

# Báo cáo bài cuối kỳ

## Môn thực hành kiến trúc máy tính

Vũ Thành Trung - 20200650  
Phùng Trung Kiên - 20204994  
Lớp Việt – Nhật 03 Trường đại học  
Bách Khoa Hà Nội,  
Email: [Trung.vt200650@sis.hust.edu.vn](mailto:Trung.vt200650@sis.hust.edu.vn)  
& [kien.pt204994@sis.hust.edu.vn](mailto:kien.pt204994@sis.hust.edu.vn)

### Đề bài được giao:

#### 7. Chương trình kiểm tra cú pháp lệnh MIPS

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 kỳ (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.

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.

#### 9. Vẽ hình bằng kí tự ASCII

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.

```
*****
*222222222222222*
*22222*****22222*
*22222*      *22222*
*22222*      *22222*      *****
*22222*      *22222*      **11111*****111*
*22222*      *22222*      **1111**      **
*22222*      *222222*      *1111*
*22222*****22222*      *11111*
*222222222222222*      *11111*
*****
---
/  o  o  \
\   >  /
-----

*****
*33333333333333*
*33333*****
*33333*
*33333*****
*33333333333333*
*33333*****
*33333*
*33333*****
*33333333333333*
*****
*11111*
*11111**
*1111*****
**11111***111*
*****
dce.hust.edu.vn
```

- 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 hoạ 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.

## Hướng dẫn thực hiện

Chia nhóm 2 sinh viên thực hiện 2 project được giao ngẫu nhiên, mỗi sinh viên thực hiện 1 bài và hiểu bài còn lại.

## Cách thức đánh giá

- Trình bày vấn đáp: chạy mô phỏng MARS, trình bày mã nguồn, trả lời câu hỏi.
- Viết báo cáo in quyển (theo nhóm) trình bày: phân tích cách làm, thuật toán, mã nguồn và kết quả chạy mô phỏng.
- Nộp chương trình hợp ngữ, bản mềm. Qui định cách đặt tên file như sau:

**xxx\_gyy\_StudentName.asm**

+ xx: số thứ tự của bài. Ví dụ SV làm bài số 8 thì xx là 08, SV làm bài số 10 thì xx là 10.

+ yy: số thứ tự của nhóm. Ví dụ SV thuộc nhóm 7 thì yy là 07, SV thuộc nhóm 18 thì yy là 18.

+ StudentName: là tên của sinh viên, tiếng Việt, không dấu, không có kí tự space và viết hoa chữ cái đầu. Ví dụ sinh viên Nguyễn Văn A thì StudentName là NguyenVanA, sinh viên Vũ Thanh Thủy thì StudentName là VuThanhThuy.

+ Ví dụ, sinh viên tên Lê Mạnh Hùng, thuộc nhóm 21, làm bài 8, thì cần nộp bản mềm với tên file là n08\_g21\_LeManhHung.asm

### Lưu ý khi thực hiện:

- Chương trình nên có giao diện menu chọn (nếu cần), có thể lặp lại nhiều lần mà không cần chạy lại chương trình.
- Các thao tác nhập dữ liệu nên có kiểm tra, xác thực tính hợp lệ của dữ liệu.
- Mã nguồn nên tổ chức thành chương trình con (nếu cần).
- Viết mã nguồn sáng sủa, dễ đọc, dễ hiểu (có căn lề, chú thích).

**Phần bài làm:****Bài 7 - Vũ Thành Trung 20200650:****SOURCE CODE:**

<https://drive.google.com/file/d/1S0kKPfdDYJY0rEdHKJQ5FU8RLTdPNvuu/view?usp=sharing>

```
# Author: Vu Thanh Trung
# Creation date: 19/07/2022

.data

# ----- #
#           Opcode library                               #
#           Rule: each opcode has length of 8 byte, seperated by type and #
#           syntax                                         #
# ----- #

# Opcode Library:
opcdLibrary:  .asciiz      "add,1  sub,1  addu,1  subu,1  mul,1  and,1  or,1
nor,1  xor,1  slt,1  addi,2  addiu,2  andi,2  ori,2  sll,2  srl,2  slti,2  sltiu,2
mult,3  div,3  move,3  lw,4  sw,4  lb,4  sb,4  lbu,4  lhu,4  ll,4  sh,4
lui,5  li,5  la,6  mfhi,7  mflo,7  jr,7  beq,8  bne,8  j,9  jal,9  "

buffer:      .space      100
opcode:      .space      10

# Prompts
border:      .asciiz      "\n# -----
----- #\n"
start_mes:   .asciiz      "\n          Opcode Checker\n"
Message:     .asciiz      "Enter string: "
correct_opcode_prompt: .asciiz      "\nCorrect opcode: "
end_prompt:  .asciiz      "\nCorrect syntax."
not_valid_register_prompt: .asciiz      "\nInvalid register syntax."
not_valid_number_prompt: .asciiz      "\nNot valid number."
not_valid_address_prompt: .asciiz      "\nNot valid address"
valid_syntax_prompt:   .asciiz      "\nCorrect MIPS syntax."
continue_prompt:       .asciiz      "\nContinue? (1. Yes 0. No): "
missing_prompt:        .asciiz      "\nThieu toan hang"
```

```

# Syntax error prompts:
missing_colon_prompt: .asciiiz      "\nSyntax error: missing colon."
invalid_opcode_prompt: .asciiiz      "\nOpcode is invalid/doesn't exist."
too_many_variable_prompt:.asciiiz    "\nSyntax has too many variables."


# Registers library #
tokenRegisters: .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    $21    $22    $23    $24    $25    $26    $27
$28    $29    $30    $31    "


.text
main:
    jal    start


read_data:
# ----- #
#           Read data                               #
# ----- #
    li     $v0, 8                                # take in input
    la     $a0, buffer                            # load byte space into address
    li     $a1, 100                              # allot the byte space for
string
    syscall
    move   $s0, $a0                              # save string to $s0
# ----- #
#           Registers used:                          #
#           $s0, $a0: string's address                #
# ----- #


clear_whitespace:

```



```
# Check library if there is matching syntax #
```

```

    la            $s2, opcdLibrary
    jal          check

    j            invalid_opcode      # jump to target

check:
    move         $a2, $s2           # a2 pointer to beginning of
library
loop_check:
    lb           $t2, 0($a2)        # load each character from
library
    beq          $t2, ',', evaluation1      # if encountered colon, evaluate
whether it is correct
    lb           $t1, 0($a1)        # load each character from
opcode
    beq          $t2, 0, jump_      # if encountered /0 then not
found valid opcode
    bne          $t1, $t2, next_opcode      # character mismatch
    addi         $a1, $a1, 1         # next character
    addi         $a2, $a2, 1
    j            loop_check

evaluation1:
    lb           $t1, 0($a1)        # load each character from
opcode
    beq          $t1, 0, opcode_done
    j            next_opcode        # jump to $ra

next_opcode:
    addi         $s2, $s2, 8         # Each opcode has length of 8
byte, thus moving to next opcode
    move         $a2, $s2
    move         $a1, $s1
    j            loop_check        # Keep looping

```

```

opcode_done:
# ----- #
#           Registers used:                               #
#           $s0, $a0: string's address                     #
#           $s1: opcode's address, $a1: at the end of the string #
# ----- #

        jal        correct_opcode

        addi        $a2, $a2, 1
        lb          $t2, 0($a2)                # Load syntax type in $t2

        jal        check_whitespace

        addi        $t2, $t2, -48              # Store type (-48 to change to
int)

        beq         $t2, 1, Type_1
        beq         $t2, 2, Type_2
        beq         $t2, 3, Type_3
        beq         $t2, 4, Type_4
        beq         $t2, 5, Type_5
        beq         $t2, 6, Type_6
        beq         $t2, 7, Type_7
        beq         $t2, 8, Type_8
        beq         $t2, 9, Type_9

end:
        j          ending                    # jump to ending

#####
#####

#####
#####

```

```
#####
#####
```

check\_whitespace:

```
# ----- #
#           Read white space before code           #
# ----- #
        move        $a0, $s0                        # Dong bo con tro $a0
voi $s0
        lb          $t1, 0($a0)                     # Doc tung ki tu cua
command
        beq         $t1, ' ', loop_whitespace       # Loop de tru het dau
cach
        jr          $ra                             # Trong truong hop khong
phai, quay tro ve chuong trinh chinh
loop_whitespace:
        lb          $t1, 0($a0)                     # Doc tung ki tu cua
command
        beq         $t1, ' ', check_whitespace_pass # Khi gap dau cach thi
tiep tuc doc
        # addi      $a0, $a0, 1                     # Dich $a0 di 1 ky tu
        move        $s0, $a0                       # Cho dia chi moi cho s0
la ky tu dau tien cua word
        jr          $ra                             # jump to $ra
check_whitespace_pass:
        addi        $a0, $a0, 1                     # Dich $a0 di 1 ky tu
        j           loop_whitespace                 # jump to
check_whitespace
# ----- #
#           Registers used: $t1, $a0, $s0, $ra      #
#           Pointer will point at next character    #
# ----- #
```

check\_colon:

```
# ----- #
#           Read colon                             #
# ----- #
        move        $a0, $s0                        # Dong bo con tro $a0
voi $s0
```



```

        lb            $t1, 0($a0)                # Doc tung ki tu cua
command
        bne           $t1, ',', missing_colon    # Khi gap dau cach thi
tiếp tục doc
        jr            $ra                        # jump to $ra
# ----- #
#           Registers used: $t1, $a0, $s0, $ra    #
#           Pointer will be at the colon          #
# ----- #

```

```

check_gap:
# ----- #
#           Check gap between code                #
# ----- #

        move         $t4, $ra
        jal          check_whitespace
        jal          check_colon
        addi         $a0, $a0, 1                # Point to
character/whitespace after colon
        move         $s0, $a0
        jal          check_whitespace
        move         $ra, $t4
        jr           $ra
# ----- #
#           Registers used: $t1, $a0, $s0, $ra    #
# ----- #

```

```

jump_:
        jr           $ra

```

OPCODE\_TYPES:

Type\_1:

```

# ----- #
#           Format xyz $1, $2, $3                  #
# ----- #
# Ari1Library_:      .asciiz      "add,    sub,    addu,    subu,    mul;    " #
Format add $1, $2, $3
# Log1Library_:      .asciiz      "and,    or,    nor,    xor;    " # Format and
and $1, $2, $3 //
# Com1Library_:      .asciiz      "slt;    " # Format slt $1,$2,$3
# move $a0, $s0      # Load and print string asking for string

```

```

# li    $v0, 4
# syscall
jal     reg_check
jal     check_gap
jal     reg_check
jal     check_gap
jal     reg_check
jal     check_end

```

Type\_2:

```

# ----- #
#      Format xyz $1, $2, 10000                                #
# ----- #
# Ari2Library_:      .asciiiz      "addi,   addiu;  " # Format addi $1,$2,100
# Log2Library_:      .asciiiz      "andi,   ori,    sll,    srl;   " # Format or
andi $1,$2,
# Con1Library_: # 1
# Com2Library:      .asciiiz      "slti,   sltiu;  " # Format slti $1, $2, 100
    jal     reg_check
    jal     check_gap
    jal     reg_check
    jal     check_gap
    jal     num_check
    jal     check_end

```

Type\_3:

```

# ----- #
#      Format mult $2,$3                                      #
# ----- #
# Ari3Library:      .asciiiz      "mult,   div;    " # Format mult $2,$3 //
# Dat5Library:      .asciiiz      "move;    " # Format move $1, $2
    jal     reg_check
    jal     check_gap
    jal     reg_check
    jal     check_end

```

Type\_4:

```

# ----- #
#      Format lw $1, 100($2)                                  #
# ----- #
# Dat1Library:      .asciiiz      "lw,     sw,     lb,     sb,     lbu,    lhu,    ll,
sh;    " # Format lw $1, 100($2) //
    jal     reg_check
    jal     check_gap
    jal     address_check

```

```

        jal    check_end

Type_5:
# ----- #
#      Format lui $1, 100                                #
# ----- #
# Dat2Library:  .asciiz      "lui,    li;    " # Format lui $1, 100 //
        jal    reg_check
        jal    check_gap
        jal    num_check
        jal    check_end

Type_6:
# ----- #
#      Format la $1,label                                #
# ----- #
# Dat3Library:  .asciiz      "la;      " # Format la $1,label //
jal    reg_check
        jal    check_gap
        jal    label_check
        jal    check_end

Type_7:
# ----- #
#      Format mfhi $2                                    #
# ----- #
# Dat4Library:  .asciiz      "mfhi,    mflo;    " # Format mfhi $2
# Jum2Library:  .asciiz      "jr;      " # Format jr $1
        jal    reg_check
        jal    check_end

Type_8:
# ----- #
#      Format beq $1, $2, label beq $1,$2,100            #
# ----- #
# Con1Library:  .asciiz      "beq,     bne;     " # Format beq $1,$2,100 ; beq $1,
# $2, label
        jal    reg_check
        jal    check_gap
        jal    reg_check
        jal    check_gap
        jal    label_check
        beq    $s7, 1, check_end

```

```

        jal    num_check

end_type8:
        jal    check_end

Type_9:
# ----- #
#      Format j 1000 ; j label                      #
# ----- #
# Jum1Library:  .asciiz      "j,      jal;  " # Format j 1000 ; j label
        jal    label_check
        beq    $s7, 1, check_end
        jal    num_check

end_type9:
        jal    check_end

check_syntax:
        check_end:
# -----
#
#      Check whether string has ended or not (space excluded)
#
# -----
#
        jal    check_whitespace
        lb     $t5, 0($s0)
        beq    $t5, '\n', valid_syntax #
        beq    $t5, '\0', valid_syntax #
        j      too_many_variable

        reg_check:
# -----
#
#      Check whether string is register or not
#
# -----
#
        la     $s3, tokenRegisters
        move   $a3, $s3                # a3 points to beginning of
register library
        move   $a0, $s0

```

```

        loop_reg_check:
            lb            $t3, 0($a3)            # load each
character from library
            lb            $t0, 0($a0)            # load each
character from string
            beq          $t3, ' ', evaluation2    # if encountered
space, evaluate whether it is correct
            beq          $t3, 0, not_valid_register # if encountered
/0 then not valid register
            bne          $t0, $t3, next_reg        # character
mismatch
            addi         $a0, $a0, 1              # next character
            addi         $a3, $a3, 1
            j            loop_reg_check
        evaluation2:
            lb            $t0, 0($a0)
            beq          $t0, ',', found_reg        # Correct
register
            beq          $t0, ' ', found_reg        # Correct
register
            beq          $t0, 0, found_reg          # Correct
register
            beq          $t0, '\n', found_reg        # Correct
register
            j            next_reg                  # jump to
next_register
        next_reg:
            addi         $s3, $s3, 8              # Move to next
register token
            move         $a3, $s3
            move         $a0, $s0
            j            loop_reg_check            # jump to
loop_reg_check
        found_reg:
            move         $s0, $a0                  # move pointer
forward
            j            jump_                      # jump to jump_

# -----
#
#           Registers used:
#
#           $s0, $a0: string's address (checking register)
#
#           $s3, $a3: register's library pointers
#

```

```

# -----
#
num_check:
# -----
#
#           Check whether string is register or not
# -----
#
                move        $a0, $s0
num_check_loop:
                lb          $t0, 0($a0)
                beq         $t0, ',', is_num          # Correct
register
                beq         $t0, ' ', is_num          # Correct
register
                beq         $t0, 0, is_num             # Correct
register
                beq         $t0, '\n', is_num          # Correct
register
                bgt         $t0, '9', not_num          # if $t0 > 9
then target
                blt         $t0, '0', not_num          # if $t0 < 0
then target
                addi        $a0, $a0, 1
                j           num_check_loop            # jump to
num_check_loop
is_num:
                move        $s0, $a0
                j           jump_                     # jump to jump_
not_num:
                j           not_num_error             # jump
to not_num_error

address_check:
adnum_check:

                num_check_loop2:
                lb          $t0, 0($a0)
                beq         $t0, '(', is_num2          #
Correct registe
                bgt         $t0, '9', not_num2        # if $t0
> 9 then target
                blt         $t0, '0', not_num2        # if $t0
< 0 then target
                addi        $a0, $a0, 1

```

```

                                j            num_check_loop2          # jump
to num_check_loop
                                is_num2:
                                move        $s0, $a0
                                j            adreg_check
# jump to jump_
                                not_num2:
                                j            not_valid_address
# jump to not_num_error
                                adreg_check:
                                reg_check2:
                                addi        $a0, $a0, 1
                                move        $s0, $a0
                                # -----
#
#                               Check whether string is register or not
#
                                # -----
#
                                la          $s3, tokenRegisters
                                move        $a3, $s3                    # a3 points to beginning of
register library
                                move        $a0, $s0

                                loop_reg_check2:
                                lb           $t3, 0($a3)                # load each
character from library
                                lb           $t0, 0($a0)                # load each
character from string
                                beq          $t3, ' ', evaluation3        # if encountered
space, evaluate whether it is correct
                                beq          $t3, 0, not_valid_address2    # if encountered
/0 then not valid register
                                bne         $t0, $t3, next_reg2          # character
mismatch
                                addi         $a0, $a0, 1                # next character
                                addi         $a3, $a3, 1
                                j            loop_reg_check2
                                evaluation3:
                                lb           $t0, 0($a0)
                                beq          $t0, ')', found_reg2        # Correct
register
                                j            next_reg2                    # jump to
next_register
                                next_reg2:

```

```

                                addi      $s3, $s3, 8                # Move to next
register token
                                move      $a3, $s3
                                move      $a0, $s0
                                j         loop_reg_check2          # jump to
loop_reg_check
                                not_valid_address2:
                                    move    $a0, $t0
                                    li      $v0, 11
                                    syscall
                                    j not_valid_address
                                found_reg2:
                                    addi    $a0, $a0, 1
                                    move    $s0, $a0                # move pointer
forward
                                jr         $ra

# -----
#
#           Registers used:
#
#           $s0, $a0: string's address (checking register)
#
#           $s3, $a3: register's library pointers
# -----
#

label_check:
    move    $a0, $s0

First_char_check: # Can't be number and can't be underscore:
    lb      $t0, ($a0)                #Load byte from 't0'th position
in buffer into $t1
    blt     $t0, 'a', not_lower        #If less than a, exit
    bgt     $t0, 'z', not_lower        #If greater than z, exit

    j       loop_label_check          # It's lower so
we jump to 2nd character

    not_lower:
        blt     $t0, 'A', fail_case    #If less than A, means not
alphabet, failcase
        bgt     $t0, 'Z', fail_case    #If greater than Z, means not
alphabet, failcase

```



```

loop_label_check: # Can be alphabet, number and underscore

                addi    $a0, $a0, 1                # Increment
                lb      $t0, ($a0)                #Load byte from 't0'th position
in buffer into $t0

                beq     $t0, ' ', valid_label      #Correct case
                beq     $t0, '\n', valid_label     #Correct case
                beq     $t0, 0, valid_label         #If ends, exit

                blt     $t0, 'a', not_lower2       #If less than a, exit
                bgt     $t0, 'z', not_lower2       #If greater than z, exit
                j       loop_label_check          # if a<= t0 <= z then lowercase
character, loop

not_lower2:
                bne     $t0, '_', not_underscore  # Self explanatory
                j       loop_label_check          # if A<= t0 <=Z then uppercase
character, loop

not_underscore:
                blt     $t0, 'A', not_upper2       #If less than A, means not
alphabet
                bgt     $t0, 'Z', not_upper2       #If greater than Z, means not
alphabet
                j       loop_label_check          # if A<= t0 <=Z then uppercase
character, loop

not_upper2:
                blt     $t0, '0', fail_case        #If less than 0, means not
number either, failcase
                bgt     $t0, '9', fail_case        #If greater than 9, means not
not number either, failcase
                j       loop_label_check          # if A<= t0 <=Z then uppercase
character, loop

fail_case:
                move    $a0, $s0                  # Reset to before so we check
other case (not using label as address but numerical)
                li      $s7, 0                    # Case checker so we know to
check numerical address
                jr      $ra                        # jump to jump_

```

```

        valid_label:
            move    $s0, $a0                # Move pointer forward
            li      $s7, 1                  # Case checker = 1 (valid)
            jr      $ra

prompts:
    start:

        la        $a0, border
        li        $v0, 4
        syscall
        la        $a0, start_mes
        li        $v0, 4
        syscall
        la        $a0, border
        li        $v0, 4
        syscall
        la        $a0, Message              # Load and print string
asking for string
        li        $v0, 4
        syscall
        jr        $ra

correct_opcode:
        la        $a0, correct_opcode_prompt
        li        $v0, 4
        syscall
        la        $a0, opcode
        li        $v0, 4
        syscall
        move      $a0, $s0                  # Return $a0
        jr        $ra

# ----- #
#           Registers used:                 #
#           $a0: prompt address             #
# ----- #

error:
    missing_colon:
        la        $a0, missing_colon_prompt # Load and print string
asking for string

```

```

        li    $v0, 4
        syscall
        j      ending                # jump to ending

invalid_opcode:
        la    $a0, invalid_opcode_prompt
        li    $v0, 4
        syscall
        j      ending                # jump to ending

too_many_variable:
        move  $a0, $s0                # Load and print string asking for string
        li    $v0, 1
        syscall
        # j      ending                # jump to ending
        la    $a0, too_many_variable_prompt
        li    $v0, 4
        syscall
        j      ending

not_valid_register:
        la    $a0, not_valid_register_prompt
        li    $v0, 4
        syscall
        j      ending

not_num_error:
        la    $a0, not_valid_number_prompt
        li    $v0, 4
        syscall
        j      ending

not_valid_address:
        la    $a0, not_valid_address_prompt
        li    $v0, 4
        syscall
        j      ending

missing_:
        la    $a0, missing_prompt
        li    $v0, 4
        syscall
        j      ending

valid_syntax:
        la    $a0, valid_syntax_prompt
        li    $v0, 4
        syscall

ending:
        la    $a0, continue_prompt
        li    $v0, 4
        syscall

```

```

    li    $v0, 5
    syscall

    beq    $v0, 1, resetAll

    li    $v0, 10
    syscall

resetAll:

    li $v0, 0
    li $v1, 0
    jal      clean_block      # jump to clean_block
    jal      clean_opcode    # jump to clean_block
    li $a0, 0
    li $a1, 0
    li $a2, 0
    li $a3, 0
    li $t0, 0
    li $t1, 0
    li $t2, 0
    li $t3, 0
    li $t4, 0
    li $t5, 0
    li $t6, 0
    li $t7, 0
    li $t8, 0
    li $t9, 0
    li $s0, 0
    li $s1, 0
    li $s2, 0
    li $s3, 0
    li $s4, 0
    li $s5, 0
    li $s6, 0
    li $s7, 0
    li $k0, 0
    li $k1, 0

    j main

```

```

clean_block:
    li $a0, 0
    li $a1, 0
    la    $s0, buffer
loop_block:
    beq    $a1, 100, jump_
    sb     $a0, 0($s0)
    addi   $s0, $s0, 1
    addi   $a1, $a1, 1
    j      loop_block

clean_opcode:
    li $a0, 0
    li $a1, 0
    la    $s1, opcode
loop_opcode:
    beq    $a1, 10, jump_
    sb     $a0, 0($s1)
    addi   $s1, $s1, 1
    addi   $a1, $a1, 1
    j      loop_opcode

```

### Trình tự thực hiện:

- Bước 1:

Tạo các lệnh đọc input, cũng như phân opcode ra khỏi string.

```

read_data:
# ----- #
#           Read data                               #
# ----- #
    li      $v0, 8                                # take in input
    la      $a0, buffer                          # load byte space into address
    li      $a1, 100                             # allot the byte space for string
    syscall
    move    $s0, $a0                             # save string to $s0
# ----- #
#           Registers used:                         #
#           $s0, $a0: string's address              #
# ----- #

```

Các tập lệnh lưu trữ địa chỉ toàn bộ lệnh opcode cần kiểm tra \$s0, \$a0

Sau đó cần phải tạo chương trình con để xử lý nhiều dấu cách ngăn cách giữa opcode với toán tử hạng cũng như các toán tử hạng khác nhau. Ta tạo chương trình con check\_whitespace:

```

check_whitespace:
# ----- #
#           Read white space before code           #
# ----- #
    move     $a0, $s0                # Dong bo con tro $a0 voi $s0
    lb       $t1, 0($a0)             # Doc tung ki tu cua command
    beq      $t1, ' ', loop_whitespace # Loop de tru het dau cach
    jr       $ra                    # Trong truong hop khong phai, quay tro ve chu
loop_whitespace:
    lb       $t1, 0($a0)             # Doc tung ki tu cua command
    beq      $t1, ' ', check_whitespace_pass # Khi gap dau cach thi tiep tục doc
    # addi    $a0, $a0, 1             # Dich $a0 di 1 ky tu
    move     $s0, $a0                # Cho dia chi moi cho s0 la ky tu dau tien cua
    jr       $ra                    # jump to $ra
check_whitespace_pass:
    addi     $a0, $a0, 1             # Dich $a0 di 1 ky tu
    j        loop_whitespace        # jump to check_whitespace
# ----- #
#           Registers used: $t1, $a0, $s0, $ra      #
#           Pointer will point at next character    #
# ----- #

```

Cũng như tạo chương trình check\_gap để kiểm tra dấu phẩy giữa các toán tử (có thể có dấu cách hoặc không) bằng cách kết hợp chương trình kiểm tra dấu phẩy và dấu cách là check\_gap

```

check_gap:
# ----- #
#           Check gap between code                 #
# ----- #
    move     $t4, $ra
    jal      check_whitespace
    jal      check_colon
    addi     $a0, $a0, 1                # Point to character/w
    move     $s0, $a0
    jal      check_whitespace
    move     $ra, $t4
    jr       $ra
# ----- #
#           Registers used: $t1, $a0, $s0, $ra      #
# ----- #

```

Tất cả các chương trình này đều chạy con trỏ \$a0 để kiểm tra, sau khi kiểm tra thành công hoặc là trả về vị trí hoặc là cho con trỏ \$s0 đồng bộ để kiểm tra ký tự tiếp theo trong lệnh

Ta chia ra làm 2 là kiểm tra opcode và kiểm tra các toán tử.

## Kiểm tra opcode

- Với kiểm tra opcode, ta tạo thư viện toàn bộ lệnh opcode cùng chia dạng (tổng cộng là 9) dựa theo syntax khác nhau có thể có để sau kiểm tra tính hợp lệ của opcode.
- Đầu tiên là thư viện thì sẽ chia theo dạng syntax như sau

Type\_1:

```

# ----- #
#   Format xyz $1, $2, $3                        #
# ----- #
# Ari1Library_: .asciiz  "add, sub, addu, subu, mul; " # Format add $1, $2, $3
# Log1Library_: .asciiz  "and, or, nor, xor; " # Format and and $1, $2, $3 //
# Com1Library_: .asciiz  "slt; " # Format slt $1,$2,$3

```

Type\_2:

```
# ----- #
#   Format xyz $1, $2, 10000                               #
# ----- #
# Ari2Library:  .asciiiz   "addi, addiu; " # Format addi $1,$2,100
# Log2Library:  .asciiiz   "andi, ori,  sll,  srl; " # Format or andi $1,$2,
# Com2Library:  .asciiiz   "slti, sltiu; " # Format slti $1, $2, 100
```

Type\_3:

```
# ----- #
#   Format mult $2,$3                                     #
# ----- #
# Ari3Library:  .asciiiz   "mult, div; " # Format mult $2,$3 //
# Dat5Library:  .asciiiz   "move; " # Format move $1, $2
```

Type\_4:

```
# ----- #
#   Format lw $1, 100($2)                                #
# ----- #
# Dat1Library:  .asciiiz   "lw,  sw,  lb,  sb,  lbu,  lhu,  ll,  sh; "
```

Type\_5:

```
# ----- #
#   Format lui $1, 100                                    #
# ----- #
# Dat2Library:  .asciiiz   "lui,  li; " # Format lui $1, 100 //
```

Type\_6:

```
# ----- #
#   Format la $1,label                                    #
# ----- #
# Dat3Library:  .asciiiz   "la; " # Format la $1,label //
```

Type\_7:

```
# ----- #
#   Format mfhi $2                                        #
# ----- #
# Dat4Library:  .asciiiz   "mfhi,  mflo; " # Format mfhi $2
# Jum2Library:  .asciiiz   "jr; " # Format jr $1
```

Type\_8:

```
# ----- #
#   Format beq $1, $2, label beq $1,$2,100               #
# ----- #
# Con1Library:  .asciiiz   "beq,  bne; " # Format beq $1,$2,100 ; beq $1, $2, label
```

Type\_9:

```
# ----- #
```

```
# Format j 1000 ; j label #
# ----- #
# Jum1Library: .asciiz    ",    jal; " # Format j 1000 ; j label
```

2 loại cuối đặc biệt là có 2 khả năng xảy ra. Sau khi ta chia ra được thành 9 loại, ta sẽ tạo dựng thư viện tương ứng để kiểm tra syntax của opcode:

```
opcdLibrary: .asciiz    "add,1 sub,1 addu,1 subu,1 mul,1 and,1 or,1 nor,1 xor,1 slt,1 addi
,2 addiu,2 andi,2 ori,2 sll,2 srl,2 slti,2 sltiu,2
mult,3 div,3 move,3 lw,4 sw,4 lb,4 sb,4 lbu,4 lhu,4 ll,4 sh,4 lui,5 li,5 la,6 mfhi,7 mflo
,7 jr,7 beq,8 bne,8 j,9 jal,9 "
```

Mỗi phần tử chiếm 8 byte, đi kèm dấu phẩy và số bên cạnh thể hiện type của nó. Sau đây ta tạo code kiểm tra tính hợp lệ của opcode:

```
check_opcode:
# ----- #
#      Check opcode's validity      #
# ----- #
    move    $a1, $s1        # Bring back $a1 to the beginning
    move    $s0, $a0        # Push pointer s0 to new space

# Check library if there is matching syntax #

    la      $s2, opcdLibrary
    jal     check

    j       invalid_opcode    # jump to target

check:
    move    $a2, $s2        # a2 pointer to beginning of library
loop_check:
    lb      $t2, 0($a2)      # load each character from library
    beq     $t2, ':', evaluation1    # if encountered colon, evaluate whether it is correct
    lb      $t1, 0($a1)      # load each character from opcode
    beq     $t2, 0, jump_    # if encountered /0 then not found valid opcode
    bne     $t1, $t2, next_opcode    # character mismatch
    addi    $a1, $a1, 1      # next character
    addi    $a2, $a2, 1
    j       loop_check

evaluation1:
    lb      $t1, 0($a1)      # load each character from opcode
    beq     $t1, 0, opcode_done
    j       next_opcode      # jump to $ra
```



```

next_opcode:
    addi    $s2, $s2, 8          # Each opcode has length of 8 byte, thus moving to next opcode
    move    $a2, $s2
    move    $a1, $s1
    j       loop_check          # Keep looping

```

opcode\_done:

Ta sẽ có 2 con trỏ khác nhau, tại 2 mảng là opcode (sau khi được tách khỏi chính) là \$s1 (\$a1) và opcdLibrary \$s2 (\$a2) (thư viện đã tạo) em sẽ lần từng opcode trong thư viện xem có trùng với \$s1 không (bằng cách so sánh) cùng với việc lấy dạng (Type) của nó nếu hợp lệ. Bởi mỗi opcode cách nhau 8 byte nên nếu k hợp lệ ta chỉ việc cho \$s2 nhảy 8 byte là được.

Sau khi ta làm xong bước này ta có lần lượt trạng thái con trỏ \$s1 chỉ dấu cách sau opcode. Ta khứ đi bằng lệnh con check\_whitespace rồi kiểm tra toán hạng.

## Kiểm tra toán hạng

Ta sẽ có 4 trường hợp toán hạng cần phải kiểm tra:

- Thanh ghi
- Số
- Địa chỉ (label)
- Địa chỉ theo thanh ghi ( 0(\$1) )

Ta lần lượt tạo 4 chương trình con để kiểm tra 4 trường hợp trên

### Thanh ghi

```

reg_check:
    # ----- #
    #      Check whether string is register or not      #
    # ----- #

    la     $s3, tokenRegisters
    move   $a3, $s3          # a3 points to beginning of register library
    move   $a0, $s0

loop_reg_check:
    lb     $t3, 0($a3)        # load each character from library
    lb     $t0, 0($a0)        # load each character from string
    beq     $t3, ' ', evaluation2    # if encountered space, evaluate whether it is correct
    beq     $t3, 0, not_valid_register    # if encountered /0 then not valid register
    bne     $t0, $t3, next_reg    # character mismatch
    addi    $a0, $a0, 1          # next character
    addi    $a3, $a3, 1
    j       loop_reg_check

```

```

evaluation2:
    lb      $t0, 0($a0)
    beq     $t0, ',', found_reg      # Correct register
    beq     $t0, '.', found_reg      # Correct register
    beq     $t0, 0, found_reg        # Correct register
    beq     $t0, '\n', found_reg     # Correct register
    j       next_reg                 # jump to next_register

next_reg:
    addi    $s3, $s3, 8              # Move to next register token
    move    $a3, $s3
    move    $a0, $s0
    j       loop_reg_check           # jump to loop_reg_check

found_reg:
    move    $s0, $a0                # move pointer forward
    j       jump_                   # jump to jump_

```

## Số

```

# ----- #
#       Check whether string is register or not       #
# ----- #

move      $a0, $s0
num_check_loop:
    lb      $t0, 0($a0)
    beq     $t0, ',', is_num        # Correct register
    beq     $t0, '.', is_num        # Correct register
    beq     $t0, 0, is_num          # Correct register
    beq     $t0, '\n', is_num       # Correct register
    bgt     $t0, '9', not_num        # if $t0 > 9 then target
    blt     $t0, '0', not_num        # if $t0 < 0 then target
    addi    $a0, $a0, 1
    j       num_check_loop          # jump to num_check_loop

is_num:
    move    $s0, $a0
    j       jump_                   # jump to jump_

not_num:
    j       not_num_error           # jump to not_num_error

```

Địa chỉ (theo thanh ghi) (Ta kết hợp kiểm tra thanh ghi và số ngăn cách bởi dấu “()”)

address\_check:

adnum\_check:

```

num_check_loop2:
    lb      $t0, 0($a0)
    beq     $t0, '[', is_num2       # Correct registe
    bgt     $t0, '9', not_num2      # if $t0 > 9 then target
    blt     $t0, '0', not_num2      # if $t0 < 0 then target
    addi    $a0, $a0, 1
    j       num_check_loop2         # jump to num_check_loop

is_num2:

```

```

        move    $s0, $a0
        j      adreg_check      # jump to jump_
not_num2:

        j      not_valid_address      # jump to not_num_error
adreg_check:
    reg_check2:
        addi   $a0, $a0, 1
        move   $s0, $a0
# ----- #
#       Check whether string is register or not       #
# ----- #
    la    $s3, tokenRegisters
    move  $a3, $s3      # a3 points to beginning of register library
    move  $a0, $s0

loop_reg_check2:
    lb    $t3, 0($a3)      # load each character from library
    lb    $t0, 0($a0)      # load each character from string
    beq   $t3, ' ', evaluation3      # if encountered space, evaluate whether it is correct
    beq   $t3, 0, not_valid_address2  # if encountered /0 then not valid register
    bne   $t0, $t3, next_reg2      # character mismatch
    addi   $a0, $a0, 1      # next character
    addi   $a3, $a3, 1
    j     loop_reg_check2
evaluation3:
    lb    $t0, 0($a0)
    beq   $t0, ' ', found_reg2      # Correct register
    j     next_reg2      # jump to next_register
next_reg2:
    addi   $s3, $s3, 8      # Move to next register token
    move   $a3, $s3
    move   $a0, $s0
    j     loop_reg_check2      # jump to loop_reg_check
not_valid_address2:
    move   $a0, $t0
    li    $v0, 11
    syscall
    j     not_valid_address
found_reg2:
    addi   $a0, $a0, 1
    move   $s0, $a0      # move pointer forward
    jr     $ra

```

Địa chỉ (label)

Với địa chỉ ta chỉ có lưu ý nhỏ là điều kiện của chữ cái đầu tiên hơi khác so với điều kiện chữ còn lại nên ta làm riêng.

label\_check:

```
move $a0, $s0
```

First\_char\_check: # Can't be number and can't be underscore:

```
lb  $t0, ($a0)      #Load byte from 't0'th position in buffer into $t1
blt $t0, 'a', not_lower  #If less than a, exit
bgt $t0, 'z', not_lower  #If greater than z, exit

j    loop_label_check    # It's lower so we jump to 2nd character
```

not\_lower:

```
blt $t0, 'A', fail_case  #If less than A, means not alphabet, failcase
bgt $t0, 'Z', fail_case  #If greater than Z, means not alphabet, failcase
```

loop\_label\_check: # Can be alphabet, number and underscore

```
addi $a0, $a0, 1      # Increment
lb  $t0, ($a0)        #Load byte from 't0'th position in buffer into $t0
```

```
beq $t0, '', valid_label  #Correct case
beq $t0, '\n', valid_label  #Correct case
beq $t0, 0, valid_label    #If ends, exit
```

```
blt $t0, 'a', not_lower2  #If less than a, exit
bgt $t0, 'z', not_lower2  #If greater than z, exit
j    loop_label_check    # if a<= t0 <= z then lowercase character, loop
```

not\_lower2:

```
bne $t0, '_', not_underscore  # Self explanatory
j    loop_label_check    # if A<= t0 <=Z then uppercase character, loop
```

not\_underscore:

```
blt $t0, 'A', not_upper2  #If less than A, means not alphabet
bgt $t0, 'Z', not_upper2  #If greater than Z, means not alphabet
j    loop_label_check    # if A<= t0 <=Z then uppercase character, loop
```

not\_upper2:

```
blt $t0, '0', fail_case  #If less than 0, means not number either, failcase
bgt $t0, '9', fail_case  #If greater than 9, means not not number either, failcase
j    loop_label_check    # if A<= t0 <=Z then uppercase character, loop
```

fail\_case:

```
move $a0, $s0      # Reset to before so we check other case (not using label as address but numerical)
li   $s7, 0        # Case checker so we know to check numerical address
jr   $ra           # jump to jump_
```

```

valid_label:
    move $s0, $a0      # Move pointer forward
    li   $s7, 1        # Case checker = 1 (valid)
    jr   $ra

```

Vậy là đã chia xong 4 trường hợp khác nhau của toán tử, ta chỉ cần cho từng trường hợp phù hợp với từng dạng (Type), ví dụ như sau:

Type\_5:

```

# ----- #
#   Format lui $1, 100                               #
# ----- #
# Dat2Library: .asciiz    "lui, li; " # Format lui $1, 100 //
    jal  reg_check
    jal  check_gap
    jal  num_check
    jal  check_end

```

Để ý rằng là mình chỉ bắt đầu kiểm tra từ toán tử đầu tiên vì phần cách và opcode đã được xử lý trước. Cũng dùng chung nguyên lý bài trước là có pointer s0 (a0) chạy kiểm tra từng ký tự 1 xem có thỏa mãn hay k, ta có thể thấy ví dụ trên

- lui \$1, 100 (\$1, 100) bao gồm:
  - \$1 là register (reg\_check)
  - Có phẩy và cách giữa \$1 và 100 (check\_gap)
  - 100 (num\_check)
  - Kiểm tra kết thúc (xem có thừa toán tử không) là '\n' (Có thể có cả dấu cách được xử lý trong chương trình con check\_end)

Cùng nguyên lý như vậy ta xử lý 7 Type có 1 trường hợp còn đối với các type có 2 ta tạo 1 branch beq đơn giản cùng 1 số hạng để check (có sẵn trong label\_check do tất cả các type có 2 dạng đều có nó) để xử lý và làm tương tự

Type\_9:

```

# ----- #
#   Format j 1000 ; j label                           #
# ----- #
# Jum1Library: .asciiz    "j, jal; " # Format j 1000 ; j label
    jal  label_check
    beq  $s7, 1, check_end
    jal  num_check

```

end\_type9:

```

    jal  check_end

```

Ví dụ chạy

```
# -----#
Opcode Checker
# -----#
Enter string: add $1, $2, $3

Correct opcode: add
Correct MIPS syntax.
Continue? (1. Yes 0. No): |
```

1

```
# -----#
Opcode Checker
# -----#
Enter string: beq $1, $2, $3

Correct opcode: beq
Not valid number.
Continue? (1. Yes 0. No): |
```

```
# -----#
Opcode Checker
# -----#
Enter string: beq $1, $2, label

Correct opcode: beq
Correct MIPS syntax.
Continue? (1. Yes 0. No): |
```

*Tài liệu tham khảo:*

- Bài giữa kỳ phần String của em
- <https://www.geeksforgeeks.org/tokenizing-a-string-cpp/>

**-- Hết phần trình bày bài 7 --**

## Bài 9 - Phùng Trung Kiên 20204994:

## SOURCE CODE:

<https://drive.google.com/file/d/11wTMhxiczlbqelpXdkXvfAC28TDUQbkl/view?usp=sharing>

```

.data
    # (21+21+21+1)x16 = 1024
    String: .asciiz "
        ***** \n*****
*3333333333333333* \n*2222222222222222* *33333* *33333***** \n*2222* *2222*
\n*2222******22222* *33333* \n*2222* *2222*
*33333***** \n*2222* *2222* ***** *3333333333333* \n*2222*
*2222* **1111*****111* *33333***** \n*2222* *2222* **1111** ** *33333*
\n*2222* *22222* *1111* *33333***** \n*2222*****22222* *1111*
*3333333333333* \n*2222222222222222* *1111*
*****
\n***** *1111* \n --- *1111**
\n / o o \ \n \ \ > /
**11111**111* \n ***** dce.hust.edu.vn
\n"
# " ***** \n"
# "***** *3333333333333* \n"
# "*2222222222222222* *33333***** \n"
# "*22222*****22222* *33333* \n"
# "*2222* *2222* *33333***** \n"
# "*2222* *2222* ***** *3333333333333* \n"
# "*2222* *2222* **1111*****111* *33333***** \n"
# "*2222* *2222* **1111** ** *33333* \n"
# "*2222* *22222* *1111* *33333***** \n"
# "*2222*****22222* *1111* *3333333333333* \n"
# "*2222222222222222* *1111* ***** \n"
# "***** *1111* \n"
# " --- *1111** \n"
# " / o o \ \n \ \ > /
# " \ \ > /
# " ----- ***** dce.hust.edu.vn \n"
M:.asciiz "\n\n\n=====MENU===== \n|1. Hien thi hinh
anh tren giao dien |\n|2. Hien thi hinh anh chi con lai vien, khong co mau o
giao|\n|3. Hien thi hinh anh sau khi hoan doi vi tri |\n|4. Nhap tu ban phim ki tu mau
cho chu D, C, E roi hien thi|\n|(Nhap exit de thoat chuong trinh)
|\n===== \n\n\n"
Mess: .asciiz "\nNhap 3 ky tu tuong ung voi 3 mau moi lan luot cua D,C,E\n\n\n"
##### Su dung lai code bai 2 Tuan 10(2) De tao Menu #####
.equv KEY_CODE 0xFFFF0004 # ASCII code from keyboard, 1 byte
.equv KEY_READY 0xFFFF0000 # =1 if has a new keycode ?
# Auto clear after 1w
.equv DISPLAY_CODE 0xFFFF000C # ASCII code to show, 1 byte
.equv DISPLAY_READY 0xFFFF0008 # =1 if the display has already to do
# Auto clear after sw
.text
li $k0, KEY_CODE
li $k1, KEY_READY

li $s0, DISPLAY_CODE
li $s1, DISPLAY_READY
#4 ky tu gan nhât
li $s2, 0
li $s3, 0
li $s4, 0
li $s5, 0
Menu:
li $v0, 4
la $a0, M
syscall
loop: nop
WaitForKey: lw $t1, 0($k1) # $t1 = [$k1] = KEY_READY
beq $t1, $zero, WaitForKey # if $t1 == 0 then Polling
ReadKey: lw $t0, 0($k0) # $t0 = [$k0] = KEY_CODE
WaitForDis: lw $t2, 0($s1) # $t2 = [$s1] = DISPLAY_READY
beq $t2, $zero, WaitForKey # if $t2 == 0 then Polling
#Luu 4 ky tu gan nhât
addi $s5, $s4, 0

```

```

    addi $s4, $s3, 0
    addi $s3, $s2, 0
    addi $s2, $t0, 0
#Thoat chuong trinh khi 4 ky tu tao thanh chu exit
    bne $s5, 101, Encrypt
    bne $s4, 120, Encrypt
    bne $s3, 105, Encrypt
    bne $s2, 116, Encrypt
    j exit

Encrypt:
    beq $t0, 49, f1
    beq $t0, 50, f2
    beq $t0, 51, f3
    beq $t0, 52, f4
    j ShowKey

ShowKey:
    sw $t0, 0($s0) # show key
    nop
    j loop

##### Chuc nang so 1 #####
f1:
    li $v0, 4
    la $a0, String
    syscall
    j Menu

##### Chuc nang so 2 #####
f2:
    addi $s6, $zero, 0
    la $s7, String          # $s7 la dia chi cua String
loop_f2:
    beq $s6, 1024, Menu      # In ra toan bo 1024 ky tu
    lb $t3, 0($s7)          # $t3 luu gia tri cua tung phan tu trong String

    bge $t3, 58, print_f2    # Tu (0-9) trong bang ma ascii tu 48-58
    bge $t3, 48, Chuso_f2
    j print_f2
Chuso_f2:
    addi $t3, $zero, 32      # space trong ascii la 32
print_f2:
    li $v0, 11              # In tung ki tu
    addi $a0, $t3, 0
    syscall

    addi $s6, $s6, 1        # s6 += 1
    addi $s7, $s7, 1        # s7 += 1
    j loop_f2

##### Chuc nang so 3 #####
f3:
    # (21+21+21+1)x16
    # DCE -> ECD
    # [ (+42) In 21 -> (-42) In 21 -> (-42) In 21 -> (+42) In 1 ] x 16
    addi $s6, $zero, 0
    la $s7, String          # $s7 la dia chi cua String

loop_f3:
    beq $s6, 16, Menu
    addi $s6, $s6, 1
    # (+42) In 21
    addi $s7, $s7, 42
    jal in_21
    # (-42) In 21
    addi $s7, $s7, -42
    jal in_21
    # (-42) In 21
    addi $s7, $s7, -42
    jal in_21
    # (+42) In 1
    addi $s7, $s7, +42
    lb $t3, 0($s7)          # $t3 luu gia tri cua tung phan tu trong String
    li $v0, 11              # In tung ki tu
    addi $a0, $t3, 0
    syscall

```



```

        addi $s7, $s7, 1    # s7 += 1
    j loop_f3

#In 21
in_21:
    addi $t4, $zero, 0
loop_2_f3:
    lb $t3, 0($s7)         # $t3 lưu giá trị của từng phần tử trong String
    li $v0, 11             # In từng kí tự
    addi $a0, $t3, 0
    syscall

    addi $s7, $s7, 1    # s7 += 1
    addi $t4, $t4, 1    # t4 += 1
    bne $t4, 21, loop_2_f3
jr $ra

##### Chuc nang so 4 #####
f4:
    li $v0, 4
    la $a0, Mess
    syscall

    la $s7, String        # $s7 là địa chỉ của String
    addi $t5, $t5, 0
loop_input_f4:
#loop:
    nop
    WaitForKey2: lw $t1, 0($k1) # $t1 = [$k1] = KEY_READY
    beq $t1, $zero, WaitForKey2 # if $t1 == 0 then Polling
    ReadKey2: lw $t0, 0($k0) # $t0 = [$k0] = KEY_CODE
    WaitForDis2: lw $t2, 0($s1) # $t2 = [$s1] = DISPLAY_READY
    beq $t2, $zero, WaitForDis2 # if $t2 == 0 then Polling
    #Luu 4 ky tu gan nhat
    addi $s5, $s4, 0
    addi $s4, $s3, 0
    addi $s3, $s2, 0
    addi $s2, $t0, 0

    addi $t5, $t5, 1    # t5 += 1
    bne $t5, 3, loop_input_f4

    addi $t6, $t6, 0
loop_print_f4:
    addi $t7, $s4, 0    # Mau của chu D
    jal in_21_f4
    addi $t7, $s3, 0    # Mau của chu C
    jal in_21_f4
    addi $t7, $s2, 0    # Mau của chu E
    jal in_21_f4
    # In \n
    lb $t3, 0($s7)         # $t3 lưu giá trị của từng phần tử trong String
    li $v0, 11             # In từng kí tự
    addi $a0, $t3, 0
    syscall
    addi $s7, $s7, 1    # s7 += 1

    addi $t6, $t6, 1    # t6 += 1
    bne $t6, 16, loop_print_f4

#FREE
li $s2, 0
li $s3, 0
li $s4, 0
li $s5, 0
li $t0, 0
j Menu

in_21_f4:
    addi $t4, $zero, 0
loop_2_f4:
    lb $t3, 0($s7)         # $t3 lưu giá trị của từng phần tử trong String

```

```

        bge $t3, 58, print_f4      # Tu (0-9) trong bang ma ascii tu 48-58
        bge $t3, 48, Chuso_f4
        j print_f4
Chuso_f4:
        addi $t3, $t7, 0          # Mau tuong ung
print_f4:
        li $v0, 11                # In tung ki tu
        addi $a0, $t3, 0
        syscall

        addi $s7, $s7, 1          # s7 += 1
        addi $t4, $t4, 1          # t4 += 1
        bne $t4, 21, loop_2_f4
        jr $ra
exit:
```

**Trình tự thực hiện:**

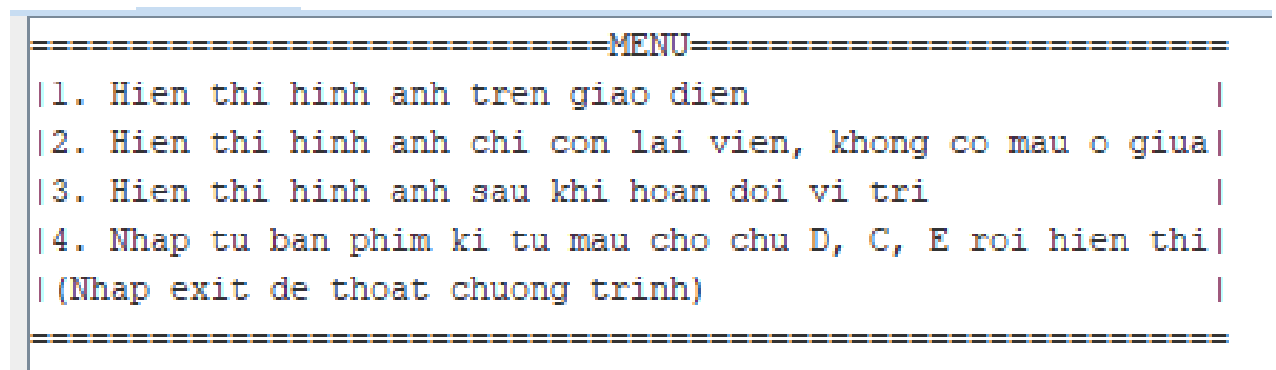
- + Nhập liệu vùng nhớ lớn chứa ảnh.

Chứa toàn bộ dữ liệu về ảnh vào String bao gồm cả ký tự xuống dòng. Mỗi dòng bao gồm 64 ký tự, 21 ký tự tương ứng với 1 chữ cái, ký tự cuối là ký tự xuống dòng (\n)

[illegible]

- + Tao Menu:

Menu sẽ hiển thị lên console mỗi khi bắt đầu chương trình.



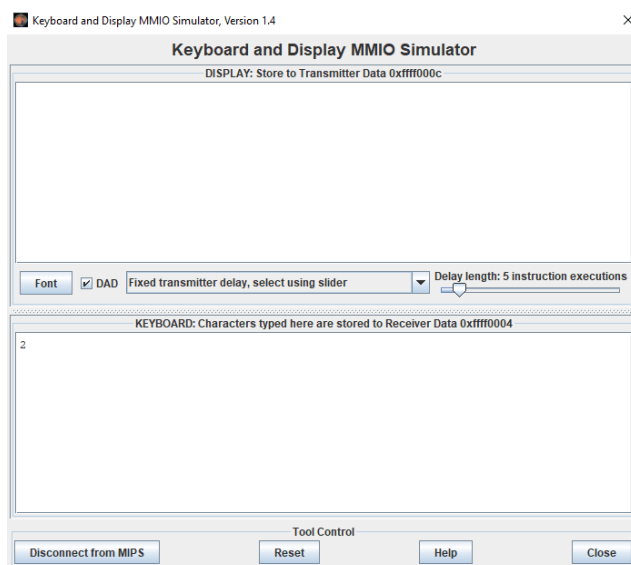
```
##### Su dung lai code bai 2 Tuan 10(2) De tao Menu #####
.eqv KEY_CODE 0xFFFF0004 # ASCII code from keyboard, 1 byte
.eqv KEY_READY 0xFFFF0000 # =1 if has a new keycode ?
# Auto clear after lw
.eqv DISPLAY_CODE 0xFFFF000C # ASCII code to show, 1 byte
.eqv DISPLAY_READY 0xFFFF0008 # =1 if the display has already to do
# Auto clear after sw
.text
li $k0, KEY_CODE
li $k1, KEY_READY

li $s0, DISPLAY_CODE
li $s1, DISPLAY_READY
#4 ky tu gan nhut
li $s2, 0
li $s3, 0
li $s4, 0
li $s5, 0
Menu:
li $v0, 4
la $a0, M
syscall
loop: nop
WaitForKey: lw $t1, 0($k1) # $t1 = [$k1] = KEY_READY
beq $t1, $zero, WaitForKey # if $t1 == 0 then Polling
ReadKey: lw $t0, 0($k0) # $t0 = [$k0] = KEY_CODE
WaitForDis: lw $t2, 0($s1) # $t2 = [$s1] = DISPLAY_READY
beq $t2, $zero, WaitForKey # if $t2 == 0 then Polling
#Luu 4 ky tu gan nhut
addi $s5, $s4, 0
addi $s4, $s3, 0
addi $s3, $s2, 0
addi $s2, $t0, 0
#Thoat chuong trinh khi 4 ky tu tao thanh chu exit
bne $s5, 101, Encrypt
bne $s4, 120, Encrypt
bne $s3, 105, Encrypt
bne $s2, 116, Encrypt
j exit

Encrypt:
beq $t0, 49, f1
beq $t0, 50, f2
beq $t0, 51, f3
beq $t0, 52, f4
j ShowKey

ShowKey:
```

Sử dụng giao diện công cụ giả lập Keyboard and Display MMIO Simulator để Input từ bàn phím. Mỗi khi nhận một lệnh vào Keyboard trong công cụ giả lập, chương trình sẽ thực hiện yêu cầu rồi tự động đưa vào trạng thái chờ lệnh kế tiếp.

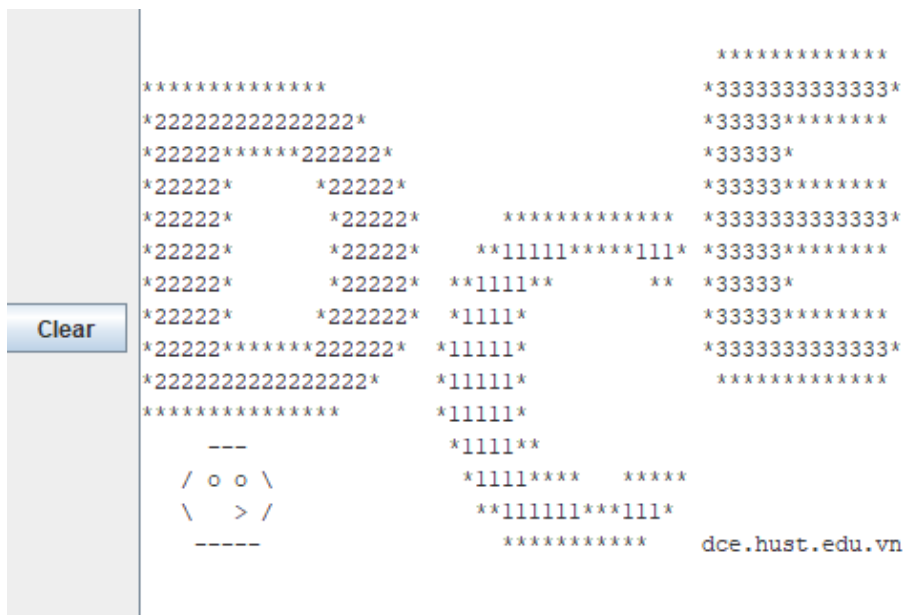


+ Chức năng 1: Diễn thị hình ảnh trên lên giao diện console.

```
##### Chuc nang so 1 #####
f1:
    li $v0, 4
    la $a0, String
    syscall
    j Menu
```

Giải thích: In ra toàn bộ String bằng syscall (\$v0 = 4). Do dữ liệu chỉ lưu trong String, nên việc in ra hình ảnh là rất dễ dàng - bằng cách in ra String.

\* Lưu ý: Ký tự “\” trong hình ảnh khi lập trình với MIPS nó sẽ không được hiểu là ký tự “\” như thông thường, do “\” thường được dùng khi tạo 1 ký tự. Nên ta thay “\” bằng “\\” thì khi in ra nó sẽ in được ký tự “\”. Nên trong phần data sẽ có “\\” thay vì “\”.



```
##### Chuc nang so 1 #####
f1:
    li $v0, 4
    la $a0, String
    syscall
    j Menu
```

Clear

```

          *****
          *3333333333333*
*222222222222222*  *33333*****
*22222*****22222*  *33333*
*22222*      *22222*  *33333*****
*22222*      *22222*  *****  *3333333333333*
*22222*      *22222*  **11111*****111*  *33333*****
*22222*      *22222*  **1111**      **  *33333*
*22222*      *22222*  *1111*          *33333*****
*22222*****22222*  *11111*          *3333333333333*
*222222222222222*  *11111*          *****
*****          *11111*
          ---      *1111*
          / o o \      *1111*****  *****
          \   > /      **111111**111*
          -----      *****  dce.hust.edu.vn
```

+ Chức năng 2:

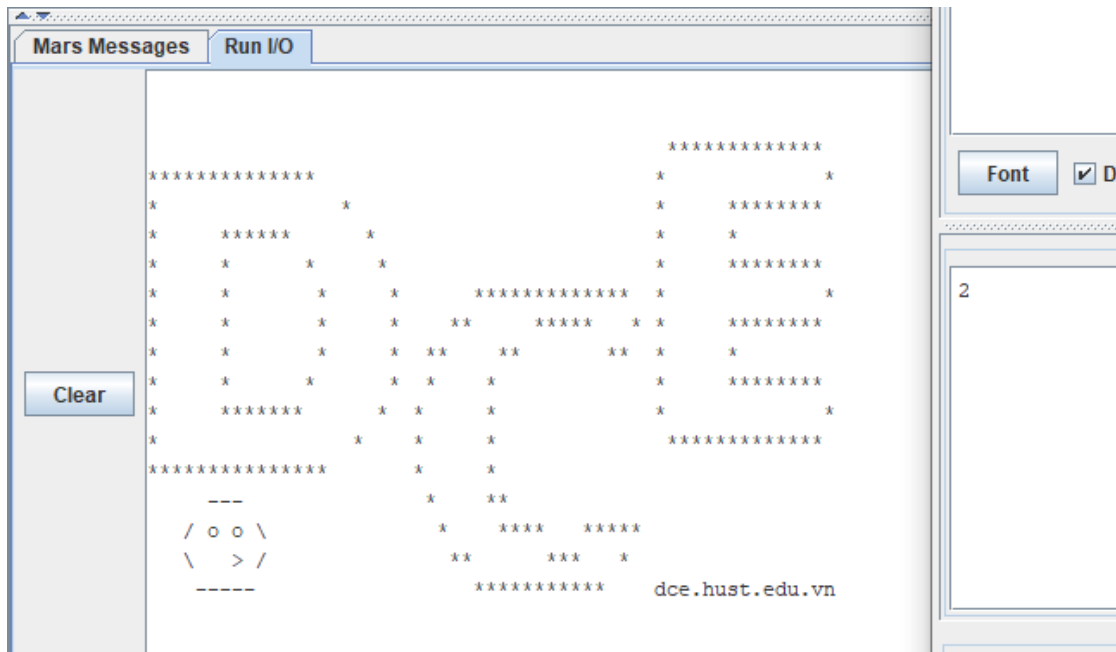
```
##### Chuc nang so 2 #####
f2:
    addi $s6, $zero, 0
    la $s7, String      # $s7 la dia chi cua String
loop_f2:
    beq $s6, 1024, Menu  # In ra toan bo 1024 ky tu
    lb $t3, 0($s7)      # $t3 luu gia tri cua tung phan tu trong String

    bge $t3, 58, print_f2  # Tu (0-9) trong bang ma ascii tu 48-58
    bge $t3, 48, Chuso_f2
    j print_f2
Chuso_f2:
    addi $t3, $zero, 32  # space trong ascii la 32
print_f2:
    li $v0, 11          # In tung ki tu
    addi $a0, $t3, 0
    syscall

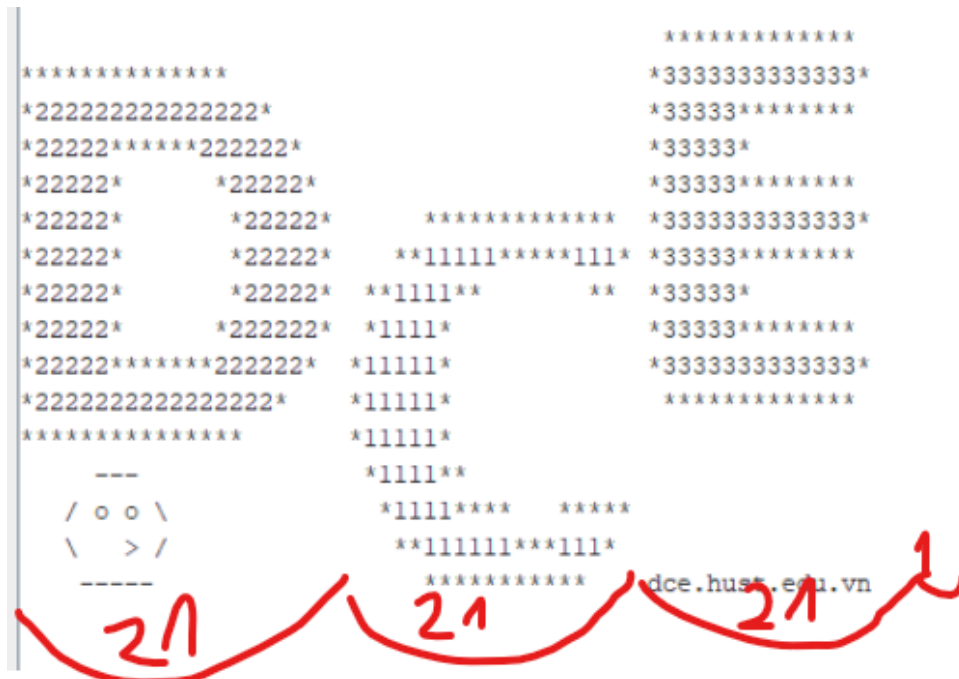
    addi $s6, $s6, 1    # s6 += 1
    addi $s7, $s7, 1    # s7 += 1
    j loop_f2
```

Giải thích: In từng ký tự (1024 ký tự) trong String bằng syscall (\$v0=11) Nếu ký tự nó nằm trong đoạn từ [0-9] thì đổi thành ký tự space.

Lưu ý: Ký tự từ [0-9] nằm trong đoạn [48-58] trong bảng mã Ascii, nên ta sẽ so sánh với 58 trước, nếu lớn hơn 58 thì sẽ không phải là một chữ số nên in ra như bình thường. Sau đó so sánh với 48, nếu lớn hơn hoặc bằng 48 và không lớn hơn 58 thì ký tự đó nằm trong đoạn [0-9], thì ra sẽ thay ký tự đó bằng ký tự space (32 trong bản mã Ascii).



+ Chức năng 3:



```
##### Chuc nang so 3 #####
f3:
    # (21+21+21+1)x16
    # DCE -> ECD
    # [ (+42) In 21 -> (-42) In 21 -> (-42) In 21 -> (+42) In 1 ] x 16
    addi $s6, $zero, 0
    la $s7, String      # $s7 la dia chi cua String

loop_f3:
    beq $s6, 16, Menu
    addi $s6, $s6, 1
    # (+42) In 21
    addi $s7, $s7, 42
    jal in_21
    # (-42) In 21
    addi $s7, $s7, -42
    jal in_21
    # (-42) In 21
    addi $s7, $s7, -42
    jal in_21
    # (+42) In 1
    addi $s7, $s7, +42
    lb $t3, 0($s7)      # $t3 luu gia tri cua tung phan tu trong String
    li $v0, 11          # In tung ki tu
    addi $a0, $t3, 0
    syscall
    addi $s7, $s7, 1    # s7 += 1
    j loop_f3

#In 21
in_21:
    addi $t4, $zero, 0
    loop_2_f3:
        lb $t3, 0($s7)      # $t3 luu gia tri cua tung phan tu trong String
        li $v0, 11          # In tung ki tu
        addi $a0, $t3, 0
        syscall

        addi $s7, $s7, 1    # s7 += 1
        addi $t4, $t4, 1    # t4 += 1
        bne $t4, 21, loop_2_f3
    jr $ra
```

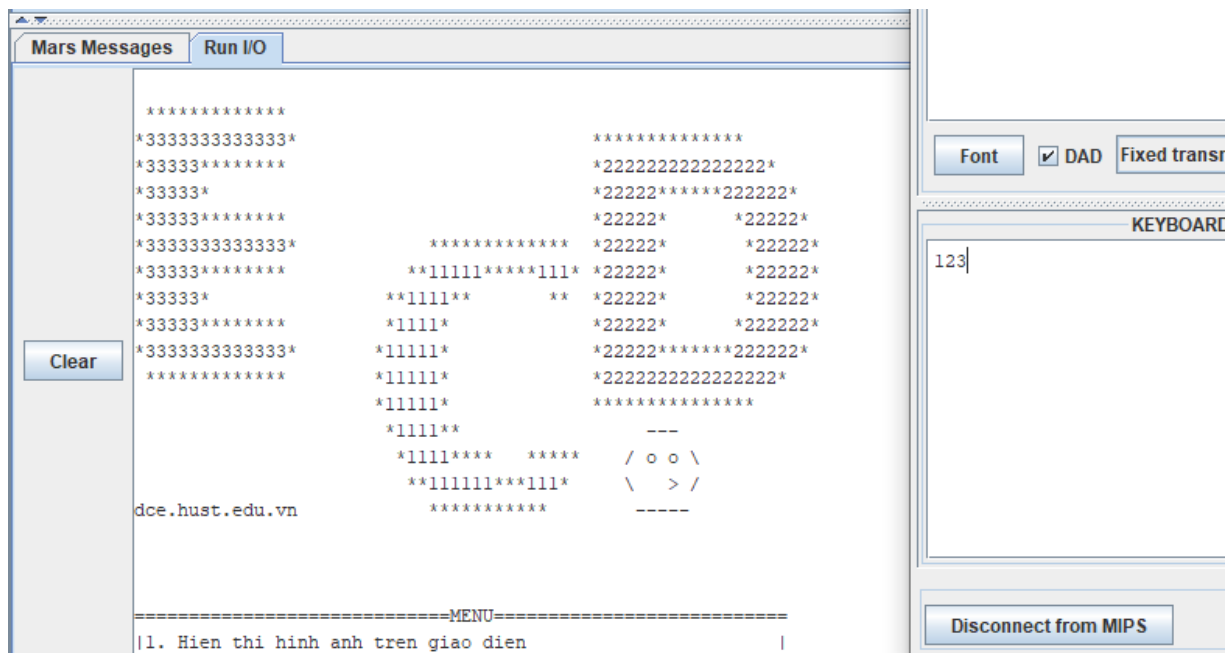
Giải thích:

Mỗi dòng có dạng  $(21+21+21+1) \times 16$  ý nghĩa là có 21 ký tự cho 1 chữ, ký tự xuống dòng ( $\backslash n$ ) ở cuối và dữ liệu này có 16 dòng.

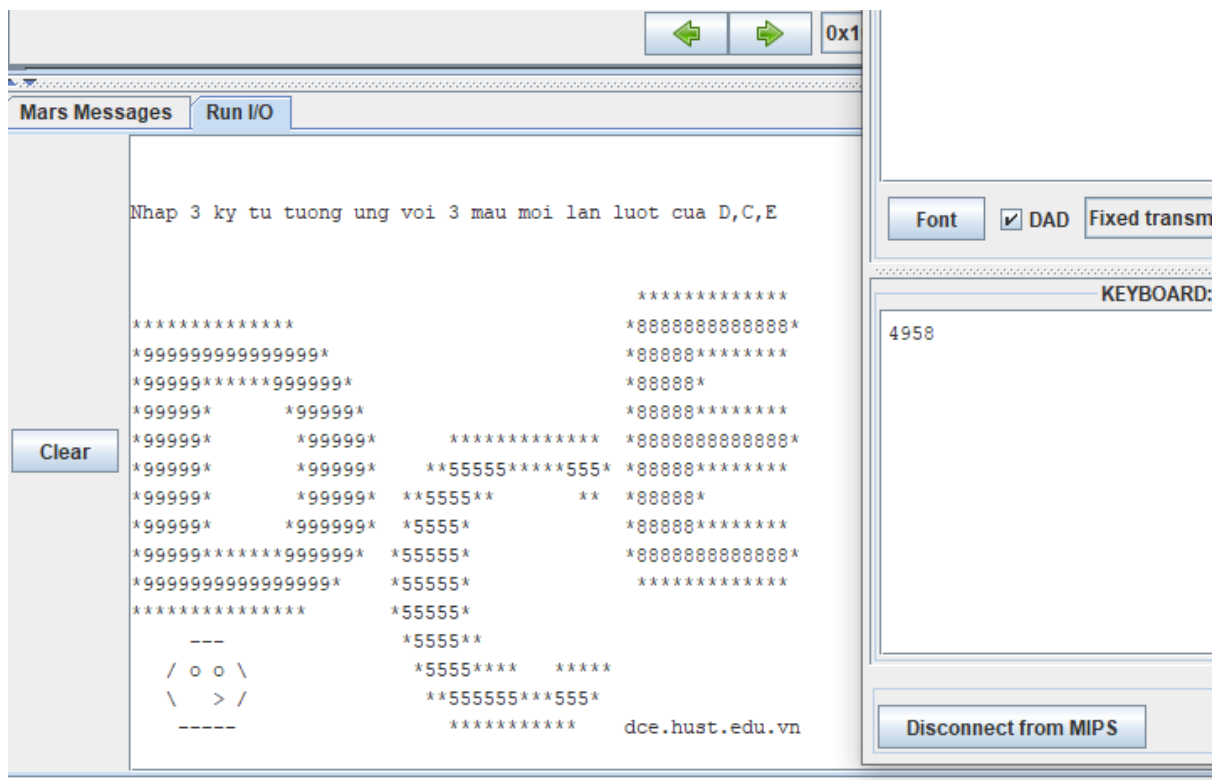
\* Lưu ý: Do ký tự E có chiều dài ngắn hơn 2 ký tự C và D, nên khi in ra sẽ bao gồm cả các ký tự space phía đằng sau E, nên sẽ tạo ra khoảng cách giữa E và C lớn hơn so với ban đầu.

Ta in từng dòng và từng cụm lần lượt theo thứ tự E -> C -> D ->  $\backslash n$ .

=> [ (+42) in 21 -> (-42) in 21 -> (-42) in 21 -> (+42) in 1 ] x 16.



+ Chức năng 4:



```
##### Chuc nang so 4 #####
f4:
    li $v0, 4
    la $a0, Mess
    syscall

    la $s7, String      # $s7 la dia chi cua String
    addi $t5, $t5, 0
loop_input_f4:
    #loop:
        nop
        WaitForKey2: lw $t1, 0($k1) # $t1 = [$k1] = KEY_READY
        beq $t1, $zero, WaitForKey2 # if $t1 == 0 then Polling
        ReadKey2: lw $t0, 0($k0) # $t0 = [$k0] = KEY_CODE
        WaitForDis2: lw $t2, 0($s1) # $t2 = [$s1] = DISPLAY_READY
        beq $t2, $zero, WaitForDis2 # if $t2 == 0 then Polling
        #Luu 4 ky tu gan nhut
        addi $s5, $s4, 0
        addi $s4, $s3, 0
        addi $s3, $s2, 0
        addi $s2, $t0, 0

        addi $t5, $t5, 1 # t5 += 1
        bne $t5, 3, loop_input_f4

    addi $t6, $t6, 0
loop_print_f4:
    addi $t7, $s4, 0 # Mau cua chu D
    jal in_21_f4
    addi $t7, $s3, 0 # Mau cua chu C
    jal in_21_f4
    addi $t7, $s2, 0 # Mau cua chu E
    jal in_21_f4
    # In \n
    lb $t3, 0($s7) # $t3 luu gia tri cua tung phan tu trong String
    li $v0, 11 # In tung ki tu
    addi $a0, $t3, 0
    syscall
    addi $s7, $s7, 1 # s7 += 1

    addi $t6, $t6, 1 # t6 += 1
    bne $t6, 16, loop_print_f4

#FREE
li $s2, 0
li $s3, 0
li $s4, 0
li $s5, 0
li $t0, 0
j Menu

in_21_f4:
    addi $t4, $zero, 0
    loop_2_f4:
        lb $t3, 0($s7) # $t3 luu gia tri cua tung phan tu trong String

        bge $t3, 58, print_f4 # Tu (0-9) trong bang ma ascii tu 48-58
        bge $t3, 48, Chuso_f4
        j print_f4
    Chuso_f4:
        addi $t3, $t7, 0 # Mau tuong ung
    print_f4:
        li $v0, 11 # In tung ki tu
        addi $a0, $t3, 0
        syscall

        addi $s7, $s7, 1 # s7 += 1
        addi $t4, $t4, 1 # t4 += 1
        bne $t4, 21, loop_2_f4
    jr $ra

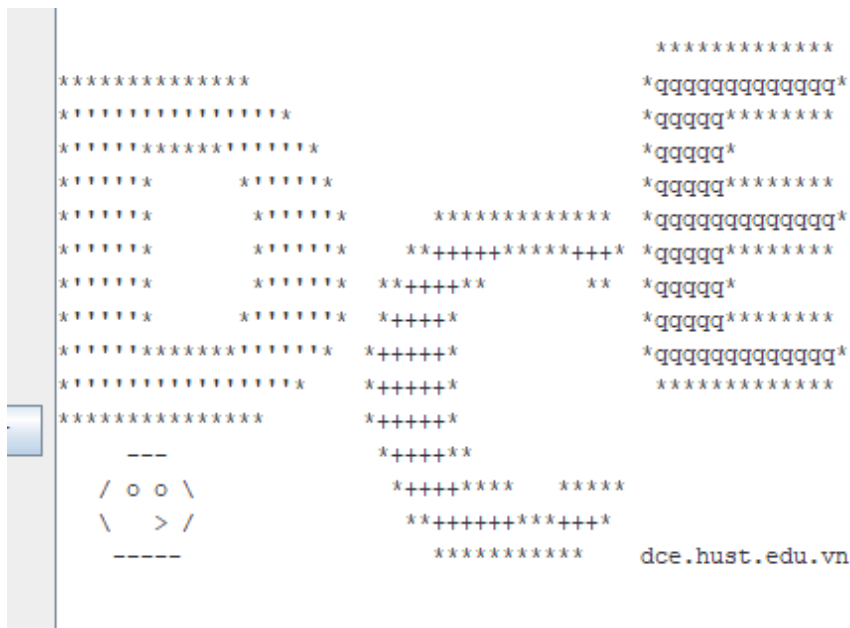
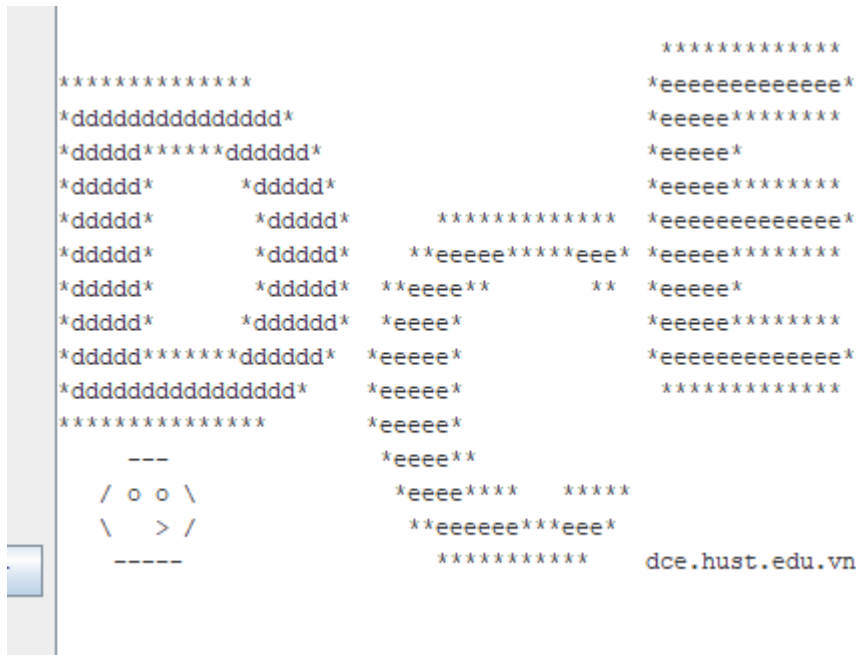
exit:
```



Là sự kết hợp của cả 3 phần đầu: In từng ký tự, chia mỗi dòng ra làm 4 phần (D,C,E,\n) thay đổi màu với mỗi phần bằng cách sử dụng 4 ký tự gần nhất.

Phần 4 ký tự gần nhất được thực hiện trong bài thực hành số 10(2), ở đây ta chỉ cần sử dụng 3 ký tự gần nhất để lấy 3 màu cho 3 hình ảnh chữ cái. Ta sẽ in ra yêu cầu nhập 3 ký tự để làm 3 màu mới của D,C,E thì 3 ký tự nhập vào KEYBOARD tiếp theo sẽ lần lượt là 3 màu của D,C,E.

\* Lưu ý: Do yêu cầu của phần 4 không bắt buộc màu mới của D,C,E phải là các chữ số, nên ta có thể nhập bất kỳ kí tự nào để làm màu mới cho D,C,E. Và ta nên tránh các ký tự đặc biệt như là ký tự "\n" hoặc các ký tự tạo cú pháp để tránh xảy ra kết quả không đúng với yêu cầu đề bài.



\* Lưu ý: Trong trường hợp yêu cầu của phần 4 bắt buộc màu phải là chữ số thì ta sẽ thêm một đoạn code tương tự phần 2 và phần 3, để check input nhập vào có phải chữ số không

```
bge $t3, 58, print_f2      # Tu (0-9) trong bang ma ascii tu 48-58
bge $t3, 48, Chuso_f2
j print_f2
```

(Code phần 2)

Check cả 3 ký tự gần nhất, nếu cả 3 ký tự đều là chữ số thì cho in ra hình ảnh, còn không phải thì sẽ không in và hiện dòng “Màu của chữ cái phải nằm trong đoạn [0-9]”

#### *Kết luận và ý nghĩa của bài thực hành:*

+ Từ bài thực hành này, ta có thể biết cách hoạt động của việc xử lý hình ảnh trong MIPS, giả sử hình ảnh không phải là các ký tự mà là các màu thì ta vẫn có thể hoàn toàn xử lý và thay đổi màu của các cá thể có trong hình ảnh bằng cách tìm và so sánh khoảng của các màu sắc, ví dụ như là màu viền là màu đen thì ta chỉ cần loại bỏ tất cả các màu khác trừ màu đen là sẽ có được bức ảnh không bao gồm màu.

+ Nếu ta biết sẵn format của hình ảnh thì sẽ dễ dàng thay đổi vị trí hoặc tạo các tổ hợp phần tử có trong hình ảnh một cách dễ dàng. Việc xác định phần tử có trong hình ảnh ta hoàn toàn có thể lập trình để nhận biết thay vì làm một cách thủ công, nhưng sẽ rất khó khăn. Trong khoảng yêu cầu của bài toán thì ta đã biết được thông số, nên việc thực hiện thay đổi bố cục không phải một vấn đề quá lớn.

+ Bài thực hành giúp ta hiểu rõ hơn về lập trình các loại dữ liệu lớn như hình ảnh và thay đổi tính chất, bố cục của chúng. Có thể áp dụng nhiều vào các chương trình như : Nhận biết vật thể, nén ảnh, ...

*Kiến nghị: Đây là một bài tập lập trình rất hay về dữ liệu và nhập liệu. Nhưng theo ý kiến chủ quan của bản thân em thì em mong đề bài sẽ có sẵn Input hoặc ít nhất là format nhập liệu Input vào, thay vì phải gõ lại từng chữ và căn chỉnh (Do slide bài thực hành theo dạng PDF không thể copy trực tiếp từ phần đề bài).*

#### *Tài liệu tham khảo:*

- Slide bài giảng của thầy Lê Bá Vui môn học thực hành kiến trúc máy tính - Trường ĐHBKHN.
- Các bài thực hành trước đã làm trong các tuần học của bản thân.
- <https://stackoverflow.com/questions/47616060/how-do-i-print-to-the-mmio-display-in-mips>
- <https://www.programminghomeworkhelp.com/memory-mapped-i-o-mips-assembly/>
- <https://inst.eecs.berkeley.edu/~cs61cl/fa08/labs/lab25.html>
- <https://github.com/BlazingRockStorm/MISP-assembly-of-HEDSPI>

*Chúng em chân thành cảm ơn thầy Lê Bá Vui đã nhiệt tình giảng dạy và truyền đạt những kiến thức trong suốt 16 tuần qua. Những kiến thức này không chỉ là nền tảng cho quá trình học tập, nghiên cứu đề tài báo cáo mà còn là hành trang quý báu giúp chúng em tự tin trong các môn học trong các kỳ sắp tới. Song không thể tránh khỏi những thiếu sót mong quý thầy cô nhận xét để bài báo cáo của em được hoàn thiện hơn.*

*Bài báo cáo cuối kỳ của chúng em đến đây là hết. Trân trọng !*

*Sinh viên thực hiện*

*Phùng Trung Kiên - 20204994*

*Vũ Thành Trung – 20200650*