Báo cáo final project

# Main: 8. Mô phỏng ổ đĩa RAID 5

## Đề bài

Hệ thống ổ đĩa RAID5 cần tối thiểu 3 ổ đĩa cứng, trong đó phần dữ liệu parity sẽ được chứa lần lượt lên 3

ổ đĩa như trong hình bên. Hãy viết chương trình 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 như trong minh họa dưới. Giới hạn chuỗi kí tự nhập vào có

độ dài là bội của 8.

Trong ví dụ sau, chuỗi kí tự nhập vào từ bàn phím (DCE.\*\*\*\*ABCD1234HUSTHUST) sẽ được chia thành

các block 4 byte. Block 4 byte đầu tiên “DCE.” sẽ được lưu trên Disk 1, Block 4 byte tiếp theo “\*\*\*\*” sẽ

lưu trên Disk 2, dữ liệu trên Disk 3 sẽ là 4 byte parity được tính từ 2 block đầu tiên với mã ASCII là

6e=’D’ xor ‘\*’ ; 69=’C’ xor ‘\*’; 6f=’E’ xor ‘\*’ ; 04=’.’ xor ‘\*’

## Ý tưởng giải quyết

* Đọc vào buffer, kiểm tra độ dài string, báo lỗi nếu độ dài không là bội của 8
* Vòng lặp xử lý:
  + Đọc vào 2 đĩa d1, d2 mỗi đĩa 4 bytes (tương ứng lw)
  + xor giá trị 2 word vừa đọc
  + lưu kết quả vào đĩa 3
  + gọi hàm in dòng tương ứng với số đếm dòng

## Code

.eqv BUF\_LEN 5000

.data

buffer: .space BUF\_LEN

d1: .space 4

d2: .space 4

d3: .space 4

msg\_input: .asciiz "Nhap chuoi ki tu : "

buf\_len\_err: .asciiz "Do dai chuoi khong hop le! Nhap lai.\n"

disk\_ls\_msg: .asciiz " Disk 1 Disk 2 Disk 3\n"

msg\_dash: .asciiz "----------------- ---------------- ----------------\n"

msg2: .asciiz "| "

msg3: .asciiz " | "

msg4: .asciiz "[[ "

msg5: .asciiz "]] "

msg\_tryagain: .asciiz "Try again?"

hex: .byte '0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'

.text

main:

main\_loop:

li $v0, 4

la $a0, msg\_input

syscall

li $v0, 8 # Get string

la $a0, buffer

li $a1, BUF\_LEN

syscall

la $a0, buffer # get input length

jal strlen

nop

addi $s0, $v0, 0

addi $a0, $s0, 0 # check if input length divisible by 8

jal mod8

nop

beq $v0, $0, main\_loop\_start\_process

nop

la $a0, buf\_len\_err # if not, warn user

li $v0, 4

syscall

b main\_loop # and start again

main\_loop\_start\_process:

li $v0, 4 # print " Disk1 Disk2 Disk3"

la $a0, disk\_ls\_msg

syscall

li $v0, 4 # print " Disk1 Disk2 Disk3"

la $a0, msg\_dash

syscall

li $t0, 0 # line count

li $t1, 0 # iterator

main\_loop\_proc\_loop:

beq $t1, $s0, main\_loop\_proc\_loop\_done

nop

# read disc1

lw $t2, buffer($t1)

addi $t3, $t2, 0

sw $t2, d1

addi $t1, $t1, 4

# read disc2

lw $t2, buffer($t1)

addi $t4, $t2, 0

sw $t2, d2

addi $t1, $t1, 4

# write disc3

xor $t3, $t3, $t4

sw $t3, d3

# print line

proc\_print\_line\_0:

bne $t0, 0, proc\_print\_line\_1

nop

jal print\_line\_0

nop

b print\_done

nop

proc\_print\_line\_1:

bne $t0, 1, proc\_print\_line\_2

nop

jal print\_line\_1

nop

b print\_done

nop

proc\_print\_line\_2:

jal print\_line\_2

nop

print\_done:

addi $t0, $t0, 1 # increment line count

bne $t0, 3, skip\_reset\_linecount

nop

li $t0, 0

skip\_reset\_linecount:

b main\_loop\_proc\_loop # loop again

nop

main\_loop\_proc\_loop\_done:

li $v0, 50 # ask if try again

la $a0, msg\_tryagain

syscall

beq $a0, 0, main\_loop # a0 : 0 = yes; 1 = NO ; 2 = cancel

nop

exit\_main:

li $v0, 10

syscall

#### PROCEDURES AND FUNCTIONS ####

# strlen

# a0: string address

# v0: returned value

strlen:

sw $ra, -4($sp)

sw $fp, -8($sp)

sw $t0, -12($sp)

sw $t1, -16($sp)

addi $sp, $sp, -16

nop

addi $v0, $0, 0

addi $t0, $a0, 0

strlen\_loop:

lb $t1, 0($t0)

beq $t1, $0, strlen\_done\_loop # null char

nop

beq $t1, '\n', strlen\_done\_loop # new line char

nop

beq $t1, '\r', strlen\_done\_loop # line feed char (windows)

nop

addi $t0, $t0, 1

addi $v0, $v0, 1

j strlen\_loop

nop

strlen\_done\_loop:

addi $sp, $sp, 16

lw $ra, -4($sp)

lw $fp, -8($sp)

lw $t0, -12($sp)

lw $t1, -16($sp)

jr $ra

nop

# mod 8

# a0: integer number

# v0=a0 mod 8

mod8:

sw $ra, -4($sp)

sw $fp, -8($sp)

sw $t0, -12($sp)

addi $sp, $sp, -12

li $v0, 0

srl $t0, $a0, 3

sll $t0, $t0, 3

sub $v0, $a0, $t0

addi $sp, $sp, 12

lw $ra, -4($sp)

lw $fp, -8($sp)

lw $t0, -12($sp)

jr $ra

nop

# print line 0

print\_line\_0:

sw $ra, -4($sp)

sw $fp, -8($sp)

addi $sp, $sp, -8

la $a0, msg2

li $v0, 4

syscall

la $a0, d1

jal print\_4bytes\_char

nop

la $a0, msg3

li $v0, 4

syscall

la $a0, msg2

li $v0, 4

syscall

la $a0, d2

jal print\_4bytes\_char

nop

la $a0, msg3

li $v0, 4

syscall

la $a0, msg4

li $v0, 4

syscall

la $a0, d3

jal print\_4bytes\_hex

nop

la $a0, msg5

li $v0, 4

syscall

li $a0, '\n'

li $v0, 11

syscall

addi $sp, $sp, 8

lw $ra, -4($sp)

lw $fp, -8($sp)

jr $ra

nop

# print line 1

print\_line\_1:

sw $ra, -4($sp)

sw $fp, -8($sp)

addi $sp, $sp, -8

la $a0, msg2

li $v0, 4

syscall

la $a0, d1

jal print\_4bytes\_char

nop

la $a0, msg3

li $v0, 4

syscall

la $a0, msg4

li $v0, 4

syscall

la $a0, d3

jal print\_4bytes\_hex

nop

la $a0, msg5

li $v0, 4

syscall

la $a0, msg2

li $v0, 4

syscall

la $a0, d2

jal print\_4bytes\_char

nop

la $a0, msg3

li $v0, 4

syscall

li $a0, '\n'

li $v0, 11

syscall

addi $sp, $sp, 8

lw $ra, -4($sp)

lw $fp, -8($sp)

jr $ra

nop

# print line 2

print\_line\_2:

sw $ra, -4($sp)

sw $fp, -8($sp)

addi $sp, $sp, -8

la $a0, msg4

li $v0, 4

syscall

la $a0, d3

jal print\_4bytes\_hex

nop

la $a0, msg5

li $v0, 4

syscall

la $a0, msg2

li $v0, 4

syscall

la $a0, d1

jal print\_4bytes\_char

nop

la $a0, msg3

li $v0, 4

syscall

la $a0, msg2

li $v0, 4

syscall

la $a0, d2

jal print\_4bytes\_char

nop

la $a0, msg3

li $v0, 4

syscall

li $a0, '\n'

li $v0, 11

syscall

addi $sp, $sp, 8

lw $ra, -4($sp)

lw $fp, -8($sp)

jr $ra

nop

# print chars

# a0: string address

print\_4bytes\_char:

sw $ra, -4($sp)

sw $fp, -8($sp)

sw $t0, -12($sp)

sw $t1, -16($sp)

addi $sp, $sp, -16

li $t0, 0

addi $t1, $a0, 0

print\_4bytes\_char\_loop:

beq $t0, 4, print\_4bytes\_char\_done\_loop

nop

add $a0, $t1, $t0

lb $a0, ($a0)

li $v0, 11

syscall

addi $t0, $t0, 1

b print\_4bytes\_char\_loop

nop

print\_4bytes\_char\_done\_loop:

addi $sp, $sp, 16

lw $ra, -4($sp)

lw $fp, -8($sp)

lw $t0, -12($sp)

lw $t1, -16($sp)

jr $ra

nop

# print hex

print\_4bytes\_hex:

sw $ra, -4($sp)

sw $fp, -8($sp)

sw $t0, -12($sp)

sw $t1, -16($sp)

sw $t2, -20($sp)

sw $t3, -24($sp)

sw $t4, -28($sp)

sw $t5, -32($sp)

addi $sp, $sp, -32

li $t0, 0

addi $t1, $a0, 0

la $t2, hex

print\_4bytes\_hex\_loop:

beq $t0, 4, print\_4bytes\_hex\_done\_loop

nop

add $t3, $t1, $t0

lb $t3, ($t3)

srl $t4, $t3, 4

sll $t5, $t4, 4

sub $t5, $t3, $t5

add $t4, $t2, $t4

lb $a0, ($t4)

li $v0, 11

syscall

add $t5, $t2, $t5

lb $a0, ($t5)

li $v0, 11

syscall

li $a0, ','

syscall

addi $t0, $t0, 1

b print\_4bytes\_hex\_loop

nop

print\_4bytes\_hex\_done\_loop:

addi $sp, $sp, 32

lw $ra, -4($sp)

lw $fp, -8($sp)

lw $t0, -12($sp)

lw $t1, -16($sp)

lw $t2, -20($sp)

lw $t3, -24($sp)

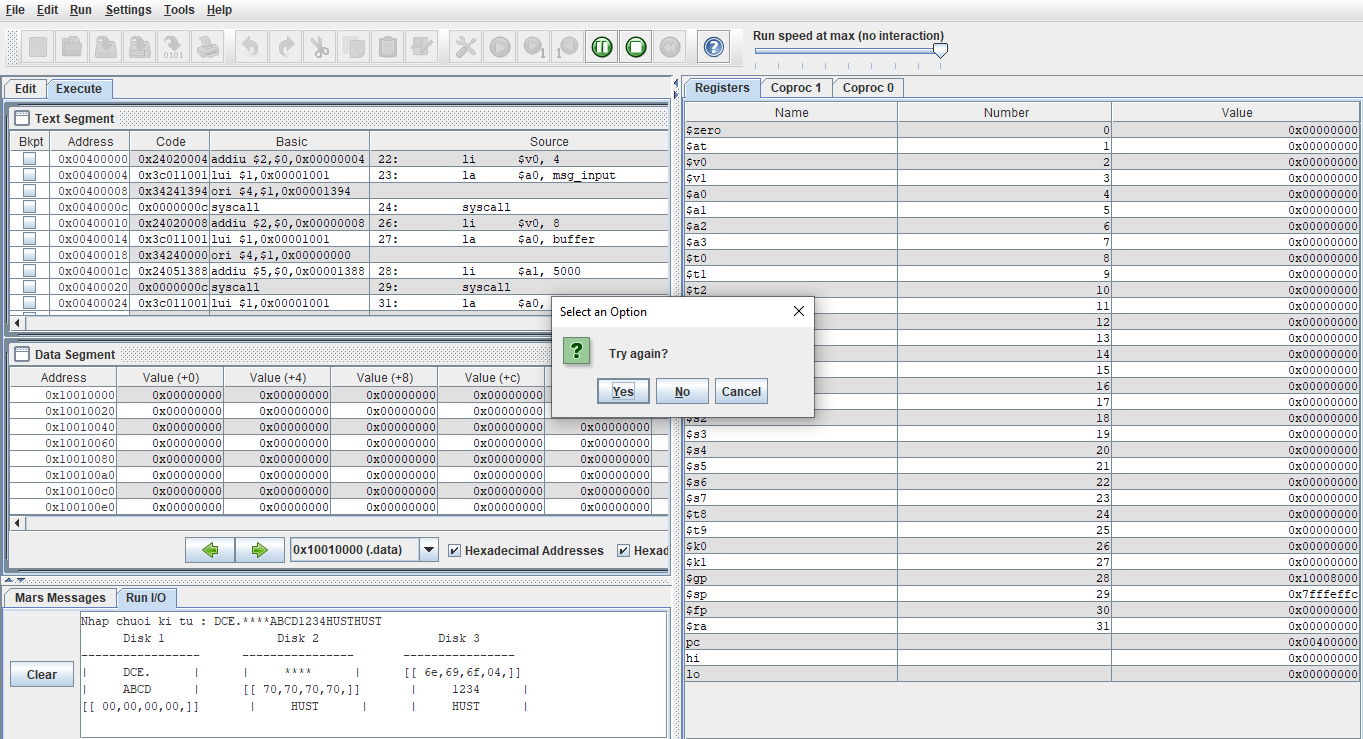
lw $t4, -28($sp)

lw $t5, -32($sp)

jr $ra

nop

## Kết quả



# Minor: 3. Kiểm tra tốc độ và độ chính xác khi gõ văn bản

## Đề bài

Kiểm tra tốc độ và độ chính xác khi gõ văn bản

Chương trình sau sẽ đo tốc độ gõ bàn phím và hiển thị kết quả bằng 2 đèn led 7 đoạn. Nguyên tắc:

- Cho một đoạn văn bản mẫu, cố định sẵn trong mã nguồn. Ví dụ “bo mon ky thuat may tinh”

- Sử dụng bộ định thời Timer (trong bộ giả lập Digi Lab Sim) để tạo ra khoảng thời gian để đo. Đây là thời

gian giữa 2 lần ngắt, chu kì ngắt.

- Trong thời khoảng đó, người dùng nhập các kí tự từ bàn phím. Ví dụ nhập “bo mOn ky 5huat may tinh”.

Chương trình cần phải đếm số kí tự đúng (trong ví dụ trên thì người dùng gõ sai chữ O và 5) mà người

dùng đã gõ và hiển thị lên các đèn led.

## Ý tưởng giải quyết

* Lưu văn bản mẫu vào một xâu kí tự (string)
* Lưu kí tự đã nhập vào xâu buffer
* So sánh 2 xâu, đếm số kí tự giống nhau tương ứng
* In ra số kí tự giống nhau, tốc độ gõ tính bằng wpm

S =

## Code

# MMIO Simulator

.eqv KEY\_CODE 0xFFFF0004

.eqv KEY\_READY 0xFFFF0000

.eqv DISPLAY\_CODE 0xFFFF000C # display

.eqv DISPLAY\_READY 0xFFFF0008

# Digi Lab Sim

.eqv COUNTER 0xFFFF0013 # Time counter

.eqv MASK\_CAUSE\_COUNTER 0x00000400

# Led 7seg

.eqv SEVENSEG\_RIGHT 0xFFFF0010

.eqv SEVENSEG\_LEFT 0xFFFF0011

.data

string: .asciiz "bo mon ki thuat may tinh"

buffer: .space 1000

count\_correct: .asciiz "Correct chars count: "

unit\_chars: .asciiz " chars\n"

speed: .asciiz "Speed: "

unit\_wpm: .asciiz " wpm\n"

repeat: .asciiz "Do you want to start typing again (Press Enter to repeat)?\n"

.text

li $t0, 1

li $a0,COUNTER

sb $t0,0($a0) # trigger counter exception

li $t0, 0

li $s0, 0 # $s0 = Cycle count

li $s1, 0 # $s1 = 0: not counting, $s1 = 1: start counting

print\_sample:

WaitForDis:

lw $t2, DISPLAY\_READY

beq $t2, $zero, WaitForDis

nop

lb $t1, string($t0) # t1 = string[i]

beq $t1, '\0', start\_readc # if string[i] == NULL break

sw $t1, DISPLAY\_CODE

add $t0, $t0, 1 # i++

j print\_sample

start\_readc:

li $t0, 0 # typed chars count

li $s0, 0 # reset count

li $s1, 1 # start counting cycles

WaitForKey:

lw $t2, KEY\_READY

beqz $t2, WaitForKey

nop

lw $t3, KEY\_CODE

beq $t3, 8, backspace # check backspace

nop

beq $t3, 10, finish # finish if enter is pressed

nop

sb $t3, buffer($t0) # save to buffer

add $t0, $t0, 1 # i++

j WaitForKey

nop

backspace:

beqz $t0, WaitForKey # if i == 0 => poll new key

nop

sb $zero, buffer($t0) # else{ buffer[i] = NULL;

add $t0, $t0, -1 # i--;}

sb $zero, buffer($t0)

j WaitForKey

nop

finish: li $s1, 0 # stop counting cycles

# speed report

li $v0, 4

la $a0, speed

syscall

addi $a0, $t0, 0

mulo $a0, $a0, 132000

divu $a0, $a0, $s0

li $v0, 1

syscall

li $v0, 4

la $a0, unit\_wpm

syscall

li $t4, 0 # $t4: correct chars count

li $t0, 0

count\_chars:

lb $t1, string($t0) # $t1 = string[i]

lb $t2, buffer($t0) # $t2 = buffer[i]

beqz $t1, report # if(string[i] == NULL) break;

nop

bne $t1, $t2, next\_char # If (String[i] == buffer[i])

nop

add $t4, $t4, 1 # $t4++;

next\_char:

add $t0, $t0, 1

b count\_chars

nop

report:

# report number of correct chars

li $v0, 4

la $a0, count\_correct

syscall

move $a0, $t4

li $v0, 1

syscall

li $v0, 4

la $a0, unit\_chars

syscall

# display on 7 seg

li $t5, 10

div $t4, $t5

mflo $t0

jal SET\_DATA\_FOR\_7SEG

move $a1, $a0

mfhi $t0

jal SET\_DATA\_FOR\_7SEG

nop

jal SHOW\_7SEG\_RIGHT

nop

jal SHOW\_7SEG\_LEFT

nop

start\_again:

la $a0, repeat

li $v0, 4

syscall

WaitForKey2:

lw $t2, KEY\_READY

beqz $t2, WaitForKey2

nop

lw $t3, KEY\_CODE

beq $t3, 10, start\_readc # if enter is pressed, start typing again

nop

exit\_main:

li $v0, 10

syscall

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

# Function SHOW\_7SEG\_RIGHT : turn on/off the 7seg

# @param [in] $a0 value to shown

# remark $t0 changed

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

SHOW\_7SEG\_RIGHT:

sb $a0, SEVENSEG\_RIGHT # assign new value

jr $ra

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

# Function SHOW\_7SEG\_LEFT : turn on/off the 7seg

# @param [in] $a1 value to shown

# remark $t0 changed

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

SHOW\_7SEG\_LEFT:

sb $a1, SEVENSEG\_LEFT # assign new value

jr $ra

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

# Function SET\_DATA\_FOR\_7SEG : decimal to LED display

# @param [in] $t0 dec value

# @return $a0 encrypted message

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

SET\_DATA\_FOR\_7SEG:

beq $t0, 0, \_\_0

beq $t0, 1, \_\_1

beq $t0, 2, \_\_2

beq $t0, 3, \_\_3

beq $t0, 4, \_\_4

beq $t0, 5, \_\_5

beq $t0, 6, \_\_6

beq $t0, 7, \_\_7

beq $t0, 8, \_\_8

beq $t0, 9, \_\_9

nop

\_\_0: li $a0, 0x3f

j END\_\_F

\_\_1: li $a0, 0x06

j END\_\_F

\_\_2: li $a0, 0x5B

j END\_\_F

\_\_3: li $a0, 0x4f

li $s2, 3

j END\_\_F

\_\_4: li $a0, 0x66

j END\_\_F

\_\_5: li $a0, 0x6D

j END\_\_F

\_\_6: li $a0, 0x7d

j END\_\_F

\_\_7: li $a0, 0x07

j END\_\_F

\_\_8: li $a0, 0x7f

j END\_\_F

\_\_9: li $a0, 0x6f

j END\_\_F

END\_\_F:

jr $ra

# Exception handler

.ktext 0x80000180

IntSR:

move $k0, $at

mfc0 $k1, $13 # cause

bne $k1, MASK\_CAUSE\_COUNTER, finish # another exception => exit

add $s0, $s0, $s1

move $at, $k0

return: eret

## Kết quả

Graphical user interface, application

Description automatically generated