

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



# FINAL PROJECT

Assembly Language and Computer Architecture

Instructor: Lê Bá Vui

Group 06:

1. Mai Hoàng Đức - 20215195
2. Lưu Yến Nhi - 20215232

Class: 143684

Hà Nội, tháng 1 năm 2024

<b>MỤC LỤC</b>	<b>2</b>
<b>GIAO ĐỀ</b>	<b>2</b>
<b>KẾT QUẢ THỰC HIỆN</b>	<b>3</b>
Bài 3: Kiểm tra tốc độ và độ chính xác khi gõ văn bản	3
Phân tích cách làm và thuật toán	3
Mã nguồn	4
Kết quả chạy mô phỏng	8
Bài 6: Hàm cấp phát bộ nhớ malloc()	10
Phân tích cách làm và Demo	10
Kết luận	15
Mã nguồn	15

#### **GIAO ĐỀ**

Student	Title
Mai Hoàng Đức - 20215195	6
Lưu Yến Nhi - 20215232	3

# KẾT QUẢ THỰC HIỆN

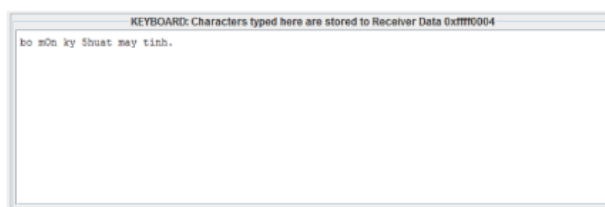
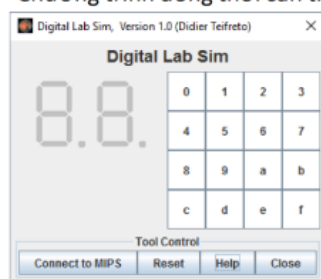
## Bài 3: Kiểm tra tốc độ và độ chính xác khi gõ văn bản

### a, Đề bài

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

Thực hiện chương trình đo tốc độ gõ bàn phím và hiển thị kết quả bằng 2 đèn led 7 đoạn. Nguyên tắc:

- Cho một đoạn văn bản mẫu, cố định sẵn trong mã nguồn. Ví dụ *"bo mon ky thuat may tinh"*
- Sử dụng bộ định thời Timer (trong bộ giả lập Digital Lab Sim) để tạo ra khoảng thời gian để đo. Đây là thời gian giữa 2 lần ngắt, chu kỳ ngắt.
- Người dùng nhập các kí tự từ bàn phím. Ví dụ nhập *"bo mOn ky 5huat may tinh"*. Chương trình cần phải đếm số kí tự đúng (trong ví dụ trên thì người dùng gõ sai chữ **O** và **5**) mà người dùng đã gõ và hiển thị lên các đèn led.
- Chương trình đồng thời cần tính được tốc độ gõ: thời gian hoàn thành và số từ trên một đơn vị thời gian.



### b, Phân tích cách làm

- Sử dụng 1 vòng lặp vô hạn.
  - Trong vòng lặp có kiểm tra giá trị tại địa chỉ KEY\_READY nếu khác 0  $\Leftrightarrow$  có kí tự nhập từ bàn phím thì nhảy đến nhãn xử lý interrupt từ bàn phím.
  - Xử lý interrupt: kiểm tra xem có ngắt từ bàn phím không, bằng cách so sánh \$t1 (lưu trạng thái KEY\_READY từ bàn phím) với 1, nếu bằng nhau tức là có ngắt từ bàn phím thì chương trình quay lại vòng lặp để đợi và xử lý interrupt tiếp theo.
- Kiểm tra loại interrupt: Bên trong vùng .ktext ta sẽ lấy giá trị bên trong thanh ghi Coproc0.cause(\$13) để kiểm tra đây là loại ngắt nào.
- Nếu loại ngắt là từ bàn phím:

- + Kiểm tra đã duyệt hết chuỗi chưa, nếu kí tự thứ i trong chuỗi là kí tự kết thúc ('\0') thì kết thúc chương trình và hiển thị số kí tự đúng ra led 7 thanh và thời gian, tốc độ gõ lên màn hình.
  - + Ngược lại, so sánh kí tự thứ i trong chuỗi với kí tự vừa nhập từ bàn phím, nếu chúng bằng nhau thì tăng biến đếm số kí tự đúng lên 1.
  - + Kiểm tra nếu kí tự vừa nhập vào là ' ' và kí tự nhập vào trước đó là khác ' ' thì tăng biến đếm số kí tự đã nhập lên 1
  - + Sau đó tăng số kí tự nhập vào trong 1s lên 1, tăng con trỏ \$a1 lên 1 để kiểm tra kí tự tiếp theo của chuỗi
- Trường hợp lệnh ngắt được thực hiện bởi bộ đếm time counter:
    - + Kiểm tra số lần tạo lệnh ngắt của timer đã đủ chưa (1s), nếu chưa đủ thì tăng biến đếm lên
    - + Nếu đã đủ thì hiển thị số ký tự đã gõ trong 1s lên Digital Lab Sim và khởi tạo lại biến đếm ký tự trong 1s, đồng thời tăng biến đếm thời gian hoàn thành nhập lên 1s
  - Hiển thị lên kết quả số kí tự nhập đúng lên led 7 thanh và thời gian, tốc độ thực hiện lên màn hình.

## B. Mã nguồn

```
#-----
#LƯU YẾN NHI
#-----
.eqv SEVENSEG_LEFT 0xFFFF0011      #Địa chỉ led 7 đoạn trái
.eqv SEVENSEG_RIGHT 0xFFFF0010     #Địa chỉ led 7 đoạn phải
.eqv IN_ADDRESS_HEXKEYBOARD 0xFFFF0012 #d/c đầu vào bàn
phím hexa
.eqv MASK_CAUSE_COUNTER 0x00000400  #Bit 10: bitmask cho ngắt
của bộ đếm
.eqv COUNTER 0xFFFF0013            #Time Counter
.eqv KEY_CODE 0xFFFF0004           #mã ASCII từ bàn phím
.eqv KEY_READY 0xFFFF0000          #=1 if has a new keycode
.data
mang_so: .byte 63, 6, 91, 79, 102, 109, 125, 7, 127, 111    #tu 0 den 9
string: .asciiz "bo mon ky thuat may tinh"
message1: .asciiz "Thoi gian hoan thanh: "
message2: .asciiz "(s) \nSo ki tu tren don vi thoi gian: "
message3: .asciiz "tu/phut\n"
```

```

#-----

.text
li    $k0, KEY_CODE
li    $k1, KEY_READY
li    $t1, COUNTER          #time counter
sb    $t1, 0($t1)
addi  $s0, $0, 0            #Dem so ky tu trong 1s
addi  $s1, $0, 0            #đếm tổng kí tự đúng
addi  $s2, $0, 1            #đếm tổng kí tự nhập vào
addi  $s3, $0, 0            #đếm số lần ngắt từ bộ đếm
addi  $s4, $0, 0            #lưu kí tự trc đó
addi  $s5, $0, 0            #đếm tgian(s)
la    $a1, string
#-----

loop:
lw    $t1, 0($k1)           #$t1 = [$k1] = KEY_READY
bne   $t1, $zero, make_Keyboard_Intr    #t1 != 0 <-> có kí tự từ bàn phím
-> nhảy đến nhãn xử lí interrupt từ bàn phím
addi  $v0, $0, 32
li    $a0, 5

syscall
b     loop
#-----

make_Keyboard_Intr:
teqi  $t1, 1    #nếu bằng 1 sẽ xác định trạng thái ngắt
b     loop      #Quay lai vong lap de cho doi su kien interrupt
tiếp theo
nop
end_Main:

#-----

.ktext 0x80000180

dis_int:li    $t1, COUNTER
sb    $zero, 0($t1)

```

```

#Kiểm tra loại interrupt
get_Caus:mfc0    $t1, $13                #$t1 = Coproc0.cause, lấy giá trị
nguyên nhân ngắt
isCount:li    $t2, MASK_CAUSE_COUNTER
and    $at, $t1,$t2
bne    $at,$t2, keyboard_Intr
#-----
#NGAT DO BO DEM COUNTER
counter_Intr:
blt    $s3, 40, continue                #biến đếm số lần ngắt đã đủ timer chưa nếu
chưa đủ, nhảy đến continue và tăng biến đếm số lần ngắt lên 1
jal    hien_thi                          #nếu đủ (1s) thì hiển thị
addi    $s3, $0, 0                       #khởi tạo lại biến đếm số lần ngắt
addi    $s5, $s5, 1                      #tăng biến đếm thời gian lên 1
j      en_int
nop
continue:
addi    $s3, $s3, 1
j      en_int
nop
keyboard_Intr:
#-----

check_Matching:
lb      $t0, 0($a1)                      #lấy kí tự thứ i trong mảng
beq     $t0, $0, end_Program              #dừng ct nếu gặp null
lb      $t1, 0($k0)                      #lấy kí tự nhập vào từ bàn phím
beq     $t1, $0, en_int
bne     $t0, $t1, check_Space             #kí tự nhập vào và kí tự từ string k khớp
-> check space
nop
addi    $s1, $s1, 1                      #còn nếu = nhau thì biến đếm kí tự đúng(s1)
tăng lên 1
check_Space:
bne     $t1, '', end_Process              #kí tự nhập vào != '' và trc nó là '' thì
tăng biến đếm số kí tự nhập vào lên
nop
beq     $s4, '', end_Process              #s4 kí tự trc đó từ bàn phím
nop

```

```

addi $s2, $s2, 1
end_Process:
addi $s0, $s0, 1           #Tang so ky tu trong 1s len 1
addi $s4, $t1, 0           #Cap nhat lai thanh ghi chua ky tu nhap vao
ban phim truooc do
addi $a1, $a1, 1           #Tang con tro len 1 <=> string+i, ktra ki tự tiếp
theo
#-----
en_int:
li    $t1, COUNTER
sb    $t1, 0($t1)
mtc0 $zero, $13
next_pc: mfc0    $at, $14
        addi $at, $at, 4
        mtc0 $at, $14
return: eret

#-----
hien_thi:
addi $sp, $sp, -4
sw    $ra, ($sp)
addi $t0, $0, 10
div   $s0, $t0
mflo  $v1           #số hàng chục
mfhi  $v0           #số hàng đơn vị
la    $a0, mang_so
add   $a0, $a0, $v1
lb    $a0, 0($a0)    #Set value for segments
jal   SHOW_7SEG_LEFT
la    $a0, mang_so
add   $a0, $a0, $v0
lb    $a0, 0($a0)    #Set value for segments
jal   SHOW_7SEG_RIGHT
addi  $s0, $0, 0     #Sau khi chieu ra man hinh thi khoi tao lai bien
dem
lw    $ra, ($sp)
addi  $sp, $sp, 4
jr    $ra
SHOW_7SEG_LEFT:

```

```

li    $t0, SEVENSEG_LEFT      #Assign port's address
sb    $a0, 0($t0)             #Assign new value
jr    $ra
SHOW_7SEG_RIGHT:
li    $t0, SEVENSEG_RIGHT     #Assign port's address
sb    $a0, 0($t0)             #Assign new value
jr    $ra
nop

```

#-----

end\_Program:

```

addi  $v0, $0, 4
la    $a0, message1
syscall
addi  $v0, $0, 1
addi  $a0, $s5, 0
syscall
addi  $v0, $0, 4
la    $a0, message2
syscall

```

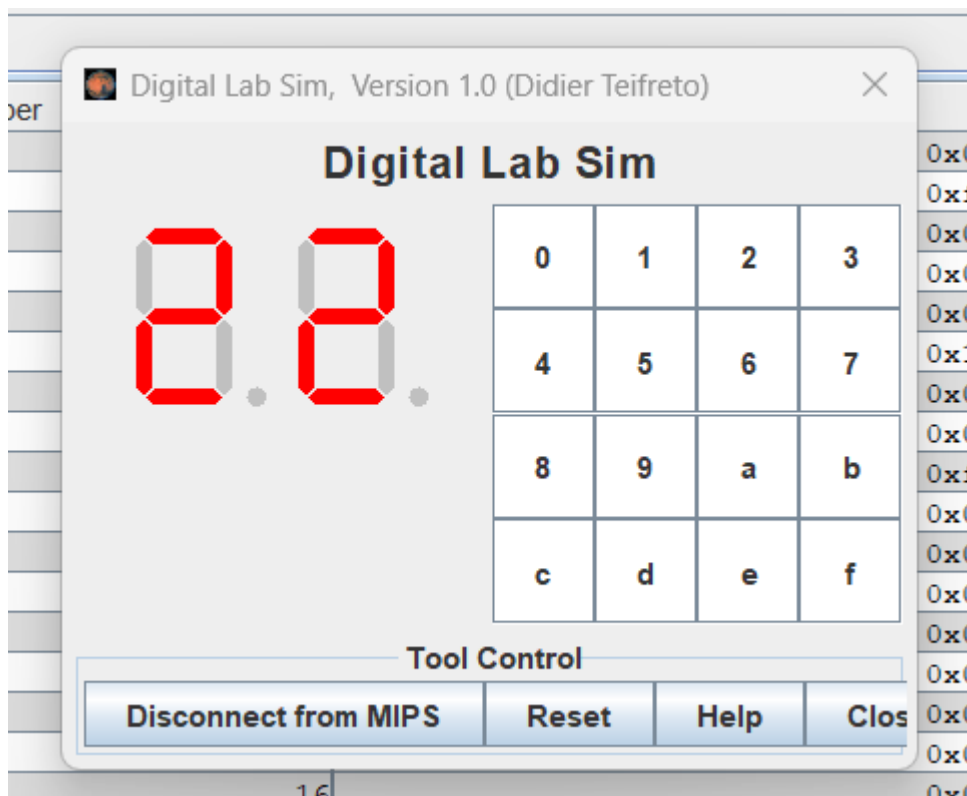
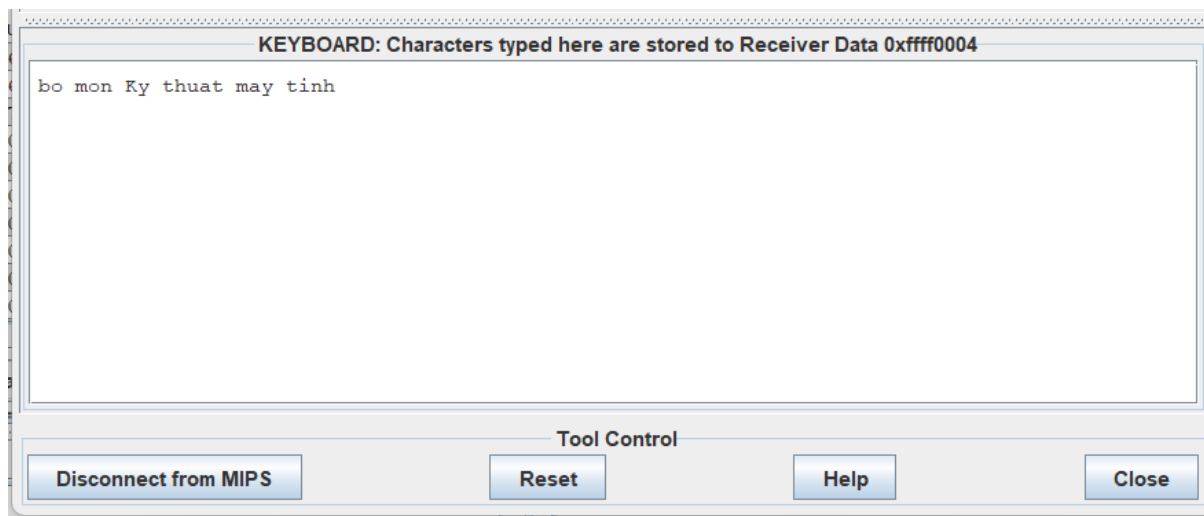
```

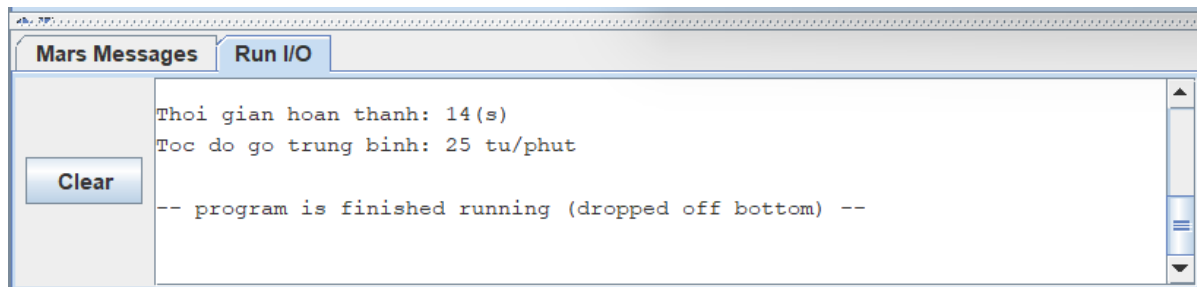
addi  $v0, $0, 1
addi  $a0, $0, 60
mult  $s2, $a0
mflo  $s2
div   $s2, $s5
mflo  $a0
syscall
addi  $v0, $0, 4
la    $a0, message3
syscall
addi  $s0, $s1, 0
jal   hien_thi

```



### C. Kết quả chạy mô phỏng





## Bài 6: Hàm cấp phát bộ nhớ malloc():

### A) Đề bài

Chương trình cho bên dưới là hàm malloc(), kèm theo đó là ví dụ minh họa, được viết bằng hợp ngữ MIPS, để cấp phát bộ nhớ cho một biến con trỏ nào đó. Hãy đọc chương trình và hiểu rõ nguyên tắc cấp phát bộ nhớ động. Trên cơ sở đó, hãy hoàn thiện chương trình như sau: (Lưu ý, ngoài viết các hàm đó, cần viết thêm một số ví dụ minh họa để thấy việc sử dụng hàm đó như thế nào)

- 1) Việc cấp phát bộ nhớ kiểu word/mảng kiểu word có 1 lỗi, đó là chưa bảo đảm bảo quy tắc địa chỉ của kiểu word phải chia hết cho 4. Hãy khắc phục lỗi này.
- 2) Viết hàm lấy giá trị của biến con trỏ.
- 3) Viết hàm lấy địa chỉ biến con trỏ.
- 4) Viết hàm thực hiện copy 2 con trỏ xâu kí tự.
- 5) Viết hàm giải phóng bộ nhớ đã cấp phát cho các biến con trỏ
- 6) Viết hàm tính toàn bộ lượng bộ nhớ đã cấp phát.
- 7) Hãy viết hàm malloc2 để cấp phát cho mảng 2 chiều kiểu .word với tham số vào gồm: a. Địa chỉ đầu của mảng b. Số dòng c. Số cột
- 8) Tiếp theo câu 7, hãy viết 2 hàm `getArray[i][j]` và `setArray[i][j]` để lấy/thiết lập giá trị cho phần tử ở dòng i cột j của mảng.

### B) Phân tích cách làm và Demo

Dựa trên chương trình malloc mẫu, em đã xây dựng chương trình hoàn chỉnh bằng MIPS Assembly thực hiện các chức năng mà yêu cầu đề bài đã đặt ra. Ý tưởng, cách thức xây dựng và thực hiện từng chức năng của chương trình cấp phát bộ nhớ động mô phỏng hàm malloc():

- Cấp phát bộ nhớ động cho các biến trong chương trình
- Thực hiện cấp phát bộ nhớ động cho các biến của chương trình, các biến ở đây do là biến con trỏ nên sẽ có giá trị là 4 bytes (chứa địa chỉ nó trỏ tới trong vùng nhớ .kdata – có thể coi vùng nhớ này tương đồng với vùng nhớ heap thực hiện cấp phát bộ nhớ cho các biến kiểu dữ liệu tham chiếu).

- Địa chỉ vùng nhớ trong .kdata tương ứng sẽ được gán giá trị phù hợp khi thực hiện khởi tạo giá trị, như trong đoạn code dưới đây mô tả cách một mảng số nguyên kiểu Word thực hiện gán từng giá trị vào vùng nhớ đã được khởi tạo từ trước đó. (Giá trị từ 0x90000010 đến 0x90000020 với mảng kiểu Word chứa 5 phần tử, mỗi phần tử 4 bytes)

```
#-----
# Khoi tao gia tri WordPtr
#-----

init_WordPtr:
    la      $t1,    Word
    addi    $t0, $t8, -4
    addi    $t2, $zero, 0

loop_init_WordPtr:
    beq     $t2, $a1, init_WordPtr_back
    addi    $t0, $t0, 4
    lw      $t3, 0($t1)
    sw      $t3, 0($t0)
    addi    $t1, $t1, 4
    addi    $t2, $t2, 1
    j       loop_init_WordPtr

init_WordPtr_back:
    jr      $ra
```

- Với các biến con trỏ kiểu word/ mảng word, ta sẽ phải đảm bảo yêu cầu rằng các biến trên sẽ trỏ đến địa chỉ đầu tiên luôn chia hết cho 4, thỏa mãn tính đúng đắn của dữ liệu. Do đó ta sẽ kiểm tra giá trị tiếp theo trong vùng nhớ .kdata tại địa chỉ 0x90000000 chứa địa chỉ còn trống tiếp theo trong vùng nhớ, sau đó nếu nó không chia hết cho 4 thì cộng phần bù vào từ thanh ghi hi sau khi chia dư cho 4 để thực hiện.

- Hiện thị giá trị của địa chỉ con trỏ, địa chỉ con trỏ trỏ đến, giá trị lưu trữ trong địa chỉ mà con trỏ trỏ đến
- Thực hiện lời gọi hệ thống với thanh ghi trả về \$v0, load các biến, địa chỉ biến trỏ đến hay giá trị tại địa chỉ mà biến trỏ đến tại thanh ghi \$a0 rồi thực hiện syscall
  - Viết hàm thực hiện copy 2 con trỏ xâu ký tự
- Tương tự như trong ngôn ngữ lập trình C, ta sẽ lấy giá trị của địa chỉ xâu mà biến đang trỏ đến (ký tự đầu của xâu) sau đó, cộng 1 sau mỗi lần thực hiện

vòng lặp ở cả xâu ký tự đã có và xâu ký tự thực hiện copy để copy xâu sang xâu mới.

- Liên quan đến cấp phát mảng động 2 chiều
- Ý tưởng: với mảng 2 chiều cỡ  $(m \times n)$  thì ta sẽ thực hiện biến đổi mảng này về mảng một chiều để có thể lưu trữ giá trị của biến. Ví dụ  $a[1][1]$  trong mảng 2 chiều cỡ  $(2 \times 3)$  sẽ được cấp phát là phần tử có địa chỉ lưu trữ giá trị là  $(\text{địa chỉ đầu tiên của mảng} + (1 \times 3 + 1)) = (\text{địa chỉ đầu tiên của mảng} + 4)$ , tương ứng là phần tử thứ 5 của mảng. Việc thực hiện cấp phát mảng động 2 chiều được nhả `malloc2` trong chương trình thực thi
- Khi duyệt mảng để lấy giá trị hay thay đổi giá trị thì ta sẽ duyệt xem chỉ số nhập vào có vượt quá giới hạn của mảng là  $m \times n$  không, nếu vượt quá thì báo lỗi, ngược lại chương trình thực thi chức năng bình thường.

### → Kết quả

- Câu 1:
  - Trước khi cấp phát mảng word, địa chỉ là `0x9000000d` không chia hết cho 4
  - Sau khi cấp phát, địa chỉ là `0x90000010` chia hết cho 4

- Câu 2, 3:

Địa chỉ của các biến con trỏ:

`&CharPtr = 0x10010000`

`&BytePtr = 0x10010004`

`&WordPtr = 0x10010008`

Địa chỉ mà các biến con trỏ trỏ tới:

`CharPtr = 0x90000004`

`BytePtr = 0x90000007`

`WordPtr = 0x90000010`

Khu con trỏ:

`*CharPtr = M`

`*BytePtr = 15`

`*WordPtr = 21`

- Câu 4:

`Char = "MaiHoangDuc-20215195"` đã được sao chép

Hàm in của địa chỉ được sao chép tới:

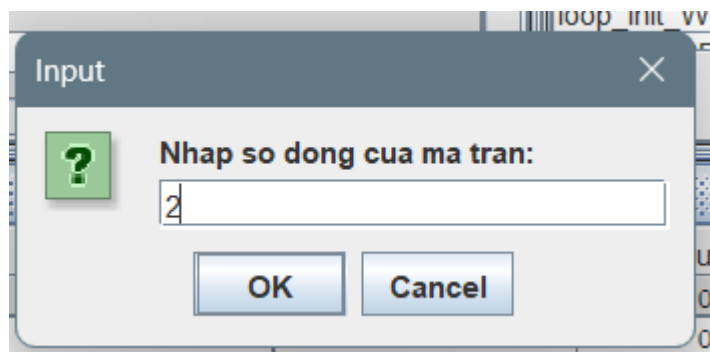
```
# Printing the copied string in newCharPtr
li    $v0, 4                # syscall for print_str
la    $a0, newCharPtr      # Load the address of newCharPtr
syscall                    # Print the string in newCharPtr
j     totalAllocatedCapacity # Jump to the end of the program
```

Ở đây, em sử dụng newCharPtr chứ không phải Char

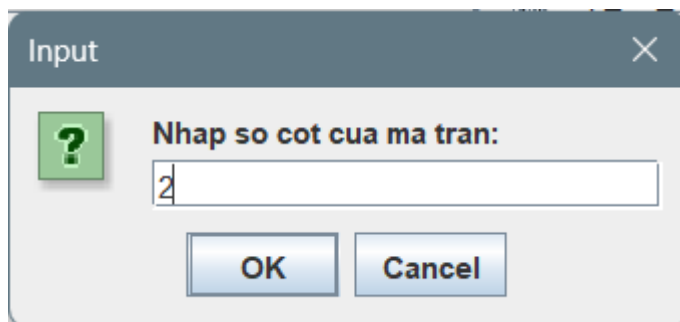
- Câu 5,6:

```
Tong bo luong bo nho da cap phat cho cac bien dong = 65 byte(s)
Da giai phong bo nho da cap phat!
```

- Câu 7, 8: mảng 2 chiều cỡ 2 x 2 với các phần tử ({1, 2}, {3, 4})

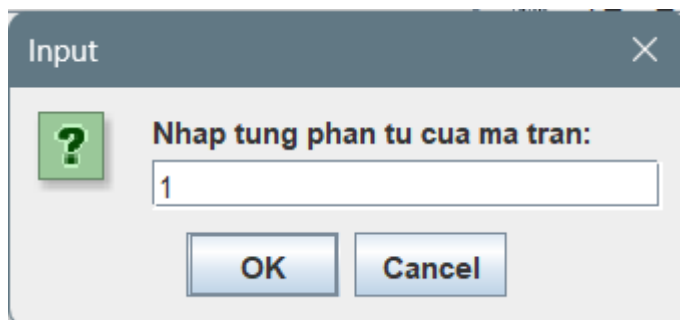


Input dialog box titled "Nhập số dòng của ma trận:" (Enter the number of rows of the matrix:). The input field contains the value "2". There are "OK" and "Cancel" buttons at the bottom.



Input dialog box titled "Nhập số cột của ma trận:" (Enter the number of columns of the matrix:). The input field contains the value "2". There are "OK" and "Cancel" buttons at the bottom.

Nhập phần tử:



Input dialog box titled "Nhập từng phần tử của ma trận:" (Enter each element of the matrix:). The input field contains the value "1". There are "OK" and "Cancel" buttons at the bottom.

Input

?

Nhap tung phan tu cua ma tran:

2

OK Cancel

Input

?

Nhap tung phan tu cua ma tran:

3

OK Cancel

Input

?

Nhap tung phan tu cua ma tran:

4

OK Cancel

GetArray:

Input

?

GetArray: Nhap i cua A[i][j]

1

OK Cancel

Input

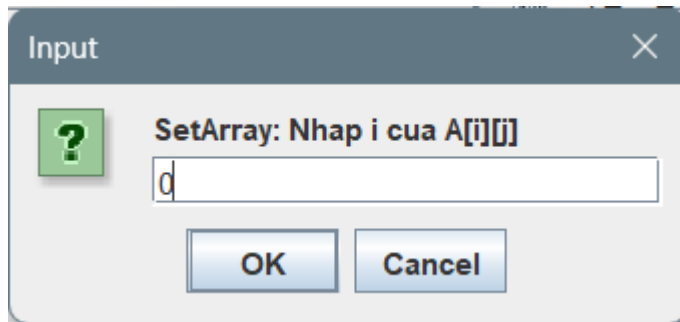
?

GetArray: Nhap j cua A[i][j]

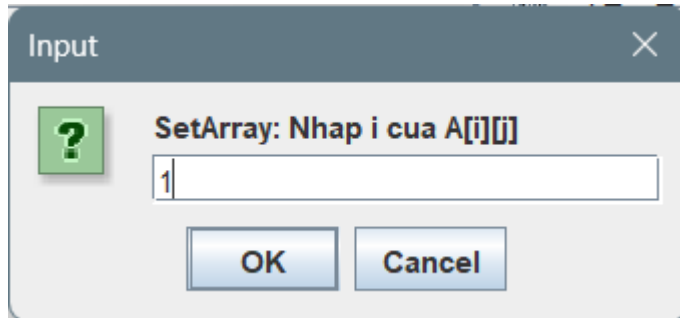
1

OK Cancel

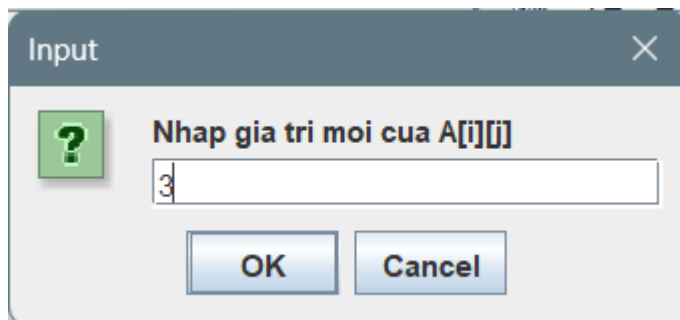
SetArray:



Input dialog box titled "SetArray: Nhap i cua A[i][j]". The input field contains the value "0". There are "OK" and "Cancel" buttons at the bottom.



Input dialog box titled "SetArray: Nhap i cua A[i][j]". The input field contains the value "1". There are "OK" and "Cancel" buttons at the bottom.



Input dialog box titled "Nhap gia tri moi cua A[i][j]". The input field contains the value "3". There are "OK" and "Cancel" buttons at the bottom.

Output:

```
GetArray[i][j] = 4  
Gia tri cua A[i][j] tro thanh: 3
```

### C) Kết luận:

- Dựa vào chương trình mà em hiểu được cách thức hoạt động của hàm malloc() trong Assembly.
- Nếu em được cho phép thêm thời gian, em sẽ hoàn thiện chương trình hơn với menu.

### D) Mã nguồn:

#Subject 6

.data

```
CharPtr: .word    0 # Bien con tro, tro toi kieu asciiz  
BytePtr: .word    0 # Bien con tro, tro toi kieu Byte  
WordPtr: .word    0 # Bien con tro, tro toi mang kieu Word  
Word2DPtr: .word 0 # Bien con tro, tro toi mang 2 chieu kieu Word  
newCharPtr: .word 0 # Bien con tro, toi kieu asciiz
```

```

Char: .asciiiz "MaiHoangDuc-20215195"
Byte: .byte 15, 16, 17, 18, 19, 20
Word: .word 21, 22, 23, 24, 25
m: .word 0
n: .word 0
gottenValue: .word
totalAllocatedMemory: .word
textCharPtr: .asciiiz "&CharPtr = "
textBytePtr: .asciiiz "&BytePtr = "
textWordPtr: .asciiiz "&WordPtr = "
textCharDePtr: .asciiiz    "*CharPtr = "
textByteDePtr: .asciiiz    "*BytePtr = "
textWordDePtr: .asciiiz    "*WordPtr = "
textCharValueOfPtr: .asciiiz "CharPtr = "
textByteValueOfPtr: .asciiiz "BytePtr = "
textWordValueOfPtr: .asciiiz "WordPtr = "
message1:  .asciiiz "Dia chi cua cac bien con tro:\n"
message2:  .asciiiz "Dia chi ma cac bien con tro tro toi:\n"
message3:  .asciiiz "Khu con tro:\n"
message4:  .asciiiz "\nTong bo luong bo nho da cap phat cho cac bien
dong = "
message5:  .asciiiz " byte(s)"
message6:  .asciiiz "Nhap so dong cua ma tran: "
message7:  .asciiiz "Nhap so cot cua ma tran: "
message8:  .asciiiz "Nhap tung phan tu cua ma tran: "
message9_1: .asciiiz "GetArray: Nhap i cua A[i][j]"
message9_2: .asciiiz "GetArray: Nhap j cua A[i][j]"
message10: .asciiiz "GetArray[i][j] = "
message11_1: .asciiiz "SetArray: Nhap i cua A[i][j]"
message11_2: .asciiiz "SetArray: Nhap i cua A[i][j]"
message12: .asciiiz "Nhap gia tri moi cua A[i][j]"
message13:  .asciiiz "Gia tri cua A[i][j] tro thanh: "
message_err: .asciiiz "Chi so phan tu khong hop le!"
message_end: .asciiiz "Da giai phong bo nho da cap phat!"
newLine:    .asciiiz "\n"
.kdata
Sys_TheTopOfFree: .word 1
Sys_MyFreeSpace:

.text
#Tong bo luong bo nho da cap phat cho cac bien dong
addi $s0, $zero, 0

```



```

        #Khoi tao vung nho cap phat dong
        jal SysInitMem
#-----
# Cap phat cho bien con tro, gom 3 phan tu,moi phan tu 1 byte
#-----
        la    $a0, CharPtr
        addi $a1, $zero, 3
        addi $a2, $zero, 1
        jal   malloc
        jal   init_CharPtr
#-----
# Cap phat cho bien con tro, gom 6 phan tu, moi phan tu 1 byte
#-----
        la    $a0, BytePtr
        addi $a1, $zero, 6
        addi $a2, $zero, 1
        jal   malloc
        jal   init_BytePtr
        nop
#-----
# Cap phat cho bien con tro, gom 5 phan tu, moi phan tu 4 byte
#-----
        la    $a0, WordPtr
        addi $a1, $zero, 5
        addi $a2, $zero, 4
        jal   malloc_WordPtr
        jal   init_WordPtr

        #lock: j lock
        #nop
        j     getPointerAddress

#-----
# Khoi tao gia tri CharPtr
#-----
init_CharPtr:
        la    $t1, Char
        addi $t0, $t8, -1
        addi $t2, $zero, 0
        # $t0 = $t8 - 1
        # $t2: dem so
        phan tu
loop_init_CharPtr:
        beq   $t2, $a1, init_CharPtr_back

```

```

    addi $t0, $t0, 1
    lb   $t3, 0($t1)
    sb   $t3, 0($t0)
    addi $t1, $t1, 1
    addi $t2, $t2, 1
    j     loop_init_CharPtr

```

init\_CharPtr\_back:

```

    jr    $ra

```

#-----

# Khoi tao gia tri BytePtr

#-----

init\_BytePtr:

```

    la    $t1, Byte
    addi $t0, $t8, -1
    addi $t2, $zero, 0

```

loop\_init\_BytePtr:

```

    beq   $t2, $a1, init_BytePtr_back
    addi  $t0, $t0, 1
    lb    $t3, 0($t1)
    sb    $t3, 0($t0)
    addi  $t1, $t1, 1
    addi  $t2, $t2, 1
    j     loop_init_BytePtr

```

init\_BytePtr\_back:

```

    jr    $ra

```

#-----

# Khoi tao gia tri WordPtr

#-----

init\_WordPtr:

```

    la    $t1, Word
    addi $t0, $t8, -4
    addi $t2, $zero, 0

```

loop\_init\_WordPtr:

```

    beq   $t2, $a1, init_WordPtr_back
    addi  $t0, $t0, 4
    lw    $t3, 0($t1)
    sw    $t3, 0($t0)
    addi  $t1, $t1, 4

```

```

        addi $t2, $t2, 1
        j     loop_init_WordPtr

init_WordPtr_back:
        jr     $ra

#-----
SysInitMem:
        la     $t9, Sys_TheTopOfFree #Lay con tro chua dau tien con trong,
khoi tao
        la     $t7, Sys_MyFreeSpace  #Lay dia chi dau tien con trong, khoi tao
        sw     $t7, 0($t9)            #Luu lai
        jr     $ra

#-----

malloc:
        la     $t9, Sys_TheTopOfFree #
        lw     $t8, 0($t9)            #Lay dia chi dau tien con trong
        sw     $t8, 0($a0)            #Cat dia chi do vao bien con tro
        addi $v0, $t8, 0              # Dong thoi la ket qua tra ve cua ham
        mul    $t7, $a1, $a2          #Tinh kich thuoc cua mang can cap
phat
        mflo   $t2                    #Cap nhat tong bo luong bo nho da
cap phat cho cac bien dong
        add    $s0, $s0, $t2
        add    $t6, $t8, $t7          #Tinh dia chi dau tien con trong
        sw     $t6, 0($t9)            #Luu tro lai dia chi dau tien do vao bien
Sys_TheTopOfFree
        jr     $ra

#-----

malloc2:
        jal    malloc_WordPtr
init2DArr:
        #la    $t1, Word2DPtr
        #lw     $t1, 0($t1)
        addi $t0, $t8, -4
        addi $t2, $zero, 0
loop_init_Word2DPtr:
        beq    $t2, $a3, init_Word2DPtr_back
        addi   $t0, $t0, 4

```

```

li    $v0, 51
la    $a0, message8
syscall

```

```

sw    $a0, 0($t0)
addi  $t2, $t2, 1
j     loop_init_Word2DPtr

```

```

init_Word2DPtr_back:
    j     getArray

```

#-----

#CAU 1: Dia chi kieu word/ mang word phai chia het cho 4 (su dung  
malloc\_WordPtr)

malloc\_WordPtr:

```

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

```

```

addi  $t5, $zero, 0x4
div   $t8, $t5
mfhi  $t4

```

```

beq   $t4, $zero, afterCheckingDivisionBy4    #Chia 4 lay du de

```

kiem tra

```

beq   $t4, 0xffffffff, missing3              #=-3 => thieu 3
beq   $t4, 0xffffffe, missing2                #=-2 => thieu 2
beq   $t4, 0xffffffff, missing1              #=-1 => thieu 1

```

missing3:

```

addi  $t3, $zero, 3
add   $t8, $t8, $t3
j     afterCheckingDivisionBy4

```

missing2:

```

addi  $t3, $zero, 2
add   $t8, $t8, $t3
j     afterCheckingDivisionBy4

```

missing1:

```

addi  $t3, $zero, 1
add   $t8, $t8, $t3
j     afterCheckingDivisionBy4

```

afterCheckingDivisionBy4:

```

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

```

addi \$v0, \$t8, 0	# Dong thoi la ket qua tra ve cua ham
mul \$t7, \$a1,\$a2	#Tinh kich thuoc cua mang can cap
phat	
mflo \$t2	#Cap nhat tong bo luong bo nho da
cap phat cho cac bien dong	
add \$s0, \$s0, \$t2	
add \$t6, \$t8, \$t7	#Tinh dia chi dau tien con trong
sw \$t6, 0(\$t9)	#Luu tro lai dia chi dau tien do vao bien
Sys_TopOfFree	
jr \$ra	

#-----

#CAU 3: Dia chi con tro

getPointerAddress:

```
li    $v0, 4
la    $a0, message1
syscall
```

#&CharPtr

```
li    $v0, 4
la    $a0, textCharPtr
syscall
li    $v0, 34
la    $a0, CharPtr
syscall
li    $v0, 4
la    $a0, newLine
syscall
```

#&BytePtr

```
li    $v0, 4
la    $a0, textBytePtr
syscall
li    $v0, 34
la    $a0, BytePtr
syscall
li    $v0, 4
la    $a0, newLine
syscall
```

#&WordPtr

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

```

syscall
li    $v0, 34
la    $a0, WordPtr
syscall
li    $v0, 4
la    $a0, newLine
syscall

```

```

nop

```

#-----

```

#CAU 3: Dia chi con tro tro den
getAddressPointedbyThePointer:

```

```

li    $v0, 4
la    $a0, newLine
syscall

```

```

li    $v0, 4
la    $a0, message2
syscall

```

```

#CharPtr
li    $v0, 4
la    $a0, textCharValueOfPtr
syscall
li    $v0, 34
la    $t0, CharPtr
lw    $a0, 0($t0)
syscall
li    $v0, 4
la    $a0, newLine
syscall

```

```

#BytePtr
li    $v0, 4
la    $a0, textByteValueOfPtr
syscall
li    $v0, 34
la    $t0, BytePtr
lw    $a0, 0($t0)
syscall
li    $v0, 4

```

```
la    $a0, newLine
syscall
```

```
#WordPtr
li    $v0, 4
la    $a0, textWordValueOfPtr
syscall
li    $v0, 34
la    $t0, WordPtr
lw    $a0, 0($t0)
syscall
li    $v0, 4
la    $a0, newLine
syscall
```

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

```
#CAU 2: Khu tham chieu
```

```
getValuebyDereferenceThePointer:
```

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

```
li    $v0, 4
la    $a0, message3
syscall
```

```
/*CharPtr
li    $v0, 4
la    $a0, textCharDePtr
syscall
```

```
la    $t1, CharPtr
lw    $t2, 0($t1)
lb    $t1, 0($t2)
la    $t3, gottenValue
sw    $t1, 0($t3)
```

```
li    $v0, 4
la    $a0, gottenValue
syscall
li    $v0, 4
la    $a0, newLine
```

syscall

##\*BytePtr

li \$v0, 4

la \$a0, textByteDePtr

syscall

la \$t1, BytePtr

lw \$t2, 0(\$t1)

lb \$t1, 0(\$t2)

la \$t3, gottenValue

sw \$t1, 0(\$t3)

li \$v0, 1

la \$t0, gottenValue

lw \$a0, 0(\$t0)

syscall

li \$v0, 4

la \$a0, newLine

syscall

##\*WordPtr

li \$v0, 4

la \$a0, textWordDePtr

syscall

la \$t1, WordPtr

lw \$t2, 0(\$t1)

lw \$t1, 0(\$t2)

la \$t3, gottenValue

sw \$t1, 0(\$t3)

li \$v0, 1

la \$t0, gottenValue

lw \$a0, 0(\$t0)

syscall

li \$v0, 4

la \$a0, newLine

syscall

#-----

#CAU 6: Malloc2 cap phat dong mang hai chieu

li \$v0, 51

la \$a0, message6



```

syscall
addi $t1, $a0, 0

li    $v0, 51
la    $a0, message7
syscall
addi $t2, $a0, 0

```

#-----

# Cap phat cho bien con tro, gom 3 phan tu,moi phan tu 1 byte

#-----

```

la    $a0, Word2DPtr
mul   $a3, $t1, $t2
addi  $a1, $a3, 0
addi  $a2, $zero, 4

```

```

la    $t0, m
sw    $t1, 0($t0)

```

```

la    $t0, n
sw    $t2, 0($t0)

```

```

jal   malloc2

```

#-----

-----

#CAU 7: GET\_ARRAY

getArray:

```

li    $v0, 51,
la    $a0, message9_1
syscall
addi  $t1, $a0, 0

```

```

li    $v0, 51
la    $a0, message9_2
syscall
addi  $t2, $a0, 0

```

```

la    $a3, Word2DPtr
lw    $a3, 0($a3)

```

```

la    $a1, m
lw    $a1, 0($a1)

```

```

la    $a2, n
lw    $a2, 0($a2)

slt   $t5, $t1, $a1
slt   $t6, $t2, $a2
beq   $t5, $zero, outOfRange
beq   $t6, $zero, outOfRange

```

```

mul   $t3, $t1, $a2
add   $t3, $t3, $t2
mul   $t3, $t3, 4

```

```

add   $a3, $a3, $t3

```

```

li    $v0, 4
la    $a0, newLine
syscall

```

```

li    $v0, 4
la    $a0, message10
syscall

```

```

li    $v0, 1
lw    $a0, 0($a3)
syscall

```

```

j      setArray

```

outOfRange:

```

li    $v0, 55
la    $a0, message_err
li    $a1, 0
syscall
j      end

```

#CAU 7: SET ARRAY-----

setArray:

```

li    $v0, 4
la    $a0, newLine
syscall

```

```

li    $v0, 51

```

```

la $a0, message11_1
syscall
addi $t1, $a0, 0

li $v0, 51
la $a0, message11_2
syscall
addi $t2, $a0, 0

la $a3, Word2DPtr
lw $a3, 0($a3)

la $a1, m
lw $a1, 0($a1)

la $a2, n
lw $a2, 0($a2)

slt $t5, $t1, $a1
slt $t6, $t2, $a2
beq $t5, $zero, outOfRange
beq $t6, $zero, outOfRange

mul $t3, $t1, $a2
add $t3, $t3, $t2
mul $t3, $t3, 4

add $a3, $a3, $t3

li $v0, 51
la $a0, message12
syscall

sw $a0, 0($a3)

li $v0, 4
la $a0, message13
syscall
li $v0, 1
lw $a0, 0($a3)
syscall

```

```

li $v0, 4
la $a0, newLine
syscall
li $v0, 4
la $a0, newLine
syscall
j copyCharPtr
#-----
# CAU 4: Copy Char to newCharPtr
copyCharPtr:
    la $a0, newCharPtr    # Load the address of newCharPtr
    addi $a1, $zero, 20    # Allocate memory for a string of length 21
    addi $a2, $zero, 1     # Size of each element in bytes
    jal malloc             # Allocate memory for newCharPtr
    jal init_newCharPtr    # Initialize newCharPtr with the copied string

    # Printing the copied string in newCharPtr
    li $v0, 4             # syscall for print_str
    la $a0, newCharPtr    # Load the address of newCharPtr
    syscall               # Print the string in newCharPtr
    j totalAllocatedCapacity # Jump to the end of the program

# Procedure to initialize newCharPtr with the string from Char
init_newCharPtr:
    la $t1, Char          # Load the address of the string
    "MaiHoangDuc20215195"
    la $t2, newCharPtr    # Load the address of newCharPtr
    li $t0, 20            # Number of characters to copy (including null terminator)
copy_loop:
    lb $t3, 0($t1)        # Load a character from "MaiHoangDuc20215195"
    sb $t3, 0($t2)        # Store the character into newCharPtr
    addi $t1, $t1, 1      # Move to the next character in
    "MaiHoangDuc20215195"
    addi $t2, $t2, 1      # Move to the next character in newCharPtr
    sub $t0, $t0, 1       # Decrement character count
    bnez $t0, copy_loop   # Continue copying until all characters are copied
    li $t3, 0             # Load the null terminator
    sb $t3, 0($t2)        # Store the null terminator into newCharPtr
    jr $ra                # Return from the procedure
#-----
# Hàm giải phóng bộ nhớ đã cấp phát
freeMemory:

```

```
lw $t0, 0($a0) # Load địa chỉ bắt đầu của bộ nhớ cần giải phóng từ tham số truyền vào
```

```
li $t1, 0 # Khởi tạo giá trị 0 (null) cho bộ nhớ được giải phóng
```

```
free_loop:
```

```
lb $t2, 0($t0) # Load byte từ vùng nhớ cần giải phóng
```

```
beq $t2, $zero, end_free # Nếu gặp ký tự kết thúc chuỗi (0/null), thoát khỏi vòng lặp
```

```
sb $t1, 0($t0) # Ghi giá trị null vào vùng nhớ được giải phóng
```

```
addi $t0, $t0, 1 # Chuyển sang vùng nhớ tiếp theo
```

```
j free_loop # Lặp lại quá trình giải phóng cho tất cả các byte
```

```
end_free:
```

```
jr $ra # Trả về từ hàm
```

#CAU 5: Tong bo luong bo nho da cap phat cho cac bien dong

```
totalAllocatedCapacity:
```

```
li $v0, 4
```

```
la $a0, newLine
```

```
syscall
```

```
li $v0, 4
```

```
la $a0, message4
```

```
syscall
```

```
la $t0, totalAllocatedMemory
```

```
sw $s0, 0($t0)
```

```
li $v0, 1
```

```
la $t0, totalAllocatedMemory
```

```
lw $a0, 0($t0)
```

```
syscall
```

```
li $v0, 4
```

```
la $a0, message5
```

```
syscall
```

```
li $v0, 4
```

```
la $a0, newLine
```

```
syscall
```

```
la $a0, CharPtr    # Đưa địa chỉ của CharPtr vào $a0 để giải phóng  
jal freeMemory     # Gọi hàm giải phóng bộ nhớ
```

```
j end
```

```
end:
```

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