

# FINAL PROJECT – REPORT

Thực hành kiến trúc máy tính - IT4182

Đề tài (7) , (9)

Nguyễn Hải Dương

Nhóm 12 – Lớp 130999

Nguyễn Hải Dương – 20194530

Lê Văn Duẩn - 20194508

## 1. Đề tài (7): Chương trình kiểm tra cú pháp lệnh MIPS

### 1.1. Nội dung 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 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.

### 1.2. Thuật toán

Mô tả bằng C:

// Mảng các opcode để kiểm tra

```

Array opCodes = {"beq", "add", "sll", "sub", "j"};
Array registers; // Mảng 32 thanh ghi

// Các biến lưu các giá trị khi đọc đầu vào
char op[40], tmp[40], tmp2[40], tmp3[40];
// Biến lưu giá trị đầu vào
char input[100];
int main()
{
    int res_op;
    gets(input);
    op = read_opCode(input);
    lowerCase(op);
    res_op = check_opCode(op);
    tmp = read_1st_operand(input);
    tmp2 = read_2nd_operand(input);
    tmp3 = read_2nd_operand(input);
    switch (res_op)
    {
        case 0:
            syntax_beq();
            break;
        case 1:
            syntax_add();
            break;
        case 2:
            syntax_sll();
            break;
        case 3:
            syntax_sub();
            break;
        case 4:
            syntax_j();

```

```

        break;
    }
    return 0;
}

```

### Các hàm con:

+ read\_opcode(input);

Hàm duyệt chuỗi input là chuỗi mà người dùng nhập lệnh vào, khi gặp kí tự ' ' thì dừng và lưu các ký tự trước đó vào biến op

+ lowerCase(op); và res\_op = check\_opcode(op);

Vì opcode có thể là chữ hoa, thường nên phải dùng hàm lowerCase()

Check\_opcode: nhận vào op là opcode, kiểm tra xem có tồn tại trong mảng opCodes hay không.

Nếu không: -> Báo lỗi, dừng chương trình

Nếu có: trả về index trong mảng để xác định được lệnh.

+ read\_1st\_operand(input);

Trả về chuỗi chứa toán hạng đầu tiên

Nhận vào input và tiếp tục duyệt sau khi đọc được opcode. Dừng và lưu toán hạng khi gặp dấu ','.

+ read\_2nd\_operand(input); -> tương tự hàm trên

+ read\_3rd\_operand(input);

Hoạt động tương tự 2 hàm trên

Vì chỉ có 3 toán hạng nên điều kiện dừng duyệt và lưu toán hạng cuối này '\n'.

+ syntax\_beq(); và các hàm syntax khác

Kiểm tra các toán hạng lưu trong các biến toàn cục có phù hợp với cú pháp của lệnh không.

Ví dụ như beq: toán hạng 1,2 là thanh ghi, toán hạng 3 là nhãn -> tmp, tmp2 phải tồn tại trong mảng registers. Tmp3 thỏa mãn nhãn.

## Mã nguồn

```
.data
reg_0: .asciiz "$0"
reg_zero: .asciiz "$zero"
reg_1: .asciiz "$1"
reg_at: .asciiz "$at"
reg_2: .asciiz "$2"
reg_v0: .asciiz "$v0"
reg_3: .asciiz "$3"
reg_v1: .asciiz "$v1"
reg_4: .asciiz "$4"
reg_a0: .asciiz "$a0"
reg_5: .asciiz "$5"
reg_a1: .asciiz "$a1"
reg_6: .asciiz "$6"
reg_a2: .asciiz "$a2"
reg_7: .asciiz "$7"
reg_a3: .asciiz "$a3"
reg_8: .asciiz "$8"
reg_t0: .asciiz "$t0"
reg_9: .asciiz "$9"
reg_t1: .asciiz "$t1"
reg_10: .asciiz "$10"
reg_t2: .asciiz "$t2"
reg_11: .asciiz "$11"
reg_t3: .asciiz "$t3"
reg_12: .asciiz "$12"
reg_t4: .asciiz "$t4"
reg_13: .asciiz "$13"
reg_t5: .asciiz "$t5"
reg_14: .asciiz "$14"
reg_t6: .asciiz "$t6"
reg_15: .asciiz "$15"
reg_t7: .asciiz "$t7"
reg_16: .asciiz "$16"
reg_s0: .asciiz "$s0"
reg_17: .asciiz "$17"
reg_s1: .asciiz "$s1"
reg_18: .asciiz "$18"
reg_s2: .asciiz "$s2"
reg_19: .asciiz "$19"
```

```

reg_s3: .asciiiz "$s3"
reg_20: .asciiiz "$20"
reg_s4: .asciiiz "$s4"
reg_21: .asciiiz "$21"
reg_s5: .asciiiz "$s5"
reg_22: .asciiiz "$22"
reg_s6: .asciiiz "$s6"
reg_23: .asciiiz "$23"
reg_s7: .asciiiz "$s7"
reg_24: .asciiiz "$24"
reg_t8: .asciiiz "$t8"
reg_25: .asciiiz "$25"
reg_t9: .asciiiz "$t9"
reg_26: .asciiiz "$26"
reg_k0: .asciiiz "$k0"
reg_27: .asciiiz "$27"
reg_k1: .asciiiz "$k1"
reg_28: .asciiiz "$28"
reg_gp: .asciiiz "$gp"
reg_29: .asciiiz "$29"
reg_sp: .asciiiz "$sp"
reg_30: .asciiiz "$30"
reg_fp: .asciiiz "$fp"
reg_31: .asciiiz "$31"
reg_ra: .asciiiz "$ra"

```

# Lưu các tên thanh ghi ở trên vào mảng resgister

registers:

```

    .word reg_0, reg_1, reg_2, reg_3, reg_4, reg_5, reg_6, reg_7, reg_8, reg_9,
reg_10, reg_11, reg_12, reg_13, reg_14, reg_15
    .word reg_16, reg_17, reg_18, reg_19, reg_20, reg_21, reg_22, reg_23,
reg_24, reg_25, reg_26, reg_27, reg_28, reg_29, reg_30, reg_31
    .word reg_zero, reg_at, reg_v0, reg_v1, reg_a0, reg_a1, reg_a2, reg_a3,
reg_t0, reg_t1, reg_t2, reg_t3, reg_t4, reg_t5, reg_t6, reg_t7
    .word reg_s0, reg_s1, reg_s2, reg_s3, reg_s4, reg_s5, reg_s6, reg_s7,
reg_t8, reg_t9, reg_k0, reg_k1, reg_gp, reg_sp, reg_fp, reg_ra

```

# Trong đề tài này, nhóm quyết định chọn 5 lệnh với opCode dưới đây

# Trong 5 lệnh đều đủ 3 khuôn dạng R(lệnh Add, Sub, Sll), I(lệnh Beq), J(lệnh J)

```

beq_op: .asciiiz "beq"
add_op: .asciiiz "add"
sll_op: .asciiiz "sll"

```

```

sub_op: .ascii "sub"
j_op: .ascii "j"
# Lưu các opCode vào cùng một mảng
opCodes: .word beq_op, add_op, sll_op, sub_op, j_op

# các biến trung gian
op: .space 4          # Biến trung gian lưu opCode khi đọc được từ đầu vào
tmp: .space 40         # Biến trung gian lưu toán hạng thứ 1 khi đọc được từ đầu vào
tmp2: .space 40        # Biến trung gian lưu toán hạng thứ 2 được từ đầu vào
tmp3: .space 40        # Biến trung gian toán hạng thứ 3 được từ đầu vào

# Các thông báo của chương trình
mess: .ascii "Nhap code mips: "
cycle: .ascii "\nSo chu ki cua lenh nay la: "

# Các kết quả, lỗi khi kiểm tra cú pháp
valid_beq: .ascii "\nopcode: beq, hop le!"
valid_add: .ascii "\nopcode: add, hop le!"
valid_sll: .ascii "\nopcode: sll, hop le!"
valid_sub: .ascii "\nopcode: sub, hop le!"
valid_j: .ascii "\nopcode: j, hop le!"
valid_operand1: .ascii "\nToan hang: "
valid_operand2: .ascii ", hop le!"
valid_command: .ascii "\nCau lenh dung cu phap!"

# Các lỗi
invalid_op: .ascii "\nopcode: khong hop le!"
invalid_operand1: .ascii "\nToan hang: "
invalid_operand2: .ascii ", khong hop le!"

# Biến chứa lệnh nhập vào
input: .space 100

.text
main: li $v0, 4          # In thông báo nhập lệnh để kiểm tra cú pháp
     la $a0, mess
     syscall

     li $v0, 8          # Đọc chuỗi nhập vào và lưu vào biến input
     la $a0, input

```

```

    li $a1, 100
    syscall

    li $t0, 0          # khởi tạo biến chạy để duyệt input

# Duyệt chuỗi input và đọc opCode khi gặp dấu ' ' đầu tiên
op_loop: lb $t1, input($t0)
    beq $t1, ' ', end_op_loop
    nop
    sb $t1, op($t0)
    addi $t0, $t0, 1
    j op_loop
end_op_loop:
    jal lowerCase      # Vì opCode nhập vào không quan tâm hoa thường
    nop               # Sử dụng hàm lowerCase sau đó so sánh với các opCod

    jal check_opCode # Kiểm tra opCode vừa đọc được lưu trong biến op
    nop

# Đọc toán hạng đầu tiên và lưu trong biến tmp
# Bằng cách đọc đến khi gặp dấu ','
    addi $t0, $t0, 1 # Tiếp tục duyệt input
    li $t9, 0        # index cho các mảng tmp, tmp2, tmp3 phục vụ việc lưu vào
đúng vị trí
loop_1: lb $t2, input($t0) # lưu giá trị phần tử input[$t0]
    sb $t2, tmp($t9)

    beq $t2, '\n', end_input1 # kết thúc chuỗi
    nop
    beq $t2, ' ', eat_space1  # đi đến hàm loại bỏ dấu ' '
    nop

    beq $t2, ',', end_1st_operand # nếu gặp dấu ',' -> kết thúc toán hạng 1
    nop

    addi $t0, $t0, 1
    addi $t9, $t9, 1
    j loop_1
eat_space1:
    addi $t0, $t0, 1 # tăng index của input, tiếp tục đọc
    j loop_1

```

```

end_1st_operand:
    sb $0, tmp($t9)
    li $t9, 0          # Đã lưu xong cho tmp, đặt lại biến index để bắt đầu lưu
vào tmp2

# Đọc toán hạng thứ 2 và lưu trong biến tmp2
# Bằng cách đọc đến khi gặp dấu ','
    addi $t0, $t0, 1
loop_2: lb $t2, input($t0)
    sb $t2, tmp2($t9)

    beq $t2, '\n', end_input2
    nop
    beq $t2, ' ', eat_space2
    nop
    beq $t2, ',', end_2nd_operand
    nop
    addi $t0, $t0, 1
    addi $t9, $t9, 1
    j loop_2
eat_space2:
    addi $t0, $t0, 1
    j loop_2
end_2nd_operand:
    sb $k0, tmp2($t9)
    li $t9, 0          # Đã lưu xong cho tmp2, đặt lại biến index để bắt đầu lưu
vào tmp3

# Đọc toán hạng thứ 3 và lưu trong biến tmp3
# Vì các lệnh chỉ tối đa 3 toán hạng nên toán hạng cuối đọc đến khi kết thúc
chuỗi
    addi $t0, $t0, 1

loop_3: lb $t2, input($t0)
    sb $t2, tmp3($t9)

    beq $t2, '\n', end_3rd_operand
    nop

    beq $t2, ' ', eat_space3
    nop

```



```

        addi $t0, $t0, 1
        addi $t9, $t9, 1
        j loop_3
eat_space3:
        addi $t0, $t0, 1
        j loop_3
end_3rd_operand:
        sb $k0, tmp3($t9)
        li $t9, 0
        addi $t0, $t0, 1

# Dựa vào giá trị trả về của hàm Check_opCode lưu trong $k1 để xác định lệnh ->
# trả về index mảng opCodes
# và thực hiện kiểm tra
switchCase:
        beqz $k1, syntax_beq # TH: $k1 = 0 -> lệnh beq
        nop                  # syntax_beq thực hiện kiểm tra các toán hạng đối với lệnh
        beq

        li $a0, 1            # TH: $k1 = 1 -> lệnh add
        beq $k1, $a0, syntax_add # syntax_add thực hiện kiểm tra các toán hạng đối
        với lệnh add
        nop

        li $a0, 2            # TH: $k1 = 2 -> lệnh sll
        beq $k1, $a0, syntax_sll # syntax_sll thực hiện kiểm tra các toán hạng đối
        với lệnh sll
        nop

        li $a0, 3            # TH: $k1 = 3 -> lệnh sub
        beq $k1, $a0, syntax_sub # syntax_sub thực hiện kiểm tra các toán hạng đối
        với lệnh sub
        nop

        li $a0, 4            # TH: $k1 = 4 -> lệnh j
        beq $k1, $a0, syntax_j # syntax_j thực hiện kiểm tra các toán hạng đối với
        lệnh j
        nop

end_main:  li $v0, 10
           syscall
#-----

```

```

#-----
# end_input1, end_input2: các nhãn xử lý khi trong quá trình đọc toán hạng 1,
toán hạng 2
#-----
end_input1: sb $k0, tmp($t9)
            j switchCase
end_input2: sb $k0, tmp2($t9)
            j switchCase

#-----
#-----
# function check_opCode()
# Kiểm tra opCode có phải là 1 trong 5 lệnh mình kiểm tra hay không
# Tham số vào: giá trị opCode được lưu trong biến opCode
# Trả về: 0 -> beq
#      1 -> add
#      2 -> sll
#      3 -> sub
#      4 -> j
#      và được lưu trong $k1
#-----
check_opCode:
    add $v1, $ra, $0      # lưu địa chỉ trả về của hàm tại $v1
    la $t8, op            # lưu địa chỉ biến op vào $t8
    li $t3, 0             # Biến chạy cho mảng
    li $t4, 20            # vì có 5 lệnh và lưu các lệnh theo word -> 4*5 = 20

loop_opCodes:    beq $t3, $t4, err_op # Khi duyệt hết mảng mà ko hợp lệ -> báo
lỗi

                lw $t9, opCodes($t3) # lấy giá trị từ phần tử opCodes[$t3]

                jal str_compare      # so sánh op đọc được với từng phần tử trong
opCodes
                nop                  # 2 tham số là 2 địa chỉ 2 chuỗi lưu ở $t8, $t9
                                     # kết quả trả về $t5: 0 -> giống nhau, 1 -> khác nhau
                beqz $t5, end_check_opCode # Nếu op trùng với 1 phần tử trong opCodes
-> thỏa mãn
                nop                  # trả về $k1 lưu index chính là $t3

                addi $t3, $t3, 4 # Tăng lên 4 vì đang làm việc với word

                j loop_opCodes      # lặp

```

```

end_check_opCode:      div $k1, $t3, 4 # chia 4 để lấy index của phần tử và lưu
vào $k1

                        jr $v1          # trở lại hàm main

#-----
#-----
# function check_register()
# Kiểm tra toán hạng đưa vào có phải là 1 trong 32 thanh ghi hay không
# So sánh toán hạng lần lượt với các phần tử của mảng registers
# Đầu vào: toán hạng cần kiểm tra lưu trong $t8
# Nếu toán hạng không hợp lệ báo lỗi và dừng chương trình
#-----

check_register: add $v1, $ra, $0 # lưu đại chỉ trả về của hàm vào $v1
                li $t3, 0        # biến chạy trong mảng registers
                li $t4, 256       # vì có 32 thanh ghi nhưng mỗi thanh ghi để có 2
cách viết
                                # và vì lưu trữ word nên:  $32 \times 2 \times 4 = 256$  -> giá trị biên
để kết thúc lặp

loop_registers: beq $t3, $t4, err_operand # dừng lặp khi đã lặp hết phần tử ->
không hợp lệ
                lw $t9, registers($t3)    # lưu giá trị phần tử registers[$t3] vào
$t9

                jal str_compare           # thực hiện so sánh chuỗi trong $t8, $t9
                nop

                beqz $t5, end_check_register # khi đã xác định hợp lệ, kết thúc kiểm
tra toán hạng
                nop

                addi $t3, $t3, 4 # tăng 4 mỗi lần lặp vì lưu word

                j loop_registers

end_check_register:  li $v0, 4 # in các thông báo hợp lệ
                    la $a0, valid_operand1
                    syscall

                    li $v0, 4
                    add $a0, $t8, $0
                    syscall

```

```

        li $v0, 4
        la $a0, valid_operand2
        syscall

        jr $v1                # trở lại với giá trị trả về đã lưu tại $v1
#-----
#-----
# function syntax_beq()
# Kiểm tra các toán hạng có phù hợp với lệnh beq hay không
# Đầu vào: các toán hạng lưu ở các biến
#-----
syntax_beq:
    li $v0, 4
    la $a0, valid_beq
    syscall
# check toán hạng thứ 1 là thanh ghi có hợp lệ ko
    la $t8, tmp
    jal check_register
    nop
# check toán hạng thứ 2 là thanh ghi có hợp lệ ko
    la $t8, tmp2
    jal check_register
    nop
# check label ở toán hạng thứ 3
    la $t8, tmp3
    jal check_label
    nop
# In thông báo và chu kì của lệnh
    li $v0, 4
    la $a0, valid_command
    syscall

    li $v0, 4
    la $a0, cycle
    syscall

    li $v0, 1
    li $a0, 2
    syscall

```

```

end_beq:
    j end_main

#-----
#-----
# function syntax_add()
# Kiểm tra các toán hạng có phù hợp với lệnh add hay không
# Đầu vào: các toán hạng lưu ở các biến, đủ 3 toán hạng
#-----

syntax_add:
    li $v0, 4
    la $a0, valid_add
    syscall

# check toán hạng thứ 1 là thanh ghi có hợp lệ ko
    la $t8, tmp
    jal check_register
    nop

# check toán hạng thứ 2 là thanh ghi có hợp lệ ko
    la $t8, tmp2
    jal check_register
    nop

# check toán hạng thứ 3 là thanh ghi có hợp lệ ko
    la $t8, tmp3
    jal check_register
    nop

# In thông báo và chu kì
    li $v0, 4
    la $a0, valid_command
    syscall

    li $v0, 4
    la $a0, cycle
    syscall

    li $v0, 1
    li $a0, 1
    syscall

end_add:
    j end_main

```

```

#-----
#-----
# function syntax_sll()
# Kiểm tra các toán hạng có phù hợp với lệnh sll hay không
# Đầu vào: các toán hạng lưu ở các biến, 2 toán hạng đầu là thanh ghi, toán hạng
3 là số
#-----
syntax_sll:
    li $v0, 4
    la $a0, valid_sll
    syscall
end_sll:
# check toán hạng thứ 1 là thanh ghi có hợp lệ ko
    la $t8, tmp
    jal check_register
    nop
# check toán hạng thứ 2 là thanh ghi có hợp lệ ko
    la $t8, tmp2
    jal check_register
    nop
# check toán hạng thứ 3 là hằng số hay không
    la $t8, tmp3
    jal check_shift
    nop
# In thông báo và chu kì của lệnh sll
    li $v0, 4
    la $a0, valid_command
    syscall

    li $v0, 4
    la $a0, cycle
    syscall

    li $v0, 1
    li $a0, 1
    syscall

    j end_main
#-----
#-----

```

```

# function syntax_sub()
# Kiểm tra các toán hạng có phù hợp với lệnh beq hay không
# Đầu vào: các toán hạng lưu ở các biến, 3 toán hạng đều là thanh ghi
#-----

syntax_sub:
    li $v0, 4
    la $a0, valid_sub
    syscall

# check toán hạng thứ 1 là thanh ghi có hợp lệ ko
    la $t8, tmp
    jal check_register
    nop

# check toán hạng thứ 2 là thanh ghi có hợp lệ ko
    la $t8, tmp2
    jal check_register
    nop

# check toán hạng thứ 2 là thanh ghi có hợp lệ ko
    la $t8, tmp3
    jal check_register
    nop

#In thông báo và chu kì của lệnh sub
    li $v0, 4
    la $a0, valid_command
    syscall

    li $v0, 4
    la $a0, cycle
    syscall

    li $v0, 1
    li $a0, 1
    syscall
end_sub:
    j end_main

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

# function syntax_j()
# Kiểm tra các toán hạng có phù hợp với lệnh jum hay không
# Đầu vào: các toán hạng lưu ở các biến, chỉ cần 1 toán hạng và là label

```

```

#-----
syntax_j:
    li $v0, 4
    la $a0, valid_j
    syscall

# check toán hạng thứ 1 là nhân có hợp lệ không
    la $t8, tmp
    jal check_label
    nop

# Nếu có toán hạng thứ 2 -> báo lỗi
    lb $t2, tmp2($0)
    bne $t2, $0, err_operand
    nop

# In thông báo và chu kì của lệnh j
    li $v0, 4
    la $a0, valid_command
    syscall

    li $v0, 4
    la $a0, cycle
    syscall

    li $v0, 1
    li $a0, 2
    syscall
end_j:    j end_main
#-----
#-----
# function str_compare()
# So sánh 2 chuỗi có địa chỉ lưu trong $t8, $t9
# Trả về: 0 -> trùng nhau
#         1 -> khác nhau
#         kết quả lưu tại thanh ghi $t5
#-----
str_compare:    li $a2, 0                # biến chạy cho cả 2 biến

loop_compare:    add $t6, $t8, $a2        # load địa chỉ của từng kí tự có
string trong $t8
                add $t7, $t9, $a2        # load địa chỉ của từng kí tự có string
trong $t9
                lb $k0, 0($t6)            # load giá trị

```



```

        lb $k1, 0($t7)                # load giá trị

        bne $k0, $k1, false           # nếu kí tự khác -> false
        nop
        beq $k0, $0, true              # nếu là kí tự kết thúc -> trùng nhau
        nop
        addi $a2, $a2, 1
        j loop_compare
false:   li $t5, 1  # -> gán giá trị trả về 1 -> khác nhau
        jr $ra
true:   li $t5, 0  # -> gán giá trị trả về 0 -> giống nhau
        jr $ra

#-----
#-----
# function lowerCase()
# Biến chuỗi đưa vào thành chữ thường
# mặc định lấy ở op ngay khi đọc được opCode tu input
#-----
lowerCase: li $k1, 0  # biến chạy
loop_tmp_str:
    lb $t1, op($k1)
    beq $t1, 0, exit_funct # nếu là kí tự kết thúc chuỗi thì thoát hàm
    blt $t1, 'A', case
    bgt $t1, 'Z', case
    addi $t1, $t1, 32  # nếu là in hoa thì giảm 32 mã ascii -> chữ thường
    sb $t1, op($k1)

case:
    addi $k1, $k1, 1
    j loop_tmp_str
exit_funct: jr $ra

#-----
#-----
# function check_shift()
# kiểm tra chuỗi trong $t8 có phải là hằng số hay không và in lỗi
# kiểm tra kí tự có phải là số hay không, phục vụ cho toán hạng thứ 3 của lệnh sll
sll
#-----

check_shift: li $k1, 0                # biến chạy
            la $t8, tmp3

```

```

loop_shift:
    lb $t1, tmp3($k1)
    beq $t1, 0, end_check_shift
    blt $t1, '0', invalid
    bgt $t1, '9', invalid
    addi $k1, $k1, 1
    j loop_shift
invalid:
    j err_operand
end_check_shift:
    li $v0, 4
    la $a0, valid_operand1
    syscall

    li $v0, 4
    add $a0, $t8, $0
    syscall

    li $v0, 4
    la $a0, valid_operand2
    syscall

    jr $ra
#-----
#-----
# function check_label()
# Kiểm tra xem toán hạng gán vào $t8 có thảo mãn là nhãn không
# Nhãn bao gồm kí tự chữ cái, số và dấu gạch dưới
#-----
check_label: li $k1, 0

loop_label: add $k0, $t8, $k1
    lb $t1, 0($k0)
    beq $t1, 0, end_check_label
    nop
    blt $t1, '0', invalid_label
    nop
    bgt $t1, '9', check_up_letter
    nop
loop:    addi $k1, $k1, 1
    j loop_label

```

```

check_up_letter:
    blt $t1, 'A', invalid_label
    nop
    bgt $t1, 'Z', check_specialChar
    nop
    j loop
check_specialChar:
    bne $t1, '_', check_low_letter
    nop
    j loop
check_low_letter:
    blt $t1, 'a', invalid_label
    nop
    bgt $t1, 'z', invalid_label
    nop
    j loop
invalid_label:
    j err_operand
end_check_label:
    li $v0, 4
    la $a0, valid_operand1
    syscall

    li $v0, 4
    add $a0, $t8, $0
    syscall

    li $v0, 4
    la $a0, valid_operand2
    syscall

    jr $ra
#-----
#-----
# err_op, err_operand
# In các lỗi liên quan đến opCode và toán hạng
# Sau đó trở về hàm main để kết thúc chương trình
# riêng hàm err_operand có in ra tên toán hạng thông qua $t8
#-----
err_op:
    li $v0, 4

```

```

    la $a0, invalid_op
    syscall
    j end_main
err_operand:
    li $v0, 4
    la $a0, invalid_operand1
    syscall

    li $v0, 4
    add $a0, $t8, $0
    syscall

    li $v0, 4
    la $a0, invalid_operand2
    syscall
    j end_main

```

### 1.3. Kết quả, giải thích

Mars Messages	Run I/O
	Nhap code mips: J NguyenHaiDuong
	opcode: j, hop le!
	Toan hang: NguyenHaiDuong, hop le!
	Cau lenh dung cu phap!
	So chu ki cua lenh nay la: 2
	-- program is finished running --
	Nhap code mips: sub \$5, \$3, \$0
	opcode: sub, hop le!
	Toan hang: \$5, hop le!
	Toan hang: \$3, hop le!
	Toan hang: \$0, hop le!
	Cau lenh dung cu phap!
	So chu ki cua lenh nay la: 1
	-- program is finished running --
	Nhap code mips: j Duong\$2
	opcode: j, hop le!
	Toan hang: Duong\$2, khong hop le!
	-- program is finished running --

Mars Messages	Run I/O
	Nhap code mips: beq \$4, \$0, HaiDuong
	opcode: beq, hop le!
	Toan hang: \$4, hop le!
	Toan hang: \$0, hop le!
	Toan hang: HaiDuong, hop le!
	Cau lenh dung cu phap!
	So chu ki cua lenh nay la: 2
	-- program is finished running --
	Nhap code mips: add \$t1, #\$\$@
	opcode: add, hop le!
	Toan hang: \$t1, hop le!
	Toan hang: #\$\$@, khong hop le!
	-- program is finished running --
	Nhap code mips: SLL \$fp, \$ra, 2
	opcode: sll, hop le!
	Toan hang: \$fp, hop le!
	Toan hang: \$ra, hop le!
	Toan hang: 2, hop le!
	Cau lenh dung cu phap!
	So chu ki cua lenh nay la: 1
	-- program is finished running --

*Trong các test đã có thử thêm tên và mssv (20194530)*

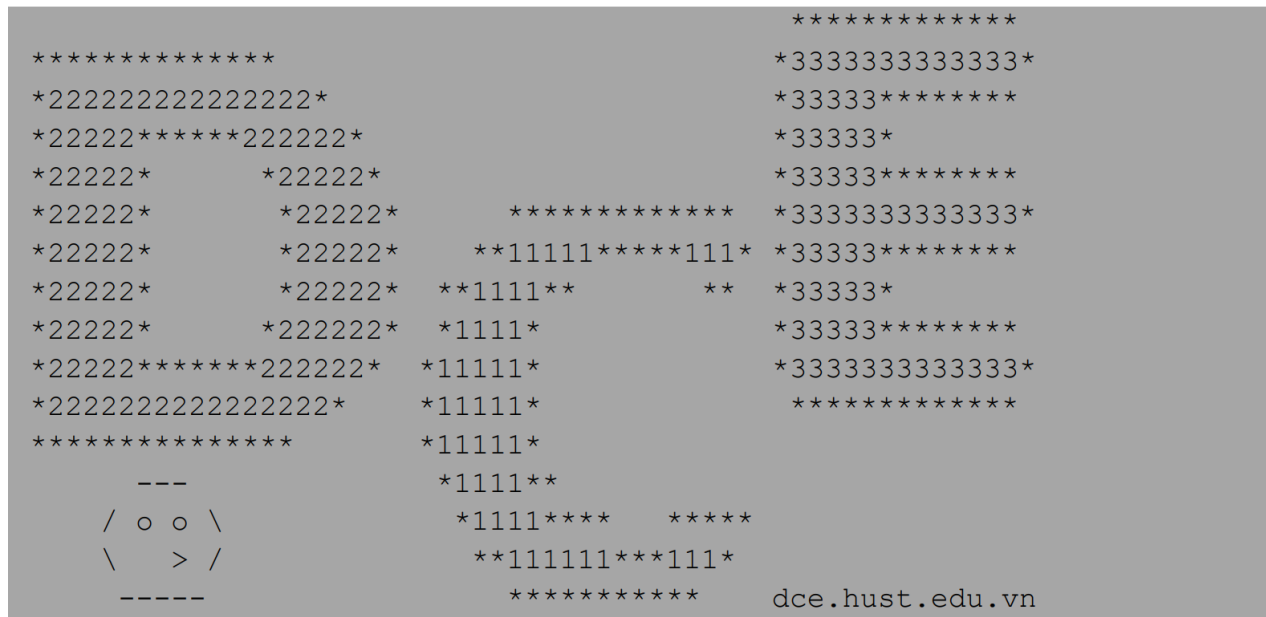
Ý nghĩa các thanh ghi:

Stt	Thanh ghi	Ý nghĩa
1	\$v0	Lưu code để gọi hàm syscall như 4 -> in string, 5 -> đọc vào integer, ...
2	\$a0	Lưu địa chỉ của string thông báo khai báo ở .data, làm biến chạy trong các hàm
5	\$s0, \$a1, \$a2, \$t0, \$t1, \$t2, \$t3, \$t4	Các thanh ghi thường xuyên được sử dụng làm biến trung gian, biến chạy
	\$t5	Lưu giá trị trả về của hàm str_compare
8	\$k1	Lưu kết quả của hàm check_opcode, là index của mảng opCodes khi có op thỏa mãn. Làm căn cứ để kiểm tra cú pháp của lệnh nào
9	\$t8, \$t9	Lưu địa chỉ của 2 string đầu vào của hàm str_compare. \$t8 thường xuyên được sử dụng để lưu các địa chỉ của các biến trong chương trình để làm đầu vào như hàm check_register, check_shift, check_label,...
10	\$ra	Lưu địa chỉ lệnh tiếp theo khi đi vào các thủ tục

## 2. Đề tài (14): Vẽ hình bằng kí tự ascii

### 2.1. Nội dung yêu cầu:

Cho hình ảnh đã được chuyển thành các kí tự ascii như hình vẽ. Đây là hình của chữ DCE có viền \* và màu là các con số.



- Hãy hiển thị hình ảnh trên lên giao diện console (hoặc giao diện Display trong công cụ giả lập Keyboard and Display MMIO Simulator)
- Hãy sửa ảnh để các chữ cái DCE chỉ còn lại viền, không còn màu số ở giữa, và hiển thị
- Hãy sửa ảnh để hoán đổi vị trí của các chữ, thành ECD, và hiển thị. Để đơn giản, các hoạt tiết đính kèm cũng được phép di chuyển theo.
- Hãy nhập từ bàn phím kí tự màu cho chữ D, C, E, rồi hiển thị ảnh trên với màu mới.

*Chú ý: ngoài vùng nhớ lớn chứa ảnh được chứa sẵn trong code, không được tạo thêm vùng nhớ mới để chứa ảnh hiệu chỉnh.*

## 2.2. Thuật toán

### Mô tả bằng C:

```
// Sử dụng mảng để lưu ảnh theo từng dòng dưới dạng chuỗi kí tự
// Sau khi phân tích và tính toán -> ảnh có 16 dòng và 64 cột.
// Sử dụng 16 chuỗi kí tự tương ứng với 16 dòng, mỗi chuỗi 64char
Array Image[16][64];
```

```
int main()
{
    int choice;
    printf_menu();
    scanf("%d",&choice);
```

```

switch (choice)
{
    case 1:
        option_1();
        break;
    case 2:
        option_2();
        break;
    case 3:
        option_3();
        break;
    case 4:
        option_4();
        break;
    case 5:
        option_5();
        break;
}
return 0;
}

```

#### Các hàm con:

// Hàm in ra màn hình ảnh theo mã ascii

```

option_1() {
    // In bằng cách in trực tiếp 16 chuỗi kí tự
    print_image();
}

```

//Hàm thực hiện xóa màu của chữ giữ lại viền

```

option_2() {
// Duyệt 16 chuỗi kí tự, nếu kí tự nào là số thì gán lại dấu cách ' '
    remove_color();
    print_image(); // in kết quả sau khi thay đổi
}

```

```

// Hàm thực thi đổi từ DCE sang ECD
option_3() {
    // Chia mỗi chuỗi kí tự làm 3 phần tương ứng với 3 chữ cái
    // Duyệt lần lượt 16 chuỗi.
    // Đối với mỗi chuỗi: swap 1/3 đầu và 1/3 cuối sẽ đổi được
    // vị trí của D và E nên DCE -> ECD
    ECD();

    print_image(); // in kết quả sau khi thay đổi
}

// Hàm thực thi việc đổi màu cho chữ
option_4() {
    // Đọc từ bàn phím 3 màu mới
    int d, c, e;
    scanf("%d %d %d", &d, &c, &e);

    // Kiểm tra các mã màu mới nhập có hợp lệ không
    // Vì màu hợp lệ là số nên là các chữ số 0,...,9
    check_color(d);
    check_color(c);
    check_color(e);

    // Hàm đổi màu chữ với 3 màu đầu vào
    // Duyệt 16 chuỗi, với mỗi chuỗi duyệt 1/3 chuỗi 1 lần.
    // Nếu gặp kí tự là chữ số thì thay bằng d, 1/3 tiếp theo là
    // c và 1/3 chuỗi còn lại là e.
    change_color(d, c, e);
    print_image(); // in kết quả sau khi thay đổi
}

```



## Mã nguồn

.data

```
ln1: .asciiz      "                                ***** \n"
ln2: .asciiz      "*****                                *33333333333333* \n"
ln3: .asciiz      "*2222222222222222*                *33333***** \n"
ln4: .asciiz      "*22222*****22222*                *33333* \n"
ln5: .asciiz      "*22222*          *22222*            *33333***** \n"
ln6: .asciiz      "*22222*          *22222*          ***** *33333333333333* \n"
ln7: .asciiz      "*22222*          *22222*          **11111*****111* *33333***** \n"
ln8: .asciiz      "*22222*          *22222*          **1111**          ** *33333* \n"
ln9: .asciiz      "*22222*          *22222*          *1111*          *33333***** \n"
ln10: .asciiz     "*22222*****22222*          *1111*          *33333333333333* \n"
ln11: .asciiz     "*2222222222222222*          *1111*          ***** \n"
ln12: .asciiz     "*****          *1111*          \n"
ln13: .asciiz     "          ---          *1111*          \n"
ln14: .asciiz     "          / o o \          *1111****          ***** \n"
ln15: .asciiz     "          \ > /          **111111***111*          \n"
ln16: .asciiz     "          -----          *****          dce.hust.edu.vn \n"
```

```
menu: .asciiz "\n\n=====Menu=====\\n1.In anh ra man hinh\\n2.Loai bo
mau cua chu\\n3.Doi vi tri thanh ECD\\n4.Doi mau chu\\n5.Exit.\\nLua chon:"
```

```
invalid_choice: "\nNhap lai lua chon!\\n"
```

```
input_D: .asciiz "\nNhap mau moi cho chu D: "
```

```
input_C: .asciiz "\nNhap mau moi cho chu C: "
```

```
input_E: .asciiz "\nNhap mau moi cho chu E: "
```

```
invalid_color: "\nMa mau khong hop le!\\n\\n"
```

.text

main:

```
jal print_menu          # In ra menu chuong trinh
```

```
nop
```

```
li $v0, 5               # Nhan vao input tu ban phim
```

```
syscall
```

```
beq $v0, 1, option_1    # Chay ham 1
```

```
nop
```

```
beq $v0, 2, option_2    # Chay ham 2
```

```
nop
```

```
beq $v0, 3, option_3    # Chay ham 3
```

```
nop
```

```
beq $v0, 4, option_4    # Chay ham 4
```

```
nop
```

```

    beq $v0, 5, end_main      # Ket thuc chuong trinh
    nop

    la $a0, invalid_choice
    li $v0, 4
    syscall

end_main:  li $v0, 10
           syscall

#-----
# function print_image
# thực hiện in 16 chuỗi kí tự tương ứng 16 dòng của ảnh
#-----
print_image:      # In ra tung dong (ln1 -> ln16)
    la $a0, ln1
    li $v0, 4
    syscall

    la $a0, ln2
    li $v0, 4
    syscall

    la $a0, ln3
    li $v0, 4
    syscall

    la $a0, ln4
    li $v0, 4
    syscall

    la $a0, ln5
    li $v0, 4
    syscall

    la $a0, ln6
    li $v0, 4
    syscall

    la $a0, ln7
    li $v0, 4
    syscall

```

```
la $a0, ln8
li $v0, 4
syscall
```

```
la $a0, ln9
li $v0, 4
syscall
```

```
la $a0, ln10
li $v0, 4
syscall
```

```
la $a0, ln11
li $v0, 4
syscall
```

```
la $a0, ln12
li $v0, 4
syscall
```

```
la $a0, ln13
li $v0, 4
syscall
```

```
la $a0, ln14
li $v0, 4
syscall
```

```
la $a0, ln15
li $v0, 4
syscall
```

```
la $a0, ln16
li $v0, 4
syscall
```

```
end_print: jr $ra
```

```
#-----
# printf_menu()
# In chuỗi chứa thông tin của menu ỏi biến menu
```

```

#-----
print_menu:
    la $a0, menu
    li $v0, 4
    syscall

    j end_print
#-----
# Các option của chương trình chính
#-----
option_1:
    jal print_image      # In 16 dòng từ ln1 -> ln16
    nop
end_option: j end_main      # Kết thúc chương trình

# Duyệt từng dòng -> Xóa tất cả ký tự '0' -> '9' của dòng đó
option_2:
    la $k0, ln1
    jal remove_color
    nop

    la $k0, ln2
    jal remove_color
    nop

    la $k0, ln3
    jal remove_color
    nop

    la $k0, ln4
    jal remove_color
    nop

    la $k0, ln5
    jal remove_color
    nop

    la $k0, ln6
    jal remove_color
    nop

```

```
la $k0, ln7
jal remove_color
nop
```

```
la $k0, ln8
jal remove_color
nop
```

```
la $k0, ln9
jal remove_color
nop
```

```
la $k0, ln10
jal remove_color
nop
```

```
la $k0, ln11
jal remove_color
nop
```

```
la $k0, ln12
jal remove_color
nop
```

```
la $k0, ln13
jal remove_color
nop
```

```
la $k0, ln14
jal remove_color
nop
```

```
la $k0, ln15
jal remove_color
nop
```

```
j option_1
```

```
# Duyet tung dong -> Thay cot 1 voi cot 3 -> thay E voi D
option_3:la $k0, ln1
jal ECD
```

nop

la \$k0, ln2

jal ECD

nop

la \$k0, ln3

jal ECD

nop

la \$k0, ln4

jal ECD

nop

la \$k0, ln5

jal ECD

nop

la \$k0, ln6

jal ECD

nop

la \$k0, ln7

jal ECD

nop

la \$k0, ln8

jal ECD

nop

la \$k0, ln9

jal ECD

nop

la \$k0, ln10

jal ECD

nop

la \$k0, ln11

jal ECD

nop

```
la $k0, ln12
jal ECD
nop
```

```
la $k0, ln13
jal ECD
nop
```

```
la $k0, ln14
jal ECD
nop
```

```
la $k0, ln15
jal ECD
nop
```

```
la $k0, ln16
jal ECD
nop
```

```
j option_1
```

```
# $s1 luu gia tri cua D
# $s2 luu gia tri cua C
# $s3 luu gia tri cua E
option_4:
```

```
la $a0, input_D
li $v0, 4
syscall
li $v0, 12
syscall
add $s1, $v0, $0
add $t1, $v0, $0
jal check_color
nop
```

```
la $a0, input_C
li $v0, 4
syscall
li $v0, 12
```

```
syscall
add $s2, $v0, $0
add $t1, $v0, $0
jal check_color
nop
```

```
la $a0, input_E
li $v0, 4
syscall
li $v0, 12
syscall
add $s3, $v0, $0
add $t1, $v0, $0
jal check_color
nop
```

```
# change_color: Ham duyet tung dong ->
# thay tat ca cot 1 thanh` $s1
# thay tat ca cot 2 thanh` $s2
# thay tat ca cot 3 thanh` $s3
la $k0, ln1
jal change_color
nop
```

```
la $k0, ln2
jal change_color
nop
```

```
la $k0, ln3
jal change_color
nop
la $k0, ln4
jal change_color
nop
```

```
la $k0, ln5
jal change_color
nop
la $k0, ln6
jal change_color
nop
```



```
la $k0, ln7
jal change_color
nop
la $k0, ln8
jal change_color
nop
```

```
la $k0, ln9
jal change_color
nop
la $k0, ln10
jal change_color
nop
```

```
la $k0, ln11
jal change_color
nop
la $k0, ln12
jal change_color
nop
```

```
la $k0, ln13
jal change_color
nop
```

```
la $k0, ln14
jal change_color
nop
```

```
la $k0, ln15
jal change_color
nop
```

```
la $k0, ln16
jal change_color
nop
```

```
la $a0, '\n'
li $v0, 12
syscall
```

```

        j option_1
#-----
#$k0 tham so dong dau vao
#Duyet -> xoa tat ca ky tu '0' < x < '9'
#-----

remove_color:    li $a1, 0                # Bien dem
loop_find_color:add $k1, $k0, $a1
                lb $t1, 0($k1)
                beq $t1, '\n', end_remove # Gap ky tu xuống dòng -> ket thuc ham`
                nop
                blt $t1, '0', skip        # < '0' -> Bo qua
                nop
                bgt $t1, '9', skip        # > '9' -> Bo qua
                nop

                la $t1, ' '              # Thay ky tu thanh` ' ' -> ~ xoa
                sb $t1, 0($k1)
skip:          add $a1, $a1, 1            # Tang bien dem len 1
        j loop_find_color
end_remove: jr $ra
#-----
# ECD -> đổi vị trí trong ảnh để thay đổi từ DCE -> ECD
# Chia 3 ảnh và đổi vị trí các kí tự ở 1/3 đầu và 1/3 cuối
#-----
ECD:  li $a1, 0    # 0->20 -> chu D, 21: dau cach
        # 22-> 42 -> chữ C, 43: dau cach
        # 43-> 63 -> chữ E, 64: dau cach
        # 65-> \n

loop_ECD:
        beq $a1, 21, end_ECD # Gap ' ' -> ket thuc ham
        nop

        add $a2, $a1, 44 #E

        add $s1, $k0, $a1 # D
        add $s2, $k0, $a2 # E

        # Dao vi tri cot 1 voi cot 3 (dao vi tri D voi E)
        lb $t1, 0($s1) #D

```

```

    lb $t2, 0($s2) #E

    sb $t1, 0($s2)
    sb $t2, 0($s1)

    addi $a1, $a1, 1 # Tang bien dem len 1
    j loop_ECD
end_ECD: jr $ra

#-----
# change_color($s1, $s2, $s3)
# $s1, $s2, $s3 chứa giá trị 3 màu tương ứng D, C, E
# Ham thay doi mau sac
#-----
change_color: li $a1, 0          # Bien dem

# lặp đến hết chữ D và thay màu bằng màu $s1
loop_fnd_color:
    beq $a1, 21, next_letter2    # 0->20 -> chu D, 21: dau cach

    add $k1, $k0, $a1
    lb $t1, 0($k1)               # Lay ra ki tu mau hien tai

    nop
    blt $t1, '0', skip_char      # < '0' -> khong phai mau -> Skip
    nop
    bgt $t1, '9', skip_char      # > '9' -> khong phai mau -> Skip
    nop

    add $t1, $0, $s1             # Thay mau` bang ki tu $s1
    sb $t1, 0($k1)               # Luu mau` vao chu D

skip_char:    add $a1, $a1, 1      # Tang bien dem len 1
              j loop_fnd_color
              add $a1, $a1, 1

# lặp đến hết chữ C và thay màu bằng màu $s2
next_letter2:
    beq $a1, 43, next_letter3    # 22-> 42 -> chữ C, 43: dau cach
    add $k1, $k0, $a1
    lb $t1, 0($k1)               # Lay ra ki tu mau hien tai

```

```

        nop
        blt $t1, '0', skip_char2    # < '0' -> không phải màu -> Skip
        nop
        bgt $t1, '9', skip_char2    # > '9' -> không phải màu -> Skip
        nop

        add $t1, $0, $s2            # Thay màu` bằng kí tự $s2
        sb $t1, 0($k1)              # Lưu màu` vào chu C
skip_char2:    add $a1, $a1, 1        # Tăng biến đếm lên 1
        j next_letter2
        add $a1, $a1, 1
# lặp đến hết chữ E và thay màu bằng màu $s3
next_letter3:
        beq $a1, 65, end_change      # 43-> 63 -> chữ E, 64: dấu cách. 65:
'\n'

        add $k1, $k0, $a1
        lb $t1, 0($k1)              # Lấy ra kí tự màu hiện tại

        nop
        blt $t1, '0', skip_char3    # < '0' -> không phải màu -> Skip
        nop
        bgt $t1, '9', skip_char3    # > '9' -> không phải màu -> Skip
        nop

        add $t1, $0, $s3            # Thay màu` bằng kí tự $s3
        sb $t1, 0($k1)              # Lưu màu` vào chu E
skip_char3:    add $a1, $a1, 1        # Tăng biến đếm lên 1
        j next_letter3
end_change: jr $ra

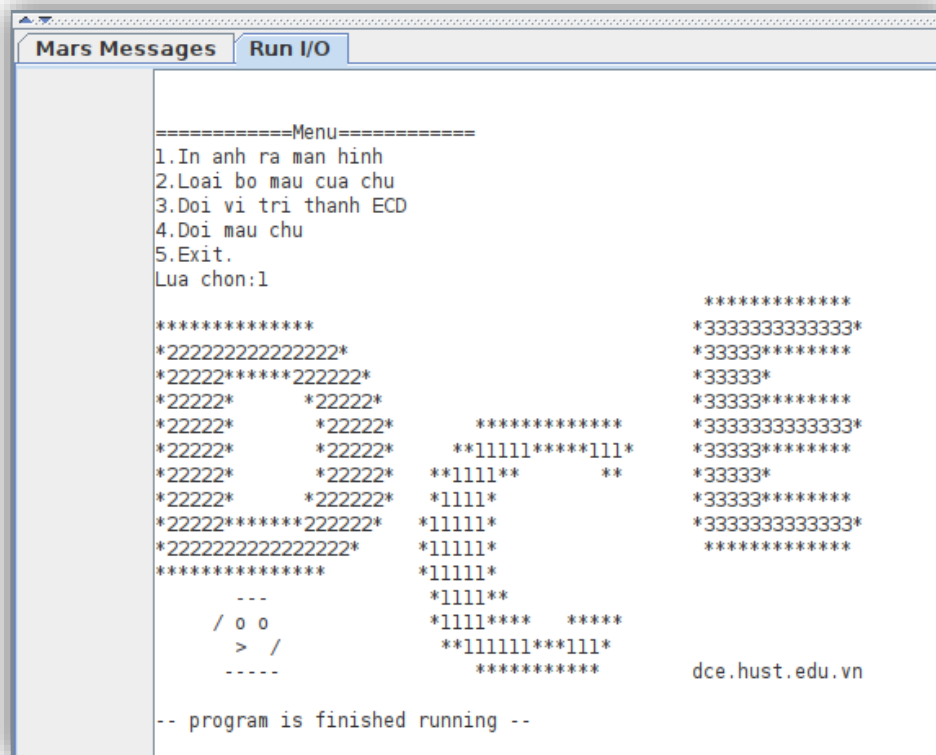
check_color:
        blt $t1, '0', invalid        # < '0' -> không phải màu -> lỗi
        nop
        bgt $t1, '9', invalid        # > '9' -> không phải màu -> lỗi
        nop
end_check: jr $ra
invalid:
        li $v0, 4
        la $a0, invalid_color        # In ra lỗi invalid_color

```

```
syscall
```

```
j end_main
```

### 2.3. Kết quả, giải thích



```
Mars Messages Run I/O

=====Menu=====
1.In anh ra man hinh
2.Loai bo mau cua chu
3.Doai vi tri thanh ECD
4.Doai mau chu
5.Exit.
Lua chon:1

*****
*2222222222222222*
*22222*****22222*
*22222*      *22222*
*22222*      *22222*      *****
*22222*      *22222*      **11111*****111*
*22222*      *22222*      **1111**          **
*22222*      *22222*      *1111*
*22222*****22222*      *11111*
*2222222222222222*      *11111*
*****                *11111*
---                  *1111**
/ o o                *1111****   *****
> /                  **111111***111*
-----              *****
                                     dce.hust.edu.vn

-- program is finished running --
```

=====Menu=====

- 1.In anh ra man hinh
- 2.Loai bo mau cua chu
- 3.Doai vi tri thanh ECD
- 4.Doai mau chu
- 5.Exit.

Lua chon:2

\*\*\*\*\*

[illegible]

```
-- program is finished running --
```

## Mars Messages

Run I/O

=====Menu=====

- 1.In anh ra man hinh
- 2.Loai bo mau cua chu
- 3.Doai vi tri thanh ECD
- 4.Doai mau chu
- 5.Exit.

Lua chon:3

[illegible]

```
-- program is finished running --
```

Clear

```
Mars Messages Run I/O

=====Menu=====
1.In anh ra man hinh
2.Loai bo mau cua chu
3.Doi vi tri thanh ECD
4.Doi mau chu
5.Exit.
Lua chon:4

Nhap mau moi cho chu D: 5
Nhap mau moi cho chu C: 0
Nhap mau moi cho chu E: 8

*****
*88888888888888*
*88888*****
*88888*
*88888*****
*88888888888888*
*88888*****
*88888*
*88888*****
*88888*****
*88888888888888*
*****
*****
*00000*****000*
*00000**
*0000*
*00000*****555555*
*00000*
*00000*
*0000**
/ o o *0000***
> / **000000***000*
----- *****
dce.hust.edu.vn

-- program is finished running --
```

TH: này sử dụng 3 số cuối mssv (Duan: 20194508)

```
Mars Messages Run I/O

=====Menu=====
1.In anh ra man hinh
2.Loai bo mau cua chu
3.Doi vi tri thanh ECD
4.Doi mau chu
5.Exit.
Lua chon:4

Nhap mau moi cho chu D: 3
Nhap mau moi cho chu C: 8
Nhap mau moi cho chu E: D
Ma mau khong hop le!

-- program is finished running --
```

Ý nghĩa các thanh ghi:

Stt	Thanh ghi	Ý nghĩa
1	\$v0	Lưu code để gọi hàm syscall như 4 -> in string, 5 -> đọc vào integer, ... Là case để biết người dùng đang sử dụng chức năng nào
2	\$a0	Lưu địa chỉ của string thông báo khai báo ở .data, in các chuỗi kí tự từng dòng của ảnh
3	\$k0	Lưu địa chỉ của chuỗi nhập vào, là đầu vào của hàm như remove_color, ECD, change_color
4	\$t1	Lưu địa chỉ của chuỗi nhập vào, là đầu vào của hàm như check_color, bên cạnh đó nó còn là biến tạm, biến chạy trong nhiều hàm
5	\$t0, \$t1, \$t2, \$t8, \$t9, \$k0, \$a1, \$a2	Các thanh ghi làm biến trung gian, biến chạy Trong suốt chương trình
6	\$sp	Lưu địa chỉ stack sử dụng trong hàm lưu các kí tự chung giữa 2 string
11	\$ra	Lưu địa chỉ lệnh tiếp theo khi đi vào các thủ tục