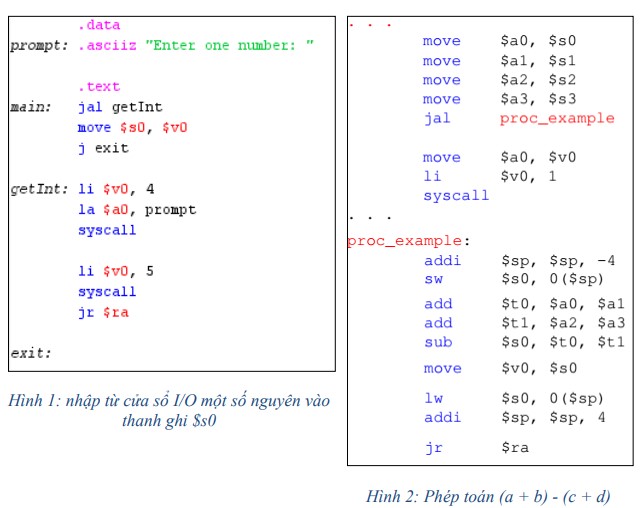
Tên: Đoàn Vũ Phú Minh

MSSV: 22520859

# LAB 4’S REPORT

# Section 1. Thực hành với thủ tục



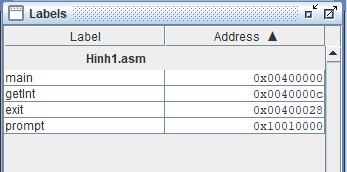
Task name:

Chạy từng bước và theo dõi sự thay đổi của thanh ghi PC, $ra, $sp, $fp trong 2 hình trên.

* Chạy toàn chương trình một lần để xem kết quả
* Với code trong Hình 1, nếu bỏ dòng code “j exit”, việc gì sẽ xảy ra?
* Viết lại code trong Hình 1, thêm vào thủ tục tên showInt để in ra cửa sổ I/O giá trị của số int nhập vào cộng thêm 1.
* Viết lại code trong Hình 2, lúc này chương trình chính cần tính giá trị của cả hai biểu thức: (a + b) – (c + d) và (a – b) + (c – d), hàm proc\_example có hai giá trị trả về và trong thân hàm sử dụng hai biến cục bộ $s0 và $s1 ($s1 lưu kết quả của (a – b) + (c – d))
* Viết lại code trong Hình 2, lúc này chương trình chính cần tính giá trị của cả hai biểu thức: (a + b) – (c + d), (e - f) hàm proc\_example có 6 input và hai giá trị trả về và trong thân hàm sử dụng hai biến cục bộ $s0 và $s1 ($s1 lưu kết quả của e – f)

## **Phần trả lời Hình 1**

• Địa chỉ ban đầu của các Label và giá trị ban đầu của các thanh ghi $sp, $fp, $ra, pc dưới dạng thập lục phân



### Hình 1: Địa chỉ ban đầu



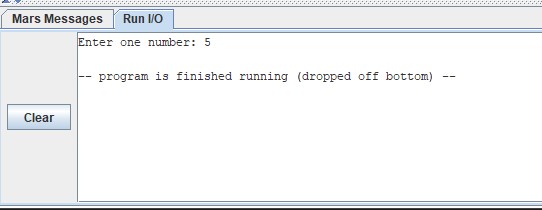
*Hình 2: Giá trị ban đầu*

### Chạy chương trình từng bước

|  |  |  |
| --- | --- | --- |
| CODE | | GIẢI THÍCH |
| .data  prompt: .asciiz "Enter one  number: "  .text main:  jal getInt    move $s0,$v0    j exit    getInt: li $v0,4    la $a0,prompt    syscall    li $v0,5 | | Chương trình bắt đầu với phần ‘ .text ’  Bắt đầu với Label main  pc = 0x00400000  # Nhảy và Liên kết đến hàm getInt pc =  0x0040000c  $ra =  0x00400004  # Di chuyển kết quả của getInt (đầu vào của người  dùng) đến $s0 pc = 0x00400004 # Nhảy đến exit pc = 0x00400008    # Nạp giá trị ngay: đặt $v0 thành 4 (syscall để in chuỗi)  pc = 0x0040000c  # Nạp địa chỉ: đặt $a0 thành địa chỉ của chuỗi prompt  pc = 0x00400010  # System call để in thông báo pc = 0x00400018 |
| exit: | syscall jr $ra | # Nạp giá trị ngay: đặt $v0 thành 5 (syscall để đọc số nguyên) pc = 0x0040001c  # System call để đọc một số nguyên từ người dùng pc = 0x00400020  # Nhảy đến địa chỉ trả về được lưu trong $ra (trả về từ hàm) pc = 0x00400024  Kết thúc chương trình bằng việc nhảy tới Label exit   pc = 0x00400028 |

Chạy toàn chương trình

 Kết quả



#### Hình 3: Kết quả sau khi chạy

 Giá trị cuối cùng của các thanh ghi sau khi kết thúc chương trình

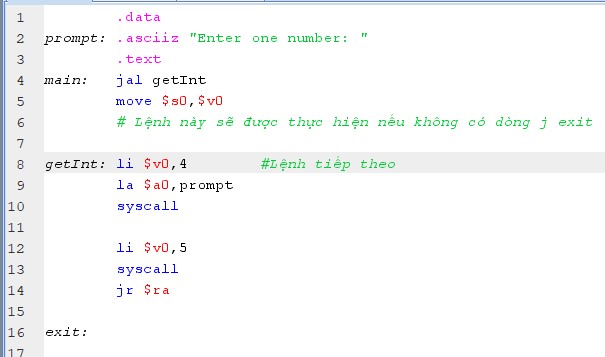


#### Hình 4: Giá trị sau khi kết thúc

* Kết quả về sự thay đổi cuối cùng của thanh ghi $sp, $fp, $ra, pc trong quá trình chạy toàn màn hình giống với quá trình chạy từng bước
* Với mỗi lệnh được thực thi, thanh ghi PC =PC + 4 với mỗi lệnh tiếp theo sẽ thực hiện và thanh ghi pc = Label’s address với mỗi lần thực hiện lệnh nhảy đến Label đó
* Sau khi thực hiện lệnh jal thì thanh ghi $ra =PC + 4. Để khi thực hiện lệnh jr $ra, chương trình sẽ thực hiện lệnh tiếp theo phía sau lệnh jal

 Với code trong Hình 1, nếu bỏ dòng code “j exit”, việc gì sẽ xảy ra?

 Sau khi thay đổi

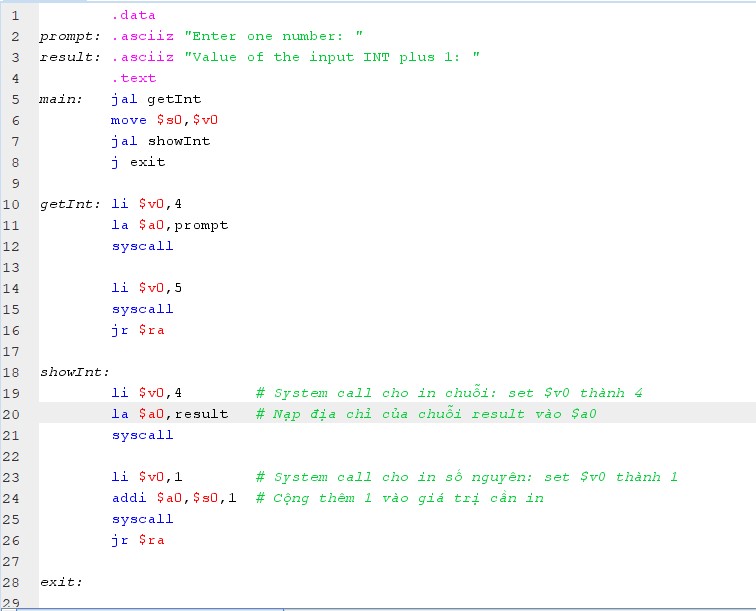


#### Hình 5: Loại bỏ lệnh “ j exit ”

* Với code trong Hình 1, nếu bỏ dòng code “j exit” thì • Thay vì chương trình nhảy đến Label exit để kết thúc chương trình thì chương trình sẽ thực hiện lệnh tiếp theo move $s0,$v0.
* Khi đó, chương trình sẽ chạy liên tục và trở thành vòng lặp vô hạn

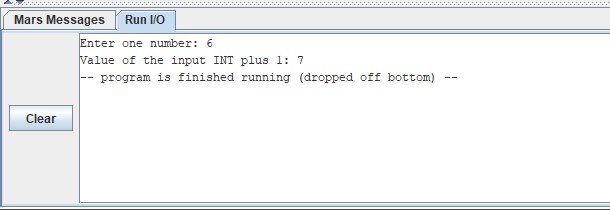
 Viết lại code trong Hình 1, thêm vào thủ tục tên showInt để in ra cửa sổ I/O giá trị của số int nhập vào cộng thêm 1.

 Code :



#### Hình 6: Sau khi thêm thủ tục showInt

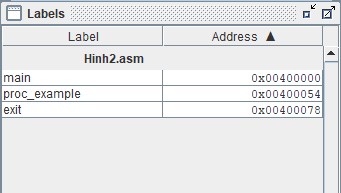
 Kết quả:



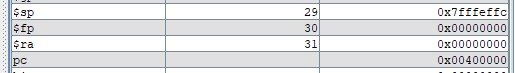
*Hình 7:* *Kết quả xuất ra màn hình I/O*

## **Phần trả lời Hình 2**

 Địa chỉ ban đầu của các Label và giá trị ban đầu của các thanh ghi $sp, $fp, $ra, pc dưới dạng thập lục phân



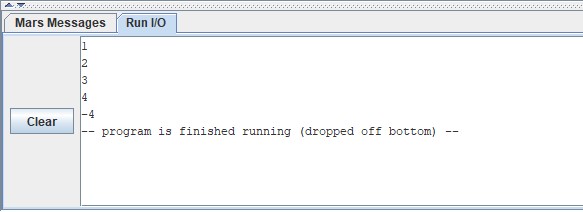
### Hình 1: Địa chỉ của các label



### Hình 2: Giá trị trước khi chạy chương trình

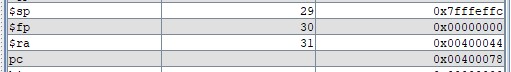
|  |  |  |
| --- | --- | --- |
| CODE | | GIẢI THÍCH |
| .data  .text main: li $v0,5  syscall move $s0,$v0    li $v0,5 syscall  move $s1,$v0    li $v0,5 syscall  move $s2,$v0 li $v0,5 syscall  move $s3,$v0    move $a0,$s0    move $a1,$s1    move $a2,$s2    move $a3,$s3    jal proc\_example | | * Chương trình bắt đầu với phần ‘ .text ’   Bắt đầu với Label main  pc = 0x00400000 Sau khi syscall để nhập số nguyên, thì giá trị thanh ghi thay đổi sau khi thực hiện lệnh move   * PC = 0x00400008       Tương tự như trên   * PC = 0x00400014       Tương tự như trên   * PC = 0x00400020     Tương tự như trên   * PC = 0x0040002c     Copy giá trị $s0 sang $a0 PC = 0x00400030    Copy giá trị $s0 sang $a0 PC = 0x00400034    Copy giá trị $s0 sang $a0 PC = 0x00400038    Copy giá trị $s0 sang $a0 PC = 0x0040003c    Nhảy và Liên kết đến hàm proc\_example |
| exit: | move $a0,$v0  li $v0,1  syscall j exit proc\_example: addi $sp,$sp,-4  sw $s0,0($sp) add $t0,$a0,$a1 add $t1,$a2,$a3 sub $s0,$t0,$t1  move $v0,$s0 lw $s0,0($sp) addi $sp,$sp,4 jr $ra | * PC = 0x00400040 * $ra = 0x00400044   Copy giá trị của $v0 sang $a0 PC= 0x00400044    Thực hiện in giá trị ra màn hình I/O  PC=  0x00400048    Thưc hiện gọi hệ thống  PC= 0x0040004c  Nhảy đến Label exit  PC= 0x00400050    Kết quả của phép tính $sp = $sp -4   * PC = 0x00400054   $sp = 0x7fffeff8  Lưu giá trị của $s0 vào vị trí đầu tiên của stack  PC = 0x00400058  Kết quả của phép toán cộng sẽ lưu vào $t0  PC = 0x0040005c  Kết quả của phép toán cộng sẽ lưu vào $t1  PC = 0x00400060  Kết quả của phép toán trừ sẽ lưu vào $s0  PC = 0x00400064  Copy giá trị của $s0 vào $v0 PC = 0x00400068  Đưa giá trị đầu tiên của stack vào $s0  PC = 0x0040006c  Cộng 4 vào thanh ghi $sp PC = 0x00400070  $sp = 0x7fffeffc  Nhảy đến địa chỉ trả về được lưu trong $ra (trả về từ hàm) PC = 0x00400074  Kết thúc chương trình bằng việc nhảy tới Label exit  PC = 0x00400078 |

Chạy toàn chương trình  Kết quả:



### Hình 3: In giá trị tính toán ra màn hình

 Giá trị cuối cùng của các thanh ghi sau khi kết thúc chương trình

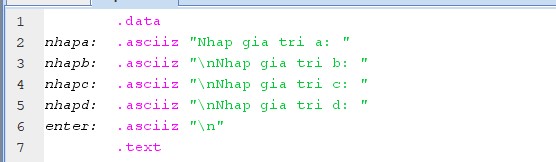


### Hình 4: Giá trị sau khi chạy chương trình

 Viết lại code trong Hình 2, lúc này chương trình chính cần tính giá trị của cả hai biểu thức: (a + b) – (c + d) và (a – b) + (c – d), hàm proc\_example có hai giá trị trả về và trong thân hàm sử dụng hai biến cục bộ $s0 và $s1 ($s1 lưu kết quả của (a – b) + (c – d))

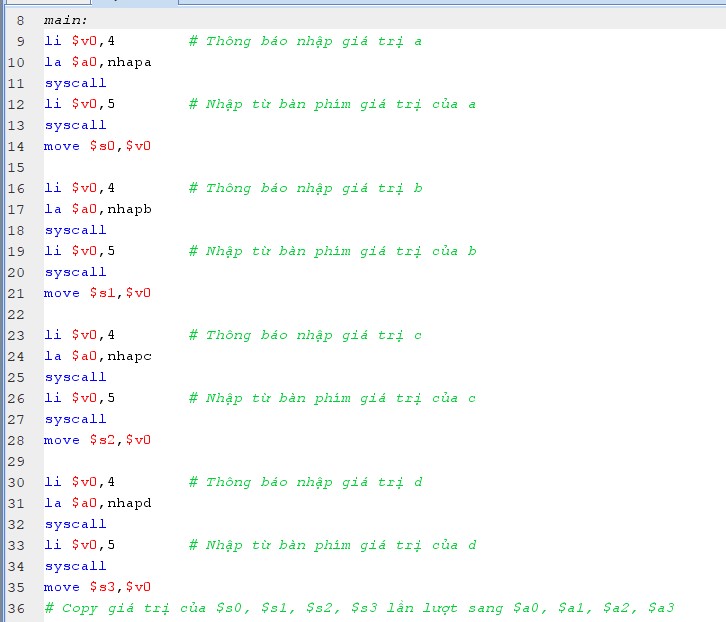
CODE

1. Khai báo



### Hình 5: Tạo các chuỗi thông báo

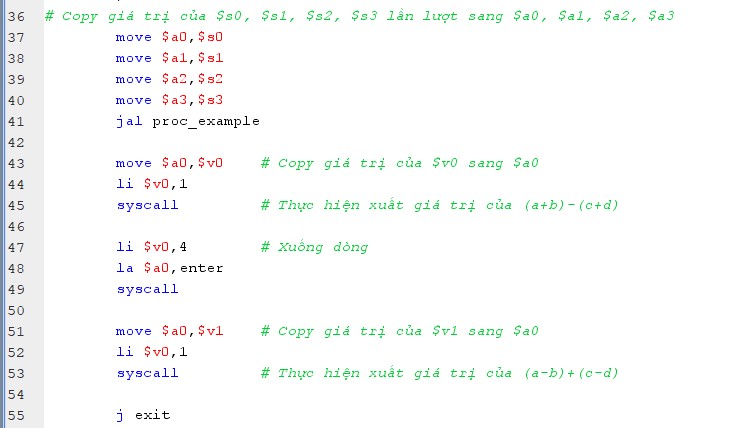
2. Thực hiện nhập giá trị vào các biến a, b, c, d



### Hình 6: Đưa giá trị từ bàn phím

3. Đưa giá trị vào các thanh ghi cần tính toán. ( dòng 37 -> 40 )  Thực hiện các lệnh jump để nhảy đến các Label cần thực hiện quá trình tính toán ( dòng 41 )

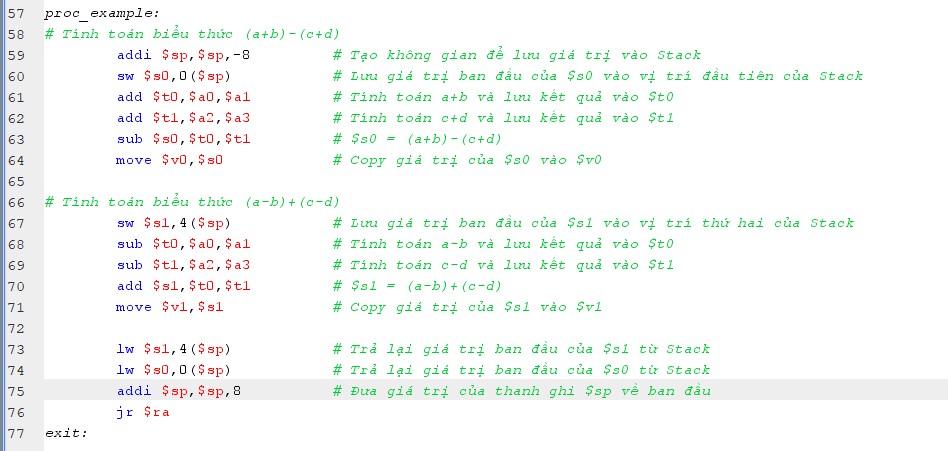
* Thực hiện quá trình xuất kết quả ra màn hình I/O ( dòng 43 -> 53 )
* Kết thúc chương trình bằng việc nhảy đến Label exit ( dòng 55 )



### Hình 7: Thực hiện các bước xuất kết quả

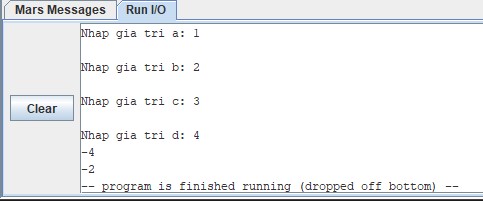
4. Thực hiện các phép tính toán trong Label proc\_example

 Sau khi thực hiện xong, chương trình sẽ nhảy tới thanh ghi $ra thông qua lệnh jr $ra



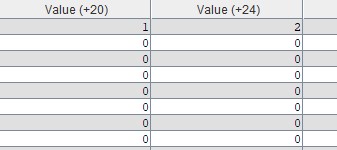
### Hình 8: Quá trình tính toán và đưa giá trị ban đầu của các biến vào Stack

 Kết quả của chương trình



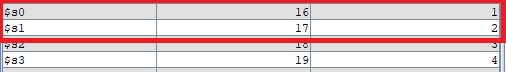
### Hình 9: Đưa ra màn hình I/O yêu cầu nhập và xuất giá trị tính toán

 Giá trị ban đầu của $s0 và $s1 khi lưu vào stack



### Hình 10: Những giá trị được lưu vào Stack

 Sau khi thực hiện tính toán, trả lại giá trị ban đầu của thanh ghi $s0 và $s1 bằng việc lấy ra từ stack và gán giá trị lần lượt vào 2 thanh ghi

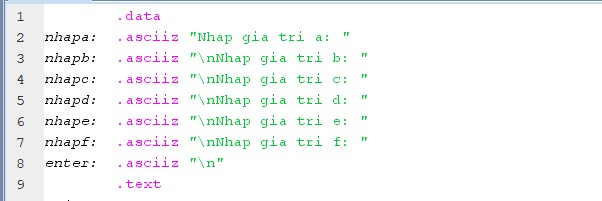


### Hình 11: Trả lại giá trị ban đầu cho các thanh ghi

 Viết lại code trong Hình 2, lúc này chương trình chính cần tính giá trị của cả hai biểu thức:(a + b) – (c + d), (e - f) hàm proc\_example có 6 input và hai giá trị trả về và trong thân hàm sử dụng hai biến cục bộ $s0 và $s1 ($s1 lưu kết quả của (e – f))

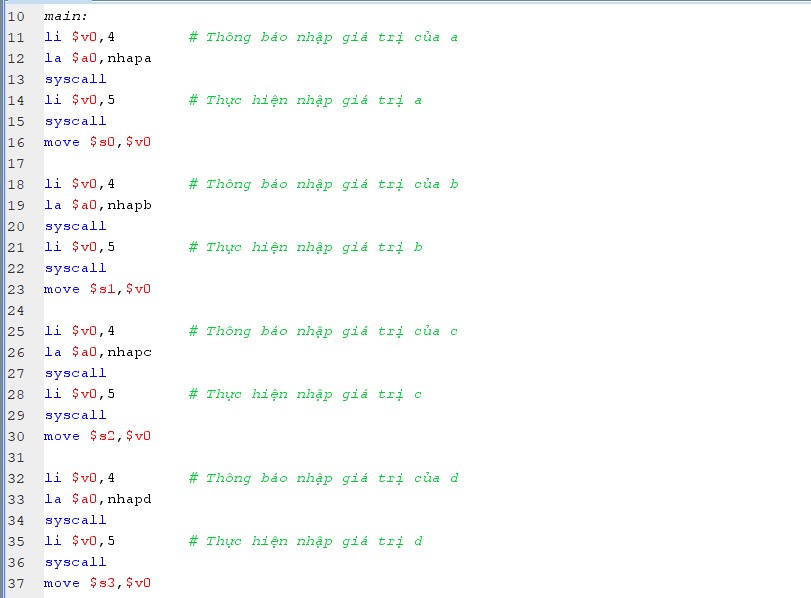
CODE

1) Khai báo

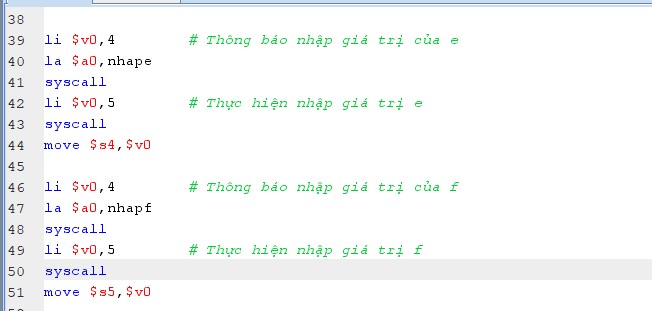


### Hình 12: Tạo ra các chuỗi yêu cầu

2) Thực hiện nhập giá trị vào các biến a, b, c, d, e, f



### Hình 13: Gán các giá trị từ bàn phím

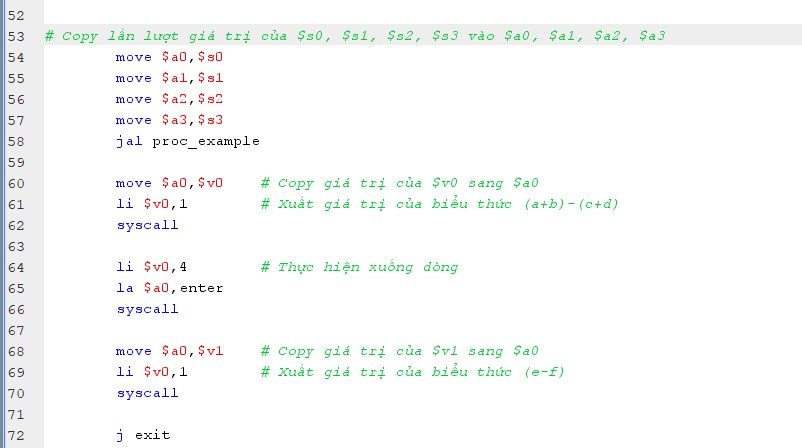


### Hình 14: Gán các giá trị từ bàn phím

3) Đưa giá trị vào các thanh ghi cần tính toán. ( dòng 54 -> 57 )

* Thực hiện các lệnh jump để nhảy đến các Label cần thực hiện quá trình tính toán ( dòng 58 )
* Thực hiện xuất kết quả của các biểu thức ra màn hình I/O ( dòng 60 -> 70 )

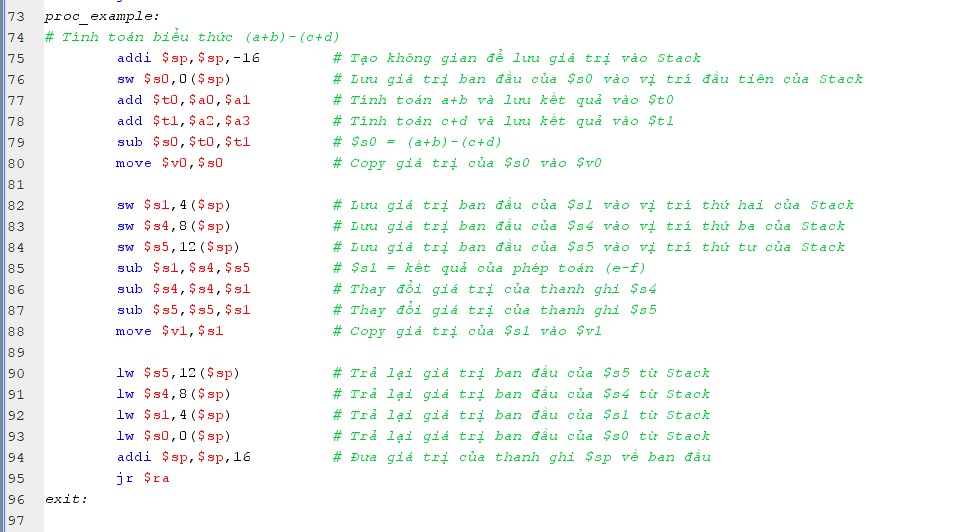
###  Lệnh nhảy đến Label exit để thoát khỏi chương trình ( dòng 72 )



#### Hình 15: Quá trình đưa kết quả ra màn hình I/O

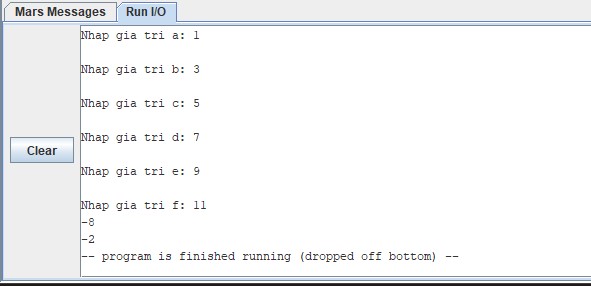
4) Thực hiện các phép toán trong Label proc\_example

 Sau khi thực hiện xong, chương trình sẽ nhảy tới thanh ghi $ra thông qua lệnh jr $ra



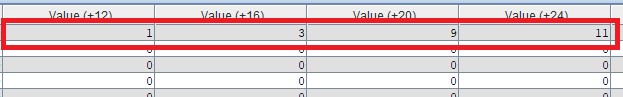
#### Hình 16: Tính toán các biểu thức và lưu các giá trị ban đầu vào Stack

 Kết quả khi chạy chương trình



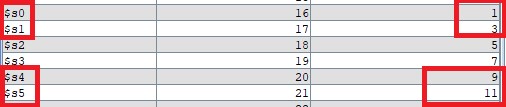
#### Hình 17: Xuất ra kết quả tính toán

 Giá trị ban đầu của $s0, $s1, $s4, $s5 sau khi lưu vào Stack



#### Hình 18: Các giá trị được lưu vào Stack

 Sau khi thực hiện tính toán, trả lại giá trị ban đầu của thanh ghi $s0 và $s1 bằng việc lấy ra từ stack và gán giá trị lần lượt vào 2 thanh ghi



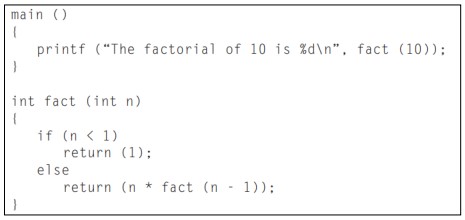
*Hình 19:* *Gán giá trị ban đầu lại cho các thanh ghi*

# Section 2. Thực hành với đệ quy

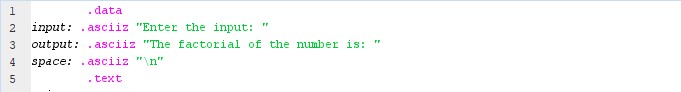
Task name:

Khi thân của một thủ tục gọi một thủ tục khác, giá trị của thanh ghi $ra ngay lập tức bị ghi đè, xóa mất giá trị cũ, dẫn tới sau khi thủ tục này thực hiện xong sẽ không có địa chỉ để quay về. Vì vậy, nếu trong trường hợp lồng thủ tục hay đệ quy, thanh ghi $ra cũng phải được lưu lại cùng các thanh ghi $s.

Viết code MIPS, chạy kiểm thử trên MARS và giải thích ý nghĩa rõ ràng cho chương trình tính giai thừa được viết bằng code C như Hình 3.

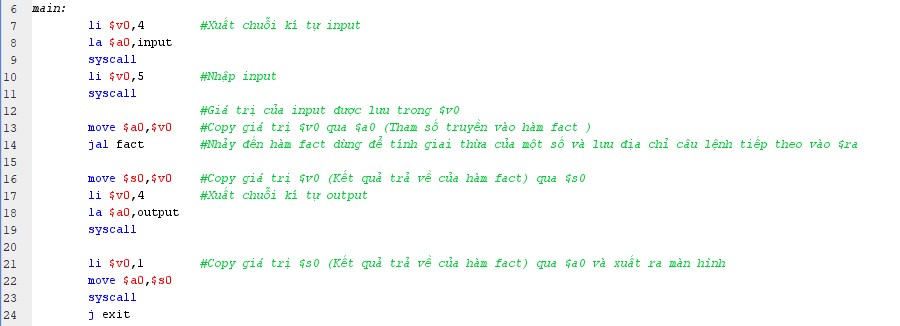


Hình 3:Tính giai thừa



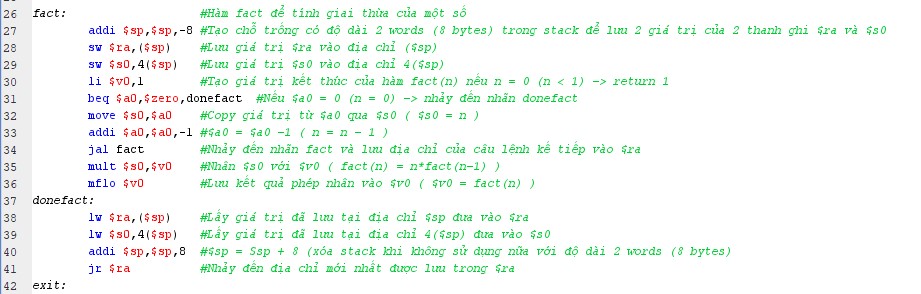
*Hình 1: Phần khai báo*

* Khai báo các chuỗi kí tự: input (nhập mảng) , output ( xuất kết quả) , space ( cách xuống dòng)



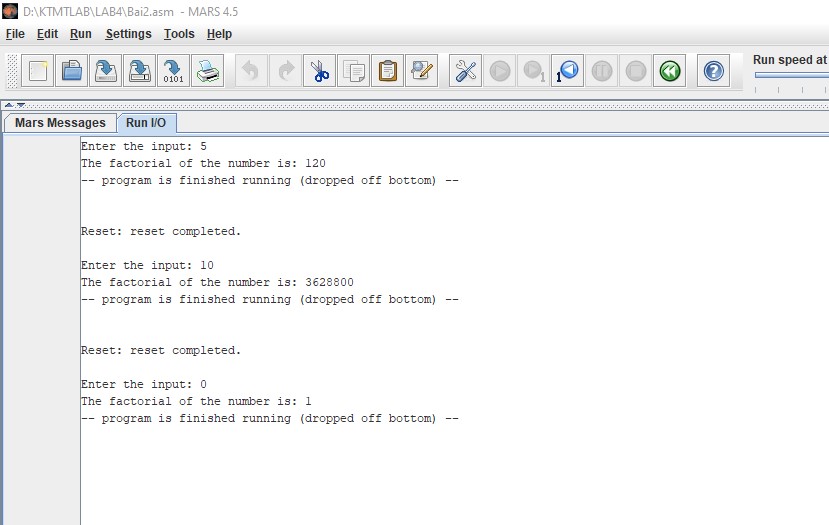
*Hình 2: Phần main*

* Xuất chuỗi kí tự input và nhập vào số nguyên
* Thực hiện tính giai thừa với số nguyên nhập vào bằng hàm fact
* Xuất chuỗi kí tự output và kết quả của hàm fact ( Giai thừa của số nguyên nhập vào )
* Thoát chương trình



*Hình 3: Hàm fact tính giai thừa*

* Lần lượt lưu các giá trị trong $ra và $s0 theo đúng thứ tự vào trong stack,tạo chỗ trong stack để lưu hai giá trị này.Thanh ghi $ra chứa địa chỉ câu lệnh cần quay về và $s0 lưu giá trị của tham số truyền vào hàm được lưu trong $a0. Giá trị của $a0 giảm dần để tính giai thừa
* Khi đã lưu đúng thứ tự các giá trị của $ra và $s0.Lần lượt lấy các giá trị từ $s0 ra để tính giai thừa,lấy giá trị $ra ra để quay về câu lệnh cần thực hiện.
* Khi lấy ra lần lượt xóa stack không cần sử dụng nữa.
* Nhảy đến địa chỉ được lưu trong $ra để thực hiện việc tính giai thừa
* Khi đã thực hiện việc tính giai thừa xong (kết thúc hàm) quay trở về câu lệnh phía trên để in kết quả của hàm tính giai thừa ra màn hình.

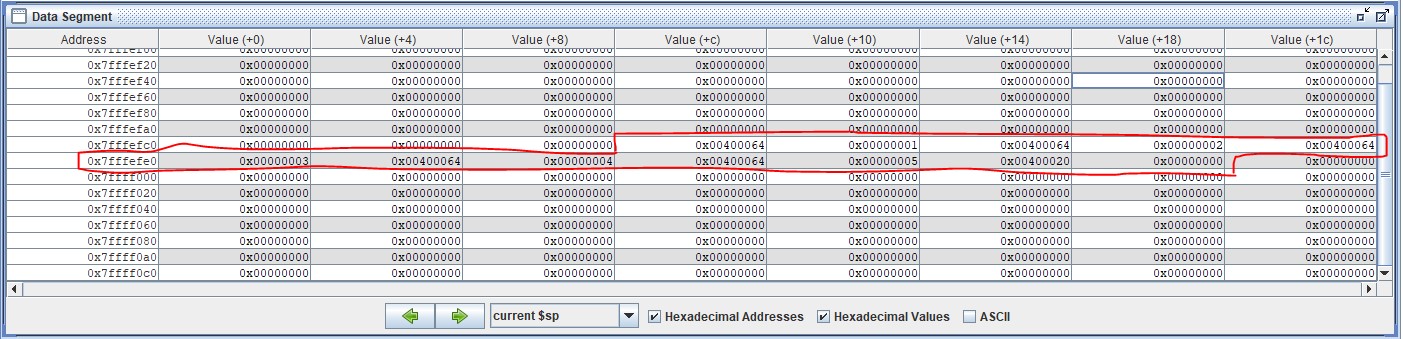


*Hình 4: Kết quả của 5!,10!,0!*

# Section 3. Bài tập

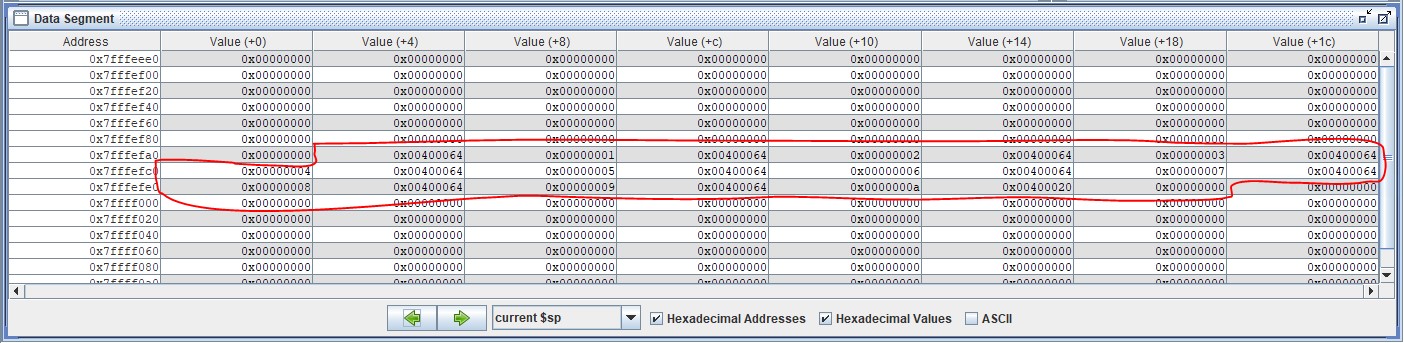
Task name 1: Tiếp tục nội dung 2. Vẽ lại hình ảnh của các stack trong trường hợp tính 5!

và 10!



*Hình 1: Hình ảnh stack trong trường hợp tính 5!*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Thứ tự lưu vào Stack | |  |
| STT | Giá trị $ra (Địa chỉ câu lệnh) | Câu lệnh | Giá trị $s0 |
| 1 | 0x00400020 | 16. move $s0,$v0 | 0 |
| 2 | 0x00400064 | 35. mult $s0,$v0 | 5 |
| 3 | 0x00400064 | 35. mult $s0,$v0 | 4 |
| 4 | 0x00400064 | 35. mult $s0,$v0 | 3 |
| 5 | 0x00400064 | 35. mult $s0,$v0 | 2 |
| 6 | 0x00400064 | 35. mult $s0,$v0 | 1 |



|  |  |  |  |
| --- | --- | --- | --- |
|  | Thứ tự lấy ra từ Stack | |  |
| STT | Giá trị $ra (Địa chỉ câu lệnh) | Câu lệnh | Giá trị $s0 |
| 1 | 0x00400064 | 35. mult $s0,$v0 | 1 |
| 2 | 0x00400064 | 35. mult $s0,$v0 | 2 |
| 3 | 0x00400064 | 35. mult $s0,$v0 | 3 |
| 4 | 0x00400064 | 35. mult $s0,$v0 | 4 |
| 5 | 0x00400064 | 35. mult $s0,$v0 | 5 |
| 6 | 0x00400020 | 16. move $s0,$v0 | 0 |

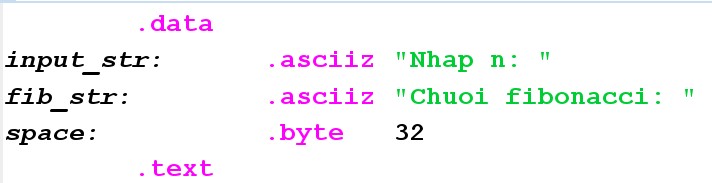
*Hình 2: Hình ảnh stack trong trường hợp tính 10!*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Thứ tự lưu vào Stack | |  |
| STT | Giá trị $ra (Địa chỉ câu lệnh) | Câu lệnh 16. move $s0,$v0 | Giá trị $s0 |
| 1 | 0x00400020 | 0 |
| 2 | 0x00400064 | 35. mult $s0,$v0 | 10 |
| 3 | 0x00400064 | 35. mult $s0,$v0 | 9 |
| 4 | 0x00400064 | 35. mult $s0,$v0 | 8 |
| 5 | 0x00400064 | 35. mult $s0,$v0 | 7 |
| 6 | 0x00400064 | 35. mult $s0,$v0 | 6 |
| 7 | 0x00400064 | 35. mult $s0,$v0 | 5 |
| 8 | 0x00400064 | 35. mult $s0,$v0 | 4 |
| 9 | 0x00400064 | 35. mult $s0,$v0 | 3 |
| 10 | 0x00400064 | 35. mult $s0,$v0 | 2 |
| 11 | 0x00400064 | 35. mult $s0,$v0 | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Thứ tự lấy ra từ Stack | |  |
| STT | Giá trị $ra (Địa chỉ câu lệnh) | Câu lệnh | Giá trị $s0 |
| 1 | 0x00400064 | 35. mult $s0,$v0 | 1 |
| 2 | 0x00400064 | 35. mult $s0,$v0 | 2 |
| 3 | 0x00400064 | 35. mult $s0,$v0 | 3 |
| 4 | 0x00400064 | 35. mult $s0,$v0 | 4 |
| 5 | 0x00400064 | 35. mult $s0,$v0 | 5 |
| 6 | 0x00400064 | 35. mult $s0,$v0 | 6 |
| 7 | 0x00400064 | 35. mult $s0,$v0 | 7 |
| 8 | 0x00400064 | 35. mult $s0,$v0 | 8 |
| 9 | 0x00400064 | 35. mult $s0,$v0 | 9 |
| 10 | 0x00400064 | 35. mult $s0,$v0 | 10 |
| 11 | 0x00400020 | 16. move $s0,$v0 | 0 |

Task name 2: Viết chương trình nhập vào n và xuất ra chuỗi Fibonacci tương ứng

- Trước tiên, khai báo các biến cần cho mô phỏng:



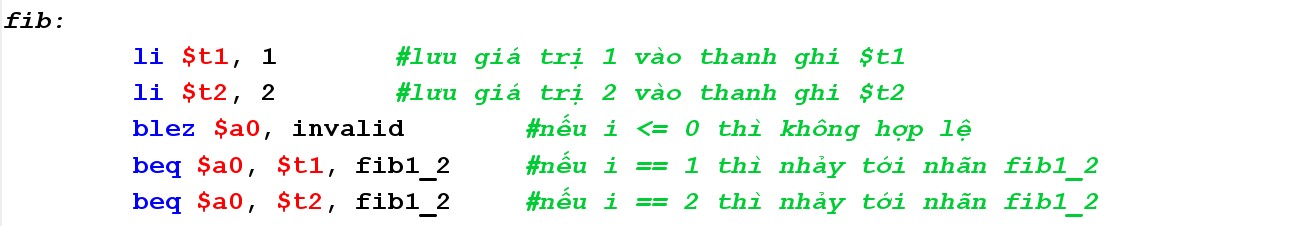
## Hình 3. Các biến đã được khai báo

 *input\_str* và *fib\_str* được dùng để lưu trữ các chuỗi với nội dung như hình.

 *space* dùng để lưu mã ASCII của kí tự khoảng trắng.

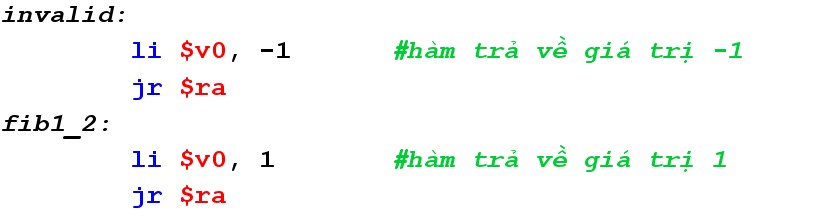
- Tiếp theo, định nghĩa hàm *fib* dùng để tính số Fibonacci thứ n:

 Đầu tiên, kiểm tra các trường hợp đặc biệt:



## Hình 4. Đoạn mã kiểm tra các trường hợp đặc biệt

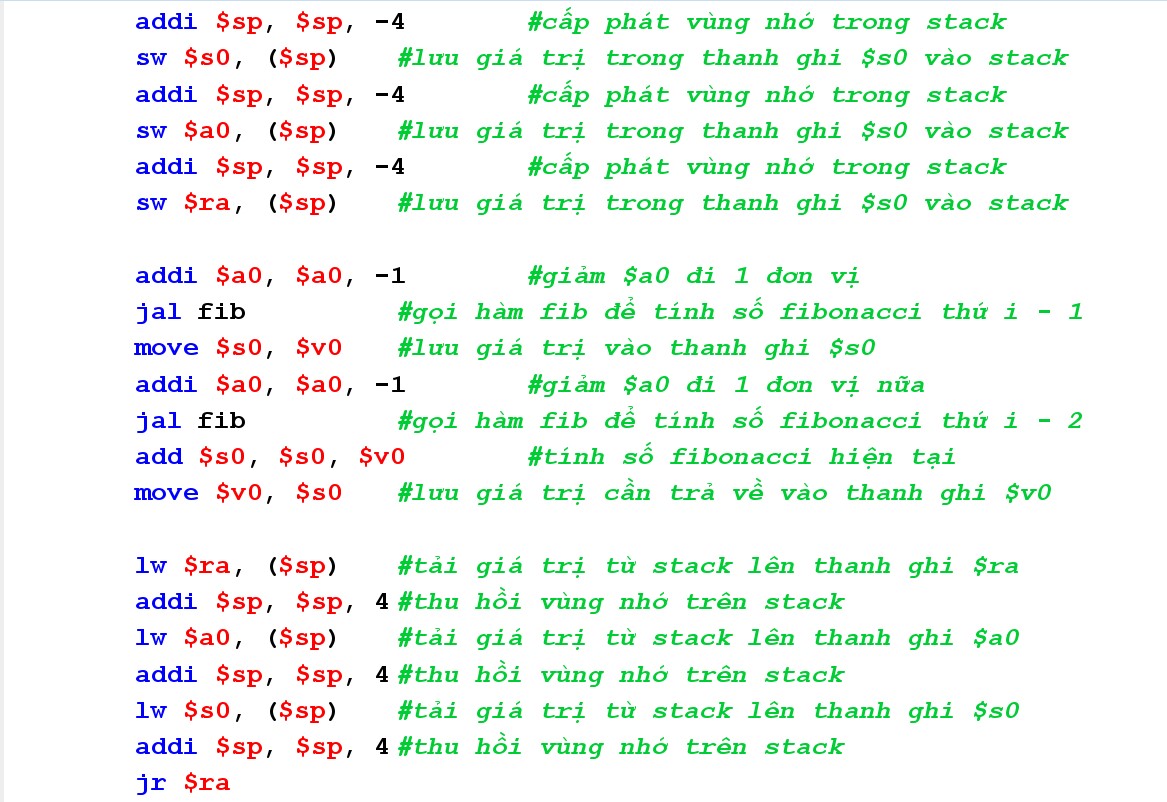
* Nếu đối số ở thanh ghi *$a0* bé hơn 0 thì là trường hợp không hợp lệ và nhảy tới nhãn *invalid*.
* Nếu đối số ở thanh ghi *$a0* bằng 1 hoặc 2 thì nhảy tới nhãn *fib1\_2*.



## Hình 5. Đoạn mã tại 2 nhãn invalid và fib1\_2

* Tại nhãn *invalid*, chương trình sẽ trả về giá trị -1 đại diện cho giá trị không hợp lệ.
* Tại nhãn *fib1\_2*, chương trình sẽ trả về giá trị 1 là số Fibonacci thứ nhất và thứ hai.

 Tiếp theo, xử lí các trường hợp còn lại:

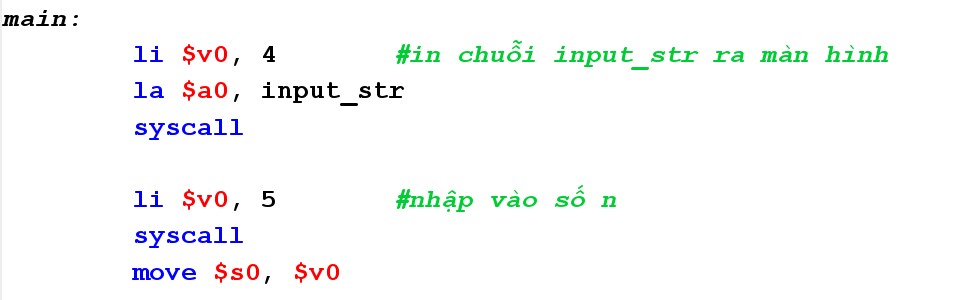


## Hình 6. Đoạn mã xử lí các trường hợp còn lại của hàm

* Lần lượt lưu vào stack những giá trị của thanh ghi *$s0, $a0* và *$ra* vì sẽ bị thay đổi giá trị trong quá trình gọi đệ quy.
* Gọi đệ quy hàm *fib* với đối số là giá trị trong thanh ghi *$a0* trừ đi 1 đơn vị.
* Lưu giá trị trả về từ hàm vào thanh ghi *$s0*.
* Gọi đệ quy hàm *fib* với đối số là giá trị trong thanh ghi *$a0* (giá trị chưa thay đổi từ khi gọi hàm *fib* trước) trừ đi 1 đơn vị.
* Tính giá trị trả về của hàm hiện tại là tổng của 2 hàm *fib* đã được gọi trước đó và lưu vào *$s0* sau đó lưu vào *$a0*.
* Cuối cùng, lần lượt trả lại giá trị cũ cho các thanh ghi *$ra*, *$a0* và *$s0* sau đó ngưng thực thi hàm hiện tại.

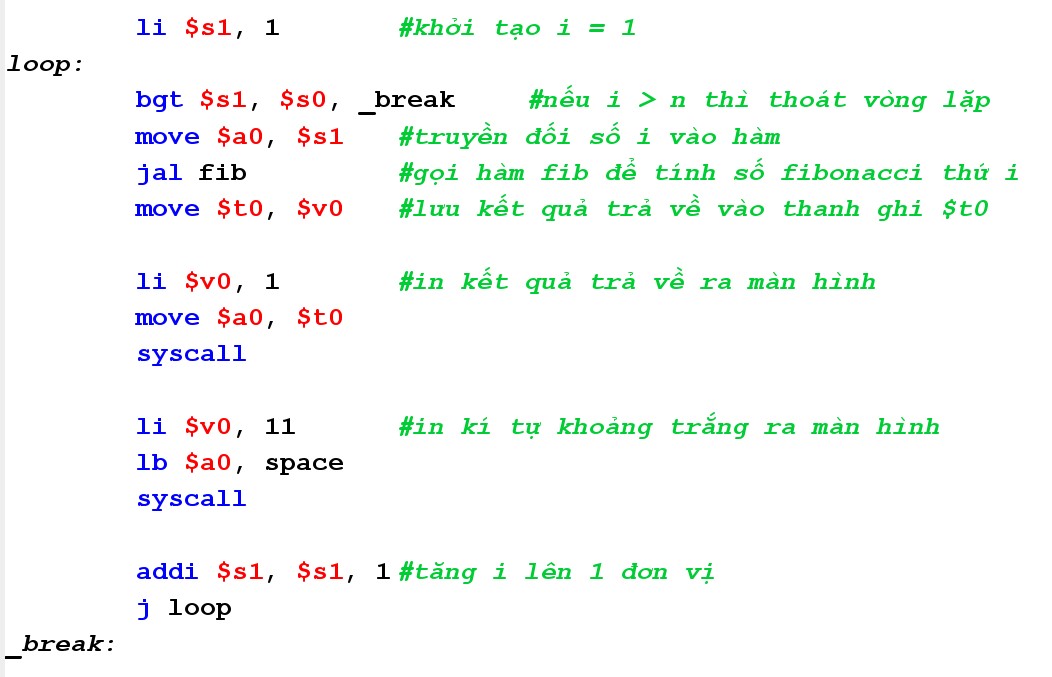
- Thực thi chương trình chính:

 Đầu tiên, nhập vào số *n* để thực hiện tính toán:



## Hình 7. Đoạn mã nhập vào số n

 Tiếp theo, tính toán và in ra các số Fibonacci có thứ tự từ 1 đến n:

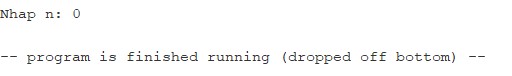


## Hình 8. Đoạn mã in chuỗi Fibonacci

* Khởi tạo giá trị *i* bằng 1 là số thứ tự trong chuỗi Fibonacci.
* Kiểm tra nếu số *i* vượt quá *n* thì thoát vòng lặp và kết thúc chương trình.
* Nếu không, gọi hàm *fib* với đối số truyền vào là *i* để tính số Fibonacci thứ *i* và lưu kết quả trả về vào thanh ghi *$t0*.
* In giá trị ở *$t0* và kí tự khoảng trắng ra màn hình.
* Tăng *i* lên 1 đơn vị.



## Hình 9. Kết quả đối với trường hợp số n là hợp lệ



## Hình 10. Kết quả đối với trường hợp số n không hợp lệ