Các vấn đề liên quan đến hàm con (hay còn gọi là thủ tục - procedure) trong MIPS, hiểu các quy ước cũng như quy tắc hoạt động của việc gọi hàm. Giải thích stack làm gì, thanh ghi PC là gì.

Chú ý: Các thuật ngữ Routine/Procedure/Function có thể gặp trong một số môi trường khác nhau; trong bài thực hành này, tất cả đều được dịch là hàm hoặc thủ tục

Nội dung:

1. Phần 1

❖ Một số quy ước khi thao tác với hàm con:

- Thanh ghi $at ($1), $k0 ($26), $k1 ($27) được dành cho hệ điều hành và assembler; không nên sử dụng trong lúc lập trình thông thường.
- Thanh ghi $a0-$a3 ($4-$7) được sử dụng để truyền bốn tham số đến một *hàm con* (nếu có hơn 4 tham số truyền vào, các tham số còn lại được lưu vào stack). Thanh ghi $v0-$v1 ($2-$3) được sử dụng để lưu giá trị trả về của hàm.
- Các thanh ghi $t0-$t9 ($9-$15, 24, 25) được sử dụng như các thanh ghi tạm. Các thanh ghi $s0-$s7 ($16-$23) được sử dụng như các thanh ghi lưu giá trị bền vững.
  (Trong MIPS, việc tính toán có thể cần một số thanh ghi trung gian, tạm thời, các thanh ghi $t nên được dùng cho mục đích này; còn kết quả cuối của phép toán nên lưu vào các thanh ghi $s)

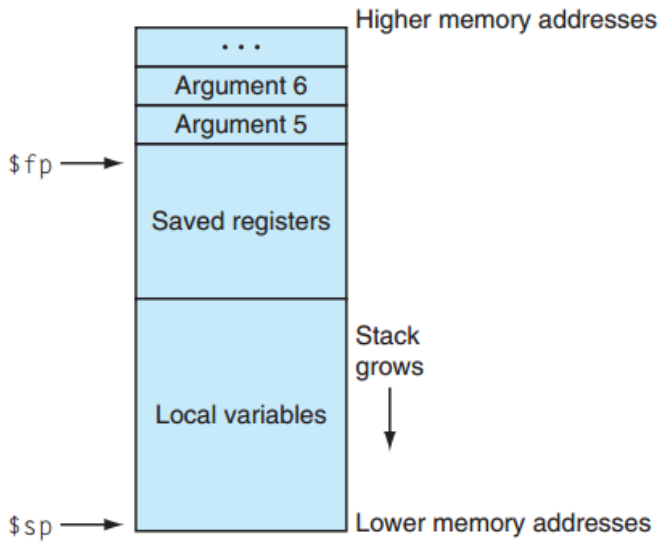
  Nếu quy ước này được tuân thủ, một hàm con nếu sử dụng bất kỳ thanh ghi loại $s nào sẽ phải lưu lại giá trị của thanh ghi $s đó trước khi thực thi hàm. Và trước khi thoát ra khỏi hàm, các giá trị cũ của các thanh ghi $s này cũng phải được trả về lại. Các thanh ghi $s này được xem như biến cục bộ của hàm
- Ngăn xếp (stack):
  Mỗi hàm con luôn cần một vùng nhớ đi cùng với nó (vì mỗi hàm con đều cần không gian để lưu lại các biến cục bộ của nó (là các thanh ghi $s được sử dụng trong thân hàm), hoặc để chứa các tham số input nếu số tham số lớn hơn 4 hoặc chứa giá trị trả về nếu số giá trị trả về lớn hơn 2). Vùng nhớ này được quy ước hoạt động như một stack.

  Thanh ghi $sp($29) goi là con trỏ stack (stack pointer), thanh ghi $fp ($30) là con trỏ frame (frame pointer).

  Stack gồm nhiều frame; frame cuối cùng trong stack được trỏ tới bởi $sp, frame đầu tiên (cho vùng lưu các thanh ghi cũng như biến cục bộ) được trỏ tới bởi $fp (xem hình 1)

  Stack được xây dựng theo kiểu từ địa chỉ cao giảm dần xuống thấp, vì thế frame pointer luôn ở trên stack pointer.

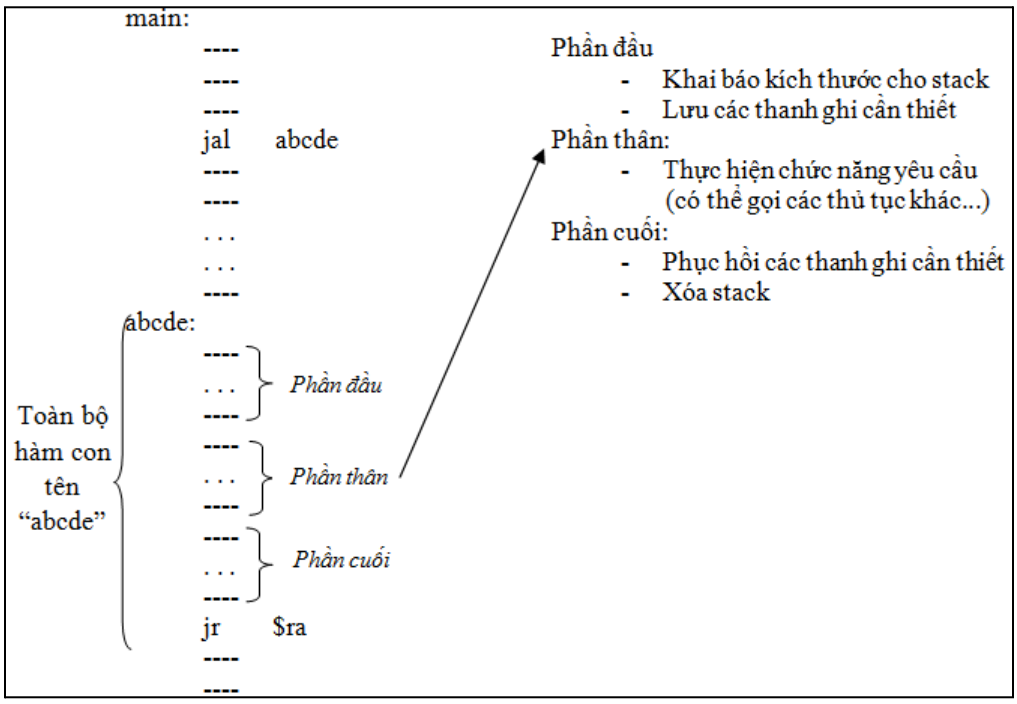
Tất nhiên stack có thể được xây dựng theo cách khác, tùy vào mỗi môi trường làm việc mà ta tuân thủ các quy ước.



Hình 1. Hình ảnh ví dụ của một stack với các frame (stack này phục vụ cho hàm con mà bốn tham số đầu vào đã được truyền thông qua thanh ghi, tham số thứ 5 và 6 được lưu trong stack như hình)

Việc lưu một phần tử vào stack gọi là *PUSH* và lấy phần tử đó ra gọi là *POP*. Theo quy ước của MIPS, khi *PUSH* một phần tử vào stack, đầu tiên $sp nên trừ đi 4 byte (1 word) rồi sau đó lưu dữ liệu vào word nhớ có địa chỉ đang chứa trong $sp. Ngược lại, khi POP một phần tử từ stack, dữ liệu được load ra từ word nhớ có địa chỉ đang chứa trong $sp rồi sau đó $sp được cộng thêm 4 (xem ví dụ 2 để hiểu rõ hơn)

❖ Cấu trúc một thủ tục:



Hình 2. Cấu trúc chung của một hàm con/thủ tục

Trong chương trình chính, khi hàm con được gọi, con trỏ PC sẽ chuyển quyền điều khiển xuống vị trí của hàm con; sau khi hàm con được thực hiện xong, con trỏ PC sẽ chuyển về thực hiện lệnh ngay sau lệnh gọi hàm con.

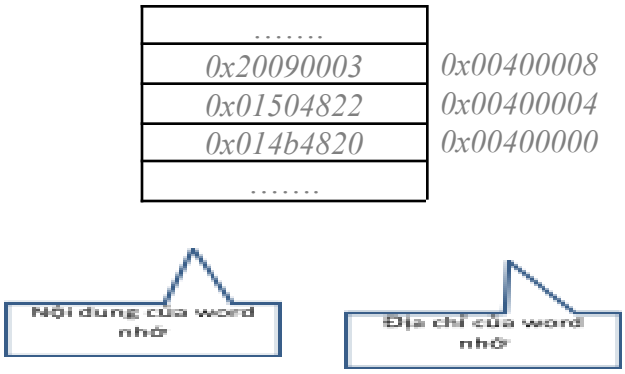
Giải thích con trỏ PC:

Khi một đoạn lệnh muốn được thực thi, từng lệnh trong đoạn lệnh này sẽ được chuyển thành mã máy và nạp lên bộ nhớ để chờ thực thi. Mã máy của mỗi lệnh sẽ được lưu ở một word nhớ. Từ đó, địa chỉ của word nhớ đang chứa mã máy của một lệnh nào đó được gọi là địa chỉ của lệnh đó.

Ví dụ cho đoan lệnh sau

add $t1, $t2, $t3 #mã máy: 0x014b4820
sub $t1, $t2, $s0 #mã máy: 0x01504822
addi $t1, $zero, 3 #mã máy: 0x20090003

Khi chạy, giả sử đoạn lệnh trên nạp vào bộ nhớ như sau



Lúc này, ta nói địa chỉ lệnh *add $t1, $t2, $t3* là *0x00400000;* địa chỉ của lệnh *sub $t1, $t2, $s0* là *0x00400004* và địa chỉ của lệnh *addi $t1, $zero, 3* là *0x00400008*

Trong bộ xử lý có một thanh ghi đặc biệt tên là PC (Program Counter). Khi một chương trình được nạp vào bộ nhớ và chuẩn bị chạy, địa chỉ của lệnh đầu tiên sẽ được lưu vào thanh ghi PC này. Khi chương trình chạy, bộ điều khiển đầu tiên sẽ vào đọc nội dung của thanh ghi PC để biết địa chỉ của lệnh cần thực thi. Sau đó mã máy của lệnh cần thực thi sẽ được load vào bộ xử lý để thực hiện; lúc này PC tự động tăng lên 4 để chứa địa chỉ của lệnh tiếp theo. Khi bộ xử lý hoàn tất lệnh hiện tại nó lại tiếp tục vào đọc nội dung PC để biết địa chỉ của lệnh tiếp theo cần thực hiện; và cứ thế tiếp tục cho đến hết chương trình.

Vì vậy, con trỏ PC thực chất là một thanh ghi tên PC (Program counter) trong bộ xử lý. Thanh ghi PC chứa địa chỉ của lệnh tiếp theo mà khối điều khiển sẽ xử lý. Vì thanh ghi PC chứa địa chỉ của lệnh, nên nó còn được gọi là con trỏ PC.

Lệnh *jal* thực hiện việc gọi hàm. Đầu tiên, địa chỉ của lệnh ngay sau lệnh *jal* phải được lưu lại để sau khi hàm con thực hiện xong, con trỏ PC có thể chuyển về lại đây (thanh ghi dùng để lưu địa chỉ trả về là $ra); sau khi địa chỉ trả về đã được lưu lại, con trỏ PC chuyển đến địa chỉ hàm con để thực hiện. Vậy có hai công việc được thực hiện trong *jal* theo thứ tự:

```
jal <address>        #   $ra = PC + 4
                     #   PC = <address>
```

Lệnh *"jr $ra"* đặt ở cuối hàm con thực hiện việc lấy giá trị đang chứa trong thanh ghi $ra gán vào PC, giúp thanh ghi quay trở về thực hiện lệnh ngay sau lệnh gọi hàm con này.

```
jr   $ra             #   PC = $ra
```

Mỗi hàm con/thủ tục được chia thành ba phần. Phần thân dùng để tính toán chức năng của hàm con này, bắt buộc phải có. Phần đầu và phần cuối có thể có hoặc không, tùy từng yêu cầu của chương trình con.

❖ **Ví dụ:**

1. Đoạn code sau thực hiện việc gọi hàm con (thủ tục) tên ABC

```
        add $t0, $t1, $t2
        jal ABC
        or $t3, $t4, $s5
        j EXIT
ABC:
        add $t0, $t3, $t5
        sub $t2, $t3, $t0
        jr $ra
EXIT:
```

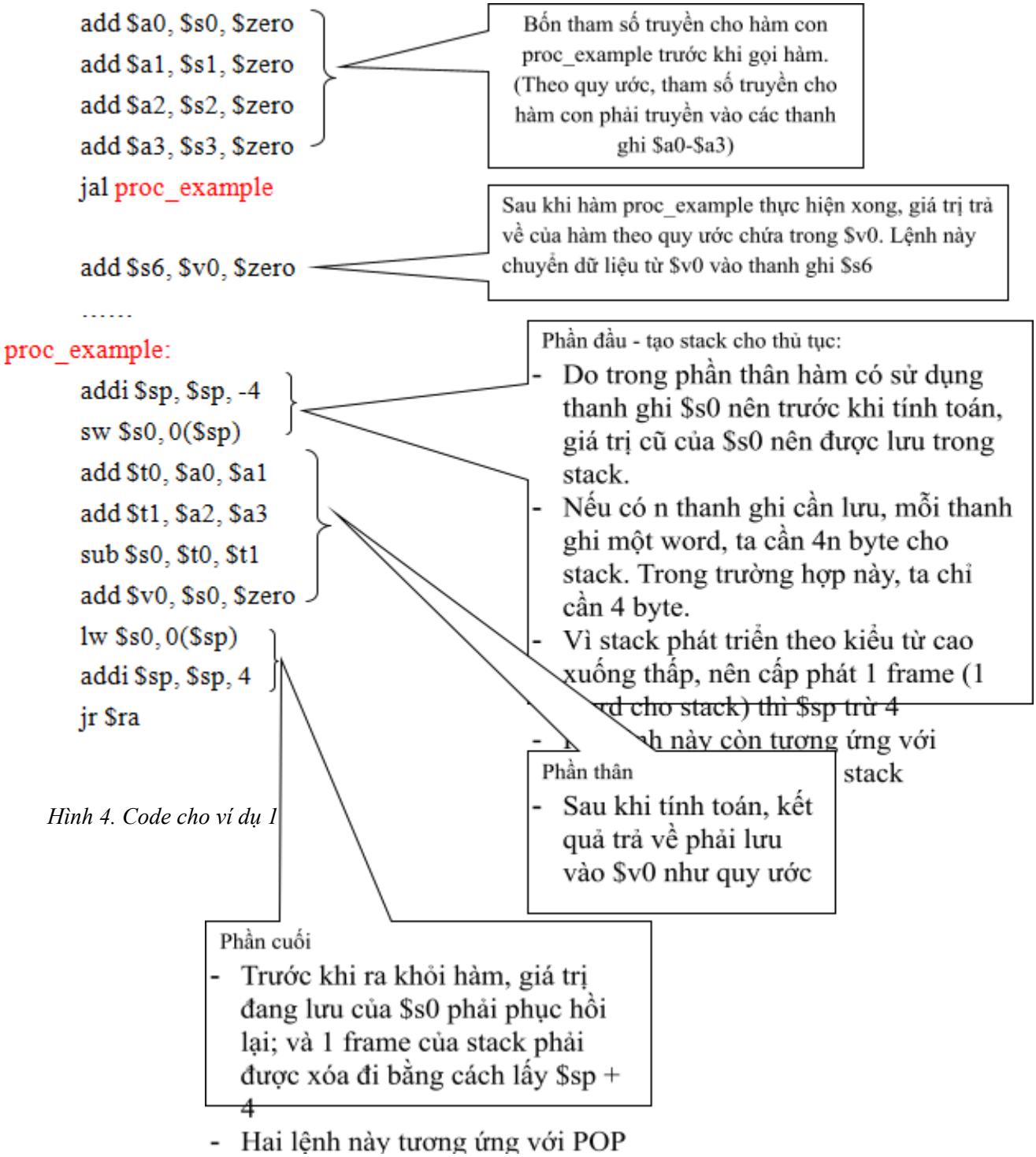
2. Viết đoạn code thực hiện

   Chương trình chính có:

   - Bốn giá trị a, b, c, d lần lượt chứa trong $s0, $1, $s2, $s3
   - Chương trình truyền bốn tham số này vào hàm con tên "proc_example" để lấy kết quả của phép toán trên. Sau đó đưa kết quả vào thanh ghi $s6

   Chương trình con có:

   - Bốn input (a, b, c, d)
   - Một output, là kết quả của phép toán: (a + b) – (c + d)
   - Quá trình tính toán sử dụng 2 thanh ghi tạm $t1, $t2 và một thanh ghi $s0



Hình 4. Code cho ví dụ 1

Trong ví dụ 1, thân hàm con không sử dụng bất kỳ thanh ghi $s nào, nên không cần khởi tạo stack để lưu các thanh ghi này lại. Trong ví dụ 2, thân hàm con có sử dụng thanh ghi $s, nên stack phải được khởi tạo để lưu; dẫn đến trước khi kết thúc hàm phải giá trị đã lưu phải trả về lại các thanh ghi và xóa stack.

(Lưu ý: Code trong ví dụ 2 có thể viết lại mà không cần dùng thanh ghi $s0; nhưng ví dụ này cố tình dùng $s0 để thấy được việc lưu trên stack phải thực hiện như thế nào)

Tóm lại:

Với một thủ tục/hàm con:

- Input: $a0, $a1, $a2, $a3
- Output: $v0, $v1
- Biến cục bộ $s0, $s1, ... , $s7 (phải được lưu trước khi sử dụng và trả lại giá trị cũ trước khi ra khỏi hàm)
- Địa chỉ quay về được lưu trong $ra

--------Hết--------