Đại học Bách Khoa Hà Nội
Trường Công nghệ Thông tin và Truyền thông

Báo cáo Project kiến trúc máy tính

Nhóm: 12
Giáo viên hướng dẫn: Lê Bá Vui
Thành viên nhóm:

STT.	Họ và tên	Mã số sinh viên	Bài làm
1	Nguyễn Kim Ngọc	20225655	Project 10
2	Hồ Đức Tú	20225676	Project 2

Mục lục

1 Đề tài 10 – Nguyễn Kim Ngọc

1.1 Mô tả bài toán

Sử dụng 2 ngoại vi là bàn phím keypad và led 7 thanh để xây dựng một máy tính bỏ túi đơn giản. Hỗ trợ các phép toán + - * / % với các toán hạng là số nguyên. Do trên bàn phím không có các phím trên nên sẽ dùng các phím:

- Bấm phím a để nhập phép tính +
- Bấm phím b để nhập phép tính –
- Bấm phím c để nhập phép tính *
- Bấm phím d để nhập phép tính /
- Bấm phím e để nhập phép tính %
- Bấm phím f để nhập phép =

Yêu cầu cụ thể như sau:

- Khi nhấn các phím số hiển thị lên LED do chỉ có 2 LED nên chỉ hiển thị 2 số cuối cùng. Ví dụ khi nhấn phím 1 → hiển thị 01. Khi nhấn thêm phím 2 → hiển thị 12. Khi nhấn thêm phím 3 → hiển thị 23.
- Sau khi nhập số sẽ nhập phép tính + - * / %
- Sau khi nhấn phím f (dấu =) tính toán và hiển thị kết quả lên LED.
- Có thể thực hiện các phép tính liên tiếp (tham khảo ứng dụng Calculator trên hệ điều hành Windows)

1.2 Hướng dẫn chạy chương trình

1. Mở công cụ Digital Lab Sim Connect to MIPS và chạy chương trình.
2. Nhập các số bằng cách nhập từ các phím trên công cụ Digital Lab Sim và màn hình led sẽ hiển thị 2 chữ số cuối.
3. Nhập xong số đầu tiên thì nhập toán tử + - * % tương ứng với a b c d e hiện trên phím của công cụ Digital Lab Sim.
4. Nhập số thứ hai.
5. Nhấn f (tương đương dấu '=') để hiển thị 2 chữ số cuối cùng của kết quả
6. Để thực hiện tiếp phép toán chỉ cần nhập toán tử sau đó nhập số thứ 2 và ấn phím f (tương đương dấu '=') để hiển thị 2 chữ số cuối của kết quả tiếp theo.
7. Để thực hiện phép toán mới chúng ta nhấn phím a 1 lần nữa để RESET (với máy tính caculator trên Windows thì có nút C và CE còn công cụ Digital Lab Sim thì không có phím nào nên mình dùng cách này).

1.3 Thuật toán

Khởi Tạo

- Tải các địa chỉ cho các đèn LED bên trái và phải và bàn phím thập lục phân.
- Khởi tạo các biến cho loại đầu vào, giá trị LED, toán tử số và lưu trữ tạm thời.
- Kích hoạt ngắt bàn phím và kiểm tra từng hàng bàn phím để nhận đầu vào.

Vòng Lặp Chính (Chờ Đầu Vào)

- Vào vòng lặp vô hạn (Loop1) chờ ngắt từ đầu vào bàn phím.

Xử Lý Ngắt

- Khi có ngắt, kiểm tra từng hàng của bàn phím để xác định phím nào được nhấn.
- Chuyển đổi phím nhấn thành giá trị hoặc toán tử tương ứng.

Chuyển Đổi Phím Nhấn Thành Giá Trị/Toán Tử

- Hàng 1: Chuyển đổi các phím 0-3 thành các giá trị LED tương ứng và lưu chữ số.
- Hàng 2: Chuyển đổi các phím 4-7 thành các giá trị LED tương ứng và lưu chữ số.
- Hàng 3: Chuyển đổi các phím 8-9 và '+' thành các giá trị LED tương ứng và lưu chữ số hoặc đặt toán tử là cộng.
- Hàng 4: Chuyển đổi các phím '-' '*' '/' và '

Đặt Số Thứ Nhất

- Lưu số thứ nhất sau khi chữ số đầu tiên được nhập và đặt lại lưu trữ tạm thời.

Đặt Số Thứ Hai

- Lưu số thứ 2 sau khi ấn phím 'f' (tương đương với dấu =)

Thực Hiện Phép Toán

- Sau khi đặt toán tử và số thứ hai, thực hiện phép toán tương ứng:
  - Cộng: Cộng hai số.
  - Trừ: Trừ số thứ hai từ số thứ nhất.
  - Nhân: Nhân hai số.
  - Chia: Chia số thứ nhất cho số thứ hai (xử lý chia cho 0).
  - Modulus: Tính phần dư của phép chia (xử lý modulus cho 0).

Hiển Thị Kết Quả

- In kết quả của phép tính ra màn hình.
- Chuyển đổi kết quả thành hai chữ số để hiển thị trên các đèn LED bên trái và phải.

Đặt Lại và Lặp Lại

- Đặt lại các biến cần thiết và lặp lại để chờ đầu vào tiếp theo.

1.4 Giải thích mã nguồn

- Khai báo:

```
.eqv    SEVENSEG_LEFT   0xFFFF0011       # Địa chỉ của LED trái
.eqv    SEVENSEG_RIGHT  0xFFFF0010       # Địa chỉ của LED phải
.eqv    IN_ADDRESS_HEXA_KEYBOARD         0xFFFF0012
.eqv    OUT_ADDRESS_HEXA_KEYBOARD        0xFFFF0014

.data
# Giá trị tương ứng với LED 7
zero:           .byte 0x3F
one:            .byte 0x06
two:            .byte 0x5B
three:          .byte 0x4F
four:           .byte 0x66
five:           .byte 0x6D
six:            .byte 0x7D
seven:          .byte 0x07
eight:          .byte 0x7F
nine:           .byte 0x6F

mess1:  .asciiz         "Can not divide by 0\n"
```

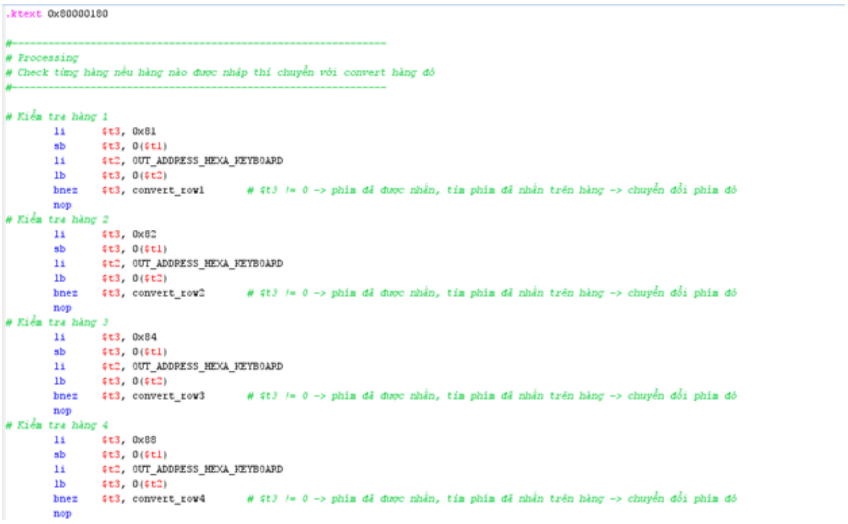
- Khởi tạo:

```
Init:
        li  $t0, SEVENSEG_LEFT          # Biến chứa giá trị của LED bên trái
        li  $t5, SEVENSEG_RIGHT         # Biến chứa giá trị của LED bên phải
        li  $s0, 0                      # Biến chứa loại đầu vào: (0: chữ số), (1: toán tử)
        li  $s1, 0                      # Biến chứa giá trị trên LED trái
        li  $s2, 0                      # Biến chứa giá trị trên LED phải
        li  $s3, 0                      # Biến chứa loại toán tử (1: +, 2: -, 3: *, 4: /, 5: %)
        li  $s4, 0                      # Số thứ nhất
        li  $s5, 0                      # Số thứ hai
        li  $s6, 0                      # Kết quả
        li  $t9, 0                      # Giá trị tạm thời

        li  $t1, IN_ADDRESS_HEXA_KEYBOARD   # Biến điều khiển hàng bàn phím và kích hoạt ngắt bàn phím
        li  $t2, OUT_ADDRESS_HEXA_KEYBOARD  # Biến chứa vị trí các phím
        li  $t3, 0x80                 # Bit dùng để kích hoạt ngắt bàn phím và kiểm tra từng hàng bàn phím
        sb  $t3, 0($t1)
        li  $t7, 0                      # Biến chứa giá trị của số trên LED
        li  $t4, 0                      # Byte hiển thị trên LED (0->9)

First_value:
        li  $t7, 0                # Giá trị cần được hiển thị bit ban đầu
        addi $sp, $sp, 4          # Đẩy lên stack
        sb  $t7, 0($sp)           # Lưu giá trị $t7 vào stack
        lb  $t4, zero             # Đọc bit đầu tiên cần được hiển thị
        addi $sp, $sp, 4          # Đẩy lên stack
        sb  $t4, 0($sp)           # Lưu giá trị $t4 vào stack
```

- Tạo vòng lặp vô hạn và chờ interrupt khi nhập ký tự vào:

```
Loop1:
        nop
        nop
        nop
        nop
        b       Loop1           # Wait for interrupt
        nop
        nop
        nop
        nop
        b       Loop1
        nop
        nop
        nop
        nop
        b       Loop1
end_loop1:
```

- Khi nhập vào thì kiểm tra coi phím đấy thuộc hàng nào rồi tiếp đến bước convert phím (bit) ra thành chữ số



```
.ktext 0x80000180

#-------------------------------------------------------------
# Processing
# Check từng hàng nếu hàng nào được nhấp thì chuyển với convert hàng đó
#-------------------------------------------------------------

# Kiểm tra hàng 1
        li      $t3, 0x81
        sb      $t3, 0($t1)
        li      $t2, OUT_ADDRESS_HEXA_KEYBOARD
        lb      $t3, 0($t2)
        bnez    $t3, convert_row1       # $t3 != 0 -> phím đã được nhấn, tìm phím đã nhấn trên hàng -> chuyển đổi phím đó
        nop
# Kiểm tra hàng 2
        li      $t3, 0x82
        sb      $t3, 0($t1)
        li      $t2, OUT_ADDRESS_HEXA_KEYBOARD
        lb      $t3, 0($t2)
        bnez    $t3, convert_row2       # $t3 != 0 -> phím đã được nhấn, tìm phím đã nhấn trên hàng -> chuyển đổi phím đó
        nop
# Kiểm tra hàng 3
        li      $t3, 0x84
        sb      $t3, 0($t1)
        li      $t2, OUT_ADDRESS_HEXA_KEYBOARD
        lb      $t3, 0($t2)
        bnez    $t3, convert_row3       # $t3 != 0 -> phím đã được nhấn, tìm phím đã nhấn trên hàng -> chuyển đổi phím đó
        nop
# Kiểm tra hàng 4
        li      $t3, 0x88
        sb      $t3, 0($t1)
        li      $t2, OUT_ADDRESS_HEXA_KEYBOARD
        lb      $t3, 0($t2)
        bnez    $t3, convert_row4       # $t3 != 0 -> phím đã được nhấn, tìm phím đã nhấn trên hàng -> chuyển đổi phím đó
        nop
```

- Thực hiện convert từ địa chỉ sang số: Nếu ký tự nhập vào là chữ số (1 - 9) thì: (ví dụ hàng 1: 0 - 3)

```
convert_row1:
        beq     $t3, 0x11, case_0               # Số 0
        beq     $t3, 0x21, case_1               # Số 1
        beq     $t3, 0x41, case_2               # Số 2
        j case_3                # Số 3
case_0:
        lb      $t4,zero        # t4 = 0x66   với $t4 là byte và $t7 là giá trị
        li      $t7,0           # t7 = 0
        j       updateDigit
case_1:
        lb      $t4,one
        li      $t7,1
        j       updateDigit
case_2:
        lb      $t4,two
        li      $t7,2
        j       updateDigit
case_3:
        lb      $t4, three
        li      $t7, 3
        j       updateDigit
```

- updateDigit là hàm cập nhật số nhập vào cho đến khi gặp ký tự và in ra màn hình

```
updateDigit:
        mul     $t9, $t9, 10
        add     $t9, $t9, $t7

# Chiếu xong 1 số -> reset LED
done:
        beq $s0,1,reset_led             # Kiểm tra xem có phải toán tử k, nếu là toán tử thì reset_led để gõ số tiếp theo
        nop
```

- Thực hiện chiếu LED và chờ nhập ký tự tiếp theo:

```
# Hàm chiếu LED trái
load_to_left_led:
        lb      $t6, 0($sp)             # Lấy bit từ stack
        add     $sp, $sp, -4
        lb      $t8, 0($sp)             # Lấy giá trị từ stack
        add     $sp, $sp, -4
        add     $s2, $t8, $zero         # s2 = value of LEFT LED
        sb      $t6, 0($t0)             # Show LEFT LED

# Hàm chiếu LED phải
load_to_right_led:
        sb      $t4, 0($t5)             # in ra số vừa nhập
        add     $sp, $sp,4              # Đưa số vừa nhập: địa chỉ và giá trị vào stack
        sb      $t7, 0($sp)
        add     $sp, $sp,4
        sb      $t4, 0($sp)
        add     $s1, $t7, $zero
        j       finish
```

- Hàm quay lại vòng lặp:

```
return:
        la      $a3, Loop1
        mtc0    $a3, $14
        eret
```

- Nếu ký tự nhập vào là toán tử (a - f) thì: (ví dụ phép cộng)

```
# Case a: Cộng
case_a:
        addi    $s0, $s0, 1             # s0 = 1, Để chỉ ra đây là 1 toán tử
        addi    $s3, $zero, 1           # s3 = 1 -> Cộng

        j       set_first_number        # Di chuyển đến hàm set số đầu tiên
```

- Nhảy tới hàm set số đầu tiên:

```
# Convert the number on LED -> value of the first number
set_first_number:
        addi    $s4, $t9, 0 # số thứ nhất được đưa vào $s4
        li      $t9, 0
        j       done
```

- Nếu là phím f thì thực hiện phép toán và in ra kết quả:

```
# Case f:
case_f:
        addi $s5, $t9, 0 # Số thứ hai được đưa vào $s5
```

- Thực hiện phép toán và in ra kết quả:

```
# Thực hiện phép toán và đưa ra kết quả
process:
        beq     $s3, 1, addition
        beq     $s3, 2, substraction
        beq     $s3, 3, multiplication
        beq     $s3, 4, division
        beq     $s3, 5, find_remainder

addition:                       # Thực hiện phép trừ
        add     $s6, $s5, $s4
        li      $s3, 0
        li      $t9, 0
        j       print_addition
        nop
```

- In ra phép tính:

```
print_addition:                 # Hàm dùng để in ra phép tính trừ
       li      $v0, 1
       move    $a0, $s4
       syscall
       li      $s4, 0           # Reset số đầu về 0

       li      $v0, 11
       li      $a0, '+'
       syscall

       li      $v0, 1
       move    $a0, $s5
       syscall
       li      $s5, 0           # Reset số sau về 0

       li      $v0, 11
       li      $a0, '='
       syscall

       li      $v0, 1
       move    $a0, $s6
       syscall
       nop

       li      $v0, 11
       li      $a0, '\n'
       syscall

       li      $s7, 100
       div     $s6, $s7
       move    $s7, $s6         # Lưu giá trị kết quả vào $s7
       mfhi    $s6              # Xuất ra 2 số cuối vào $s6
       j show_result_in_led     # Di chuyển đến hàm chiếu kết quả
       nop
```

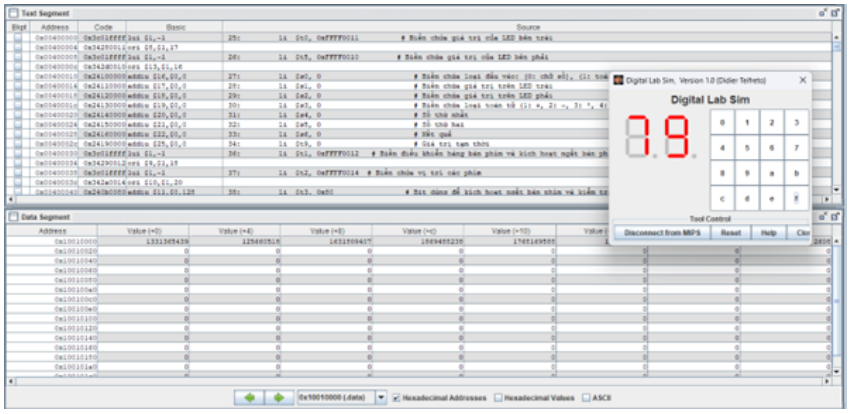
- Tương tự với các trạng thái còn lại và cuối cùng là in kết quả.

1.5 Kết quả chạy mô phỏng

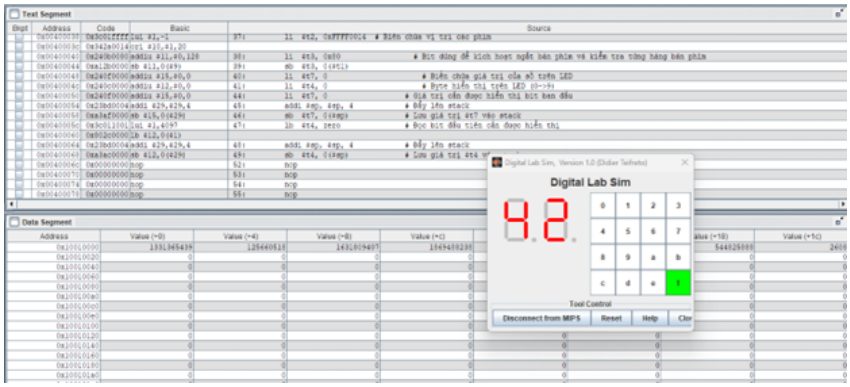
1.5.1 Trường hợp cơ bản:

- 123 + 456 = 579 (kết quả in ra màn hình số 79)

```
123+456=579
```

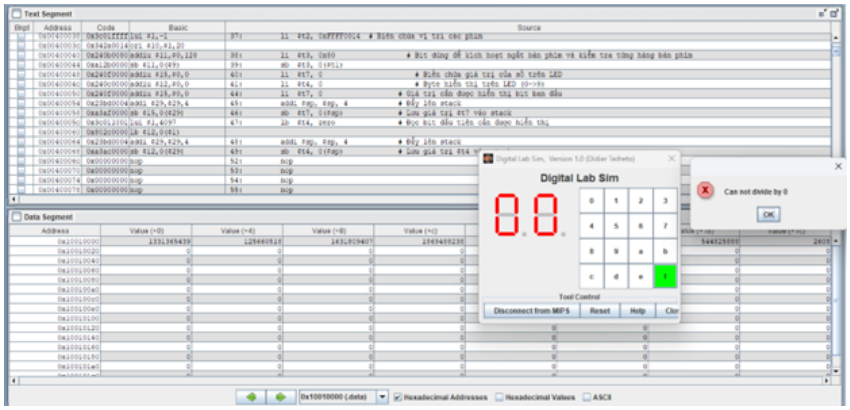


- 66123 * 54 = 3570642 in ra màn hình số 42

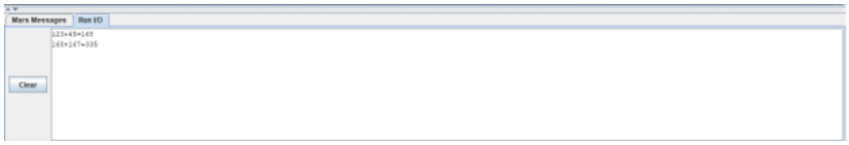


1.5.2 Trường hợp nâng cao:

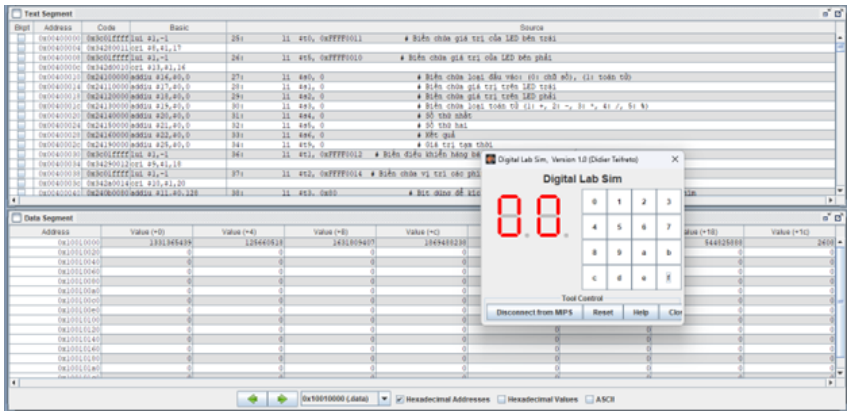
- Chia cho 0: In ra "can not divide by 0"



- 135 + 45 = 168 168 + 167 = 335 (in ra số 35)



- Trừ ra số âm: 23 – 45 = -22 Hiển thị ra số 0



2 Đề tài 2 – Hồ Đức Tú

2.1 Mô tả bài toán

Vị trí mặc định là trung tâm của màn hình. Tạo một chương trình hiển thị một quả bóng tròn có thể di chuyển trên màn hình bitmap. Nếu quả bóng chạm vào mép màn hình nó sẽ di chuyển theo hướng ngược lại. Yêu cầu:

- Đặt chiều rộng và chiều cao màn hình là 512 pixel đơn vị chiều rộng và chiều cao là 1 pixel.
- Giả sử ban đầu quả bóng di chuyển theo chiều từ trái sang phải, từ trên xuống dưới.
- Sau một khoảng thời gian T (ms), cập nhật vị trí của quả bóng.
- Vẽ quả bóng tại vị trí mới.
- Quả bóng di chuyển ngược lại khi chạm vào mép màn hình.

2.2 Giải thích thuật toán và code

Mô tả thuật toán:

- Đặt biến khởi tạo cho vị trí ban đầu của quả bóng (giả sử ở chính giữa màn hình).
- Đặt các thông số khởi tạo khác như chiều rộng và chiều cao của màn hình (512 x 512 pixel).
- Đặt hướng di chuyển ban đầu của quả bóng.

Vòng lặp chính (chạy mã Assembly):

- Vẽ quả bóng tại vị trí hiện tại.

- Sau khoảng thời gian T (ms), cập nhật vị trí mới của quả bóng.
- Nếu quả bóng chạm vào mép màn hình (x hoặc y vượt quá giới hạn của màn hình), thay đổi hướng di chuyển của quả bóng.
- Quay lại bước 1.

2.3 Giải thích mã nguồn

- Khởi tạo các giá trị ban đầu của hình tròn

```
main:
        li $t0, MONITOR_SCREEN
        addi $s0, $zero, 2048 # offset 1 line 512*4
        addi $t3, $zero, 235 #vị trí giữa màn hình, x=235 và y=235

        li $k0, KEY_CODE
        li $k1, KEY_READY

        mult $s0, $t3
        mflo $t4
        add $t0, $t0, $t4

        sll $t4, $t3, 2
        add $t0, $t0, $t4

        #tọa độ biên trên dưới, trái phải của hình tròn
        addi $s4, $zero, 236    # xA
        addi $s5, $zero, 236    # yA
        addi $s6, $zero, 269    # xB
        addi $s7, $zero, 269    # yB

        add $t8, $t0, 0
        li $t1, YELLOW
        jal drawing_circle

        addi $t9, $zero, 0      # Huong di chuyen (1, 2, 3, 4) = (len, xuong, trai, phai)
```

- Nghỉ giữa hai lần hình tròn di chuyển, tăng giảm thời gian nghỉ để tăng giảm tốc độ của hình tròn

```
moving:
    lw $t1, speed
    li $v0, 32
    move_sleep:
        li $a0, 1
        beq $t1, $zero, sleep_done
        syscall
        nop
        subi $t1, $t1, 1
        j move_sleep
    sleep_done:
    lw $t1, 0($k1)
    beq $t1, $zero, continue
```

- Xác định hướng di chuyển của hình tròn và chạy hàm tương ứng

```
teqi $t1, 1

    continue:
    beq $t9, 1, up
    beq $t9, 2, down
    beq $t9, 3, left
    beq $t9, 4, right
    nop
    j moving
```

- Di chuyển hình tròn đi lên bằng cách xóa hình tròn ở vị trí cũ bằng màu đen, di chuyển sang vị trí mới và vẽ lại bằng màu vàng.

```
up:
add $t0, $t8, 0
li $t1, BLACK
jal drawing_circle
add $t0, $t8, 0
sub $t0, $t0, $s0
sub $t0, $t0, $s0
add $t8, $t0, 0
li $t1, YELLOW
jal drawing_circle
nop
addi $s5, $s5, -2
addi $s7, $s7, -2
slti $t1, $s5, 3
bne $t1, $zero, to_down
j moving
to_down:
addi $t9, $zero, 2
j moving
```

- Di chuyển hình tròn đi xuống bằng cách xóa hình tròn ở vị trí cũ bằng màu đen, di chuyển sang vị trí mới và vẽ lại bằng màu vàng.

```
down:
add $t0, $t8, 0
li $t1, BLACK
jal drawing_circle
add $t0, $t8, 0
add $t0, $t0, $s0
add $t0, $t0, $s0
add $t8, $t0, 0
li $t1, YELLOW
jal drawing_circle
nop
addi $s5, $s5, 2
addi $s7, $s7, 2
slti $t1, $s7, 511
beq $t1, $zero, to_up
j moving
to_up:
addi $t9, $zero, 1
j moving
```

- Di chuyển hình tròn sang trái bằng cách xóa hình tròn ở vị trí cũ bằng màu đen, di chuyển sang vị trí mới và vẽ lại bằng màu vàng.

```
left:
add $t0, $t8, 0
li $t1, BLACK
jal drawing_circle
add $t0, $t8, 0
addi $t0, $t0, -8
add $t8, $t0, 0
li $t1, YELLOW
jal drawing_circle
nop
addi $s4, $s4, -2
addi $s6, $s6, -2
slti $t1, $s4, 3
bne $t1, $zero, to_right
j moving
to_right:
addi $t9, $zero, 4
j moving
```

- Di chuyển hình tròn sang phải bằng cách xóa hình tròn ở vị trí cũ bằng màu đen, di chuyển sang vị trí mới và vẽ lại bằng màu vàng.

```
right:
add $t0, $t8, 0
li $t1, BLACK
jal drawing_circle
add $t0, $t8, 0
addi $t0, $t0, 8
add $t8, $t0, 0
li $t1, YELLOW
jal drawing_circle
nop
addi $s4, $s4, 2
addi $s6, $s6, 2
slti $t1, $s6, 511
beq $t1, $zero, to_left
j moving
to_left:
addi $t9, $zero, 3
j moving
```

- Vẽ từng lớp của hình tròn (tổng cộng 33 lớp) bằng cách vẽ đường thẳng và để trống một số pixel ở giữa

```
drawing_circle:
        addi $t7, $ra, 0
        addi $s1, $zero, 13     # so pixel trong o dau
        addi $s2, $zero, 4      # so pixel can ve
        addi $s3, $zero, 0      # so pixel trong o giua
        jal drawing_line        # 1
        nop
```

- Hàm vẽ đường thẳng, cho phép bỏ qua một số pixel ở giữa đường thẳng

```
drawing_line:
        sll $t3, $s1, 2
        add $t0, $t0, $t3
        li $t2, 1
loop_draw_1:
        sw $t1, 0($t0)
        addi $t0, $t0, 4
        beq $t2, $s2, end_draw_1
        addi $t2, $t2, 1
        j loop_draw_1
end_draw_1:
        sll $t3, $s3, 2
        add $t0, $t0, $t3
        li $t2, 1
loop_draw_2:
        sw $t1, 0($t0)
        addi $t0, $t0, 4
        beq $t2, $s2, end_draw_2
        addi $t2, $t2, 1
        j loop_draw_2
end_draw_2:
        sll $t3, $s3, 2
        sub $t0, $t0, $t3
        sll $t3, $s2, 3
        sub $t0, $t0, $t3
        sll $t3, $s1, 2
        sub $t0, $t0, $t3
        add $t0, $t0, $s0
        jr $ra
```

- Khởi tạo hàm để nhận diện phím bấm và chạy hàm tương ứng.

```
.ktext 0x80000180
get_cause:
        mfc0 $t1, $13

Is_keyboar_interrupt:
        li $t2, MASK_CAUSE_KEYBOARD
        and $at, $t1, $t2
        beq $at, $t2, Keyboard_Intr
other_cause:
        nop
        j end_process

Keyboard_Intr:
        nop
        lb $t3, 0($k0)
        beq $t3, 119, up_direction      # w
        beq $t3, 115, down_direction    # s
        beq $t3, 97, left_direction     # a
        beq $t3, 100, right_direction   # d
        beq $t3, 122, increase_speed   # z
        beq $t3, 120, decrease_speed   # x
        j end_process
```

- Tăng giảm tốc độ

```
increase_speed:
lw $t4, speed
addi $t4, $t4, 1
slti $t5, $t4, 16  # Kiểm tra nếu tốc độ nhỏ hơn 16
beq $t5, $zero, end_process  # Nếu lớn hơn hoặc bằng 16 thì không thay đổi
sw $t4, speed
j end_process

decrease_speed:
lw $t4, speed
addi $t4, $t4, -1
slti $t5, $t4, 5  # Kiểm tra nếu tốc độ nhỏ hơn 5
bne $t5, $zero, end_process  # Nếu nhỏ hơn 5 thì không thay đổi
sw $t4, speed
j end_process
```

2.4 Kết quả chạy mô phỏng