BUỔI THỨ 7 – Chương trình con (Subroutine)

Thời lượng: 3 tiết.

Biên soạn	*Trần Trung Tín*
Phiên bản	21.08.2016
Mọi góp ý đều được đón nhận chân thành, xin gửi email cho tôi đến tttin@tdt.edu.vn	

Lưu ý: không có một tài liệu hay phương pháp nào là chính thức nói về "Gọi/trả về trong MIPS". Hướng dẫn này là một trong những cách tổ chức thanh ghi, stack và các lệnh nhảy để hiện thực một số bài toán có sử dụng thủ tục, chương trình con.

Thanh ghi

* Nhắc lại các thanh ghi và cách sử dụng chúng, lưu ý các dòng tô xám là những thanh ghi dùng trong hướng dẫn này.

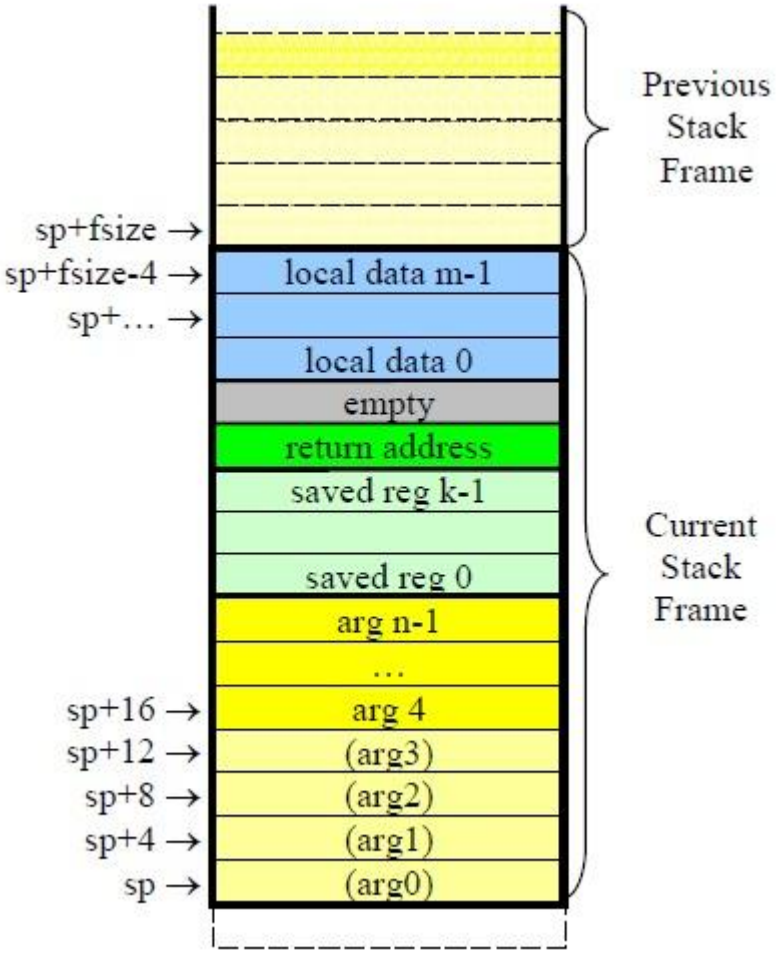
Register Number	Alternative Name	Description
0	zero	the value 0
1	$at	(**a**ssembler **t**emporary) reserved by the assembler
2-3	$v0 - $v1	(**v**alues) from expression evaluation and function results
4-7	$a0 - $a3	(**a**rguments) First four parameters for subroutine. Not preserved across procedure calls
8-15	$t0 - $t7	(**t**emporaries) Caller saved if needed. Subroutines can use w/out saving. Not preserved across procedure calls
16-23	$s0 - $s7	(**s**aved values) - Callee saved. A subroutine using one of these must save original and restore it before exiting. Preserved across procedure calls
24-25	$t8 - $t9	(**t**emporaries) Caller saved if needed. Subroutines can use w/out saving. These are in addition to $t0 - $t7 above. Not preserved across procedure calls.
26-27	$k0 - $k1	reserved for use by the interrupt/trap handler. KHÔNG SỬ DỤNG
28	$gp	**g**lobal **p**ointer. Points to the middle of the 64K block of memory in the static data segment.
29	$sp	**s**tack **p**ointer Trỏ đến đỉnh stack.
30	$s8/$fp	**s**aved value / **f**rame **p**ointer Preserved across procedure calls
31	$ra	**r**eturn **a**ddress Nhảy về địa chỉ.

Tổ chức stack

Ngăn xếp là không gian lưu trữ những tham số truyền nhận và lưu trữ tạm thời giá trị các thanh ghi của chương trình gọi trong lúc chờ chương trình được gọi hoàn tất thực thi. Một ngăn xếp (theo đề nghị) cần có các thành phần:

- Phần bộ nhớ lưu trữ các tham số truyền cho chương trình con.
- Nơi để lưu trữ giá trị các thanh ghi $s0 đến $s7.
- Nơi lưu trữ giá trị địa chỉ trở về ($ra).
- Nơi lưu trữ các biến số cục bộ của chương trình con.

Hình sau đây mô tả tổ chức các Khung ngăn xếp, chú ý đỉnh stack nằm từ phía dưới hình.



Stack được chia làm 5 vùng sử dụng:

1. **Phần tham số** của một ngăn xếp ngăn chứa khoảng trống để lưu trữ các tham số được truyền cho bất kỳ chương trình con nào được gọi bởi chương trình con hiện tại (ví dụ: chương trình con có ngăn xếp chồng lên trên đầu của ngăn xếp). Bốn Words đầu tiên của phần này không bao giờ được sử dụng bởi chương trình con hiện tại; chúng được dành riêng cho việc sử dụng bởi bất kỳ chương trình con nào được gọi bởi chương trình con hiện tại. (Gọi lại bốn đối số có thể được truyền cho một chương trình con trong các thanh ghi đối số ($a0 đến $a3) .Nếu có nhiều hơn bốn đối số, chương trình con hiện tại lưu trữ chúng trên stack, tại địa chỉ sp+16, sp+20, sp+24, v.v ...
2. **Phần các thanh ghi được lưu trữ** của ngăn xếp lưu các giá trị của các thanh ghi $s0 to $s7 trước khi nhảy đến chương trình con. Chương trình con đang thực thi có thể sử dụng bất cứ thanh ghi $s0 đến $s7, và có thể thay đổi giá trị của thanh ghi này. Nhưng ngay trước khi chương trình con trả về, các giá trị

của thanh ghi $s sẽ được phục hồi. Nói cách khác, với chương trình con được gọi thì các thanh ghi $s có tầm vực cục bộ, còn với chương trình gọi, các thanh ghi $s sẽ không bị thay đổi sau khi gọi một chương trình con.

3. Phần "Return Address" được sử dụng để lưu trữ giá trị địa chỉ sẽ trở về, $ra. Giá trị này được sao chép vào ngăn xếp khi bắt đầu thực thi một chương trình con và sao chép trở ra thanh thi $ra khi hoàn tất chương trình con.
4. Phần "Pad" là vùng trống để đảm bảo rằng phần tiếp theo Local Data Storage luôn có kích thước là bội số của 8 bytes.
5. Phần Local Data Storage dùng để lưu trữ các biến số địa phương. Chương trình con hiện tại phải dự trữ đủ kích thước lưu trữ trong khu vực này cho tất cả các dữ liệu cục bộ của nó, bao gồm không gian để lưu trữ giá trị của các thanh ghi tạm ($ t0 đến $ t9) khi mà nó lại gọi một chương trình con khác. Phần này có kích thước luôn là bộ số của 8 bytes.

Ví dụ "Simple Leaf"

Cho chương trình con.

```
int g( int x, int y ) {   return (x +
y);
}
```

```
g:    add   $v0,$a0,$a1 # result is sum of args
jr    $ra # return
```

Gọi hàm

```
//giả sử a, b đang lần lượt lưu trữ trong $a0, $a1.
j    g # go to g
lw    $t0, $v0  // $t9
```

Chương trình con bên trên không cần phải sử dụng ngăn xếp.

Ví dụ Leaf With Data

Cho hàm con, giả sử rằng mảng a[] là mảng địa phương dùng trong thân hàm con này.

```
int g( int x, int y ) {   int a[32];
  ... (calculate using x, y, a);   return a[0];
}
```

Hàm này không gọi chương trình con khác, nên không cần lưu trữ $ra, nhưng mảng a[] thì cần nên ngăn xếp sẽ được tổ chức để có phần không gian lưu trữ cho a[]

```
g:   # start of prologue
       addiu $sp,$sp,(-128) # push stack frame
       # end of prologue

       . . .                # calculate using $a0, $a1 and a
                            # array a is stored at addresses
                                 # 0($sp) to 124($sp)

       lw    $v0,0($sp)     # result is a[0]

       # start of epilogue          addiu $sp,$sp,128
# pop stack frame
       # end of epilogue
              jr    $ra             # return
```

// Phần đọc bổ sung LAB_7_advance.

Tính giai thừa (đệ qui)

```
# int fact(int n): return n <= 0 ? 1 : n * fact(n-1);
fact:
addi $sp, $sp, -8 # space for two words
sw $ra, 4($sp) # save return address
sw $a0, 0($sp) # temporary variable to hold n
li $v0, 1
ble $a0, $zero, fact_return
addi $a0, $a0, -1
jal fact
lw $a0, 0($sp) # retrieve original n
mul $v0, $v0, $a0 # n * fact(n - 1)
fact_return:
lw $ra 4($sp) # restore $ra
addi $sp, $sp, 8 # restore $sp
jr $ra # back to caller
```

Bài tập:

1. Hoàn tất đầy đủ đoạn code tính giai thừa bằng phương pháp đệ qui, chương trình cho phép nhận n có giá trị hợp lệ từ 0 đến 10, thông báo ra màn hình giá trị nhập không hợp lệ hoặc giá trị n! nếu n hợp lệ.

Dãy số Fibonacci (đệ qui)

```
# int fib(int n): return n < 2 ? n : fib(n-1) + fib(n-2)
fib: addi $sp, $sp, -8 # room for $ra and one temporary
sw $ra, 4($sp) # save $ra
move $v0, $a0 # pre-load return value as n
blt $a0, 2, fib_rt # if(n < 2) return n
sw $a0, 0($sp) # save a copy of n
addi $a0, $a0, -1 # n - 1
jal fib # fib(n - 1)
lw $a0, 0($sp) # retrieve n
sw $v0, 0($sp) # save result of fib(n - 1)
addi $a0, $a0, -2 # n - 2
jal fib # fib(n - 2)
lw $v1, 0($sp) # retrieve fib(n - 1)
add $v0, $v0, $v1 # fib(n - 1) + fib(n - 2)
fib_rt: lw $ra, 4($sp) # restore $ra
addi $sp, $sp, 8 # restore $sp
jr $ra # back to caller
```

Bài tập:

2. Hoàn tất đầy đủ đoạn code trên, chương trình cho phép nhận n có giá trị hợp lệ từ 0 đến 100, thông báo ra màn hình giá trị nhập không hợp lệ hoặc in n số Fibonacci ra màn hình.