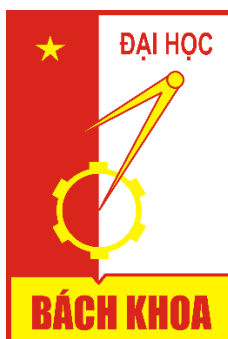


ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



ĐỒ ÁN MÔN HỌC

Đề tài: Hàm cấp phát bộ nhớ malloc().

Chương trình kiểm tra cú pháp lệnh MIPS

Giảng viên: **ĐỖ CÔNG THUẦN**

Nhóm sinh viên thực hiện: Nhóm 1

STT	Họ và tên	MSSV
1	Phạm Việt Anh	20225599
2	Mạch Ngọc Đức Anh	20225595

Hà Nội, năm 2024

1) Khai báo các biến liên quan đến vùng nhớ kernel

kdata

#Biên chứa địa chỉ đầu tiên của vùng nhớ con trong

Sys_TopOfFree: .word 1

#Vùng không gian tự do, dùng để cấp bộ nhớ cho các biến con. tro

Sys_MyFreeSpace

2) Tạo biến đếm bộ nhớ

.text

#Khởi tạo biến đếm bộ nhớ dùng thực

li \$t0, 0

#Khởi tạo biến đếm bộ nhớ dùng sau chuẩn hóa

li \$t3, 0

3) Các hàm theo yêu cầu đề bài

+) Hàm khởi tạo bộ nhớ để cấp phát động

 SysInitMem:

 la \$t9, Sys_TheTopOfFree #Lay con tro chua dau tien con trong, khoi tao

 la \$t7, Sys_MyFreeSpace #Lay dia chi dau tien con trong, khoi tao

 sw \$t7, 0(\$t9) #Luu lai

jr \$ra

+) Hàm malloc

 malloc:

 la \$t9, Sys_TheTopOfFree

 lw \$t8, 0(\$t9) #Lay dia chi dau tien con trong

 sub \$t3,\$t3,\$t8

 add \$s2, \$zero, \$a0 # luu gia trij dia chi con tro vao bien tam thoi \$s2

 li \$v0, 34

 add \$a0, \$t8, \$zero

 syscall

 li \$v0, 4

 la \$a0, Space

 syscall

 checkDivide4:

 li \$t4, 4

 div \$t8, \$t4

 mfhi \$t5

 beq \$t5, \$zero, done

 addi \$t8, \$t8,1

j checkDivide4

done:

add \$t3, \$t8, \$t3

li \$v0, 34

add \$a0, \$t8, \$zero

syscall

li \$v0, 4

la \$a0, newline

syscall

add \$a0, \$zero, \$s2 #gan lai gia tri bien tam thoi vao lai \$a0

sw \$t8, 0(\$a0) #Cat dia chi do vao bien con tro

addi \$v0, \$t8, 0 #Dong thoi la ket qua tra ve cua ham

mul \$t7, \$a1, \$a2 #Tinh kich thuoc cua mang can cap nhat

add \$t0, \$t0, \$t7

add \$t3, \$t3, \$t7

add \$t6, \$t8, \$t7 #Tinh dia chi dau tien con trong

sw \$t6, 0(\$t9) #Luu tro lai dia chi dau tien do vao bien Sys_TheTopOfFree

jr \$ra

+)Ham lay gia tri cua bien con tro

getValue:

lw \$v0, 0(\$a0)

jr \$ra

+)Ham lay dia chi cua bien con tro

getAddress:

add \$v0, \$zero, \$a0

jr \$ra

+)Ham Copy 2 con tro xau ki tu

CopyPointer:

sw \$a0, 0(\$a1)

jr \$ra

+) Hàm tính số bộ nhớ đã cấp phát

CalculateMemory:

add \$a0, \$zero, \$t0

jr \$ra

CalculateMemory2:

add \$a0, \$t3, \$zero

jr \$ra

+)Ham cap phat bo nho dong cho cac bien con tro

\$a0 Chua dia chi dau tien

\$a1 So dong

\$a2 So cot

\$v0 Dia chi cua vung nho can cap nhat

malloc2:

la \$t9, Sys_TheTopOfFree

lw \$t8, 0(\$t9) #Lay dia chi dau tien con trong

sub \$t3,\$t3, \$t8

checkDevide4_2:

li \$t4, 4

div \$t8, \$t4

mfhi \$t5

beq \$t5, \$zero, done_2

addi \$t8, \$t8,1

j checkDevide4_2

done_2:

add \$t3,\$t3,\$t8

sw \$t8, 0(\$a0) #Cat địa chỉ do vào biến con trỏ

addi \$v0, \$t8, 0 #Đồng thời là kết quả trả về của hàm

mul \$t7, \$a1, \$a2 #Tính số lượng phần tử

mul \$t7, \$t7, 4 #Tính kích thước bộ nhớ cần cung cấp

add \$t0, \$t0, \$t7

add \$t3, \$t3, \$t7

add \$t6, \$t8, \$t7 #Tính địa chỉ đầu tiên con trỏ

sw \$t6, 0(\$t9) #Lưu trữ lại địa chỉ đầu tiên do vào biến Sys_TheTopOfFree

jr \$ra

\$a0 địaChỉĐầu

\$a1 dòng

\$a2 cột

\$v0 kết quả

Hàm lấy ra giá trị ở dòng i cột j

GetArrayAt:

la \$t1, MaxDòng

lw \$t1, 0(\$t1)

la \$t2, MaxCột

lw \$t2, 0(\$t2)

mul \$t7, \$a1, \$t2

add \$t7, \$t7, \$a2

mul \$t7, \$t7, 4

add \$a0, \$a0, \$t7

lw \$v0, 0(\$a0)

jr \$ra

Hàm thiết lập giá trị ở cột i dòng j

SetArrayAt:

la \$t1, MaxDong

lw \$t1, 0(\$t1)

la \$t2, MaxCot

lw \$t2, 0(\$t2)

mul \$t7, \$a1, \$t2

add \$t7, \$t7, \$a2

mul \$t7, \$t7, 4

add \$a0, \$a0, \$t7

sw \$v0, 0(\$a0)

jr \$ra

+) Nếu $i > \text{maxdong}$ hoặc $j > \text{maxcot}$

exit:

li \$v0, 4

la \$a0, exitMess

syscall

li \$v0, 10

syscall

+) Kết thúc chương trình

end:

li \$v0, 4

la \$a0, endmess

Chạy với ví dụ

Cap phát cho biến con trỏ, gồm 3 phần tử, mỗi phần tử 1 byte

la \$a0, CharPtr

addi \$a1, \$zero, 3

addi \$a2, \$zero, 1

li \$v0, 4

la \$a0, Message11

syscall

jal malloc

#-----

li \$v0, 4

la \$a0, Message21

syscall

la \$a0, CharPtr

jal getValue

add \$a0, \$v0, \$zero

li \$v0, 1

syscall

li \$v0, 4

la \$a0, newline

syscall

li \$v0, 4

la \$a0, Message31

syscall

la \$a0, CharPtr

jal getAddress

add \$a0, \$v0, \$zero

li \$v0, 34

syscall


```

li $v0, 4
la $a0, newline
syscall

#-----
# Cap phat cho bien con tro, gom 6 phan tu, moi phan tu 1 byte
#-----

la $a0, BytePtr
addi $a1, $zero, 6
addi $a2, $zero, 1
li $v0, 100
sw $v0, 0($a0)
li $v0, 4
la $a0, Message12
syscall
jal malloc

li $v0, 4
la $a0, Message22
syscall

la $a0, BytePtr
jal getValue
add $a0, $v0, $zero
li $v0, 1
syscall

```

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

```
li $v0, 4
la $a0, Message32
syscall
```

```
la $a0, BytePtr
jal getAddress
add $a0, $v0, $zero
li $v0, 34
syscall
```

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

```
#-----
# Cap phat cho bien con tro, gom 5 phan tu, moi phan tu 4 byte
#-----
```

```
la $a0, WordPtr
addi $a1, $zero, 5
addi $a2, $zero, 4
```

```
li $v0, 4
la $a0, Message13
```

syscall

jal malloc

li \$v0, 4

la \$a0, Message23

syscall

la \$a0, WordPtr

jal getValue

add \$a0, \$v0, \$zero

li \$v0, 1

syscall

li \$v0, 4

la \$a0, newline

syscall

li \$v0, 4

la \$a0, Message33

syscall

la \$a0, WordPtr

jal getAddress

add \$a0, \$v0, \$zero

li \$v0, 34

syscall

li \$v0, 4

la \$a0, newline

syscall

#-----

Question 4: Copy hai con tro

#-----

li \$v0, 4

la \$a0, Message4

syscall

la \$a0, CharPtr

li \$v0, 34

syscall

li \$v0, 4

la \$a0, Space

syscall

la \$a0, CharPtr

la \$a1, CpyCharPtr

jal CopyPointer

add \$a1,\$a1,\$zero

li \$v0, 34

syscall

li \$v0, 4

la \$a0, newline

syscall

#-----

#Question 5: Tinh tong bo nho su dung

#-----

li \$v0, 4

la \$a0, Message5

syscall

jal CalculateMemory

li \$v0, 1

syscall

li \$v0, 4

la \$a0, Space

syscall

jal CalculateMemory2

li \$v0, 1

syscall

li \$v0, 4

la \$a0, newline

syscall

#-----

Cap phat cho mang con tro word 2 chieu, gom 5 dong, 5 cot moi phan tu 4 byte

#-----

la \$a0, ArrayWordPtr

la \$a1, MaxDong

lw \$a1, 0(\$a1)

la \$a2, MaxCot

lw \$a2, 0(\$a2)

jal malloc2

li \$v0, 4

la \$a0, Message6

syscall

jal CalculateMemory

li \$v0, 1

syscall

li \$v0, 4

la \$a0, Space

syscall

jal CalculateMemory2

li \$v0, 1

syscall

li \$v0, 4

la \$a0, newline

syscall

```
li $v0, 4
la $a0, Message71
syscall
```

```
li $v0, 1007
la $a0, ArrayWordPtr
li $a1, 3
la $s0, MaxDong
lw $s0, 0($s0)
bge $a1, $s0, exit
li $a2, 3
la $s0, MaxCot
lw $s0, 0($s0)
bge $a2, $s0, exit
jal SetArrayAt
li $v0, 34
syscall
```

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

```
li $v0, 4
la $a0, Message72
syscall
```

la \$a0, ArrayWordPtr

li \$a1, 3

la \$s0, MaxDong

lw \$s0, 0(\$s0)

bge \$a1, \$s0, exit

li \$a2, 3

la \$s0, MaxCot

lw \$s0, 0(\$s0)

bge \$a2, \$s0, exit

jal GetArrayAt

add \$a0, \$v0, \$zero

li \$v0, 1

syscall

li \$v0, 4

la \$a0, newline

syscall

j end

Project 6

CHƯƠNG II. Chương trình kiểm tra cú pháp lệnh MIPS

II.1. YÊU CẦU

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 hiển 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 thuộc dạng lệnh nào (R, I, J) và cần bao nhiêu chu kì thì mới thực hiện xong.

II.2. Ý tưởng thực hiện

1. Thiết lập tên các câu lệnh , thành phần của từng câu lệnh, dạng lệnh (I,R,J) và chu kỳ của từng câu lệnh
2. Nhập câu lệnh muốn kiểm tra và lưu vào bộ nhớ
3. Tách từng thành phần của câu lệnh và lưu vào bộ nhớ: opcode, operand 1, operand 2, operand 3
4. Kiểm tra từng thành phần của câu lệnh, nếu sai ở thành phần nào thì chương trình sẽ dừng lại và không kiểm tra tiếp. Nếu đúng thì in ra dạng lệnh và số chu kỳ.

II.3. Khởi tạo dữ liệu ban đầu

```

type: .asciiiz "x", "r", "i", "f", "l", "s"
register: .asciiiz
    "$zero", "$at", "$v0", "$v1", "$a0", "$a1", "$a2", "$a3",
    "$t0", "$t1", "$t2", "$t3", "$t4", "$t5", "$t6", "$t7",
    "$s0", "$s1", "$s2", "$s3", "$s4", "$s5", "$s6", "$s7",
    "$t8", "$t9", "$k0", "$k1", "$gp", "$sp", "$fp", "$ra",
    "$0", "$1", "$2", "$3", "$4", "$5", "$6", "$7", "$8",
    "$9", "$10", "$11", "$12", "$13", "$14", "$15", "$16",
    "$17", "$18", "$19", "$20", "$21", "$22", "$23", "$24",
    "$25", "$26", "$27", "$28", "$29", "$30", "$31", "$32", "\0"

float: .asciiiz
    "$f0", "$f1", "$f2", "$f3", "$f4", "$f5", "$f6", "$f7",
    "$f8", "$f9", "$f10", "$f11", "$f12", "$f13", "$f14", "$f15",
    "$f16", "$f17", "$f18", "$f19", "$f20", "$f21", "$f22", "$f23",
    "$f24", "$f25", "$f26", "$f27", "$f28", "$f29", "$f30", "$f31", "\0"

```

-Type: chứa các thành phần của lệnh:

Trong đó:

x: không chứa gì

r: thanh ghi (register)

i: số (immediately)

f: thanh ghi số thực (float)

l: nhãn dán (label)

s: thành phần đặc biệt

-Register: chứa tên của 32 thanh ghi và có được sắp xếp

-Float: chứa tên của các thanh ghi làm việc với số thực

```

instructions: .asciiiz
    "abs.d", "ffx", " ", "1",
    "abs.s", "ffx", " ", "1",
    "add", "rrr", "R", "4",
    "add.d", "rrr", " ", "1",
    "add.s", "rrr", " ", "1",
    "addi", "rri", "I", "4",
    "addiu", "rri", "I", "4",
    "addu", "rrr", "R", "4",
    "and", "rrr", "R", "4",
    "andi", "rri", "I", "4",
    "bcif", "lxx", " ", "2",
    "bcit", "lxx", " ", "2",
    "beq", "rrl", "I", "3",
    "bgez", "rlx", "I", "3",
    "bgezal", "rlx", "I", "3",
    "bgtz", "rlx", "I", "3",
    "blez", "rlx", "I", "3",
    "bltz", "rlx", "I", "3",
    "bltzal", "rlx", "I", "3",
    "bne", "rrl", "I", "3",
    "break", "xxx", "R", "4",
    "c.eq.d", "ffx", " ", "2",
    "c.eq.s", "ffx", " ", "2",
    "c.le.d", "ffx", " ", "2",
    "c.le.s", "ffx", " ", "2",
    "c.lt.d", "ffx", " ", "2",
    "c.lt.s", "ffx", " ", "2",

```

Instructions: chứa các lệnh được sắp xếp theo trình tự trong bảng chữ cái, các thành phần có trong lệnh, dạng lệnh và số chu kỳ thực hiện

```

#character
tab      : "\t"
null     : "\0"
enter    : "\n"
space    : " "
comma    : ","
mark     : "*"
colon    : ":"

#say sth
Message:  .asciiiz "Enter an Assembly command: "
Message1: .asciiiz "Type of Instruction: "
valid:    .asciiiz "valid"
invalid:  .asciiiz "invalid"
say_opcode: .asciiiz "Opcode "
say_operand: .asciiiz "Operand "
say_cycle: .asciiiz "Number of Clock Cycles: "

```

Khởi tạo các ký tự đặc biệt và các câu dẫn dắt khi thực hiện chương trình

II.4. Thực hiện chương trình

Bước 1 : Lấy ra opcode

```

opcode_take:
    addi $s1, $s1, 1
    sb $s2, 0($s1)
    addi $s0, $s0, 1
    lb $s2, 0($s0)
    lb $s3, 0($t0)
    beq $s3, $s2, end_take_opcode
    lb $s3, 0($t1)
    beq $s3, $s2, end_take_opcode
    lb $s3, 0($t2)
    beq $s3, $s2, end_take_opcode
    lb $s3, 0($t3)
    beq $s3, $s2, print_opcode
    j opcode_take
end_take_opcode:
    addi $s0, $s0, 1
    lb $s2, 0($s0)
    lb $s3, 0($t0)
    beq $s3, $s2, end_take_opcode
    lb $s3, 0($t3)
    beq $s3, $s2, end_take_opcode
    lb $s3, 0($t1)
    beq $s3, $s2, end_take_opcode
    lb $s3, 0($t2)
    beq $s3, $s2, end_take_opcode
    la $s4, opr1
    addi $s4, $s4, -1

```

Bước 2 : Lấy ra toán tử đầu tiên

```

opr1_take:
    addi $s4, $s4, 1
    sb $s2, 0($s4)
    addi $s0, $s0, 1
    lb $s2, 0($s0)
    lb $s3, 0($t0)
    beq $s3, $s2, end_take_opr1
    lb $s3, 0($t1)
    beq $s3, $s2, end_take_opr1
    lb $s3, 0($t2)
    beq $s3, $s2, end_take_opr1
    lb $s3, 0($t3)
    beq $s3, $s2, print_opcode
    j opr1_take
end_take_opr1:
    addi $s0, $s0, 1
    lb $s2, 0($s0)
    lb $s3, 0($t0)
    beq $s3, $s2, end_take_opr1
    lb $s3, 0($t3)
    beq $s3, $s2, end_take_opr1
    lb $s3, 0($t1)
    beq $s3, $s2, end_take_opr1
    lb $s3, 0($t2)
    beq $s3, $s2, end_take_opr1
    la $s4, opr2
    addi $s4, $s4, -1

```

Bước 3 : Lấy ra toán tử thứ 2

```

opr2_take:
    addi $s4, $s4, 1
    sb $s2, 0($s4)
    addi $s0, $s0, 1
    lb $s2, 0($s0)
    lb $s3, 0($t0)
    beq $s3, $s2, end_take_opr2
    lb $s3, 0($t1)
    beq $s3, $s2, end_take_opr2
    lb $s3, 0($t2)
    beq $s3, $s2, end_take_opr2
    lb $s3, 0($t3)
    beq $s3, $s2, print_opcode
    j opr2_take
end_take_opr2:
    addi $s0, $s0, 1
    lb $s2, 0($s0)
    lb $s3, 0($t0)
    beq $s3, $s2, end_take_opr2
    lb $s3, 0($t2)
    beq $s3, $s2, end_take_opr2
    lb $s3, 0($t3)
    beq $s3, $s2, end_take_opr2
    lb $s3, 0($t1)
    beq $s3, $s2, end_take_opr2
    la $s4, opr3
    addi $s4, $s4, -1

```

Bước 4 : Lấy ra toán tử cuối cùng và in ra opcode đã lấy ra

```

opr3_take:
    addi $s4, $s4, 1
    sb $s2, 0($s4)
    addi $s0, $s0, 1
    lb $s2, 0($s0)
    lb $s3, 0($t3)
    beq $s3, $s2, print_opcode
    j opr3_take
print_opcode:
    la $s0, instructions
    la $s1, opcode
    li $v0, 4
    la $a0, say_opcode
    syscall
    li $v0, 4
    la $a0, mark
    syscall
    li $v0, 4
    la $a0, opcode
    syscall
    li $v0, 4
    la $a0, mark
    syscall
    li $v0, 4
    la $a0, colon
    syscall
    li $v0, 4
    la $a0, space
    syscall

```

Bước 5 : Kiểm tra opcode đã lưu có hợp lệ hay không bằng cách kiểm tra xem nó có trong phần dữ liệu đã khai báo hay không?

```
check_opcode:
    lb $s2, 0($s0)
    lb $s3, 0($s1)
    bne $s2, $s3, next_opcode
    addi $s0, $s0, 1
    addi $s1, $s1, 1
    lb $s4, 0($s1)
    beqz $s4, opcode_endcheck
    j check_opcode
next_opcode:
    beqz $s2, next_instruct
    addi $s0, $s0, 1
    lb $s2, 0($s0)
    j next_opcode
next_instruct:
    addi $s0, $s0, 9
    lb $s2, 0($s0)
    beqz $s2, invalid_opcode
    la $s1, opcode
    j check_opcode
opcode_endcheck:
    lb $t9, 0($s0)
    beqz $t9, valid_opcode
```

Bước 6 : Kiểm tra lần lượt 3 toán tử đã lưu xem có đúng với thành phần lệnh đã lưu khi khai báo hay không?

```
check_opr1:
    la $s2, opr1
    lb $t4, 0($s0)
    jal check_type
check_opr2:
    addi $s0, $s0, 1
    la $s2, opr2
    lb $t4, 0($s0)
    jal check_type
check_opr3:
    addi $s0, $s0, 1
    la $s2, opr3
    lb $t4, 0($s0)
    jal check_type
    j print_type
```

Kiểm tra lần lượt từng thành phần so với type

```

check_type:
    la $s4, register
    la $s3, type
    lb $t5, 0($s3)
    beq $t4, $t5, x_check
    li $v0, 4
    la $a0, say_operand
    syscall
    li $v0, 4
    la $a0, mark
    syscall
    li $v0, 4
    addi $a0, $s2, 0
    syscall
    li $v0, 4
    la $a0, mark
    syscall
    li $v0, 4
    la $a0, colon
    syscall
    li $v0, 4
    la $a0, space
    syscall
    addi $s3, $s3, 2
    lb $t5, 0($s3)
    beq $t4, $t5, check_r
    addi $s3, $s3, 2
    lb $t5, 0($s3)
    beq $t4, $t5, check_i

    addi $s3, $s3, 2
    lb $t5, 0($s3)
    beq $t4, $t5, check_f
    addi $s3, $s3, 2
    lb $t5, 0($s3)
    beq $t4, $t5, check_l
    addi $s3, $s3, 2
    lb $t5, 0($s3)
    la $s4, register
    beq $t4, $t5, check_s

```

Bước 7 : Kiểm tra với từng thành phần

Kiểm tra thành phần rỗng (x)

```

x_check:
    lb $t6, 0($s2)
    bnez $t6, invalid_x
    jr $ra
invalid_x:
    li      $v0, 4
    la      $a0, say_operand
    syscall
    li      $v0, 4
    la      $a0, mark
    syscall
    li      $v0, 4
    addi    $a0, $s2, 0
    syscall
    li      $v0, 4
    la      $a0, mark
    syscall
    li      $v0, 4
    la      $a0, colon
    syscall
    li      $v0, 4
    la      $a0, space
    syscall

```

Kiểm tra thành phần thanh ghi (r)

```

check_r:
    la $s4, register
    add $s7, $s2, $0
    j r_loop
r_loop:
    lb $t5, 0($s4)
    lb $t6, 0($s7)
    bne $t5, $t6, r_next
    addi $s4, $s4, 1
    addi $s7, $s7, 1
    lb $t8, 0($s7)
    beqz $t8, return_valid
    j r_loop
r_next:
    addi $s4, $s4, 1
    lb $t7, 0($s4)
    bnez $t7, r_next
    addi $s4, $s4, 1
    lb $t7, 0($s4)
    beqz $t7, return_invalid
    add $s7, $s2, $0
    j r_loop

```

Kiểm tra thành phần số nguyên (i)


```

check_i:
    lb $t5, 0($s2)
    beq $t5, '-', i_next
    beq $t5, '+', i_next
i_loop:
    lb $t9, 0($s2)
    blt $t9, 0x30, return_invalid
    bgt $t9, 0x39, return_invalid
    addi $s2, $s2, 1
    lb $t9, ($s2)
    beqz $t9, return_valid
    j i_loop
i_next:
    addi $s2, $s2, 1
    j check_i

```

Kiểm tra thành phần thanh ghi số thực (f)

```

check_f:
    la $s4, float
    add $s7, $s2, $0
    j f_loop
f_loop:
    lb $t5, 0($s4)
    lb $t6, 0($s7)
    bne $t5, $t6, f_next
    addi $s4, $s4, 1
    addi $s7, $s7, 1
    lb $t8, 0($s7)
    beqz $t8, return_valid
    j f_loop
f_next:
    addi $s4, $s4, 1
    lb $t7, 0($s4)
    bnez $t7, f_next
    addi $s4, $s4, 1
    lb $t7, 0($s4)
    beqz $t7, return_invalid
    add $s7, $s2, $0
    j f_loop

```

Kiểm tra thành phần nhãn dán (l)

```

check_l:
    lb $t5, 0($s2)
    addi $s2, $s2, 1
    blt $t5, 'A', return_invalid
    ble $t5, 'Z', l_next
    blt $t5, 'a', return_invalid
    ble $t5, 'z', l_next

l_loop:
    lb $t5, 0($s2)
    beq $t5, '_', i_next
    blt $t5, 'A', return_invalid
    ble $t5, 'Z', l_next
    blt $t5, 'a', return_invalid
    ble $t5, 'z', l_next

l_next:
    addi $s2, $s2, 1
    lb $t5, 0($s2)
    beqz $t5, return_valid
    j l_loop

```

Kiểm tra thành phần đặc biệt (s)

```

check_s:
    lb $t5, 0($s2)
    beq $t5, '+', s_next
    beq $t5, '-', s_next

s_loopnumber:
    beq $t5, '(', s_rcheck
    blt $t5, '0', return_invalid
    bgt $t5, '9', return_invalid

s_next:
    addi $s2, $s2, 1
    lb $t5, 0($s2)
    beqz $t5, return_valid
    j s_loopnumber

s_rcheck:
    addi $s2, $s2, 1
    add $s7, $s2, $0
    j s_rloop

s_rloop:
    lb $t5, 0($s4)
    lb $t6, 0($s7)
    bne $t5, $t6, s_rnext
    addi $s4, $s4, 1
    addi $s7, $s7, 1
    lb $t8, 0($s7)
    beq $t8, ')', return_valid
    j s_rloop

s_rnext:
    addi $s4, $s4, 1
    lb $t7, 0($s4)
    bnez $t7, s_rnext
    addi $s4, $s4, 1
    lb $t7, 0($s4)
    beqz $t7, return_invalid
    add $s7, $s2, $0
    j s_rloop

```

Bước 8 : In ra dạng lệnh và số chu kỳ thực hiện lệnh

```

print_type:
    addi $s0, $s0, 2
    lb $t5, 0($s0)
    beq $t5, ' ', clock_cycles
    li $v0, 4
    la $a0, Message1
    syscall
    li $v0, 11
    add $a0, $t5, $0
    syscall
clock_cycles:
    addi $s0, $s0, 2
    lb $t5, 0($s0)
    li $v0, 4
    la $a0, enter
    syscall
    li $v0, 4
    la $a0, say_cycle
    syscall
    li $v0, 11
    add $a0, $t5, $0
    syscall
    j endmain

```

II.5. Kết quả mô phỏng

TH 1 : Với dữ liệu đầu vào là add \$t2, \$t3, \$t4

Ta thu được kết quả sau :

```

Enter an Assembly command: add $t2, $t3, $t4
Opcode 'add': valid
Operand '$t2': valid
Operand '$t3': valid
Operand '$t4': valid
Type of Instruction: R
Number of Clock Cycles: 4

```

⇒ Kết quả đúng

TH2 : Với dữ liệu đầu vào là sb \$t2, -45(\$f2)

Ta thu được kết quả sau :

```

Enter an Assembly command: sb $t2, -45($f2)
Opcode 'sb': valid
Operand '$t2': valid
Operand '-45($f2)': invalid
-- program is finished running (dropped off bottom) --

```

⇒ Kết quả đúng

TH3 : Với dữ liệu đầu vào là j vietanh

Ta thu được kết quả sau :

```
Enter an Assembly command: j vietanh
Opcode 'j': valid
Operand 'vietanh': valid
Type of Instruction: J
Number of Clock Cycles: 3
-- program is finished running (dropped off bottom) --
```

⇒ Kết quả đúng

KẾT LUẬN

Với đề tài : Chương trình kiểm tra cú pháp lệnh MIPS

Chương trình giúp ta kiểm tra tính hợp lý của các câu lệnh trước khi dịch câu lệnh đó sang mã máy (machine code) giúp tiết kiệm thời gian và tiết kiệm dung lượng bộ nhớ

Kết quả thu được sau khi hoàn thành đồ án :

- +) Thành thạo sử dụng ngôn ngữ lập trình hợp ngữ
- +) Hiểu rõ bản chất của cấu trúc lệnh và cách thức hoạt động của MIPS
- +) Cải thiện kỹ năng làm việc nhóm

TÀI LIỆU THAM KHẢO

[Tài liệu: https://en.wikipedia.org/wiki/Cycles_per_instruction

1

]

[Tài liệu: Computer Organization and Design 4th

2

]

