Bài thực hành số 2

Mục tiêu:

- Thực hành các vấn đề liên quan đến thủ tục/hàm con trong MIPS, hiểu các quy ước cũng như quy tắc hoạt động của việc gọi hàm.

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:**

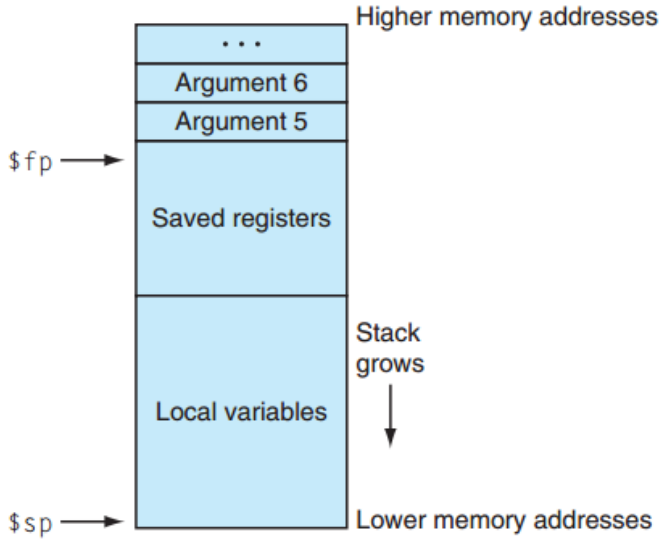
- Thanh ghi $at ($1), $k0 ($26), $k1 ($27) được dành cho hệ điều hành và assembler; không nên sử dụng trong lúc lập trình thông thường.
- Thanh ghi $a0-$a3 ($4-$7) được sử dụng để truyền bốn tham số đến một *hàm con* (nếu có hơn 4 tham số truyền vào, các tham số còn lại được lưu vào stack). Thanh ghi $v0-$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): Luôn cần một vùng nhớ đi kèm với mỗi hàm con (vì mỗi hàm con đều cần không gian để lưu lại các thanh ghi $s nếu có sử dụng trong hàm, hoặc để chứa các tham số input nếu số tham số lớn hơn 4). 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 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)

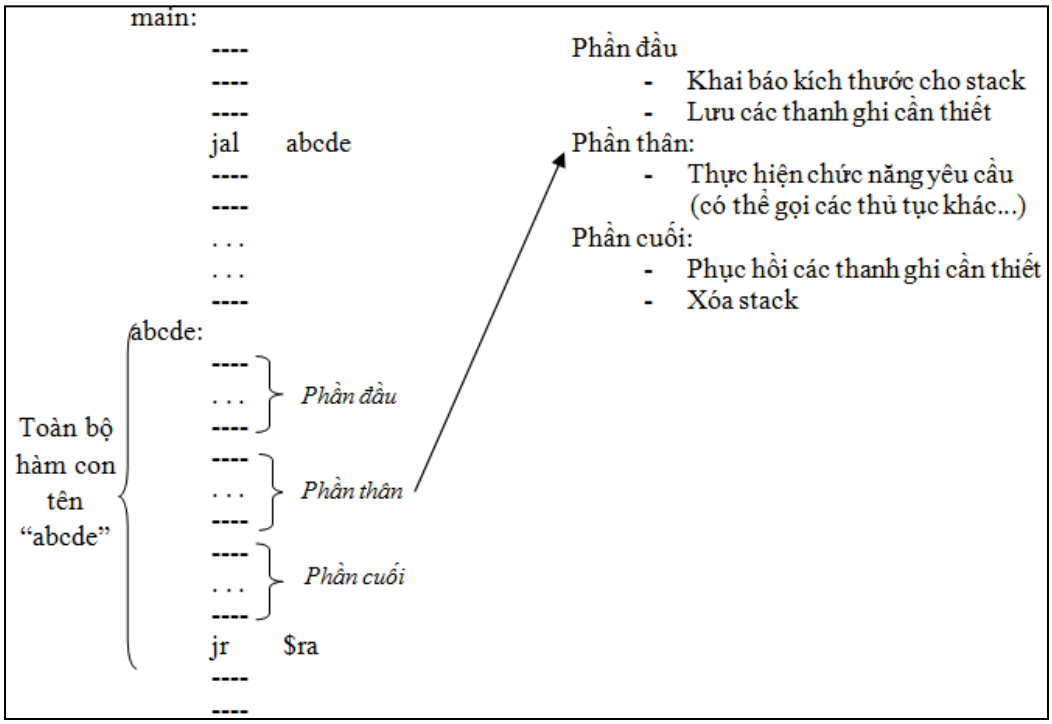
  Stack được xây dựng theo kiểu từ địa chỉ cao giảm dần xuống thấp, vì thế frame pointer luôn ở trên stack pointer.

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)

❖ **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.

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 trong hình 3 thực hiện việc nhập từ cửa sổ I/O một số nguyên vào thanh ghi $s0. Việc nhập được gọi thông qua hàm con tên getInt

```
        .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
        syscall
        jr $ra

exit:
```

Hình 3. Code cho ví dụ 1

2. Đoạn code trong hình 4 thực hiện phép toán (a + b) – (c + d)
   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 "example" để lấy kết quả của phép toán trên. Sau đó kết quả được xuất ra cửa sổ I/O

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

```
. . .
        move    $a0, $s0
        move    $a1, $s1
        move    $a2, $s2
        move    $a3, $s3
        jal     proc_example

        move    $a0, $v0
        li      $v0, 1
        syscall
. . .
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
```

Bốn tham số truyền cho hàm con proc_example trước khi gọi hàm. (Theo quy ước, tham số truyền cho hàm con phải truyền vào các thanh ghi $a0-$a3)

Sau khi hàm proc_example thực hiện xong, giá trị trả về của hàm theo quy ước chứa trong $v0

Phần đầu - tạo stack cho thủ tục:
- Do trong phần thân hàm có sử dụng thanh ghi $s0 nên trước khi tính toán, giá trị cũ của $s0 nên được lưu trong stack.
- Nếu có n thanh ghi cần lưu, mỗi thanh ghi một word, ta cần 4n byte cho stack. Trong trường hợp này, ta chỉ cần 4 byte.
- Vì stack phát triển theo kiểu từ cao xuống thấp, nên cấp phát 1 frame (1 word cho stack) thì $sp trừ 4

Phần thân
- Sau khi tính toán, kết quả trả về phải lưu vào $v0 như quy ước

Phần cuối
- Trước khi ra khỏi hàm, giá trị đang lưu của $s0 phải phục hồi lại; và 1 frame của stack phải được xóa đi bằng cách lấy $sp + 4

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)

Tóm lại:

Với một thủ tục/hàm con:

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

2. Phần 2

Chạy đoạn code như trong hình 3 và hình 4 trong MARS:

- Chạy từng bước và theo dõi sự thay đổi của thanh ghi PC, $ra, $sp, $fp
- Chạy toàn chương trình một lần để xem kết quả
- Với code trong hình 3, nếu bỏ dòng code "j exit", việc gì sẽ xảy ra?
- Viết lại code trong hình 3, 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 4, 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 4, 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)

3. Phần 3 - Hàm lồng hàm và đệ quy

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

1. 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 5.

```
main ()
{
   printf ("The factorial of 10 is %d\n", fact (10));
}

int fact (int n)
{
   if (n < 1)
      return (1);
   else
      return (n * fact (n - 1));
}
```

Hình 5. Tính giai thừa được viết bằng C

Vẽ lại hình ảnh của các stack trong trường hợp tính 5! Và 10!

2. 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 6.

```
int tak (int x, int y, int z)
{
    if (y < x)
        return 1+ tak (tak (x - 1, y, z),
          tak (y - 1, z, x),
          tak (z - 1, x, y));
    else
        return z;
}

int main ()
{
    tak(18, 12, 6);
}
```

Hình 6. Ví dụ được viết bằng C

Chú ý: Tham khảo phần "Procedure Call Example", trang B27, mục "Appendix B. Assemblers, Linkers, and the SPIM Simulator" của sách tham khảo chính môn Kiến Trúc Máy Tính.,

4. Phần 4 – Macro

Macro là gì, macro và thủ tục khác nhau như thế nào, cho 3 ví dụ và chạy ví dụ trên MARS?

(Tham khảo:

[1] – Appendix B, Computer Organization and Design: The Hardware/Software Interface, Fourth Edition, Patterson and Hennessy, Morgan Kauffman Publishers, 2011

[2] – Trong chương trình MARS, mở Help, chọn mục Macro và đọc hướng dẫn)

5. Phần 5

Bài tập thêm:

1. Viết lại tất cả các bài tập trong bài thực hành số 1 dưới dạng được hàm con nếu có thể, lưu ý:
   - Việc nhập vào từ cửa sổ I/O phải đưa vào hàm riêng, tương tự việc in ra cũng đưa vào hàm riêng
   - Các chức năng còn lại tùy từng yêu cầu và sinh viên tự phân tích
2. Viết chương trình nhập vào n và xuất ra chuỗi Fibonacci tương ứng

--------*Hết*--------