một số hợp lệ hay không.
  - Hàm `Split_Register_and_Number` được gọi để tách số ra khỏi chuỗi lệnh.
  - Sau đó, hàm `Check_Number` được gọi để kiểm tra xem số này có hợp lệ hay không.
- **Kiểm tra ký tự cuối cùng (`$t9`)**:
  - Tương tự với hàm kiểm tra khuôn lệnh R

c. Hàm kiểm tra khuôn lệnh I đặc biệt. (bao gồm 2 câu lệnh `beq`, `bne`)

```
    I_2_Check_Register_and_Number:
    jal Right_opcode
    jal Split_Register_and_Number
    jal Check_Register
    jal Split_Register_and_Number
    jal Check_Register
    jal Split_Register_and_Number
    addi    $t5, $zero, 10
    beq $t9, $t5, R_1_Check_Label
    addi    $t5, $zero, 0
    beq $t9, $t5, R_1_Check_Label
    j   False_code
    R_1_Check_Label:
    jal Check_Label
```

- **Gọi hàm** `Right_opcode`
- **Kiểm tra hai thanh ghi đầu tiên:** Giống lệnh kiểm tra I bên trên.
- **Kiểm tra phần tử cuối cùng (nhãn)**:
  - Sau khi kiểm tra xong các thanh ghi, ký tự cuối cùng của chuỗi lệnh (`$t9`) được so sánh với ký tự xuống dòng (newline, mã ASCII là 10) và ký tự NULL (mã ASCII là 0).
  - Nếu ký tự cuối cùng là newline hoặc NULL, chương trình sẽ nhảy đến nhãn `R_1_Check_Label` để kiểm tra nhãn.

  - Nếu không, lệnh được xác nhận là sai (`False_code`).

- **Kiểm tra nhãn (`Check_Label`):**
  - Tại nhãn `R_1_Check_Label`, hàm `Check_Label` được gọi để kiểm tra tính hợp lệ của nhãn trong lệnh.

d. Kiểm tra khuôn lệnh L (bao gồm các lệnh `/lb/lbu/lh/lhu/ll/lw/sb/sc/....`)

```
    L_Check_Register_and_Number:
    jal Right_opcode
    jal Split_Register_and_Number
    jal Check_Register
    jal Check_Sign_ExtImm
```

- **Gọi hàm** `Right_opcode`
- **Tách và kiểm tra thanh ghi**:
  - Hàm `Split_Register_and_Number` được gọi để tách thanh ghi ra khỏi chuỗi lệnh.
  - Sau đó, hàm `Check_Register` được gọi để kiểm tra xem thanh ghi đó có hợp lệ hay không.
- **Kiểm tra (`Sign_ExtImm`):**
  - Cuối cùng, vì khuôn dạng như loại này thì nó có cái cụm phía sau khá đặc biệt nên ta sẽ xây dựng một hàm riêng để vừa cả tách đồng thời kiểm tra xem có thỏa mãn hay không.

2.3.5 Hàm tách thanh ghi và số

```
#-----------------------------------------------------
#Tach ma thanh ghi va so
Split_Register_and_Number:
    li  $s0, 0      #Vi tri phan tu cuoi cua mang chain_check
    add $t1, $s5, $zero #i=vi tri dang doc trong cau lenh=s5
Loop1_Split: #Bo qua space truoc thi ghi/so va load ki tu dau
    add $a2, $s1, $t1   #a2 = Dia chi cua ky tu dang load
    add $a3, $s2, $s0   #a3 = Dia chi dang nap vao hang doi
    lb  $t0, 0($a2) #t0 = Ky tu dang Load
    add $t9, $zero, $t0 #t9 = Ky tu cuoi cung duoc load
    beq $t0, $zero, EndLoop_Split   #Gap null => ket thuc vong
       lap 1
    beq $t0, $s6, Loop1_Split_them  #Gap Space -> Chay qua Space
    li  $t5, 44             #t5=44~'dau phay,'
    beq $t0, $t5, False_code
    sb  $t0, 0($a3)         #Nap ky tu vao hang doi
    addi    $s0, $s0, 1     #Dich chuyen vi tri cuoi cua hang
       doi sang phai
    addi    $t1, $t1, 1
Loop2_Split: #load cac gia tri tiep theo vao hang doi
    add $a2, $s1, $t1   #a2 = Dia chi cua ky tu dang load
    add $a3, $s2, $s0   #a3 = Dia chi dang nap vao hang doi
    lb  $t0, 0($a2)
    add $t9, $zero, $t0 #t9 = Ky tu cuoi cung duoc load
```

```
23      beq $t0, $zero, EndLoop_Split#Gap null => ket thuc vong lap
24      beq $t0, $s6, Loop3_Split   #Gap space => Chay qua Space
25      li  $t5, 10          #t5=newline
26      beq $t0, $t5, EndLoop_Split #Gap newline => ket thuc vong
           lap
27      li  $t5, 44          #t5=44~'dau phay,'
28      beq $t0, $t5, EndLoop_Split #Gap dau phay => ket thuc vong
           lap
29      sb  $t0, 0($a3)          #Nap ky tu vao hang doi
30      addi    $s0, $s0, 1      #Dich chuyen vi tri cuoi cua hang
           doi sang phai
31      addi    $t1, $t1, 1
32      j   Loop2_Split
33  Loop3_Split: #Loai bo cac ki tu khoang trang dung sau thanh ghi
       va so
34      add $a2, $s1, $t1    #a2 = Dia chi cua ky tu dang load
35      add $a3, $s2, $s0    #a3 = Dia chi dang nap vao hang doi
36      lb  $t0, 0($a2) #t0 = Ky tu dang Load
37      add $t9, $zero, $t0 #t9 = Ky tu cuoi cung duoc load
38      beq $t0, $zero, EndLoop_Split   #Gap null => ket thuc vong
           lap 1
39      beq $t0, $s6, Loop3_Split_them  #Gap Space -> Chay qua Space
40      li  $t5, 44      #t5=44~'dau phay,'
41      beq $t0, $t5, EndLoop_Split
42      li  $t5, 10      #t5=10~'New line'
43      beq $t0, $t5, EndLoop_Split
44      j   False_code
45  EndLoop_Split: #hoan thanh viec tach
46      #Chen ky tu NULL cho hang doi
47      sb  $zero, 0($a3)
48      addi    $s5, $t1, 1 #Luu vi tri ki tu dang doc vao s5
49      addi    $s0, $s0, -1
50      jr  $ra
```

Giải thích

- **Loop1_Split:**
  - *Mục đích:* Bỏ qua các ký tự khoảng trắng (space) đứng trước thanh ghi hoặc số và load ký tự đầu vào hàng đợi.
  - *Cách thực hiện:*
    - Load ký tự từ địa chỉ `$s1` với vị trí `$t1`.
    - Nếu ký tự là `null`, kết thúc vòng lặp.
    - Nếu ký tự là space, bỏ qua và tiếp tục vòng lặp.
    - Nếu ký tự là dấu phẩy (`,`), chuyển đến hàm `False_code`.
    - Lưu ký tự vào hàng đợi và di chuyển đến ký tự tiếp theo.
- **Loop2_Split:**
  - *Mục đích:* Load các ký tự tiếp theo vào hàng đợi.
  - *Cách thực hiện:*
    - Load ký tự từ địa chỉ `$s1` với vị trí `$t1`.
    - Nếu ký tự là `null`, kết thúc vòng lặp.
    - Nếu ký tự là space, chuyển sang `Loop3_Split`.

    * Nếu ký tự là `newline`, kết thúc vòng lặp.
    * Nếu ký tự là dấu phẩy (`,`), kết thúc vòng lặp.
    * Lưu ký tự vào hàng đợi và di chuyển đến ký tự tiếp theo.

- **Loop3_Split:**
  - *Mục đích:* Loại bỏ các ký tự khoảng trắng đứng sau thanh ghi hoặc số.
  - *Cách thực hiện:*
    * Load ký tự từ địa chỉ `$s1` với vị trí `$t1`.
    * Nếu ký tự là `null`, kết thúc vòng lặp.
    * Nếu ký tự là `space`, bỏ qua và tiếp tục vòng lặp.
    * Nếu ký tự là dấu phẩy (`,`), kết thúc vòng lặp.
    * Nếu ký tự là `newline`, kết thúc vòng lặp.
    * Nếu ký tự không hợp lệ, chuyển đến hàm `False_code`.
- **EndLoop_Split:**
  - *Mục đích:* Hoàn tất quá trình tách thanh ghi và số.
  - *Cách thực hiện:*
    * Chèn ký tự `NULL` vào cuối hàng đợi để đánh dấu kết thúc chuỗi.
    * Cập nhật giá trị `$s5` và `$s0`.
    * Quay về chương trình chính.

2.3.6 Hàm kiểm tra thanh ghi

```
    #-----------------------------------------------------
#-----------------------------------------------------
#Check Register
Check_Register:
    la  $s3, Register_Check #dia chi cua list thanh ghi mau
    li  $t1, 0          #i=0
Loop1_Reg: #Duyet qua cac ki tu cua thanh ghi mau va bo qua ki tu '/'
    add $a3, $s3, $t1   #load byte cua thanh ghi mau
    lb  $t3, 0($a3)
    addi    $t1, $t1, 1
    bne $t3, $s7, Loop1_Reg #s7=/
    li  $t0, 0      #So ki tu cua thanh ghi mau
Loop2_Reg: #so sanh thanh ghi mau va thanh ghi kiem tra.
    add $a3, $s3, $t1   #load byte cua thanh ghi mau
    lb  $t3, 0($a3)
    add $a2, $s2, $t0   #Load byte cua thanh ghi can check
    lb  $t2, 0($a2)
    beq $t3, $s7, Check_Reg
    beq $t3, $s6, False_code
    bne $t2, $t3, Loop1_Reg_them    #Kiem tra xem thanh ghi check va thanh ghi mau co giong nhau khong
    beq $t2, $t3, Loop2_Reg_them
End_Loop_Reg:

#-----------------------------------------------------

Check_Reg: #kiem tra ki tu cuoi cung cua Reg co khop voi thanh ghi mau khong?
```

```
27      addi    $t0, $t0, -1
28      beq $s0, $t0, Reg_True
29      j   Loop1_Reg
30  Loop1_Reg_them:
31      addi    $t1, $t1, 1
32      j   Loop1_Reg
33  Loop2_Reg_them:
34      addi    $t1, $t1, 1
35      addi    $t0, $t0, 1
36      j   Loop2_Reg
37  Reg_True:
38      add     $t8, $zero, $ra
39      jal Right_Register
40      jr  $t8
```

a. Giải thích chức năng của các thanh ghi

Thanh ghi	Ý nghĩa
$s2	Địa chỉ của ký tự đầu tiên trong hàng đợi
$s3	Địa chỉ của chuỗi thanh ghi mẫu đã lưu trữ trước đó
$s0	Vị trí cuối cùng trong hàng đợi
$a2	Địa chỉ của ký tự thanh ghi đang được nạp từ hàng đợi để kiểm tra
$a3	Địa chỉ của ký tự đang được nạp từ chuỗi thanh ghi mẫu
$t1	Biến đếm i, khởi tạo bằng 0
$t2	Ký tự được nạp từ $a2
$t3	Ký tự được nạp từ $a3
$t0	Số lượng ký tự của chuỗi thanh ghi mẫu

Bảng 2.2: Chức năng các thanh ghi trong hàm Check_Register

b. Giải thích code

- **Khởi tạo giá trị ban đầu:**
  - $s3: Địa chỉ chứa danh sách các thanh ghi hợp lệ (Register_Check).
  - $t1: Biến đếm, khởi tạo bằng 0 (i=0).
- **Vòng lặp** Loop1_Reg**:**
  - Tải byte của thanh ghi mẫu từ địa chỉ $s3 và lưu vào $t3.
  - Tăng biến đếm $t1 lên 1.
  - Nếu ký tự $t3 là ký tự '/', tiếp tục vòng lặp Loop1_Reg.
  - Đặt $t0 là số ký tự của thanh ghi mẫu.

  => Dùng để duyệt qua các ký tự của thanh ghi mẫu và bỏ qua các ký tự '/'.
- **Vòng lặp** Loop2_Reg**:**
  - Tải byte của thanh ghi mẫu từ địa chỉ $s3 và lưu vào $t3.
  - Tải byte của thanh ghi cần kiểm tra từ hàng đợi ($s2) và lưu vào $t2.
  - Nếu ký tự $t3 là ký tự '/', nhảy đến Check_Reg.
  - Nếu ký tự $t3 là ký tự ' ', nhảy đến False_code.
  - Nếu ký tự $t2 và $t3 không khớp, nhảy đến Loop1_Reg_them.
  - Nếu ký tự $t2 và $t3 khớp, tiếp tục vòng lặp Loop2_Reg_them.

  => Dùng để kiểm tra xem thanh ghi cần kiểm tra có khớp với thanh ghi mẫu hay không. Nếu không khớp, nó sẽ nhảy đến Loop1_Reg_them hoặc False_code.

- **Vòng lặp** `Loop1_Reg_them`**:**
  - Tăng biến đếm `$t1` lên 1.
  - Quay lại `Loop1_Reg`.

  => Dùng để tăng biến đếm `$t1` **và quay lại** `Loop1_Reg`.
- **Vòng lặp** `Loop2_Reg_them`**:**
  - Tăng biến đếm `$t1` lên 1.
  - Tăng `$t0` lên 1.
  - Quay lại `Loop2_Reg`.

  => Dùng để tăng biến đếm `$t1`**,** `$t0` **và quay lại** `Loop2_Reg`**.**
- **Hàm** `Check_Reg`**:**
  - Giảm `$t0` đi 1.
  - Nếu `$s0` bằng `$t0`, nhảy đến `Reg_True`.
  - Quay lại `Loop1_Reg`.

  => Dùng để kiểm tra xem ký tự cuối cùng của thanh ghi cần kiểm tra có khớp với thanh ghi mẫu không. Nếu khớp, nhảy đến `Reg_True`**.**
- **Nhãn** `Reg_True`**:**
  - Đặt `$t8` bằng giá trị trả về của thanh ghi ra.
  - Gọi hàm `Right_Register` để in ra thông báo rằng thanh ghi hợp lệ.
  - Quay lại địa chỉ trong `$t8`.

  => Xác nhận thanh ghi hợp lệ và gọi hàm `Right_Register` **để in thông báo**

2.3.7 Kiểm tra số

```
#-----------------------------------------------------
#Check Number
Check_Number:
    li  $t1, 0      #i = 0
    j   Check_Mark
Check_Mark_done: #Kiem tra cac ki tu tiep theo
    add $a2, $s2, $t1   #Kiem tra so dau tien
    lb  $t2, 0($a2)
    li  $t5, 10     #t5 = newline
    beq $t2, $t5, False_code
    beq $t2, $zero, False_code
    li  $t5, 48     #t5 = zero
    bne $t2, $t5, Loop_Number_1
    slti    $t4, $t2, 48
    bne $t4, $zero, False_code
    slti    $t4, $t2, 58
    beq $t4, $zero, False_code
    addi    $t1, $t1, 1 #Kiem tra so thu hai(co the la x trong
       so hexa)
    add $a2, $s2, $t1
    lb  $t2, 0($a2)
    #Kiem tra xem cac ki tu co nam trong khoang so hop le khong
       (0-9, A-F, a-f)
    beq $t2, $zero, Right_Number
    li  $t5, 120
```

```
    beq $t2, $t5, Loop_Number_them
    li  $t5, 88
    beq $t2, $t5, Loop_Number_them
    slti    $t4, $t2, 48
    bne $t4, $zero, False_code
    slti    $t4, $t2, 58
    beq $t4, $zero, False_code
Loop_Number: #kiem tra cac ki tu so
    add $a2, $s2, $t1
    lb  $t2, 0($a2)
    beq $t2, $zero, Right_Number
    li  $t5, 48
    beq $t2, $t5, Loop_Number_them
    li  $t5, 49
    beq $t2, $t5, Loop_Number_them
    li  $t5, 50
    beq $t2, $t5, Loop_Number_them
    li  $t5, 51
    beq $t2, $t5, Loop_Number_them
    li  $t5, 52
    beq $t2, $t5, Loop_Number_them
    li  $t5, 53
    beq $t2, $t5, Loop_Number_them
    li  $t5, 54
    beq $t2, $t5, Loop_Number_them
    li  $t5, 55
    beq $t2, $t5, Loop_Number_them
    li  $t5, 56
    beq $t2, $t5, Loop_Number_them
    li  $t5, 57
    beq $t2, $t5, Loop_Number_them
    li  $t5, 65
    beq $t2, $t5, Loop_Number_them
    li  $t5, 66
    beq $t2, $t5, Loop_Number_them
    li  $t5, 67
    beq $t2, $t5, Loop_Number_them
    li  $t5, 68
    beq $t2, $t5, Loop_Number_them
    li  $t5, 69
    beq $t2, $t5, Loop_Number_them
    li  $t5, 70
    beq $t2, $t5, Loop_Number_them
    li  $t5, 97
    beq $t2, $t5, Loop_Number_them
    li  $t5, 98
    beq $t2, $t5, Loop_Number_them
    li  $t5, 99
    beq $t2, $t5, Loop_Number_them
    li  $t5, 100
    beq $t2, $t5, Loop_Number_them
    li  $t5, 101
    beq $t2, $t5, Loop_Number_them
```

```
77      li   $t5, 102
78      beq $t2, $t5, Loop_Number_them
79      j    False_code
80  Loop_Number_1:
81      add $a2, $s2, $t1
82      lb   $t2, 0($a2)
83      beq $t2, $zero, Right_Number
84      li   $t5, 48
85      beq $t2, $t5, Loop_Number_them_1
86      li   $t5, 49
87      beq $t2, $t5, Loop_Number_them_1
88      li   $t5, 50
89      beq $t2, $t5, Loop_Number_them_1
90      li   $t5, 51
91      beq $t2, $t5, Loop_Number_them_1
92      li   $t5, 52
93      beq $t2, $t5, Loop_Number_them_1
94      li   $t5, 53
95      beq $t2, $t5, Loop_Number_them_1
96      li   $t5, 54
97      beq $t2, $t5, Loop_Number_them_1
98      li   $t5, 55
99      beq $t2, $t5, Loop_Number_them_1
100     li   $t5, 56
101     beq $t2, $t5, Loop_Number_them_1
102     li   $t5, 57
103     j    False_code
104 #-----------------------------------------------------
105 Right_Number:
106     add $t8, $zero, $ra
107     jal      Print_Right_Number
108     jr   $t8
109 #-----------------------------------------------------
110 Check_Mark: #Ham kiem tra dau cua imm
111     add $a2, $s2, $t1    #Kiem tra xem ki tu dau tien cua Imm co
           phai dau + hay - khong?
112     lb   $t2, 0($a2)
113     li   $t5, 43      #t5 =43 ~ '+'
114     beq $t2, $t5, Check_Mark_them
115     li   $t5, 45      #t5 =45 ~ '-'
116     beq $t2, $t5, Check_Mark_them
117     j    Check_Mark_done
```

a. Giải thích ý nghĩa của thanh ghi

$s2	Địa chỉ ký tự đầu của hàng đợi
$s0	Vị trí cuối của hàng đợi
$a2	Địa chỉ ký tự thanh ghi đang load từ hàng đợi đang cần kiểm tra
$t1	Biến đếm, mặc định ban đầu = 0
$t2	Ký tự load từ $a2
$t3	Ký tự load từ $a3
$t5	Ký tự tạm thời dùng để kiểm tra

Bảng 2.3: Giải thích chức năng các thanh ghi

b. Giải thích chương trình

- **Khởi tạo biến đếm:**
  - Đặt biến đếm $t1 về 0 và nhảy đến Check_Mark để kiểm tra dấu của số.
- **Kiểm tra dấu số:**
  - Kiểm tra xem ký tự đầu tiên có phải là dấu cộng hoặc trừ không. Nếu đúng, nhảy tới Check_Mark_them để bỏ qua dấu này và tiếp tục kiểm tra các ký tự tiếp theo. Nếu không phải, nhảy đến Check_Mark_done.
- **Kiểm tra các ký tự tiếp theo:**
  - Kiểm tra ký tự đầu tiên sau dấu (nếu có). Nếu gặp ký tự newline hoặc NULL thì nhảy đến False_code.
  - Nếu ký tự đầu tiên không phải là '0', chuyển sang Loop_Number.
  - Nếu ký tự đầu tiên là '0', kiểm tra ký tự tiếp theo có phải là 'x' (ký hiệu số hexa) không.
  - Kiểm tra các ký tự tiếp theo xem có nằm trong khoảng số hợp lệ (0-9, A-F, a-f).
- **Vòng lặp kiểm tra các ký tự số:**
  - Vòng lặp Loop_Number kiểm tra từng ký tự trong chuỗi xem có phải là ký tự số hợp lệ hay không.
  - Nếu ký tự là số (0-9) hoặc chữ cái trong dải hợp lệ cho số hexa (A-F, a-f), chuyển đến Loop_Number_them để tiếp tục kiểm tra ký tự tiếp theo.
  - Nếu gặp ký tự không hợp lệ, nhảy đến False_code.
- **Kiểm tra ký tự tiếp theo trong số:**
  - Tăng biến đếm $t1 và quay lại vòng lặp Loop_Number để kiểm tra ký tự tiếp theo.
- **Xác nhận số hợp lệ:**
  - Nếu tất cả các ký tự trong chuỗi đều hợp lệ, hàm Right_Number được gọi để xác nhận số hợp lệ và quay lại địa chỉ trước đó.

2.3.8 Kiểm tra nhãn Label

```
#-----------------------------------------------------
#Check Label
Check_Label:
    li   $t1, 0       #i = 0
    add $a2, $s2, $t1
    lb   $t2, 0($a2)
```

```
7       beq $t2, $zero, False_code
8       li  $t5, 10      #t5 = 'New line'
9       beq $t2, $t5, False_code
10      slti    $t4, $t2, 48
11      bne $t4, $zero, False_code
12      li  $t5, 58
13      beq $t2, $t5, False_code
14      li  $t5, 59
15      beq $t2, $t5, False_code
16      li  $t5, 60
17      beq $t2, $t5, False_code
18      li  $t5, 61
19      beq $t2, $t5, False_code
20      li  $t5, 62
21      beq $t2, $t5, False_code
22      li  $t5, 63
23      beq $t2, $t5, False_code
24      li  $t5, 64
25      beq $t2, $t5, False_code
26      li  $t5, 91
27      beq $t2, $t5, False_code
28      li  $t5, 92
29      beq $t2, $t5, False_code
30      li  $t5, 93
31      beq $t2, $t5, False_code
32      li  $t5, 94
33      beq $t2, $t5, False_code
34      li  $t5, 96
35      beq $t2, $t5, False_code
36      slti    $t4, $t2, 123
37      beq $t4, $zero, False_code
38      addi    $t1, $t1, 1
39  Loop_Label:
40      add $a2, $s2, $t1       # Lay dia chi cua ky tu tiep theo tu
            hang doi
41      lb $t2, 0($a2)          # Lay ky tu tiep theo va luu vao $t2
42      beq $t2, $zero, True_Label      # Neu gap ky tu NULL, nhay
           den True_Label
43      li $t5, 10                      # t5 = 'New line'
44      beq $t2, $t5, True_Label    # Neu gap ky tu newline, nhay
           den True_Label
45
46      slti $t4, $t2, 48
47      bne $t4, $zero, False_code      # Neu ky tu nho hon '0' (ma
            ASCII 48), nhay den False_code
48      li $t5, 58
49      beq $t2, $t5, False_code        # Neu ky tu la ':', nhay
           den False_code
50      li $t5, 59
51      beq $t2, $t5, False_code        # Neu ky tu la ';', nhay
           den False_code
52      li $t5, 60
```

```
53      beq $t2, $t5, False_code          # Neu ky tu la '<', nhay
           den False_code
54      li $t5, 61
55      beq $t2, $t5, False_code          # Neu ky tu la '=', nhay
           den False_code
56      li $t5, 62
57      beq $t2, $t5, False_code          # Neu ky tu la '>', nhay
           den False_code
58      li $t5, 63
59      beq $t2, $t5, False_code          # Neu ky tu la '?', nhay
           den False_code
60      li $t5, 64
61      beq $t2, $t5, False_code          # Neu ky tu la '@', nhay
           den False_code
62      li $t5, 91
63      beq $t2, $t5, False_code          # Neu ky tu la '[', nhay
           den False_code
64      li $t5, 92
65      beq $t2, $t5, False_code          # Neu ky tu la '\', nhay
           den False_code
66      li $t5, 93
67      beq $t2, $t5, False_code          # Neu ky tu la ']', nhay
           den False_code
68      li $t5, 94
69      beq $t2, $t5, False_code          # Neu ky tu la '^', nhay
           den False_code
70      li $t5, 96
71      beq $t2, $t5, False_code          # Neu ky tu la '`', nhay
           den False_code
72      slti $t4, $t2, 123
73      beq $t4, $zero, False_code        # Neu ky tu lon hon 'z' (ma
            ASCII 122), nhay den False_code
74
75
76      addi    $t1, $t1, 1
77      j   Loop_Label
78
79  #-----------------------------------------------------
80  True_Label:
81      jal     Print_Right_Label
82      j   Right_code
```

Giải thích Giải thích

- **Check_Label**
  - Khởi tạo giá trị ban đầu:
    - $t1 được đặt về 0 để bắt đầu đếm từ ký tự đầu tiên của chuỗi cần kiểm tra.
    - Lấy ký tự đầu tiên từ địa chỉ `$s2` (hàng đợi) và lưu vào `$t2` để kiểm tra.
  - Kiểm tra ký tự đầu tiên:
    - Nếu ký tự đầu tiên là `NULL` hoặc `newline`, chương trình kết thúc với `False_code` vì nhãn không hợp lệ.

    * Kiểm tra ký tự đầu tiên có nằm trong dải ký tự số hoặc ký tự hợp lệ cho nhãn hay không. Nếu không hợp lệ, nhảy đến `False_code`.

- **Loop_Label**
  - Vòng lặp này tiếp tục kiểm tra các ký tự tiếp theo trong chuỗi.
  - Lấy ký tự tiếp theo từ hàng đợi và lưu vào `$t2`.
  - Nếu ký tự là `NULL` hoặc `newline`, nhảy đến `True_Label`.
  - Kiểm tra ký tự có nằm trong dải ký tự số hoặc ký tự hợp lệ cho nhãn hay không. Nếu không hợp lệ, nhảy đến `False_code`.
  - Tăng biến đếm `$t1` và tiếp tục vòng lặp để kiểm tra ký tự tiếp theo.
- **True_Label**
  - Khi tất cả các ký tự của nhãn đều hợp lệ, hàm `Print_Right_Label` được gọi để in thông báo nhãn hợp lệ.
  - Chương trình tiếp tục với nhãn `Right_code`.

2.3.9 Tách và kiểm tra cấu trúc đặc biệt (lw,sw,lb,sb,lh,sh)

a. Tách

```
    #-----------------------------------------------------
#Tach Sign ExtImm
Split_Sign_ExtImm:
    li  $s0, 0      #Vi tri phan tu cuoi cua mang chain_check
    add $t1, $s5, $zero #i=vi tri dang doc trong cau lenh=s5
Loop1_Sign:
    add $a2, $s1, $t1   #a2 = Dia chi cua ky tu dang load
    add $a3, $s2, $s0   #a3 = Dia chi dang nap vao hang doi
    lb  $t0, 0($a2) #t0 = Ky tu dang Load
    add $t9, $zero, $t0 #t9 = Ky tu cuoi cung duoc load
    beq $t0, $zero, EndLoop_Sign_them_2#Check_Reg_and_Num   #Gap
        null => ket thuc vong lap 1
    li  $t5, 10     #t5=10~'New line'
    beq $t0, $t5, EndLoop_Sign_them_2
    beq $t0, $s6, Loop1_Sign_them   #Gap Space -> Chay qua Space
    li  $t5, 44     #t5=44~'dau phay,'
    beq $t0, $t5, False_code
    sb  $t0, 0($a3)         #Nap ky tu vao hang doi
    li  $t5, 40         #Thay dau ( thi ket thuc
    beq $t0, $t5, EndLoop_Sign_them
    li  $t5, 41         #Thay dau ) thi ket thuc
    beq $t0, $t5, EndLoop_Sign_them_3
    addi    $s0, $s0, 1     #Dich chuyen vi tri cuoi cua hang
       doi sang phai
    addi    $t1, $t1, 1
Loop2_Sign:
    add $a2, $s1, $t1   #a2 = Dia chi cua ky tu dang load
    add $a3, $s2, $s0   #a3 = Dia chi dang nap vao hang doi
    lb  $t0, 0($a2)
    add $t9, $zero, $t0 #t9 = Ky tu cuoi cung duoc load
    beq $t0, $zero, EndLoop_Sign_them_2#Check_Reg_and_Num   #Gap
        null => ket thuc vong lap 1
```

```
    li   $t5, 10      #t5=10~'New line'
    beq $t0, $t5, EndLoop_Sign_them_2
    beq $t0, $s6, EndLoop_Sign  #Gap space => Chay qua Space
    li   $t5, 10          #t5=newline
    beq $t0, $t5, EndLoop_Sign  #Check_Reg_and_Num  #Gap newline
        => ket thuc vong lap 1
    li   $t5, 44          #t5=44~'dau phay,'
    beq $t0, $t5, EndLoop_Sign  #Gap dau phay => ket thuc vong
       lap 1
    li   $t5, 40          #Thay dau ( thi ket thuc
    beq $t0, $t5, EndLoop_Sign_them_1
    li   $t5, 41          #Thay dau ) thi ket thuc
    beq $t0, $t5, EndLoop_Sign_them_1
    sb   $t0, 0($a3)          #Nap ky tu vao hang doi
    addi     $s0, $s0, 1      #Dich chuyen vi tri cuoi cua hang
       doi sang phai
    addi     $t1, $t1, 1
    j    Loop2_Sign
EndLoop_Sign:
    #Chen ky tu NULL cho hang doi
    sb   $zero, 0($a3)
    addi     $s5, $t1, 0
    addi     $s0, $s0, -1
    jr   $ra

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

Giải thích

- **Khởi tạo giá trị ban đầu**
  - `$s0`: Đặt về 0, vị trí phần tử cuối của hàng đợi.
  - `$t1`: Đặt bằng giá trị của `$s5`, vị trí đang đọc trong chuỗi lệnh.

  Vòng lặp Loop1_Sign
  - Lấy địa chỉ của ký tự đang load từ chuỗi lệnh vào `$a2`.
  - Lấy địa chỉ của ký tự đang load từ hàng đợi vào `$a3`.
  - Tải ký tự từ chuỗi lệnh vào `$t0`.
  - Lưu ký tự cuối cùng được load vào `$t9`.
  - Kiểm tra ký tự:
    - Nếu là ký tự `NULL`, nhảy đến `EndLoop_Sign_them_2`.
    - Nếu là ký tự `newline`, nhảy đến `EndLoop_Sign_them_2`.
    - Nếu là ký tự `space`, bỏ qua và tiếp tục vòng lặp.
    - Nếu là dấu phẩy, nhảy đến `False_code`.
  - Lưu ký tự vào hàng đợi.
  - Kiểm tra các ký tự đặc biệt:
    - Nếu là dấu `(`, nhảy đến `EndLoop_Sign_them`.
    - Nếu là dấu `)`, nhảy đến `EndLoop_Sign_them_3`.
  - Cập nhật vị trí cuối của hàng đợi và tiếp tục vòng lặp.
- **Vòng lặp Loop2_Sign**
  - Tương tự như `Loop1_Sign`, tiếp tục tải và xử lý các ký tự.

  - Kiểm tra ký tự:
    * Nếu là ký tự `NULL`, nhảy đến `EndLoop_Sign_them_2`.
    * Nếu là ký tự `newline`, nhảy đến `EndLoop_Sign_them_2`.
    * Nếu là ký tự `space`, kết thúc vòng lặp.
    * Nếu là ký tự `newline`, kết thúc vòng lặp.
    * Nếu là dấu phẩy, kết thúc vòng lặp.
    * Nếu là dấu `(`, nhảy đến `EndLoop_Sign_them_1`.
    * Nếu là dấu `)`, nhảy đến `EndLoop_Sign_them_1`.
  - Lưu ký tự vào hàng đợi.
  - Cập nhật vị trí cuối của hàng đợi và tiếp tục vòng lặp.

- **Kết thúc vòng lặp**
  - `EndLoop_Sign`: Thêm ký tự `NULL` vào cuối hàng đợi để đánh dấu kết thúc chuỗi.
  - Cập nhật giá trị `$s5`: Lưu vị trí ký tự đang đọc vào `$s5`.
  - Cập nhật vị trí cuối của hàng đợi: Giảm `$s0` đi 1.
  - Quay trở lại hàm chính.

b. Check Sign_ExtImm

```
   #-----------------------------------------------------
#Check Sign_ExtImm
Check_Sign_ExtImm:
    add $t8, $zero, $ra     #Luu dia chi tro ve chuong trinh vao
        -> t8
    jal Split_Sign_ExtImm   #tach va luu gia tri cua imm vao
       queue
    jal Check_Number        #kiem tra xem gia tri imm hop le
       khong
    jal Split_Sign_ExtImm   #tiep tuc xu li phan con lai cua
       chuoi lenh
    jal Check_Parentheses_1 #kiem tra ki tu '('
    jal Split_Sign_ExtImm   #tiep tuc xu li phan con lai cua
       chuoi lenh
    jal Check_Register      #Kiem tra xem thanh ghi co hop le
       khong
    jal Split_Sign_ExtImm   #tiep tuc xu li phan con lai cua
       chuoi lenh
    jal Check_Parentheses_2 #xu li ky tu ')'
    addi    $t5, $zero, 10      #10: newline
    beq $t9, $t5, Right_code
    addi    $t5, $zero, 0
    beq $t9, $t5, Right_code
    addi    $t5, $zero, 41      #t5 ~ ')'
    beq $t9, $t5, Right_code
    j   False_code

#Check_Parentheses_1   Kiem tra dau (
Check_Parentheses_1:
    li  $t1, 0  #i = 0
    add $a2, $s2, $t1
    lb  $t2, 0($a2)
    li  $t5, 40
```

```
27      bne $t2, $t5, False_code
28      addi    $t1, $t1, 1
29      add $a2, $s2, $t1
30      lb  $t2, 0($a2)
31      bne $zero, $t2, False_code
32      jr  $ra
33
34  #Check_Parentheses_2   Kiem tra dau )
35  Check_Parentheses_2:
36      li  $t1, 0  #i = 0
37      add $a2, $s2, $t1
38      lb  $t2, 0($a2)
39      li  $t5, 41
40      bne $t2, $t5, False_code
41      addi    $t1, $t1, 1
42      add $a2, $s2, $t1
43      lb  $t2, 0($a2)
44      bne $zero, $t2, False_code
45      jr  $ra
```

Giải thích hàm Check_Sign_ExtImm

- **Lưu địa chỉ trở về:**
  - Giá trị của thanh ghi `$ra` (địa chỉ trở về) được lưu vào thanh ghi tạm thời `$t8`.
- **Gọi các hàm phụ trợ:**
  - Gọi hàm `Split_Sign_ExtImm` để tách và lưu giá trị của Immediate (số mở rộng dấu) vào hàng đợi.
  - Gọi hàm `Check_Number` để kiểm tra xem giá trị Immediate có hợp lệ không.
  - Gọi lại hàm `Split_Sign_ExtImm` để tiếp tục xử lý phần còn lại của chuỗi lệnh.
  - Gọi hàm `Check_Parentheses_1` để kiểm tra ký tự `'('`.
  - Gọi lại hàm `Split_Sign_ExtImm` để tiếp tục xử lý phần còn lại của chuỗi lệnh.
  - Gọi hàm `Check_Register` để kiểm tra xem thanh ghi có hợp lệ không.
  - Gọi lại hàm `Split_Sign_ExtImm` để tiếp tục xử lý phần còn lại của chuỗi lệnh.
  - Gọi hàm `Check_Parentheses_2` để kiểm tra ký tự `')'`.
- **Kiểm tra ký tự cuối cùng:**
  - Kiểm tra ký tự cuối cùng được lưu trong `$t9`.
  - Nếu ký tự cuối cùng là newline (`10`), nhảy đến `Right_code`.
  - Nếu ký tự cuối cùng là NULL (`0`), nhảy đến `Right_code`.
  - Nếu ký tự cuối cùng là dấu `')'` (`41`), nhảy đến `Right_code`.
  - Nếu không, nhảy đến `False_code`.

2.3.10 In ra kết quả

Các xâu kí tự từ `Message 1` - `Message 11` đã được khai báo và giải thích ở mục **2.3.1 Khai báo dữ liệu.**

a. In opcode

Trường hợp opcode đúng thoả mãn yêu cầu đề bài

```
    False_opcode:
    #Print "Opcode"
    li $v0, 4
    la $a0, Message2
    syscall
    nop
    #Print Opcode Input
    li $v0, 4
    add $a0, $zero, $s2
    syscall
    nop
    #Print "Khong hop le!"
    li $v0, 4
    la $a0, Message4
    syscall
    nop
    jal False_code
    j   End_main
```

Trường hợp opcode sai

```
    Right_opcode:
    #Print "Opcode"
    li $v0, 4
    la $a0, Message2
    syscall
    nop
    #Print Opcode Input
    li $v0, 4
    add $a0, $zero, $s2
    syscall
    nop
    #Print ", hop le!"
    li $v0, 4
    la $a0, Message3
    syscall
    nop
    jr  $ra
```

b. In thanh ghi, số, nhãn label

Ta xét các trường hợp thanh ghi, số, nhãn thoả mãn đề bài.

```
    Right_Register:
    #Print "\n"
    li $v0, 4
    la $a0, Message7
    syscall
    nop
```

```
    #Print "Thanh ghi"
    li $v0, 4
    la $a0, Message8
    syscall
    nop
    #Print Register Input
    li $v0, 4
    add $a0, $zero, $s2
    syscall
    nop
    #Print ", hop le!"
    li $v0, 4
    la $a0, Message3
    syscall
    nop
    jr  $ra
Print_Right_Number:
    #Print "\n"
    li $v0, 4
    la $a0, Message7
    syscall
    nop
    #Print "So "
    li $v0, 4
    la $a0, Message9
    syscall
    nop
    #Print so trong hang doi
    li $v0, 4
    add $a0, $zero, $s2
    syscall
    nop
    #Print ", hop le!"
    li $v0, 4
    la $a0, Message3
    syscall
    nop
    jr  $ra
Print_Right_Label:
    #Print "\n"
    li $v0, 4
    la $a0, Message7
    syscall
    nop
    #Print "So "
    li $v0, 4
    la $a0, Message10
    syscall
    nop
    #Print label trong hang doi
    li $v0, 4
    add $a0, $zero, $s2
    syscall
```

```
    nop
    #Print ", hop le!"
    li $v0, 4
    la $a0, Message3
    syscall
    nop
    jr  $ra
```

c. In khẳng định câu lệnh đúng/ sai

```
    Right_Register:
    #Print "\n"
    li $v0, 4
    la $a0, Message7
    syscall
    nop
    #Print "Thanh ghi"
    li $v0, 4
    la $a0, Message8
    syscall
    nop
    #Print Register Input
    li $v0, 4
    add $a0, $zero, $s2
    syscall
    nop
    #Print ", hop le!"
    li $v0, 4
    la $a0, Message3
    syscall
    nop
    jr  $ra
Print_Right_Number:
    #Print "\n"
    li $v0, 4
    la $a0, Message7
    syscall
    nop
    #Print "So "
    li $v0, 4
    la $a0, Message9
    syscall
    nop
    #Print so trong hang doi
    li $v0, 4
    add $a0, $zero, $s2
    syscall
    nop
    #Print ", hop le!"
    li $v0, 4
    la $a0, Message3
    syscall
```

```
    nop
    jr  $ra
Print_Right_Label:
    #Print "\n"
    li $v0, 4
    la $a0, Message7
    syscall
    nop
    #Print "So "
    li $v0, 4
    la $a0, Message10
    syscall
    nop
    #Print label trong hang doi
    li $v0, 4
    add $a0, $zero, $s2
    syscall
    nop
    #Print ", hop le!"
    li $v0, 4
    la $a0, Message3
    syscall
    nop
    jr  $ra
```

d. Đưa ra câu hỏi và lặp lại chương trình

```
    Run_Again:  li $v0, 50
        la $a0, Message11
        syscall
        nop
        beq $a0, $zero, clear
        nop
        j exit
        nop
# clear: dua string ve trang thai ban dau de thuc hien lai qua
   trinh
clear:      add $s3, $zero, $s1
Loop_Null:
        lb  $t3, 0($s3)
        li  $t5, 10
        beq $t3, $t5, Loop_Null_them
        nop
        sb  $zero, 0($s3)
        addi    $s3, $s3, 1
        j   Loop_Null
Loop_Null_them:
        sb  $zero, 0($s3)
        j start
        nop
```

Giải thích

- **Run_Again:**
  - Hiển thị thông báo hỏi người dùng có muốn tiếp tục kiểm tra lệnh khác hay không.
  - Nếu người dùng nhập 0 (nghĩa là không muốn tiếp tục), chương trình nhảy đến nhãn `clear`.
  - Nếu không, chương trình kết thúc.
- **clear:**
  - Xóa nội dung của chuỗi lệnh đã kiểm tra bằng cách thay thế từng ký tự trong chuỗi bằng `NULL`.
- **Loop_Null:**
  - Vòng lặp xóa các ký tự trong chuỗi lệnh. Nếu gặp ký tự newline, nhảy đến nhãn `Loop_Null_them` để xóa ký tự newline và sau đó quay lại đầu chương trình (`start`) để người dùng nhập lệnh mới.

Đoạn mã này thực hiện chức năng hỏi người dùng có muốn tiếp tục kiểm tra lệnh khác hay không. Nếu người dùng chọn tiếp tục, chương trình sẽ xóa nội dung của chuỗi lệnh đã kiểm tra và quay lại từ đầu để người dùng nhập lệnh mới. Nếu người dùng chọn không tiếp tục, chương trình sẽ kết thúc.

2.4 KẾT QUẢ

Dạng lệnh R (Register)

Hợp lệ

- `add $t1, $t2, $t3`
  Thêm giá trị của thanh ghi `$t2` và `$t3`, lưu kết quả vào `$t1`.

Không hợp lệ

- `add $t1, $t2, $s32`
  Lỗi vì `$s32` không phải là thanh ghi hợp lệ.
- `mul $t1 $t2 $t3`
  Lỗi vì thiếu dấu phẩy giữa các tham số.

Hình 2.2: Kết quả chạy 1 số câu lệnh loại R

Dạng lệnh I (Immediate)

Hợp lệ

- `addi $t1, $t2, 10`
  Thêm giá trị tức thời `10` vào giá trị của thanh ghi `$t2`, lưu kết quả vào `$t1`.

Không hợp lệ

- `addi $t1, $t2, 0xG`
  Lỗi vì `0xG` không phải là giá trị tức thời hợp lệ.
- `addi $t1, $t2 10`
  Lỗi vì thiếu dấu phẩy giữa các tham số.

Hình 2.3: Kết quả chạy 1 số câu lệnh loại I

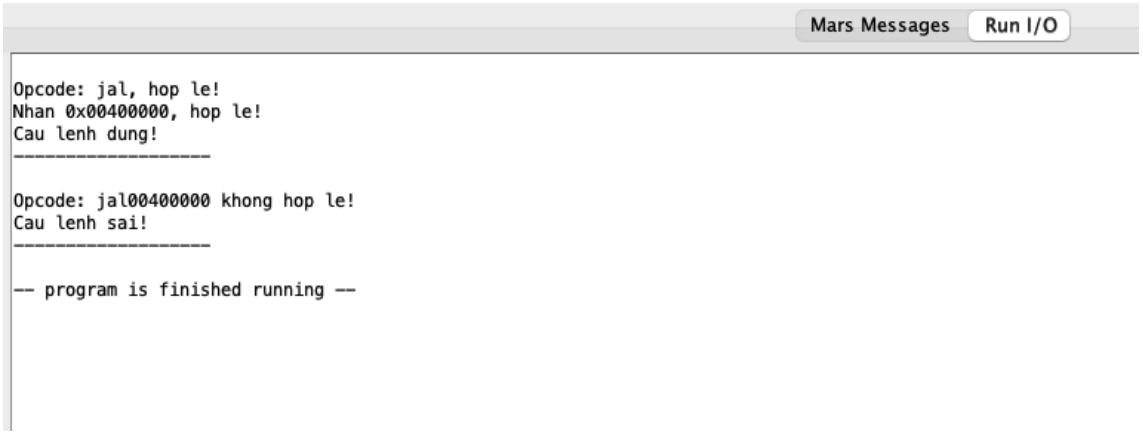
Dạng lệnh J (Jump)

Hợp lệ

- `jal 0x00400000`
  Nhảy tới địa chỉ `0x00400000` và lưu địa chỉ của lệnh tiếp theo vào thanh ghi `$ra`.

Không hợp lệ

- `jal00400000`
  Lỗi vì thiếu khoảng cách giữa opcode và địa chỉ.



Hình 2.4: Kết quả chạy 1 số câu lệnh loại J

Dạng lệnh L (Load/Store)

Hợp lệ

- `lw $t1, 4($t2)`
  Tải giá trị từ địa chỉ tính bằng giá trị của thanh ghi `$t2` cộng với `4` vào thanh ghi `$t1`.

Không hợp lệ

- `lw $t1 $t2 4`
  Lỗi vì thiếu dấu phẩy và định dạng không đúng.
- `sw $s1, ($s2)8`
  Lỗi vì thứ tự các tham số không đúng.

Hình 2.5: Kết quả chạy 1 số câu lệnh loại L

Dạng lệnh đặc biệt (Special)

Hợp lệ

- `syscall`
  Gọi hệ thống.
- `nop`
  Không làm gì (*no operation*).

Hình 2.6: Kết quả chạy 1 số câu lệnh đặc biệt

2.5 SOURCE CODE FULL