

CPS104 Computer Organization

Lecture 12

MIPS Assembly: Functions, Subroutines, Methods

September 30, 2009

Gershon Kedem

Administratrivia

- **Midterm Exam:** Wednesday October 7, in class, closed books.
 - ◆ Bring SPIM manual (or Appendix B)
 - ◆ Common MIPS instructions.
- **Homework 3 is posted. Due: October 12 (11:59 pm)**
Please note change of due date!
- **Reading: Chapter 2.**
- **Reading: Appendix A**
 - ◆ Available on: . . . cps104/Handouts/spim_appendix.pdf

Functions Subroutines & Methods

Review: MIPS/GCC Procedure Calling Conventions

Calling Procedure:

- **Step-1: Save caller-saved registers**
 - ◆ Save registers `$t0-$t9` if they contain **live values** at the call site.
- **Step-2: 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-3: Execute a `jal` instruction.**

Review: MIPS/GCC Procedure Calling Conventions (cont.)

Called Routine

- **Step-1: Establish stack frame.**
 - ◆ Subtract the frame size from the stack pointer.
`subiu $sp, $sp, <frame-size>`
 - ◆ Typically, minimum frame size is 32 bytes (8 words).
- **Step-2: Save callee saved registers in the frame.**
 - ◆ Register `$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.

Review: 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 `$sp` and other saved registers.
`[$ra, $a0-$a3, $s0-$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`

Review: Example2B

```
# Example for CPS 104
# Program to add together list of 9 numbers.

        .text                # Code
        .align 2
        .globl main

main:
        subu    $sp, 40      # MAIN procedure Entrance
                               # \ Push the stack
        sw     $ra, 36($sp)  # \ Save return address
        sw     $s3, 32($sp)  # \
        sw     $s2, 28($sp)  # > Entry Housekeeping
        sw     $s1, 24($sp)  # / save registers on stack
        sw     $s0, 20($sp)  # /
        move   $v0, $0       #/ initialize exit code to 0

        move   $s1, $0       #\
        la    $s0, list      # \ Initialization
        la    $s2, msg       # /
        la    $s3, list+36   #/
```

Example2B (cont.)

```
#                               Main code segment

again:                          #   Begin main loop
    lw      $t6, 0($s0)         #\
    add     $s1, $s1, $t6      #/   Actual "work"
                                #   SPIM I/O
    li     $v0, 4               #\
    move   $a0, $s2            # >   Print a string
    syscall                                #/
    li     $v0, 1               #\
    move   $a0, $s1            # >   Print a number
    syscall                                #/
    li     $v0, 4               #\
    la     $a0, nl             # >   Print a string (eol)
    syscall                                #/

    addiu  $s0, $s0, 4          #\   index update and
    bne   $s0, $s3, again      #/   end of loop
```

Example2B (cont.)

```
#                               Exit Code

    move    $v0, $0              # \
    lw      $s0, 20($sp)        # \
    lw      $s1, 24($sp)        # \
    lw      $s2, 28($sp)        # \ Closing Housekeeping
    lw      $s3, 32($sp)        # / restore registers
    lw      $ra, 36($sp)        # / load return address
    addu    $sp, 40              # / Pop the stack
    jr      $ra                  #/ exit(0) ;
    .end    main                 # end of program
```

```
#                               Data Segment

    .data                          # Start of data segment
list:  .word  35, 16, 42, 19, 55, 91, 24, 61, 53
msg:   .asciiz "The sum is "
nl:   .asciiz "\n"
```

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

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 $ra
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       $a0,28($sp)    # save argument(n) on stack
    lw       $v0,28($sp)    # load n from stack
    bgtz     $v0, L2        # if n>0 go to $L2
    li       $v0, 1         #
    j        L1             # return(1)
```

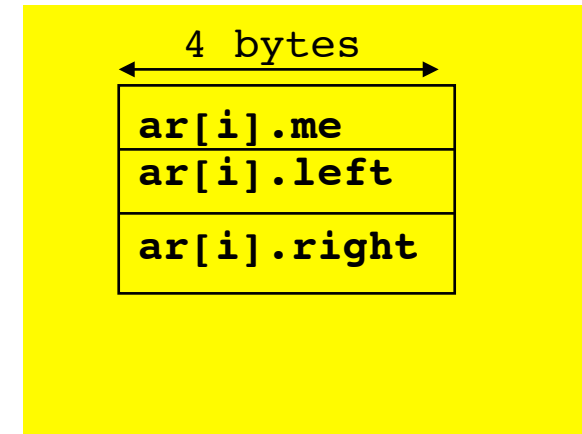
```
L2:
    lw       $v1,28($sp)    # load n
    sub      $v0,$v1,1      # compute n-1
    move     $a0,$v0        # load argument (n-1) into $a0
    jal     fact            # call fact
    lw       $v1,28($sp)    # load n
    mul      $v0,$v0,$v1    # fact(n-1)*n
```

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

Example: Binary Tree

```
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

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

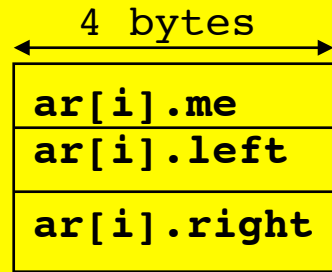
    if( p == NULL) return;           // if the tree is null, return

    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

```
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  
  
# 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

```
# 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

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

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

Example (cont.)

.text

print_tree:

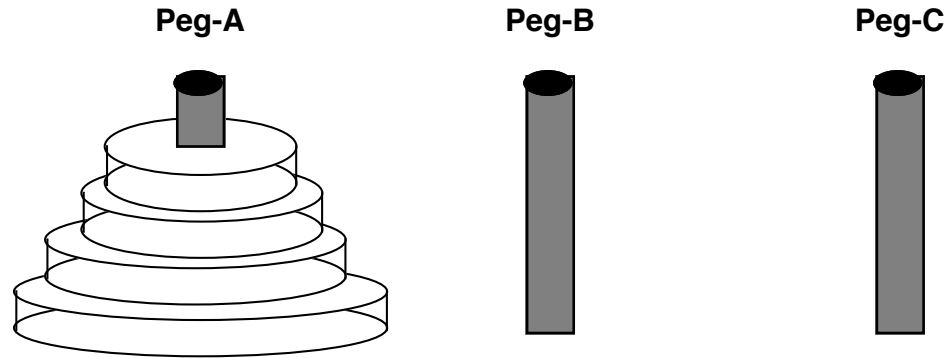
```
subiu  $sp, $sp, 32 # stack frame 32 bytes
sw     $ra, 20($sp) # save return address
sw     $a0, 12($sp) # save node_ptr
beqz   $a0, pr_done # if null, return
li     $v0, 1       # prepare to print int
syscall                                # print it out
li     $v0, 4
la     $a0, mystr   #print 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     $ra, 20($sp) # restore return address
addu   $sp, $sp, 32 # remove frame
j      $ra
```

.data

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

Example: Towers of Hanoi



The puzzle has the following recursive solution.

- ★ Move $N-1$ disks from **peg-a** to **peg-b** (using **peg-c** as auxiliary peg);
- ★ Move the largest disk from **peg-a** to **peg-c**;
- ★ Move $N-1$ disks from **peg-b** to **peg-c** (using **peg-a** as auxiliary peg);

Example: Towers of Hanoi

```
void hanoi (int n, char * a, char* b, char* c)
// Towers of Hanoi puzzle
// Move n disks from a to C using b as auxiliary peg
{
    if (n == 1) { // if one disk, move the disk from a to c
        cout << "move disk from " << a << " to " ;
        cout << c << endl ;
    }
    else{          // if n > 1
        hanoi(n-1, a, c, b) ; // move n-1 disks from a to b
        // move the last disk from a to c
        cout << "move disk from " << a << " to " ;
        cout << c << endl ;
        hanoi(n-1, b, a, c); // move n-1 disks from b to c
    }
}

main()
{    hanoi(10, "peg-A", "peg-B", "peg-C");}
```

```

# Towers of Hanoi MIPS assembly program
.align 2
.data          # Begin Data Segment
movedisk: .asciiz "Move disk from "
to: .asciiz " to "
nl: .asciiz "\n"
peg1: .asciiz "Peg-A"
peg2: .asciiz "Peg-B"
peg3: .asciiz "Peg-C"
.text         # Begin Text Segment
main:  subu $sp,$sp,28      # Push a frame into stack
      sw $ra,24($sp)      # save return address
      li $a0,10           # set n=10
      la $a1,peg1         # pass to hanoi 4 variables
      la $a2,peg2         # n, #-of disks in $a0
      la $a3,peg3         # names of pegs in $a1-$a3
      jal hanoi           # call hanoi
      lw $ra,24($sp)      # load return address
      addi $sp,$sp,28     # Pop the stack frame
      jr $ra              # exit

```

```

hanoi:
    addi $sp,$sp,-20    # Push frame into stack
    sw $ra,0($sp)      # save the return address
    sw $a0,4($sp)      # save n
    sw $a1,8($sp)      # save peg A
    sw $a2,12($sp)     # save peg B
    sw $a3,16($sp)     # save peg C

    li $t0,1           # make $t0 = 1
    bne $t0,$a0,recurse # if (n != 1) recurse

    li $v0,4           # print
    la $a0,movedisk    # "Move disk from "
    syscall            #
    move $a0,$a1        # copy peg-A into a0
    syscall            #
    la $a0,to          # print " to "
    syscall            #
    move $a0,$a3        # copy peg-C into a0
    syscall            #
    la $a0,\n          # print end-of-line
    syscall

endcase:              # exit code
    lw $ra,0($sp)      # load variables back
    lw $a0,4($sp)      # load n
    lw $a1,8($sp)      # load peg-A
    lw $a2,12($sp)     # load peg-B
    lw $a3,16($sp)     # load peg-C
    addi $sp,$sp,20    # Pop the stack frame
    jr $ra             # return

```

```

recurse:
    lw $a0,4($sp)           # Load n (num disks)
    addi $a0,$a0,-1        # n=(n-1)
    lw $a1,8($sp)          # load peg-A
    lw $a2,16($sp)         # load peg-C
    lw $a3,12($sp)         # load peg-B
    jal hanoi              # call hanoi

    li $v0,4               # print
    la $a0,movedisk        # "Move disk from "
    syscall                #
    move $a0,$a1           # copy peg-A into a0
    syscall                #
    la $a0,to              # print " to "
    syscall                #
    move $a0,$a3           # copy peg-C into a0
    syscall                #
    la $a0,nln             # print end-of-line
    syscall

    lw $a0,4($sp)          # Load n (num disks)
    addi $a0,$a0,-1        # n=(n-1)
    lw $a1,12($sp)         # load peg-B
    lw $a2,8($sp)          # load peg-A
    lw $a3,16($sp)         # load peg-C
    jal hanoi              # call hanoi

j endcase                 # goto exit code (return)

```

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
```