

**CPS 104**  
**Computer Organization**  
**Lecture-8 : Procedure Calls**

**Sep. 22, 1999**

**Dietolf (Dee) Ramm**

**<http://www.cs.duke.edu/~dr/cps104.html>**

# Overview of Today's Lecture:

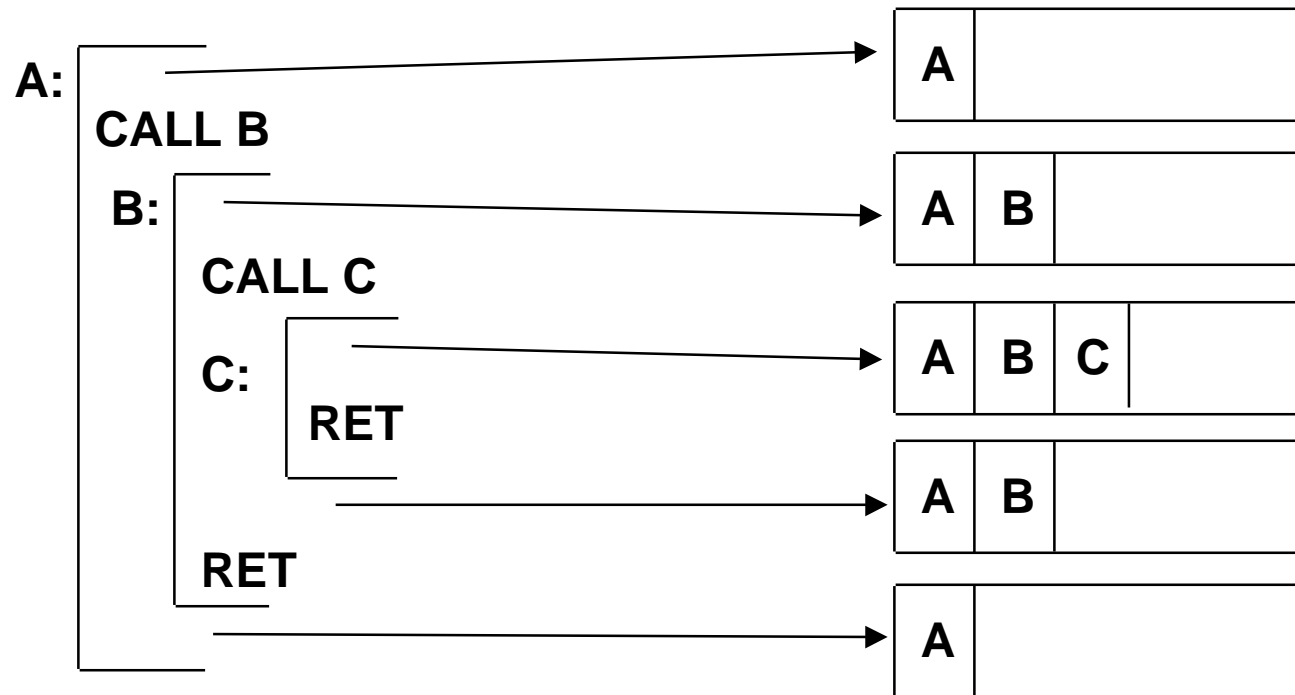
- **Data structures in assembler.**
- **The Stack**
- **Writing Functions in Assembler.**

# Administrivia

- Homework Due Sep 24.
- Look for UTAs “office hours”
- Look for on-line **SPIM manual** (Similar to Appendix).
- **On-line grades:** To come.

# Calls: Why Are Stacks So Great?

*Stacking of Subroutine Calls & Returns and Environments:*



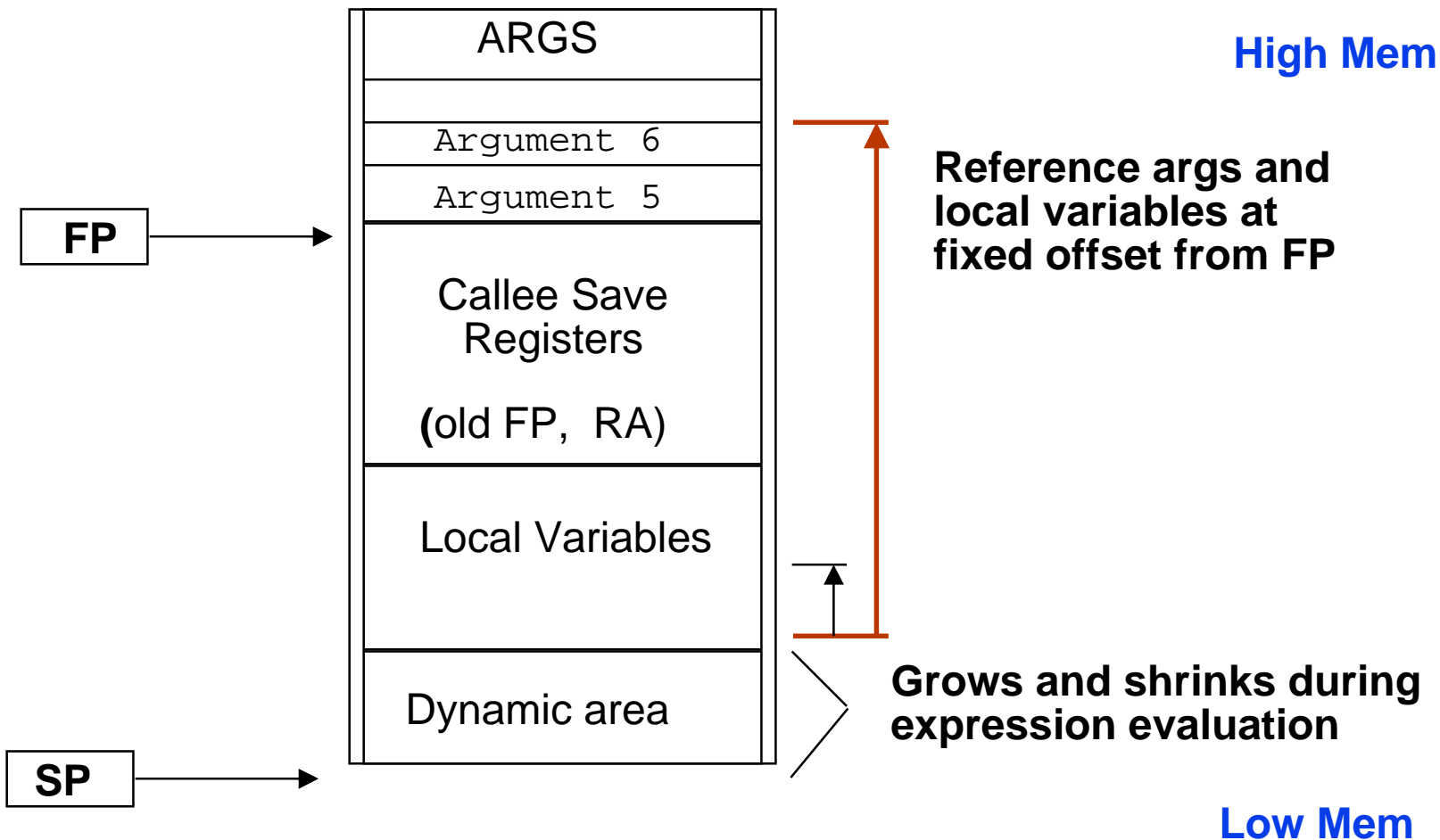
Some machines provide a memory stack as part of the architecture  
(e.g., VAX)

Sometimes stacks are implemented via software convention  
(e.g., MIPS)

# Procedure Call (Stack) Frame

- Procedures use a frame in the stack to:
  - Hold values passed to procedures as arguments.
  - Save registers that a procedure may modify, but which the procedure's caller does not want changed. (ex: **\$s0 - \$s7**)
  - Save the procedure return address (**\$ra**), and frame pointer (**\$fp**)
  - provide space for local variables (variables with local scope)
  - Used to evaluate complex expressions.
- There are two special registers **\$sp** and **\$fp** that are used as special data reference
  - The stack pointer **\$sp** points to the top of the stack.
  - The **\$fp** points to the the frame beginning.

# Call-Return Linkage: Stack Frames



- Many variations on stacks possible (up/down, last pushed / next )
- Block structured languages contain link to lexically enclosing frame
- **Compilers normally keep scalar variables in registers, not memory!**

# MIPS/GCC Procedure Calling Conventions

## Calling Procedure:

- Step-1: Pass the arguments:
  - The **first four arguments** are passed in registers **\$a0-\$a3**
  - Remaining arguments are pushed into the stack
    - ➔ (in reversed order **arg5** is at the top of the stack).
- Step-2: Save caller-saved registers
  - Save registers **\$t0-\$t9** if they contain live values at the call site.
- Step-3: Execute a **jal** instruction.

# MIPS/GCC Procedure Calling Conventions (cont.)

## Called Routine

- Step-1: Establish stack frame.
  - Subtract the frame size from the stack pointer.  
`subiu $sp, $sp, <frame-size>`
  - Typically, minimum frame size is 32 bytes (8 words).
- Step-2: Save callee saved registers in the frame.
  - Register `$fp` is always saved.
  - Register `$ra` is saved if routine makes a call.
  - Registers `$a0-$a3` are saved if they are changed.
  - Registers `$s0-$s7` are saved if they are used.
- Step-3: Establish Frame pointer `$fp`
  - Add the stack `<frame size -4>` to the address in `$sp`  
`addiu $fp, $sp, <frame-size> - 4`

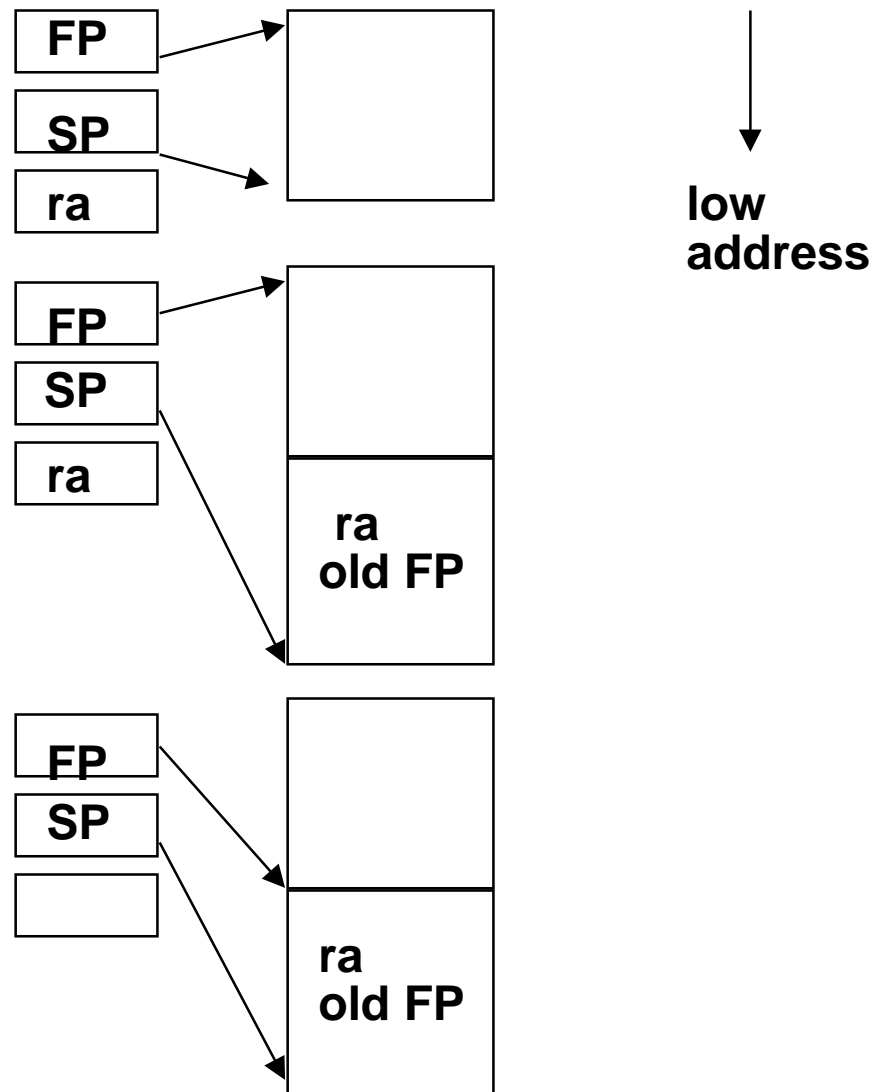
# MIPS/GCC Procedure Calling Conventions (cont.)

## On return from a call

- Step-1: Put returned values in registers `$v0`, [`$v1`].  
(if values are returned)
- Step-2: Restore callee-saved registers.
  - Restore `$fp` and other saved registers. [`$ra`, `$s0 - $s7`]
- Step-3: Pop the stack
  - Add the frame size to `$sp`.  
`addiu $sp, $sp, <frame-size>`
- Step-4: Return
  - Jump to the address in `$ra`.  
`jr $ra`

# MIPS / GCC Calling Conventions

```
fact:  
    subiu $sp, $sp, 32  
    sw    $ra, 20($sp)  
    sw    $fp, 16($sp)  
    addiu $fp, $sp, 28  
    . . .  
    sw    $a0, 0($fp)  
    ...  
    lw    $ra, 20($sp)  
    lw    $fp, 16($sp)  
    addiu $sp, $sp, 32  
    jr    $ra
```



**First four arguments are passed in registers.**

## Example: Factorial

```
main()  
{  
    printf("The factorial of 10 is  
    %d\n", fact(10));  
}  
  
int fact (int n)  
{  
    if (n < 1) return(1);  
    return (n * fact (n-1));  
}
```

```

.text
.global main
main:
    subiu    $sp, $sp, 32      #stack frame size is 32 bytes
    sw      $ra,20($sp)      #save return address
    sw      $fp,16($sp)      #save frame pointer
    addu    $fp, $sp,28      # set frame pointer

    li      $a0,10          # load argument (10) in $a0
    jal     fact            #call fact
    la     $a0,LC           #load string address in $a0
    move   $a1,$v0         #load fact result in $a1
    jal     printf          # call printf

    lw     $ra,20($sp)      # restore $sp
    lw     $fp,16($sp)      # restore $fp
    addu   $sp, $sp,32      # pop the stack
    jr     $ra             # exit()

.rdata
LC:
.asciiz "The factorial of 10 is %d\n"

```

```
.text
fact:
```

```
    subiu    $sp,$sp,32      # stack frame is 32 bytes
    sw      $ra,20($sp)     #save return address
    sw      $fp,16($sp)     #save frame pointer
    addiu   $fp, $sp,28     # set frame pointer
```

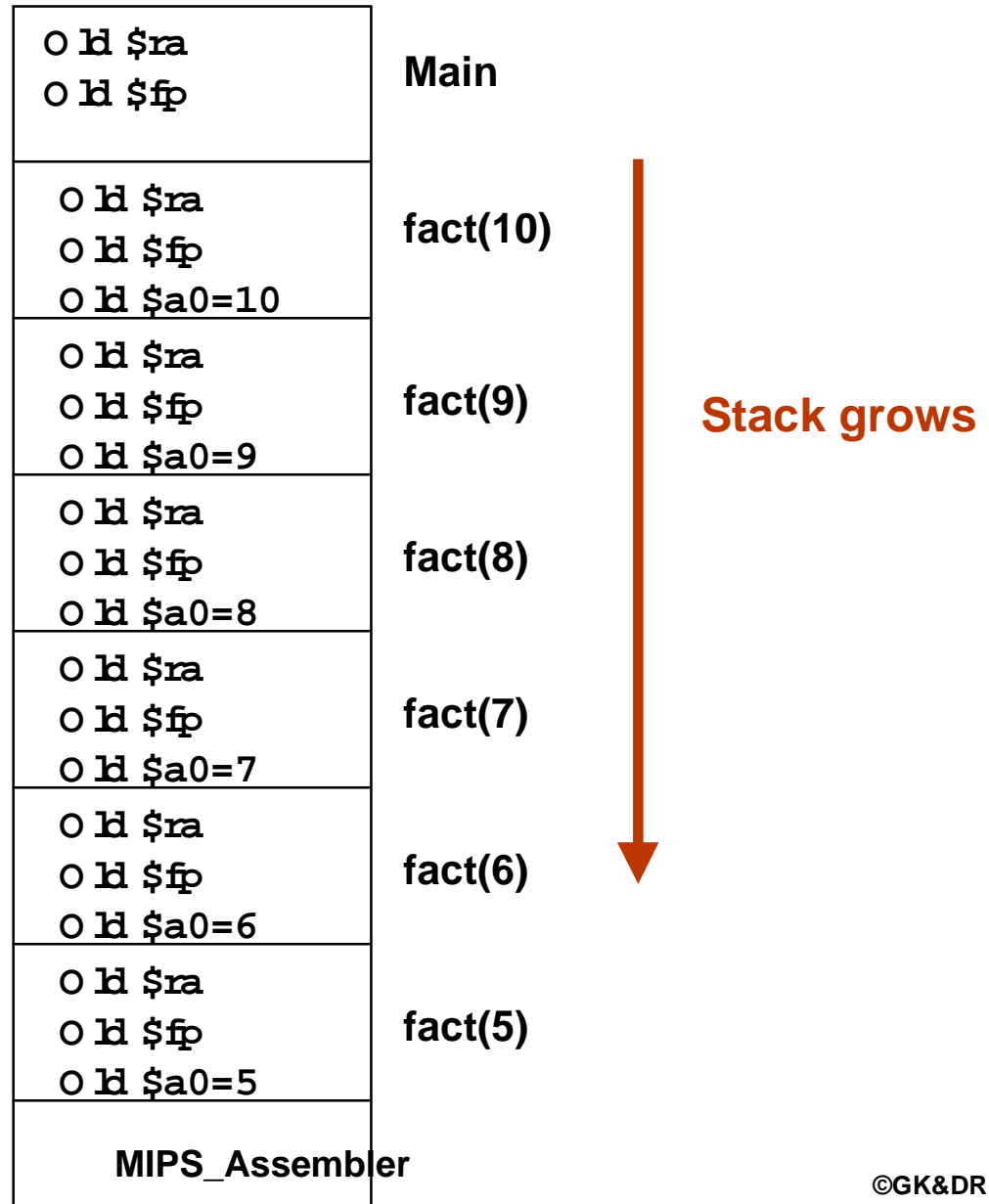
```
    sw      $a0,0($fp)     # save argument(n)
    lw      $v0,0($fp)     # load n
    bgtz    $v0, L2        # if n>0 go to $L2
    li      $v0, 1         #
    j       L1             # return(1)

L2:
    lw      $v1, 0($fp)    # load n
    sub     $v0,$v1,1      # compute n-1
    move    $a0,$v0        # load argument (n-1) in $a0
    jal    fact           # call fact
    lw      $v1,0($fp)     # load n
    mul     $v0,$v0,$v1    # fact(n-1)*n
```

```
L1:
    lw      $ra,20($sp)    # return (result in $v0)
    lw      $fp, 16($sp)  # restore $ra
    addiu   $sp,$sp,32    # restore $fp
    jr     $ra            # pop the stack
                                #return
```

# Example: Factorial

## Stack



# Example: Binary Tree of Homework-1

```
class Tree_node {
public:
    int me;                // node number
    Tree_node *left;      // left sub-tree pointer
    Tree_node *right;     // right sub-tree pointer
} ;

main()
{
    Tree_node *ar = new Tree_node[31]; // The tree has 31 nodes.
    Tree_node *p = ar;                // p = <top of the array>.
    int k;
    for (k = 0; k < 31; k++){        // initialize all nodes.
        ar[k].me = k;
        if( (2*k+2) < 31 ) {        // if it is an interior node,
            ar[k].left = &ar[2*k+1]; // set left and right pointers
            ar[k].right = &ar[2*k+2];
        }
        else{                        // if it is a leaf node
            ar[k].left = NULL;        // set the left and right pointers
            ar[k].right = NULL;      // to NULL.
        }
    }
    print_tree(p);                  // print the tree nodes in preorder.
}
```

## Example: Binary Tree of Homework-1(cont.)

```
void print_tree(Tree_node * p)
{
// a recursive procedure to print a binary tree in preorder.

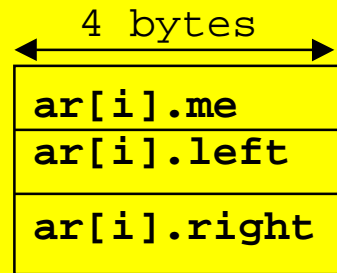
    if( p == NULL) return;           // if the tree is null, return

    cout <<  p << endl;             // print the node address

    print_tree(p->left);             // print the left subtree
    print_tree(p->right);           // print the right subtree
}
```

# Example: Binary Tree of Homework-1

```
class Tree_node {
public:
    int me;
    Tree_node *left;
    Tree_node *right;
};
```



```
&(ar[i].me)      = ?
&(ar[i].left)   = ?
&(ar[i].right)  = ?
```

```
Tree_node ar[31];
```

```
.align 2
.text
.globl main
main:
```

```
    subiu    $sp, $sp, 32      # stack frame 32 bytes
    sw      $ra, 20($sp)      # save return address
    sw      $fp, 16($sp)     # save old frame pointer
    addiu   $fp, $sp, 28     # set up new frame pointer
```

```
# allocate the tree node array:    ar = new Tree_node[31] ;
```

```
    li $a0 372                # 31 nodes, 12 bytes each.
    li $v0 9                  # get a block from sbrk
    syscall                   # $v0 points to the root $v0=ar ;
```

## Example: Binary Tree of Homework-1

```
# for (k=0; k<31 ; k++) {
    move $t0 $0           # k = 0;
LP:   mul  $t1, $t0, 12    # $t1 = k*12
      addu $t2, $v0, $t1  # $t2 = &ar[k]
      sw  $t0, 0($t2)     # ar[k].me = k ;
      mul $t3, $t0, 2     #
      addi $t3, $t3, 2    # $t3 = 2*k+2
      mul $t4, $t3, 12   #
      addu $t5, $v0, $t4  # $t5 = &ar[2*k+2];
      bge $t3, 32, L1    # if( (2*k+2) < 31 ) then{
      sw  $t5, 8($t2)    #     ar[k].right = &ar[2*k+2] ;
      subiu $t5, $t5, 12 #     $t5 = &ar[2*k+1] ;
      sw  $t5, 4($t2)    #     ar[k].left = &ar[2*k+1]}
      j   LD             #     go to LD
L1:   sw  $0, 4($t2)     # else { ar[k].left = NULL;
      sw  $0, 8($t2)     #         ar[k].right = NULL;}
LD:   addi $t0, $t0, 1   # k++
      blt $t0, 31 LP    # if k < 31 go to LP
#                                     end of for() loop
}
```

## Example: Binary Tree of Homework-1 (cont.)

```
move    $a0, $v0           # $a points to root node
jal     print_tree        # go print it

li      $v0 ,0             # do exit(0) ;
lw      $fp, 16($sp)      # restore old frame ptr
lw      $ra, 20($sp)      # restore return address
addiu   $sp, $sp, 32      # pop the stack frame
j       $ra
```

## Example (cont.)

```
.text
print_tree:
    subiu    $sp, $sp, 32      # stack frame 32 bytes
    sw      $ra, 20($sp)     # save return address
    sw      $fp, 16($sp)     # save old frame pointer
    addiu   $fp, $sp, 28     # set up new frame pointer
    sw      $a0, 12($sp)     # save node_ptr
    beqz    $a0, pr_done     # if null, return
    li      $v0, 1           # prepare to print int
    syscall                               # print it out
    li      $v0, 4
    la      $a0, mystr       #print end_of_line
    syscall
    lw      $a0, 12($sp)     # restore node_ptr
    lw      $a0, 4($a0)      # load left ptr
    jal     print_tree
    lw      $a0, 12($sp)     # restore node_ptr
    lw      $a0, 8($a0)     # load right ptr
    jal     print_tree

pr_done:                               # return code
    lw      $fp, 16($sp)     # restore old frame ptr
    lw      $ra, 20($sp)     # restore return address
    addu    $sp, $sp, 32     # remove frame
    j      $ra

.data
mystr:    .asciiz "\n"
```

# Simple functions

If a function does not call other functions one can simplify!

```
int min(int a, int b)
{
    if(a <= b) return(a) ;
    return(b);
}
```

```
min:    bgt  $a0, $a1, L1
        move $v0, $a0
        jr   $ra
L1:     move $v0, $a1
        jr   $ra
```