**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**\*\*\*\*\*\*\*\*\*\*\*\***

**A picture containing logo

Description automatically generated**

**BÁO CÁO BÀI TẬP LỚN**

**MÔN THỰC HÀNH KIẾN TRÚC MÁY TÍNH**

**Giảng viên hướng dẫn: TS. Lê Bá Vui**

**Nhóm 10:**

Nguyễn Công Việt – 20184230

Nguyễn Viết Trường – 20184210

**ĐỀ TÀI 7:**

**Sinh viên thực hiện: Nguyễn Công Việt**

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 s1 là hợp lệ, 31 là không hợp lệ, t4 thì khỏi phải kiểm tra nữa vì toán hạng trước đã bị sai rồi.

- Cho biết lệnh hợp ngữ đó cần bao nhiêu chu kì thì mới thực hiện xong. 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, số chu kì thực hiện.

1. **Cách thực hiện chương trình:**

• Yêu cầu người dùng nhập vào một câu lệnh mips

• Một câu lệnh hợp ngữ gồm 4 phần: mã opcode, toán hạng 1, toán hạng 2, toán hạng 3. Quy ước các dạng của toán hạng: 0 – null, 1 – register, 2 – constant, 3 – label, 4 - cụm offset(base).

• Tách một thành phần của chuỗi tương ứng với thành phần của câu lệnh rồi so sánh với kiểu dữ liệu quy ước. Nếu phù hợp thì tiếp tục so sánh các phần phía sau. Nếu không phù hợp thì báo không hợp lệ

+ Đầu chương trình, khởi tạo thư viện opcode chuẩn, opcode sau khi cắt được thì sẽ so sánh với thư viện đã tạo

+ Nếu không có opcode phù hợp -> báo opcode không hợp lệ và exit. Nếu có opcode hợp lệ -> tìm khuôn dạng các toán hạng phù hợp với opcode. Đầu chương trình khi tạo chuỗi opcode chuẩn thì sẽ tạo một chuỗi khuôn dạng tương ứng với từng lệnh bằng cách so vị trí lệnh và khuôn lệnh. Nếu opcode phù hợp thì sẽ lấy được khuôn dạng lệnh và tiến hành kiểm tra lần lượt các toán hạng. Nếu có 1 toán hạng không hợp lệ -> báo ra màn hình run I/O và exit

+ Tương tự opcode chuẩn thì cũng có một chuỗi register chuẩn được tạo ở đầu chương trình. Nếu toán hạng có mã là 1 -> nó là thanh ghi và sẽ so sánh với chuỗi các regisster chuẩn.

+ Trường hợp toán hạng là constant: một constant hợp lệ có thể có kí tự đầu là dấu trừ ‘- ’ hoặc dấu cộng ‘+’, các ký tự phía sau bắt buộc phải từ 0 đến 9.

+ Trường hợp toán hạng là label: label hợp lệ có thể có kí tự đầu là dấu gach dưới ‘\_’ hoặc chữ thường, chữ hoa, nếu có các ký tự tiếp theo, thì các ký tự này bắt buộc phải là chữ cái, chữ số hoặc dấu gạch dưới.

+ Trường hợp toán hạng null: kiểm tra xem chuỗi vừa cắt được có kí tự nào hay không, nếu có -> không hợp lệ, nếu không có kí tự nào -> hợp lệ và thông báo câu lệnh hợp lệ -> Thoát chương trình.

+ Trường hợp toán hạng là một cụm offset(base): kiểm tra xem dấu mở ngoặc ‘(’ và đóng ngoặc ‘)’ có hợp lệ không? Có đủ 2 dấu đóng và mở ngoặc hay không. Nếu không -> không hợp lệ. Nếu có hợp lệ -> cut chuỗi này thành 2 thành phần là constant và register -> kiểm tra từng thành phần. Nếu có bất kì một thành phần nào sai -> Không hợp lệ.

Ngược lại -> hợp lệ.

• Sau khi kiểm tra xong 3 toán hạng, chương trình sẽ kiểm tra xem còn kí tự nào khác hay không. Nếu còn -> câu lệnh không hợp lệ. Và báo kết quả của câu lệnh vừa nhập -> Hỏi người dùng có muốn kiểm tra tiếp một câu lệnh khác hay không?

• Nếu người dùng chọn không -> Kết thúc chương trình.

1. **Kết quả chạy chương trình:**

Ví dụ : add $s1,$s1,$s1

Ảnh có chứa văn bản, máy tính, trong nhà, ảnh chụp màn hình

Mô tả được tạo tự động

Trường hợp lỗi :

Ảnh có chứa văn bản, máy tính, trong nhà, ảnh chụp màn hình

Mô tả được tạo tự động

**ĐỀ TÀI SỐ 8: MÔ PHỎNG Ổ ĐĨA RAID 5**

**Sinh viên thực hiện: Nguyễn Viết Trường**

**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 ‘\*’.*

**Cách làm:**

Bước 1: Nhập chuỗi đầu vào

Bước 2: Kiểm tra độ dài chuỗi đầu vào có phải là bội của 8

Bước 3: Thực hiện thuật toán phân tách chuỗi đầu vào

Bước 4: In ra kết quả và hỏi xem có tiếp tục chương trình

**Thuật toán phân tách chuỗi đầu vào:**

*Phân tách theo từng 8 kí tự với 2 block 4 kí tự cho 2 disk không chứa mã parity*

+ Copy từng kí tự vào array1 và array2 là các disk chứa block dữ liệu

+ Thực hiện phép xor từng cặp kí tự tương ứng lưu trong các disk trên để lưu vào mảng lưu dữ liệu parity

+ Thực hiện phép dịch bit đối với phần tử trong mảng để lấy mã parity và in ra disk3

+ Tương tự đối với disk1, disk3 chứa dữ liệu, disk2 chứa mã parity và disk2, disk3 chứa dữ liệu, disk3 chứa mã parity

**Mã nguồn:**

#==============================Ex8: Mo phong o dia RAID 5===========================

.data

mInput: .asciiz "Nhap chuoi ky tu: "

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

array1: .space 4 # disk chua du lieu

array2: .space 4 # disk chua du lieu

.align 2 # dua dia chi arrayparity ve dia chi boi cua 4

arrayParity: .space 32

stringInput: .space 5000 #xau ki tu nhap vao

enter: .asciiz "\n"

error\_length: .asciiz "Do dai chuoi khong hop le! Nhap lai.\n"

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

m2: .asciiz "---------------- ---------------- ----------------\n"

m3: .asciiz "| "

m4: .asciiz " | "

m5: .asciiz "[[ "

m6: .asciiz "]] "

comma: .asciiz ","

ms: .asciiz "Try again?"

.text

main:

jal input # nhap du lieu

nop

jal check\_lengthInput # check du lieu

nop

jal split # phan tach du lieu vao cac disk

nop

exit: li $v0, 10 # ket thuc chuong trinh

syscall

#=====================Nhap du lieu dau vào===========================

input:

li $v0, 4 # nhap ten (chuoi)

la $a0, mInput # doc message yeu cau nhap

syscall

li $v0, 8 # doc chuoi

la $a0, stringInput # a0 luu chuoi ki tu nhap vao

li $a1, 1000

syscall

move $s0, $a0 # s0 chua dia chi chuoi moi nhap

li $v0, 4

la $a0, m

syscall

li $v0, 4

la $a0, m2

syscall

jr $ra

#======================check du lieu dau vào chia het cho 8=======================

check\_lengthInput:

addi $t3, $zero, 0 # t3 = length

addi $t0, $zero, 0 # t0 = index

count\_char:

add $t1, $s0, $t0 # t1 = address of string[i]

lb $t2, 0($t1) # t2 = string[i]

nop

beq $t2, 10, check\_length # t2 = '\n' ket thuc xau

nop

addi $t3, $t3, 1 # length++

addi $t0, $t0, 1 # index++

j count\_char

nop

check\_length:

move $t5, $t3

and $t1, $t3, 0x0000000f # xoa het cac byte cua $t3 ve 0, chi giu lai byte cuoi

beq $t1, 0, nCheck # byte\_cuoi # 0 -> test1

beq $t1, 8, nCheck

j error1

nop

error1:

li $v0, 4 # dua ra canh báo nếu input không đúng

la $a0, error\_length

syscall

j main

nop

nCheck:

jr $ra

nop

#=====================Phan tach block du lieu va in ra man hinh=======================

split:

jal reset

nop

jal splitting

nop

la $a0, m3 # in "| "

jal print\_message

nop

jal print\_disk1 # in ra block 4 ky tu trong disk luu du lieu dau tien

nop

la $a0, m4 # in " | "

jal print\_message

nop

la $a0, m3 # in "| "

jal print\_message

nop

jal print\_disk2 # in ra block 4 ky tu trong disk luu du lieu thu hai

nop

la $a0, m4 # in " | "

jal print\_message

nop

jal print\_arrayParity # in ra du lieu parity

nop

la $a0, enter

jal print\_message

nop

beq $t3, 0, check\_exit

nop

#---------------split2--------------------------

addi $s0, $s0, 4

jal reset

nop

jal splitting

nop

la $a0, m3

jal print\_message

nop

jal print\_disk1

nop

la $a0, m4

jal print\_message

nop

jal print\_arrayParity

nop

la $a0, m3

jal print\_message

nop

jal print\_disk2

nop

la $a0, m4

jal print\_message

nop

la $a0, enter

jal print\_message

nop

beq $t3, 0, check\_exit

nop

#---------------split3--------------------------

addi $s0, $s0, 4

jal reset

nop

jal splitting

nop

jal print\_arrayParity

nop

la $a0, m3

jal print\_message

nop

jal print\_disk1

nop

la $a0, m4

jal print\_message

nop

la $a0, m3

jal print\_message

nop

jal print\_disk2

nop

la $a0, m4

jal print\_message

nop

la $a0, enter

jal print\_message

nop

beq $t3, 0, check\_exit

nop

addi $s0, $s0, 4

j split

nop

#=========================D?ch bit L?y mã Parity==============================

HEX: li $t4, 1

loopH:

sll $s6, $t4, 2 # s6 = t4\*4 vd: ... 0000 0011 = 3 -> ... 0000 1100 =12

srlv $a0, $t8, $s6 # a0 = t8>>s6: vd: s6 =4 & t8: 0x00000021 => a0 = 0x00000002

andi $a0, $a0, 0x0000000f # a0 = a0 & ... 0000 1111 => lay byte cuoi cung cua a0

la $t7, hex # doc dia chi mang hex vao t7

add $t7, $t7, $a0 # cong gia tri a0 vs t7 de dua ra dung ki tu theo gia tri a0

lb $a0, 0($t7) # print hex[a0]

li $v0, 11 # print character

syscall

addi $t4, $t4, -1 # t4--

bgez $t4, loopH # neu t4>=0 thi tiêp tuc

jr $ra

nop

#=======================================================================

reset:

addi $t0, $zero,0 # t0: index

la $s1, array1 # disk1,2 luu du lieu

la $s2, array2

la $a2, arrayParity # luu du lieu parity

jr $ra

nop

splitting:

lb $t1, ($s0) # t1 = input[0]...

addi $t3, $t3, -1 # length: t3 = t3-1

sb $t1, ($s1) # luu input[0] -> disk1[0]

addi $s5, $s0, 4 # s5 = input[4]

lb $t2, ($s5) # t2 chua dia chi tung byte cua dick 2

addi $t3, $t3, -1 # length--

sb $t2, ($s2) # luu input[4] -> disk2[0]

xor $a3, $t1, $t2 # xor disk1[0] vs disk2[0]-> a3

sw $a3, ($a2) # a3-> array (disk3)

addi $a2, $a2, 4 # array[++]

addi $t0, $t0, 1 # numEle in disk ++

addi $s0, $s0, 1 # input++

addi $s1, $s1, 1 # disk1++

addi $s2, $s2, 1 # disk2++

bgt $t0, 3, reset # neu so phan tu trong disk ==4 -> reset disk

j splitting

nop

print\_disk1:

lb $a0, ($s1) # print disk1[++]

li $v0, 11

syscall

addi $t0, $t0, 1 # index++

addi $s1, $s1, 1 # disk1++

bgt $t0, 3, nprint # neu so phan tu in ra >4 thi ket thuc print

j print\_disk1

print\_disk2:

lb $a0, ($s2) # tuong tu print disk2

li $v0, 11

syscall

addi $t0, $t0, 1 # index\_disk2++

addi $s2, $s2, 1 # disk2\*++

bgt $t0, 3, nprint # neu so phan tu in ra >4 thi ket thuc print

j print\_disk2

print\_arrayParity:

addi $sp, $sp,-4

sw $ra, ($sp) # luu dia chi de ket thúc print\_arrayParity nhay ve vi trí lúc yêu cau gui hàm print\_arrayParity

la $a0, m5 # in "[[ "

jal print\_message

nop

loopParity:

lb $t8, ($a2) # t8 = array[i]

jal HEX # jump -> lay ma parity

li $v0, 4

la $a0, comma #print " , "

syscall

addi $t0, $t0, 1 #index++

addi $a2, $a2, 4 #array++

bgt $t0, 2, end\_printArray # in ra 3 parity dau co dau ",", parity cuoi cung k co

j loopParity

end\_printArray:

lb $t8, ($a2) #array[3]

jal HEX # jump -> lay ma parity

li $v0, 4

la $a0, m6 # in "}] "

syscall

addi $t0, $zero,0 # tra index ve 0

lw $ra, ($sp) # doc lai dia chi jjump den printArray ban dau

jr $ra

nop

nprint:

addi $t0, $zero,0 # tra index ve 0

jr $ra

nop

#================Try Again======================

ask: li $v0, 50 # hoi xem tiep tuc chuong trinh hay ko

la $a0, ms

syscall

beq $a0, 0, clear # neu co thi clear tring ve trang thai ban dau

nop

j exit # neu ko thi exit chuong trinh

nop

# clear: dua string ve trang thai ban dau de thuc hien lai qua trinh

clear:

la $s0, stringInput

add $s3, $s0, $t5 # s2: dia chi byte cuoi cung duoc su dung trong string

li $t1, 0

goAgain:

sb $t1, ($s3) # set byte o dia chi s0 thanh 0

nop

addi $s0, $s0, 1 # array\* ++

bge $s0, $s2, main # s0>s2 --> input

nop

j goAgain

nop

#=======================Exit=====================

check\_exit:

li $v0, 4 # in ra --------------- --------------- ----------------

la $a0, m2

syscall

j ask # jump ask

#======================= print các message=======================

print\_message:

li $v0, 4

syscall

jr $ra

nop

**Kết quả:**

**Graphical user interface

Description automatically generated**