ĐẠI HỌC BÁCH KHOA HÀ NỘI

BÁO CÁO BÀI TẬP LỚN

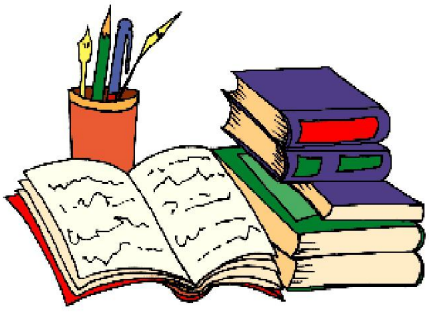
MÔN THỰC HÀNH KIẾN TRÚC MÁY TÍNH

Giảng viên hướng dẫn: Hoàng Văn Hiệp

Sinh viên thực hiện:

Hoàng Đức Quân – 20143635 – bài 2

Nguyễn Nhật Tân – 20133347 – bài 5



Bài 2: Vẽ hình trên màn hình Bitmap

Sinh viên thực hiện: Hoàng Đức Quân - 20143635

Đề bài: Viết một chương trình sử dụng MIPS để vẽ một quả bóng di chuyển trên màn hình mô phỏng Bitmap của Mars). Nếu đối tượng đập vào cạnh của màn hình thì sẽ di chuyển theo chiều ngược lại.

Yêu cầu:

- Thiết lập màn hình ở kích thước 512x512. Kích thước 1 pixel 1x1.

- Quả bóng là một đường tròn Chiều di chuyển phụ thuộc vào phím người dùng bấm, gồm có (di chuyển lên (W), di chuyển xuống (S), Sang trái (A), Sang phải (D) trong bộ giả lập Keyboard and Display MMIO Simulator). Tốc độ bóng di chuyển là không đổi. Vị trí bóng ban đầu ở giữa màn hình

1. ***Mã nguồn:***

```
################################################################
#############
#                         Circle                         #
################################################################
#############
#                                                        #
################################################################
#############
#       This program requires the Keyboard and Display MMIO       #
#     and the Bitmap Display to be connected to MIPS.          #
#                                                              #
#     Bitmap Display Settings:                           #
#       Unit Width: 1                                          #
#       Unit Height: 1                                         #
```

```
#       Display Width: 512                                      #
#       Display Height: 512                                             #
#       Base Address for Display: 0x10040000(heap)                      #
##################################################################
#############


#Author: Hoang Duc Quan

#Create date: 27/04/2017

#Hanoi university of science and technology.
.eqv KEY_CODE 0xFFFF0004  # ASCII code to show, 1 byte
.eqv KEY_READY 0xFFFF0000       # =1 if has a new keycode ?
                                # Auto clear after lw
.eqv DISPLAY_CODE 0xFFFF000C # ASCII code to show, 1 byte
.eqv DISPLAY_READY 0xFFFF0008  # =1 if the display has already to do
                                # Auto clear after sw

.data
L :     .asciiz "a"
R :     .asciiz "d"
U:      .asciiz "w"
D:      .asciiz "s"
points:
        .word           200, 10
        .word           170, 40
        .word 240, 60
        .word           240, 30
.text
        li $k0, KEY_CODE    # chua ký tu nhap vao
        li $k1, KEY_READY  # kiem tra da nhap phim nao chua
        li $s2, DISPLAY_CODE       # hien thi ky tu
        li $s1, DISPLAY_READY     # kiem tra xem man hinh da san sang
hien thi chua

        addi    $s7, $0, 512                    #store the width in s7
        #addi $s0, $0, 0x00FF0000                #pass the colour through
to s0 - for every param the colour is s0, a0-3 are the xy + size params
        #circle:
```

```
        addi   $a0, $0, 256         #x = 256
        addi   $a1, $0, 256         #y = 256
        addi   $a2, $0, 20          #r = 20
        addi   $s0, $0, 0x00FFFF66
        jal    DrawCircle
        nop
moving:

        beq $t0,97,left
        beq $t0,100,right

        beq $t0,115,down
        beq $t0,119,up
        j Input
        left:
                addi $s0,$0,0x00000000
                jal DrawCircle
                addi $a0,$a0,-1
                add $a1,$a1, $0
                addi $s0,$0,0x00FFFF66
                jal DrawCircle
                jal Pause
                bltu $a0,20,reboundRight
                j Input
        right:
                addi $s0,$0,0x00000000
                jal DrawCircle
                addi $a0,$a0,1
                add $a1,$a1, $0
                addi $s0,$0,0x00FFFF66
                jal DrawCircle
                jal Pause
                bgtu $a0,492,reboundLeft
                j Input
        up:
                addi $s0,$0,0x00000000
                jal DrawCircle
                addi $a1,$a1,-1
                add $a0,$a0,$0
                addi $s0,$0,0x00FFFF66
```

```
                jal DrawCircle
                jal Pause
                bltu $a1,20,reboundDown
                j Input
        down:
                addi $s0,$0,0x00000000
                jal DrawCircle
                addi $a1,$a1,1
                add $a0,$a0,$0
                addi $s0,$0,0x00FFFF66
                jal DrawCircle
                jal Pause
                bgtu $a1,492,reboundUp
                j Input
        reboundLeft:
                li $t3 97
                sw $t3,0($k0)
                j Input
        reboundRight:
                li $t3 100
                sw $t3,0($k0)
                j Input
        reboundDown:
                li $t3 115
                sw $t3,0($k0)
                j Input
        reboundUp:
                li $t3 119
                sw $t3,0($k0)
                j Input
endMoving:...
Input:
        ReadKey: lw $t0, 0($k0) # $t0 = [$k0] = KEY_CODE
        j moving


Pause:
        addiu $sp,$sp,-4
        sw $a0, ($sp)
        la $a0,0                # speed =20ms
```

```
        li $v0, 32      #syscall value for sleep
        syscall
        lw $a0,($sp)
        addiu $sp,$sp,4
        jr $ra
DrawCircle:
        #a0 = cx
        #a1 = cy
        #a2 = radius
        #s0 = colour

        addiu $sp, $sp, -32
        sw    $ra, 28($sp)
        sw    $a0, 24($sp)
        sw    $a1, 20($sp)
        sw    $a2, 16($sp)
        sw    $s4, 12($sp)
        sw    $s3, 8($sp)
        sw    $s2, 4($sp)
        sw    $s0, ($sp)

        #code goes here
        sub   $s2, $0, $a2              #error =  -radius
        add   $s3, $0, $a2              #x = radius
        add   $s4, $0, $0               #y = 0 (s4)

        DrawCircleLoop:
        bgt   $s4, $s3, exitDrawCircle  #if y is greater than x, break the
loop (while loop x >= y)
        nop

        #plots 4 points along the right of the circle, then swaps the x
and y and plots the opposite 4 points
        jal   plot8points
        nop

        add   $s2, $s2, $s4                     #error += y
        addi  $s4, $s4, 1               #++y
        add   $s2, $s2, $s4                     #error += y
```

```
        blt     $s2, 0, DrawCircleLoop          #if error >= 0, start loop
again
        nop

        sub     $s3, $s3, 1             #--x
        sub     $s2, $s2, $s3                   #error -= x
        sub     $s2, $s2, $s3                   #error -= x

        j       DrawCircleLoop
        nop

        exitDrawCircle:

        lw      $s0, ($sp)
        lw      $s2, 4($sp)
        lw      $s3, 8($sp)
        lw      $s4, 12($sp)
        lw      $a2, 16($sp)
        lw      $a1, 20($sp)
        lw      $a0, 24($sp)
        lw      $ra, 28($sp)

        addiu $sp, $sp, 32

        jr      $ra
        nop

plot8points:
        addiu $sp, $sp -4
        sw      $ra, ($sp)

        jal     plot4points
        nop

        beq     $s4, $s3, skipSecondplot
        nop

        #swap y and x, and do it again
        add     $t2, $0, $s4            #puts y into t2
        add     $s4, $0, $s3            #puts x in to y
```

```
        add     $s3, $0, $t2                #puts y in to x

        jal     plot4points
        nop

        #swap them back
        add     $t2, $0, $s4                #puts y into t2
        add     $s4, $0, $s3                #puts x in to y
        add     $s3, $0, $t2                #puts y in to x

        skipSecondplot:

        lw      $ra, ($sp)
        addiu $sp, $sp, 4

        jr      $ra
        nop

plot4points:
        #plots 4 points along the right side of the circle, then swaps the
cd and cy values to do the opposite side
        #if statements are for optimisation, they work if the branches
are removed
        addiu $sp, $sp -4
        sw      $ra, ($sp)

        #$a0 = a0 + s3, $a2 = a1 + s4
        add     $t0, $0, $a0                #store a0 (cx in t0)
        add     $t1, $0, $a1                #store a2 (cy in t1)

        add     $a0, $t0, $s3                   #set a0 (x for the setpixel,
to cx + x)
        add     $a2, $t1, $s4                   #set a2 (y for setPixel to
cy + y)

        jal     SetPixel                    #draw the first pixel
        nop

        sub     $a0, $t0, $s3                   #cx - x
        #add    $a2, $t1, $s4                   #cy + y
```

```
        beq     $s3, $0, skipXnotequal0  #if s3 (x) equals 0, skip
        nop

        jal     SetPixel                 #if x!=0 (cx - x, cy + y)
        nop

        skipXnotequal0:
        sub     $a2, $t1, $s4                    #cy - y (a0 already equals
cx - x
        jal     SetPixel                 #no if  (cx - x, cy - y)
        nop

        add     $a0, $t0, $s3

        beq     $s4, $0, skipYnotequal0  #if s4 (y) equals 0, skip
        nop

        jal     SetPixel                 #if y!=0 (cx + x, cy - y)
        nop

        skipYnotequal0:

        add     $a0, $0, $t0
        add     $a2, $0, $t1

        lw      $ra, ($sp)
        addiu $sp, $sp, 4

        jr      $ra
        nop
SetPixel:
        #a0 x
        #a1 y
        #s0 colour
        addiu $sp, $sp, -20              # Save return address on stack
        sw      $ra, 16($sp)
        sw      $s1, 12($sp)
        sw      $s0, 8($sp)              # Save original values of a0, s0,
a2
```

```
        sw      $a0, 4($sp)
        sw      $a2, ($sp)

        lui     $s1, 0x1004             #starting address of the screen
        sll     $a0, $a0, 2             #multiply 4
        add     $s1, $s1, $a0               #x co-ord addded to pixel
position
        mul     $a2, $a2, $s7               #multiply by width (s7
declared at top of program, never saved and loaded and it should
never be changed)
        mul     $a2, $a2, 4             #myltiply by the size of the
pixels (4)
        add     $s1, $s1, $a2               #add y co-ord to pixel
position

        sw      $s0, ($s1)              #stores the value of colour into
the pixels memory address

        lw      $a2, ($sp)              #retrieve original values and
return address
        lw      $a0, 4($sp)
        lw      $s0, 8($sp)
        lw      $s1, 12($sp)
        lw      $ra, 16($sp)
        addiu $sp, $sp, 20

        jr      $ra
        nop
```

Bài 5: Biểu thức trung tố hậu tố

Sinh viên thực hiện: Nguyễn Nhật Tân - 20133347

Đề bài: Viết chương trình tính giá trị biểu thức bất kỳ bằng phương pháp duyệt biểu thức hậu tố.
Các yêu cầu cụ thể: 1. Nhập vào biểu thức trung tố, ví dụ: 9 + 2 + 8 * 6 2. In ra biểu thức ở dạng hậu tố, ví dụ: 9 2 + 8 6 * + 3. Tính ra giá trị của biểu thức vừa nhập

Các hằng số là số nguyên, trong phạm vi từ 0 đến 99.
Toán tử bao gồm phép cộng, trừ, nhân, chia lấy thương

Thuật toán :

a) Đổi biểu thức trung tố sang hậu tố:

Để đổi biểu thức trung tố sang hậu tố, ta sẽ dùng ngăn xếp và xâu
B1: Đưa 1 biểu thức trung tố vào 1 xâu kí tự và đặt tên là str
B2: Tạo ra 1 xâu mới để lưu biểu thức hậu tố, đặt tên là str2
B3: Sắp xếp lại xâu:
- Nếu kí tự là số thì lưu vào str2
- Nếu kí tự là toán tử, nếu ngăn xếp trống thì đẩy vào ngăn xếp
- Nếu toán tử đang xét có bậc cao hơn toán tử ở đỉnh ngăn xếp thì đẩy toán tử vào ngăn xếp
- Nếu toán tử đang xét có bậc bằng toán tử ở đỉnh ngăn xếp thì lấy toán tử đỉnh ngăn xếp ra, xếp vào str2 và đẩy toán tử đang xét vào ngăn xếp
-Nếu toán tử đang xét có bậc nhỏ hơn toán tử ở đỉnh ngăn xếp thì lấy toán tử đang xét và xếp vào str2
B4: Thực hiện bước 3 cho đến khi kết thúc biểu thức và tất cả các toán tử toán hạng được xếp ào str2, khi đó ta có biểu thức hậu tố

b) Tính giá trị biểu thức hậu tố:

B1: Quét toàn bộ biểu thức hậu tố từ trái sang phải
B2: Tạo 1 ngăn xếp mới
B3: Nếu phần tử được quét là toán hạng thì đưa vào ngăn xếp
B4: Nếu phần tử được quét là toán tử thì lấy 2 toán hạng trong ngăn xếp ra, sau đó tính toán giá trị của chúng dựa vào toán tử này, sau đó đẩy lại vào ngăn xếp
B5: Thực hiện bước 3 và bước 4 cho đến khi kết thúc biểu thức và trong ngăn xếp còn 1 giá trị duy nhất. Đó chính là giá trị biểu thức hậu tố

Mã nguồn :

```
# Author: Nguyen Nhat Tan

# Create date: 01/05/2017

# Hanoi university of science and technology.

.data
infix: .space 256
postfix: .space 256
stack: .space 256
prompt:      .asciiz "Enter String contain infix expression¥n(note) Input expression has number must be integer and positive number:"
newLine: .asciiz "¥n"
prompt_postfix: .asciiz "Postfix is: "
prompt_result: .asciiz "Result is: "
prompt_infix: .asciiz "Infix is: "

# get infix
.text
 li $v0, 54
 la $a0, prompt
 la $a1, infix
 la $a2, 256
 syscall


 la $a0, prompt_infix
li $v0, 4
syscall

la $a0, infix
li $v0, 4
syscall

# convert to postfix

li $s6, -1 # counter
li $s7, -1 # Scounter
li $t7, -1 # Pcounter
```

```
while:
    la $s1, infix  #buffer = $s1
    la $t5, postfix #postfix = $t5
    la $t6, stack #stack = $t6
    li $s2, '+'
    li $s3, '-'
    li $s4, '*'
    li $s5, '/'
      addi $s6, $s6, 1  # counter ++

      # get buffer[counter]
      add $s1, $s1, $s6
      lb $t1, 0($s1)          # t1 = value of buffer[counter]



      beq $t1, $s2, operator # '+'
      nop
      beq $t1, $s3, operator # '-'
      nop
      beq $t1, $s4, operator # '*'
      nop
      beq $t1, $s5, operator # '/'
      nop
      beq $t1, 10, n_operator # '¥n'
      nop
      beq $t1, 32, n_operator # ' '
      nop
      beq $t1, $zero, endWhile
      nop

      # push number to postfix
      addi $t7, $t7, 1
      add $t5, $t5, $t7

      sb $t1, 0($t5)


      lb $a0, 1($s1)
```

```
        jal check_number
        beq $v0, 1, n_operator
        nop

        add_space:
        add $t1, $zero, 32
        sb $t1, 1($t5)
        addi $t7, $t7, 1

        j n_operator
        nop

        operator:
        # add to stack ...

        beq $s7, -1, pushToStack
        nop

        add $t6, $t6, $s7
        lb $t2, 0($t6) # t2 = value of stack[counter]

        # check t1 precedence
        beq $t1, $s2, t1to1
        nop
        beq $t1, $s3, t1to1
        nop

        li $t3, 2

        j check_t2
        nop

t1to1:
        li $t3, 1

        # check t2 precedence
check_t2:

        beq $t2, $s2, t2to1
```

```
        nop
        beq $t2, $s3, t2to1
        nop

        li $t4, 2

        j compare_precedence
        nop


t2to1:
        li $t4, 1

compare_precedence:


        beq $t3, $t4, equal_precedence
        nop
        slt $s1, $t3, $t4
        beqz $s1, t3_large_t4
        nop
#################
# t3 < t4
# pop t2 from stack  and t2 ==> postfix
# get new top stack do again

        sb $zero, 0($t6)
        addi $s7, $s7, -1  # scounter ++
        addi $t6, $t6, -1
        la $t5, postfix #postfix = $t5
        addi $t7, $t7, 1
        add $t5, $t5, $t7
        sb $t2, 0($t5)

        #addi $s7, $s7, -1  # scounter = scounter - 1
        j operator
        nop

#################
t3_large_t4:
```

```
# push t1 to stack
        j pushToStack
        nop
#################
equal_precedence:
# pop t2  from stack  and t2 ==> postfix
# push to stack

        sb $zero, 0($t6)
        addi $s7, $s7, -1  # scounter ++
        addi $t6, $t6, -1
        la $t5, postfix #postfix = $t5
        addi $t7, $t7, 1 # pcounter ++
        add $t5, $t5, $t7

        sb $t2, 0($t5)
        j pushToStack
        nop
#################
pushToStack:

        la $t6, stack #stack = $t6
        addi $s7, $s7, 1  # scounter ++
        add $t6, $t6, $s7
        sb $t1, 0($t6)

        n_operator:
        j while
        nop

########################
endWhile:

        addi $s1, $zero, 32
        add $t7, $t7, 1
        add $t5, $t5, $t7
        la $t6, stack
        add $t6, $t6, $s7

popallstack:
```

```
        lb $t2, 0($t6) # t2 = value of stack[counter]
        beq $t2, 0, endPostfix
        sb $zero, 0($t6)
        addi $s7, $s7, -2
        add $t6, $t6, $s7

        sb $t2, 0($t5)
        add $t5, $t5, 1


        j popallstack
        nop

endPostfix:
################################################################
################### END POSTFIX
# print postfix
la $a0, prompt_postfix
li $v0, 4
syscall

la $a0, postfix
li $v0, 4
syscall

la $a0, newLine
li $v0, 4
syscall


################################################################
################### Caculate

li $s3, 0 # counter
la $s2, stack #stack = $s2


# postfix to stack
while_p_s:
```

```
la $s1, postfix #postfix = $s1

add $s1, $s1, $s3
lb $t1, 0($s1)


# if null
beqz $t1 end_while_p_s
nop


add $a0, $zero, $t1
jal check_number
nop

beqz $v0, is_operator
nop

jal add_number_to_stack
nop

j continue
nop


is_operator:

jal pop
nop

add $a1, $zero, $v0 # b

jal pop
nop

add $a0, $zero, $v0 # a

add $a2, $zero, $t1 # op

jal caculate
```

```
        continue:




        add $s3, $s3, 1 # counter++

        j while_p_s
        nop


#---------------------------------------------------------------
#Procedure caculate
# @brief caculate the number ("a op b")
# @param[int] a0 : (int) a
# @param[int] a1 : (int) b
# @param[int] a2 : operator(op) as character
#---------------------------------------------------------------
caculate:
        sw $ra, 0($sp)
        li $v0, 0
        beq $t1, '*', cal_case_mul
        nop
        beq $t1, '/', cal_case_div
        nop
        beq $t1, '+', cal_case_plus
        nop
        beq $t1, '-', cal_case_sub

        cal_case_mul:
                mul $v0, $a0, $a1
                j cal_push
        cal_case_div:
                div $a0, $a1
                mflo $v0
                j cal_push
        cal_case_plus:
```

```
            add $v0, $a0, $a1
            j cal_push
        cal_case_sub:
            sub $v0, $a0, $a1
            j cal_push

        cal_push:
            add $a0, $v0, $zero
            jal push
            nop
            lw $ra, 0($sp)
            jr $ra
            nop



#---------------------------------------------------------------
#Procedure add_number_to_stack
# @brief get the number and add number to stack at $s2
# @param[in] s3 : counter for postfix string
# @param[in] s1 : postfix string
# @param[in] t1 : current value
#---------------------------------------------------------------
add_number_to_stack:
        # save $ra
        sw $ra, 0($sp)
        li $v0, 0

        while_ants:
            beq $t1, '0', ants_case_0
            nop
            beq $t1, '1', ants_case_1
            nop
            beq $t1, '2', ants_case_2
            nop
            beq $t1, '3', ants_case_3
            nop
            beq $t1, '4', ants_case_4
            nop
            beq $t1, '5', ants_case_5
```

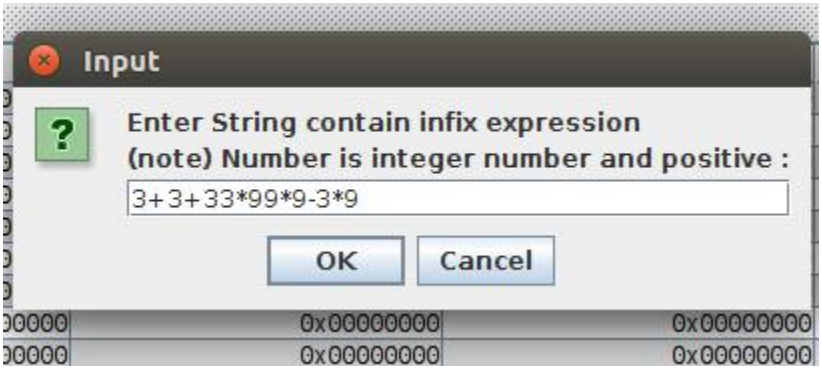
```
nop
beq $t1, '6', ants_case_6
nop
beq $t1, '7', ants_case_7
nop
beq $t1, '8', ants_case_8
nop
beq $t1, '9', ants_case_9
nop

ants_case_0:
        j ants_end_sw_c
ants_case_1:
        addi $v0, $v0, 1
        j ants_end_sw_c
        nop
ants_case_2:
        addi $v0, $v0, 2
        j ants_end_sw_c
        nop
ants_case_3:
        addi $v0, $v0, 3
        j ants_end_sw_c
        nop
ants_case_4:
        addi $v0, $v0, 4
        j ants_end_sw_c
        nop
ants_case_5:
        addi $v0, $v0, 5
        j ants_end_sw_c
        nop
ants_case_6:
        addi $v0, $v0, 6
        j ants_end_sw_c
        nop
ants_case_7:
        addi $v0, $v0, 7
        j ants_end_sw_c
        nop
```

```
            ants_case_8:
                addi $v0, $v0, 8
                j ants_end_sw_c
                nop
            ants_case_9:
                addi $v0, $v0, 9
                j ants_end_sw_c
                nop
            ants_end_sw_c:

                add $s3, $s3, 1 # counter++
                la $s1, postfix #postfix = $s1

                add $s1, $s1, $s3
                lb $t1, 0($s1)

                beq $t1, $zero, end_while_ants
                beq $t1, ' ', end_while_ants

                mul $v0, $v0, 10

                j while_ants

        end_while_ants:
            add $a0, $zero, $v0
            jal push
            # get $ra
            lw $ra, 0($sp)
            jr $ra
            nop


#---------------------------------------------------------------
#Procedure check_number
# @brief check character is number or not
# @param[int] a0 : character to check
# @param[out] v0 : 1 = true; 0 = false
#---------------------------------------------------------------
check_number:
```

```
        li $t8, '0'
        li $t9, '9'

        beq $t8, $a0, check_number_true
        beq $t9, $a0, check_number_true

        slt $v0, $t8, $a0
        beqz $v0, check_number_false

        slt $v0, $a0, $t9
        beqz $v0, check_number_false


        check_number_true:

        li $v0, 1
        jr $ra
        nop
        check_number_false:

        li $v0, 0

        jr $ra
        nop


#-------------------------------------------------------------
#Procedure pop
# @brief pop from stack at $s2
# @param[out] v0 : value to popped
#-------------------------------------------------------------
pop:
        lw $v0, -4($s2)
        sw $zero, -4($s2)
        add $s2, $s2, -4
        jr $ra
        nop

#-------------------------------------------------------------
#Procedure push
```

```
# @brief push to stack at $s2
# @param[in] a0 : value to push
#-------------------------------------------------------------
push:
        sw $a0, 0($s2)
        add $s2, $s2, 4
        jr $ra
        nop


end_while_p_s:

# add null to end of stack


# print postfix
la $a0, prompt_result
li $v0, 4
syscall


jal pop
add $a0, $zero, $v0
li $v0, 1
syscall

la $a0, newLine
li $v0, 4
syscall
```

Hình ảnh:




```
Infix is: 3+3+33*99*9-3*9
Postfix is: 3 3 +33 99 *9 *+3 9 *-
Result is: 29382

-- program is finished running (dropped off bottom) --
```