**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

**A picture containing text, sign

Description automatically generated**

**BÁO CÁO BÀI TẬP CUỐI KỲ**

**HỌC PHẦN IT3280: THỰC HÀNH KIẾN TRÚC MÁY TÍNH**

**Đề 2: Vẽ hình trên BITMAP**

**Đề 8: Mô phỏng ổ đĩa RAID 5**

|  |  |  |
| --- | --- | --- |
| **Giảng viên hướng dẫn:** | ThS. Lê Bá Vui |  |
|  |  |  |
| **Sinh viên thực hiện:** | Cù Minh Hoàng - 20204975  Nguyễn Thành Tiến - 20205034 | Bài 2  Bài 8 |

**Hà Nội, 22/07/2022**

**BÀI 2: VẼ HÌNH TRÊN MÀN HÌNH BITMAP**

1. **Trình bày bài toán đặt ra:  
   -** Viết chương trình vẽ một quả bóng hình tròn 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.
2. **Cách thực hiện:  
   1. Vẽ hình tròn:  
   - Áp dụng thuật toán Midpoint Circle Algo:  
   Text

   Description automatically generated  
   2. Dịch chuyển bóng:  
   - Với mỗi lần dịch chuyển bóng là chúng ta sẽ dịch chuyển tâm của bóng và vẽ bóng mới tại vị trí tâm mới và đồng thời xóa bỏ bóng cũ  
   - Khi bóng chạm tường ( tức chạm biên) thì ta sẽ cho bóng nảy lại, để làm được điều này ta sẽ tính khoảng cách từ tâm tới vị trí biên  
   3. Tăng tốc, giảm tốc:  
   - Việc tăng tốc, giảm tốc chúng ta sẽ thay đổi khoảng cách nhảy tâm mỗi lần bóng di chuyển, ở đây em đã lưu trữ điều đó vào thanh ghi $s4  
   - Với mỗi lần tăng thì mình sẽ \*2 khoảng cách nhảy tâm ( max 16) và giảm thì cx giảm 1 nửa( min = 1)**
3. **Mã nguồn:**

**.eqv KEY\_CODE 0xFFFF0004 # ASCII code to show, 1 byte**

**.data**

**LEFT: .asciiz "a"**

**RIGHT : .asciiz "d"**

**UP: .asciiz "w"**

**DOWN: .asciiz "s"**

**.text**

**li $k0, KEY\_CODE #Enter the key to direct ball**

**.eqv YELLOW 0x00FFFF00**

**.eqv MONITOR\_SCREEN 0x10010000**

**.text**

**#Draw the circle at the center of screen**

**# Center point is (x0, y0)**

**#s0 = x0**

**#s1 = y0**

**#s2 = color**

**#s3 = radius**

**li $v1, MONITOR\_SCREEN**

**#Set the first value**

**li $s0, 256 # set x\_center point at center of screen**

**li $s1, 256 # set y\_center point at center of screen**

**li $s3, 20 # value of radius**

**li $s2, YELLOW**

**li $s4, 1**

**addi $s7, $0, 512 #save the large to $s7**

**jal DrawCircle**

**nop**

**#-------------------------------------------------------------------------------------------------------------------------------**

**MMove: #Moving the circle**

**Readkey:**

**lw $t0, 0($k0)**

**beq $t0, 97, left #$t0 = 'a'**

**beq $t0, 100, right #$t0 = 'd'**

**beq $t0, 115, down #$t0 = 's'**

**beq $t0, 119, up #$t0 = 'w'**

**beq $t0, 120, slow #$t0 = 'x'**

**beq $t0, 122, speed #$t0 = 'z'**

**j Readkey**

**EndReadkey:**

**#----------------------------------------------------------------------------#**

**left:**

**addi $t0, $0, 97**

**sw $t0, 0($k0)**

**bltu $s0, 20, right # If go the the left margin, back right**

**li $s2, 0x00000000**

**jal DrawCircle**

**li $a0, 0**

**li $v0, 32**

**syscall**

**li $s2, YELLOW**

**sub $s0, $s0, $s4**

**jal DrawCircle**

**jal Readkey**

**end\_left:**

**#---------------------------------------------#**

**right:**

**addi $t0, $0, 100**

**sw $t0, 0($k0)**

**bgtu $s0, 492, left**

**li $s2, 0x00000000**

**jal DrawCircle**

**li $a0, 0**

**li $v0, 32**

**syscall**

**li $s2, YELLOW**

**add $s0, $s0, $s4**

**jal DrawCircle**

**jal Readkey**

**end\_right:**

**#---------------------------------------------#**

**up:**

**addi $t0, $0, 119**

**sw $t0, 0($k0)**

**bltu $s1, 20, down**

**li $s2, 0x00000000**

**jal DrawCircle**

**li $a0, 0**

**li $v0, 32**

**syscall**

**li $s2, YELLOW**

**sub $s1, $s1, $s4**

**jal DrawCircle**

**jal Readkey**

**end\_up:**

**#---------------------------------------------#**

**down:**

**addi $t0, $0, 115**

**sw $t0, 0($k0)**

**bgtu $s1, 492, up**

**li $s2, 0x00000000**

**jal DrawCircle**

**li $a0, 0**

**li $v0, 32**

**syscall**

**li $s2, YELLOW**

**addu $s1, $s1, $s4**

**jal DrawCircle**

**jal Readkey**

**end\_down:**

**#---------------------------------------------#**

**speed:**

**sw $0, 0($k0)**

**sll $s4, $s4, 1**

**bgt $s4, 16, update\_speed**

**j Readkey**

**end\_speed:**

**update\_speed:**

**addi $s4, $0, 16**

**j Readkey**

**end\_update\_speed:**

**slow:**

**sw $0, 0($k0)**

**srl $s4, $s4, 1**

**blt $s4, 1, update\_slow**

**j Readkey**

**end\_slow:**

**update\_slow:**

**addi $s4, $0, 1**

**j Readkey**

**end\_update\_slow:**

**end\_MMove:**

**#------------------------------------------------------------------------------------------------------------**

**DrawCircle: #Using Midpoint Circle Algo**

**#CREATE STACK TO STRORE DATAL POINT, COLOR, ...**

**addi $sp, $sp, -20 #Make room on stack for 1 words**

**sw $ra, 0($sp) #Store $ra on element 0 of stack**

**sw $s0, 4($sp) #Store $a0 on element 1 of stack**

**sw $s1, 8($sp) #Store $a1 on element 2 of stack**

**sw $s2, 12($sp) #Store $a2 on element 3 of stack**

**sw $s3, 16($sp) #Store $a3 on element 4 of stack**

**#VARIABLES**

**move $t0, $s0 #x0**

**move $t1, $s1 #y0**

**move $t2, $s3 #radius**

**addi $t3, $t2, 0 #x**

**li $t4, 0 #y**

**li $t7, 0 #Err**

**#While(x >= y)**

**circleLoop:**

**blt $t3, $t4, skipCircleLoop #If x < y, skip circleLoop**

**#s5 = x, s6 = y**

**#Draw Dot (x0 + x, y0 + y)**

**addu $s5, $t0, $t3**

**addu $s6, $t1, $t4**

**lw $s2, 12($sp)**

**jal drawDot #Jump to drawDot**

**#Draw Dot (x0 + y, y0 + x)**

**addu $s5, $t0, $t4**

**addu $s6, $t1, $t3**

**lw $s2, 12($sp)**

**jal drawDot #Jump to drawDot**

**#Draw Dot (x0 - y, y0 + x)**

**subu $s5, $t0, $t4**

**addu $s6, $t1, $t3**

**lw $s2, 12($sp)**

**jal drawDot #Jump to drawDot**

**#Draw Dot (x0 - x, y0 + y)**

**subu $s5, $t0, $t3**

**addu $s6, $t1, $t4**

**lw $s2, 12($sp)**

**jal drawDot #Jump to drawDot**

**#Draw Dot (x0 - x, y0 - y)**

**subu $s5, $t0, $t3**

**subu $s6, $t1, $t4**

**lw $s2, 12($sp)**

**jal drawDot #Jump to drawDot**

**#Draw Dot (x0 - y, y0 - x)**

**subu $s5, $t0, $t4**

**subu $s6, $t1, $t3**

**lw $s2, 12($sp)**

**jal drawDot #Jump to drawDot**

**#Draw Dot (x0 + y, y0 - x)**

**addu $s5, $t0, $t4**

**subu $s6, $t1, $t3**

**lw $a2, 12($sp)**

**jal drawDot #Jump to drawDot**

**#Draw Dot (x0 + x, y0 - y)**

**addu $s5, $t0, $t3**

**subu $s6, $t1, $t4**

**lw $s2, 12($sp)**

**jal drawDot #Jump to drawDot**

**#If (err <= 0)**

**bgtz $t7, doElse**

**addi $t4, $t4, 1 #y++**

**sll $t8, $t4, 1 #Bitshift y left 1**

**addi $t8, $t8, 1 #2y + 1**

**addu $t7, $t7, $t8 #Add e + (2y + 1)**

**j circleContinue #Skip else stmt**

**#Else If (err > 0)**

**doElse:**

**addi $t3, $t3, -1 #x--**

**sll $t8, $t3, 1 #Bitshift x left 1**

**addi $t8, $t8, 1 #2x + 1**

**subu $t7, $t7, $t8 #Subtract e - (2x + 1)**

**j circleContinue**

**circleContinue:**

**#LOOP**

**j circleLoop**

**#CONTINUE**

**skipCircleLoop:**

**#RESTORE $RA**

**lw $ra, 0($sp) #Restore $ra from stack**

**addiu $sp, $sp, 20 #Readjust stack**

**jr $ra**

**nop**

**drawDot:**

**#li $a2, YELLOW**

**add $at, $s6, $0**

**sll $at, $at, 9 # calculate offset in $at: at = y\_pos \* 512**

**add $at, $at, $s5 # at = y\_pos \* 512 + x\_pos = "index"**

**sll $at, $at, 2 # at = (y\_pos \* 512 + x\_pos)\*4 = "offset"**

**add $at, $at, $v1 # at = v1 + offset**

**sw $s2, ($at) # draw it!**

**jr $ra**

**.Kết quả chạy thử:  
Graphical user interface, application

Description automatically generated**

**Shape

Description automatically generated with medium confidence**

**Graphical user interface, application, Word

Description automatically generated**

**BÀI 8: MÔ PHỎNG Ổ ĐĨA RAID 5**

1. Trình bày bài toán đặt ra

* Bài toán mô phỏng raid 5 với 3 ổ đĩa cứng, trong đó ổ đĩa parity sẽ được chứa như hình 1
* Và ta cần phải mô phỏng  hoạt động của RAID 5 với 3 ổ đĩa, với giả định rằng mỗi block dữ liệu  có 4 kí tự. giao diện sau khi in ra như hình 1 giới hạn chuỗi kí tự nhập vào có độ dài là bội của 8

Text, letter

Description automatically generated

1. Cách thức thực hiện
2. Thực hiện tính toán ổ đĩa parity

* ta sẽ xử lý các bước tính toán ổ đĩa parity trước như vậy ta cần lấy dữ liệu từ string input gồm 8 byte để có thể trên một lần vậy ta tạm lấy ra 4 byte vào trong thanh ghi $t0 và 4 byte vào trong thanh ghi $t1 ta xor 2 thanh ghi $t0 và $t1 với nhau lưu vào trong thanh ghi $s1

#xor $s1, $t0, $t1

* tiếp đến chúng ta xử lý đến trích xuất từng kết quả dạng hex của từng byte trong 4 byte của $t0 xor với 4 byte của $t1 tức là 4 byte của $s1 đầu tiên ta trích xuất từng byte của $s1 vào $a0 bằng cách sử dụng lệnh and $s1 với 0xff

           #and $a0, $s1, 0xff

* Với các bước tiếp theo ta chỉ cần việc loại bỏ đi byte ta đã xét hoàn thiện bằng việc sử dụng lệnh srl (shift right logical) 8 bit ta lặp lại việc srl cho đến khi thanh ghi $s1 đã dịch đủ 32 bit (4 byte)

#srl $s1, $s1, 8

1. Thực hiện kiểm tra đầu vào

* với việc kiểm tra đầu vào ta cần phải đếm số kí tự thuộc về chuỗi mà ta nhập vào sau đó ta kiểm tra số kí tự có thực sự chia hết cho 8 không nếu không ta thông báo ra error

CHECK\_LENGTH:

add $a1,$a0,$t1

lb $t0,0($a1)

beq $0,$t0,END\_CHECK

addi $t1,$t1,1

j CHECK\_LENGTH

END\_CHECK:

addi $t1,$t1,-1

addi $t8,$0,8

div $t1,$t8

mfhi  $t8

bne $t8,$0,ERROR

1. Thực hiện in kết quả

* việc in ra các thành phần ngoài lề ta thực hiện rất dễ dàng với việc thực hiện in ra phần đầu và phần đuôi như hình 1 tất nhiên việc in phần đầu chỉ 1 lần và phần đuôi ta thực hiện khá đơn giản
* trước hết ta sử dụng một vòng lặp để tạo menu tránh việc ta phải chạy lại chương trình nhiều lần Graphical user interface, application

  Description automatically generated
* tiếp đến ta thực hiện nhận lấy đầu vào từ bàn phím sau đó check như ở mục 2 sau sau đó ta sử dụng 1 vòng lặp để có thể in ra các block sau mỗi vòng lặp ta có thể in ra tối đa 9 block tương đương với 3 block của mỗi disk để nhằm thực hiện việc tạo các disk parity ở thứ tự ở dòng đầu tiên của vòng lặp tại vị trí disk 3 dòng thứ 2 disk thứ 2 và dòng thứ ba nằm tại vị trí disk 1 ta tạm chia mỗi lần print 1 dòng của 3 disk là 1 split vậy 1 vòng lặp ta sở hữu 3 split sau mỗi lần split ta sẽ thực hiện thay đổi thanh ghi lưu giữ địa chỉ nhớ  để xét đến 2 block cần xét tiếp theo Graphical user interface, text, application, chat or text message

  Description automatically generated
* như mục một ta sẽ tính toán tại disk parity sau đó ta sẽ chuyển kết quả về dạng string để có thể in ra được hiển thị bằng cách xét từng byte (8 bit) sau đó ta lấy ra 4 bit đầu bằng cách and với 0xf và ta kiểm tra giả sử như giá trị nằm trong khoảng 0->9 ta sẽ cộng thêm một lượng bằng 0x30 = 48 (asscii giá trị 48 trong hệ 10 = ‘0’) nếu giá trị lớn hơn 9 (trong khoảng 10->15) ta sẽ cộng thêm 1 khoảng bằng ‘a' - 0xa = 87 với sau đó ta cộng giá trị 4 bit mới vào thanh ghi v0 lúc này thanh ghi v0 sẽ có dạng byte 3 - byte 2 - byte 1 - (4 bit mới) lúc này ta dịch bit v0 đi một khoảng bằng 4 bit dạng mới của thanh ghi v0 sẽ có dạng là byte 3 - byte 2  - (4 bit mới) - byte 0 sau đó ta thực hiện việc lấy 4 bit còn lại trong 8 bit ta xét bằng cách sử dụng srl 4 bit và ta thực hiện việc kiểm tra như 4 bit đầu trong khoảng nào để có giá trị cộng thêm tương ứng sau đó ta cộng vào thanh ghi v0 sau đó thanh ghi v0 sẽ có dạng mới byte 3 - byte 2 -(4 bit đầu 4 bit low đã cộng thêm lượng tương ứng - (4 bit còn lại 4 bit trên đã cộng thêm lượng tương ứng) sau đó chúng ta in ra và do việc lưu trữ theo dạng little-endian nên khi in ra sẽ đảo ngược lại thành đúng 2 kí tự đại diện cho kết quả của phép xor 2 kí tự tương ứng về vị trí trong 2 block mà ta đang xét Graphical user interface, text, application

  Description automatically generated

1. Mã nguồn:

.data

string: .space 128

input: .asciiz "Nhap chuoi: "

errormess: .asciiz "Chuoi Nhap Vao Khong Phai Boi Cua 8\n"

menumess: .asciiz "Menu Chuong trinh\n 1 .NhapChuoi\n 2.Quit"

endmess: .asciiz "\n exit success!"

space: .asciiz " "

header: .asciiz " Disk 1 Disk 2 Disk 3\n"

line: .asciiz " -------------- -------------- --------------\n"

print\_beginP: .asciiz "[[ "

print\_endP: .asciiz "]]"

.align 2

data: .space 4

.text

addi $s6,$0,1

addi $s7,$s0,2

NEW:

li $v0, 51 # In Ra menu De Lua Chon

la $a0, menumess

syscall

beq $s7,$a0,QUIT

bne $s6,$a0,NEW

li $v0, 4

la $a0, input

syscall

li $v0, 8 # Nhap chuoi

la $a0, string # Dia Chi

li $a1, 128 # gioi han cua chuoi nhap vao

syscall

addi $t1,$0,0

CHECK\_LENGTH:

add $a1,$a0,$t1

lb $t0,0($a1)

beq $0,$t0,END\_CHECK

addi $t1,$t1,1

j CHECK\_LENGTH

END\_CHECK:

addi $t1,$t1,-1

addi $t8,$0,8

div $t1,$t8

mfhi $t8

bne $t8,$0,ERROR

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

# Ve giao dien

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

first\_line:

li $v0, 4

la $a0, header

syscall

la $a0, line

syscall

la $s0, string

loop:

lw $t0, 0($s0)

lw $t1, 4($s0)

li $t2, 10

beq $t0, $t2, end\_loop # check line feed "kiem tra ki tu xuong dong de ket thuc thuat toan"

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

# Ve split thu 1

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

move $a1, $t0

jal print\_block

move $a1, $t1

jal print\_block

jal caculator\_and\_print\_partition

li $v0, 11

li $a0, 10

syscall

addi $s0, $s0, 8 #thay doi $s0 den vi tri can xet tiep theo trong string

lw $t0, 0($s0)

lw $t1, 4($s0)

li $t2, 10

beq $t0, $t2, end\_loop

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

# Ve split thu 2

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

move $a1, $t0

jal print\_block

jal caculator\_and\_print\_partition

move $a1, $t1

jal print\_block

li $v0, 11

li $a0, 10

syscall

addi $s0, $s0, 8

lw $t0, 0($s0)

lw $t1, 4($s0)

li $t2, 10

beq $t0, $t2, end\_loop

jal caculator\_and\_print\_partition

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

# Ve split thu 3

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

move $a1, $t0

jal print\_block

move $a1, $t1

jal print\_block

li $v0, 11

li $a0, 10

syscall

addi $s0, $s0, 8

j loop

end\_loop:

li $v0, 4

la $a0, line

syscall

j NEW

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

# chuyen doi tung ki tu hexa sang dang string

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

changeto\_character:

and $t8, $a0, 0xf # lay 4 bit dau trong 8 bit da lay ra

bgt $t8, 9, check1

li $v0, 48

add $v0, $v0, $t8 # "0" + $t8 = string("$t8")

j continue

check1:

li $v0, 87

add $v0, $v0, $t8 # 'a'(97) - 10(0xa) + $t8 = string("$t8")

continue:

sll $v0, $v0, 8 # lay khoang trong de cong tiep gia tri cua 4bit xet sau

srl $t8, $a0, 4 # xoa di cac bit vua check o tren

bgt $t8, 0x9, check2

addi $v0, $v0, 48

add $v0, $v0, $t8

j end\_change

check2:

addi $v0, $v0, 87

add $v0, $v0, $t8

end\_change:

jr $ra

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

# In ra giao dien 1 block gom 4 ki tu trong string da nhap vao

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

print\_block:

li $v0, 11

li $a0, 0x7c # in ra | (0x7c = "|")

syscall

li $v0, 4

la $a0, space

syscall

la $a0, data # print string 4 character is 0($a) -> 3($a) litte-endian

sw $a1, 0($a0)

syscall

la $a0, space

syscall

li $v0, 11

li $a0, 0x7c

syscall

li $v0, 4

la $a0, space

syscall

jr $ra

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

# Tinh Toan

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

caculator\_and\_print\_partition:

li $v0, 4

sw $ra, 0($sp)

la $a0, print\_beginP

syscall

xor $s1, $t0, $t1

and $a0, $s1, 0xff #lay 8 bit dau cua $s1 ghi vao thanh ghi a0

jal changeto\_character #chuyen doi ki tu hexa sang cac ki tu character

la $a0, data

sw $v0, 0($a0)

li $v0, 4

syscall

li $a0, ','

li $v0, 11

syscall

srl $s1, $s1, 8

and $a0, $s1, 0xff

jal changeto\_character

la $a0, data

sw $v0, 0($a0)

li $v0, 4

syscall

li $a0, ','

li $v0, 11

syscall

srl $s1, $s1, 8

and $a0, $s1, 0xff

jal changeto\_character

la $a0, data

sw $v0, 0($a0)

li $v0, 4

syscall

li $a0, ','

li $v0, 11

syscall

srl $s1, $s1, 8

and $a0, $s1, 0xff

jal changeto\_character

la $a0, data

sw $v0, 0($a0)

li $v0, 4

syscall

la $a0, print\_endP

syscall

la $a0, space

syscall

lw $ra, 0($sp)

jr $ra

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

# In ra Loi neu chuoi dau vao co length khong phai boi cua 8

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

ERROR:

li $v0, 4

la $a0, errormess

syscall

j NEW

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

# Phan Ket Thuc Chuong trinh

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

QUIT:

li $v0, 4

la $a0, endmess

syscall

IV.   Kết quả thực nghiệm

* Giao diện menuGraphical user interface, application, Word

  Description automatically generated
* Với giá trị nhập vào như trong đề tham khảo: **DCE.\*\*\*\*ABCD1234HUSTHUSTGraphical user interface, application, Word

  Description automatically generated**
* Với giá trị nhập vào khác tham khảo **chungtakhongthuocvenhau! Graphical user interface, application, Word

  Description automatically generated**
* Với giá trị nhập vào không chia hết cho 8 Graphical user interface, application, Word

  Description automatically generated
* Khi thoát chương trình

**Graphical user interface, application

Description automatically generated**

**IV.Kết Luận**

* Như vậy chúng ta đã mô phỏng thành công ổ đĩa raid 5 với các giới hạn về nhiều disk hơn ta cũng có thể làm tương tự
* Ổ đĩa raid ta mô phỏng hay trong bài hay các ổ đĩa raid khác có tác dụng trong việc khôi phục dữ liệu khi ta mất hoàn toàn dữ liệu của 1 trong 3 ổ đĩa ta hoàn toàn có thể dựa vào 2 ổ đĩa còn lại để khôi phục lại dữ liệu cho ổ đĩa đã mất đó