

# Functions, part 1

## Lecture B14



Lecture notes section B1 4

2002-02-05

CSE1 303 Part B lecture notes

1

## Last time

- Pointers
  - implementation
  - pointer operations
    - dereference (\*) operator
    - address (&) operator
- Arrays
  - implementation
  - indices and addresses
  - relationship to pointers

2002-02-05

CSE1 303 Part B lecture notes

2

## In this lecture

- Functions
  - properties
- Implementing functions in MIPS
  - jal and jr instructions
- Using the system stack
- The function calling convention
- Stack frames

2002-02-05

CSE1 303 Part B lecture notes

3

## Why functions?

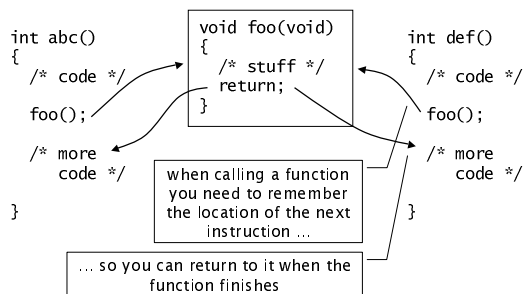
- The basic unit of code re-use
  - functions can be called repeatedly
  - functions can generalize by taking parameters
  - functions can inform through return values
  - functions are self-contained
- Abstraction of implementation
  - functions hide complexity from caller
  - functions can have their own private (local) variables/data
  - functions can call other functions
  - functions don't affect caller's environment

2002-02-05

CSE1 303 Part B lecture notes

4

## Function calling



2002-02-05

CSE1 303 Part B lecture notes

5

## Function calling in MIPS

- MIPS provides two instructions for function calling
  - Jump and link (jal): like jump (j), but first saves address of instruction following the jal in register \$ra

```
[0x004001B4] li $t0, 5
[0x004001B8] sw $t0, 0($sp)
[0x004001BC] jal foo
[0x004001C0] sw $v0, -8($fp)
[0x004001C4] add $sp, $sp, 4
```

```
[0x0041F304] foo: li $v0, 5
[0x0041F308] lw $t0, x
```

jal jumps to foo (at 0x0041F304) and puts address of next instruction (0x004001C0) in register \$ra

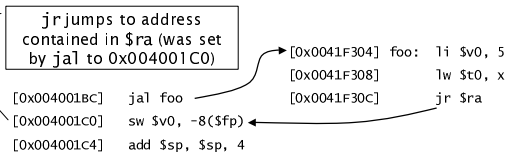
2002-02-05

CSE1 303 Part B lecture notes

6

## Function calling in MIPS

- MIPS provides two instructions for function calling
  - Jump register (jr) jumps to address specified by a register (usually \$ra)



2002-02-05

CSE1 303 Part B lecture notes

7

## Function calling in MIPS

- To write a function
  - put label at start of function
  - write body of function
  - end function with jr \$ra
- To call a function
  - write jal label
  - when function returns, program will continue from next instruction

2002-02-05

CSE1 303 Part B lecture notes

8

## Passing data

- Some functions take parameters
  - need a way of passing these parameters from caller to function
- Some functions return values
  - need a way of getting the return value safely back to caller
- Reserve some registers for these tasks
  - use system calls as prior art
  - pass parameters in \$a0, \$a1, \$a2, \$a3
  - return values in \$v0, \$v1
  - this convention has some problems
    - will make better system later

2002-02-05

CSE1 303 Part B lecture notes

9

## Example

```

# Program that uses a function.
.text
main: # Read int.
      li $v0, 5
      syscall
      move $t0, $v0

      # Pass parameter in $a0.
      move $a0, $t0
      # Calculate ($t0)!.
      jal fact # Do call.
      # Put result in $t1.
      move $t1, $v0

      # Print factorial.
      li $v0, 1
      move $a0, $t1
      syscall

      # Exit
      li $v0, 10
      syscall

# Factorial function.
fact: li $v0, 1 # For result.
      # Repeat until parameter
      # is <= 1.
loop: bgt $a0, 1, end
      # Multiply result by
      # parameter.
      mul $v0, $v0, $a0
      # Decrement parameter.
      sub $a0, $a0, 1
      # Lather, rinse, repeat.
      j loop
      # At end, $v0 contains
      # factorial.
end: jr $ra # Return.
    
```

2002-02-05

CSE1 303 Part B lecture notes

10

## Limitations

- This simple function-calling convention works, but has limits
  - function must not call other functions (i.e., it is a "leaf function")
  - function is limited to four parameters (\$a0-\$a3)
  - function must only write to "safe" registers \$v0-\$v1, \$a0-\$a3, \$t0-\$t9
  - function must not have/use local variables
- For more sophisticated function calling, need more sophisticated convention

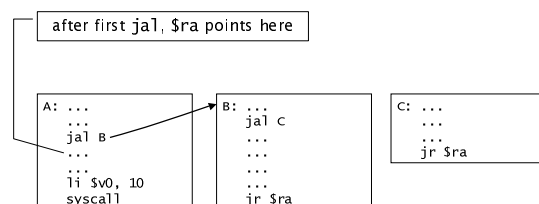
2002-02-05

CSE1 303 Part B lecture notes

11

## Saving the return address

- Problem: return address register (\$ra) can hold only one value



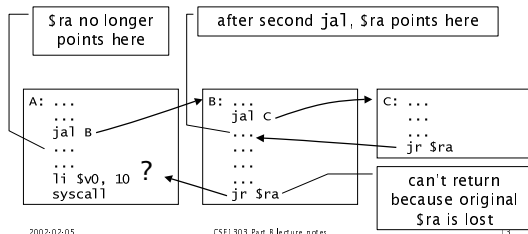
2002-02-05

CSE1 303 Part B lecture notes

12

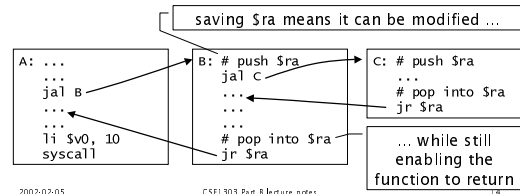
## Saving the return address

- Problem: return address register (\$ra) can hold only one value



## Saving the return address

- Problem: return address register (\$ra) can hold only one value
- Solution: save and restore \$ra register on function entry/exit



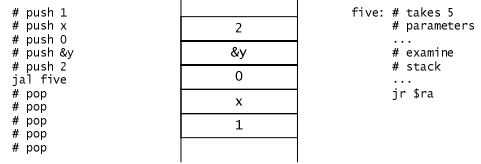
## Passing arguments

- Problem: not enough registers (4) to pass arguments to some functions



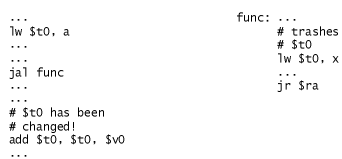
## Passing arguments

- Problem: not enough registers (4) to pass arguments to some functions
- Solution: pass arguments on stack



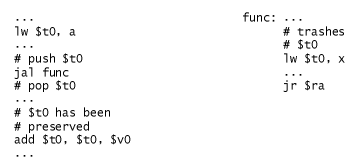
## Saving registers

- Problem: function may use a register which holds important value



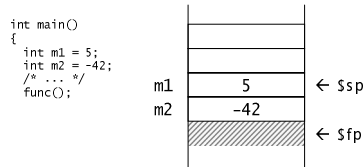
## Saving registers

- Problem: function may use registers which holds important values
- Solution: save/restore registers on stack



## Local variables

- Problem: function needs its own local variable storage



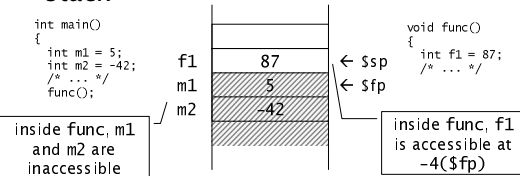
2002-02-05

CSEI 303 Part B lecture notes

19

## Local variables

- Problem: function needs its own local variable storage
- Solution: adjust \$fp to new top of stack



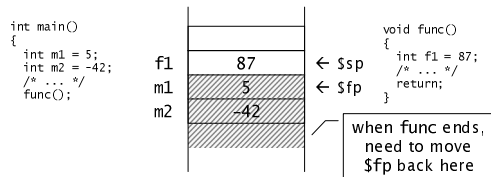
2002-02-05

CSEI 303 Part B lecture notes

20

## Saving the frame pointer

- Problem: on function return, stack state (including \$fp) must be restored



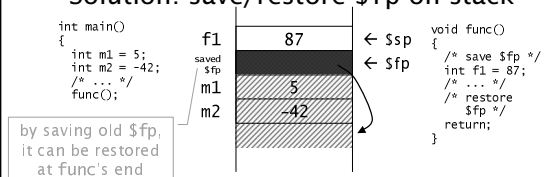
2002-02-05

CSEI 303 Part B lecture notes

21

## Saving the frame pointer

- Problem: on function return, stack state (including \$fp) must be restored
- Solution: save/restore \$fp on stack



2002-02-05

CSEI 303 Part B lecture notes

22

## Function calling convention

- These steps must be performed every time a function starts and returns
  - save/restore temporary registers
  - pass/clear arguments
  - save/restore \$ra register
  - save/restore \$fp register
  - allocate/deallocate local variables
- Need to assign responsibility to caller or callee to perform each task

2002-02-05

CSEI 303 Part B lecture notes

23

## Function calling convention

task	whose responsibility?
save/restore temporary registers	caller
pass/clear arguments	caller
save/restore \$ra	callee
save/restore \$fp	callee
allocate/deallocate local variables	callee

2002-02-05

CSEI 303 Part B lecture notes

24

## Function calling convention

- When calling a function, caller:
  1. saves temporary registers by pushing their values on stack
  2. pushes arguments on stack
  3. calls the function with `jal` instruction
- On function entry, callee:
  1. saves `$ra` by pushing its value on stack
  2. saves `$fp` by pushing its value on stack
  3. copies `$sp` to `$fp`
  4. allocates local variables

2002-02-05

CSE1 303 Part B lecture notes

25

## Example: caller

/\* C program which calls a function. \*/

```
#include <stdio.h>
#include <stdlib.h>
```

```
int power(int b, int e);
```

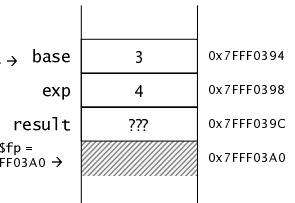
```
int main()
```

```
{
    int base;
    int exp;
    int result;

    scanf("%d", &base);
    scanf("%d", &exp);

    result = power(base, exp);
    printf("%d", result);
    exit(0);
}
```

assume user has entered 3 for base and 4 for exp



2002-02-05

CSE1 303 Part B lecture notes

26

## Example: caller

/\* C program which calls a function. \*/

```
#include <stdio.h>
#include <stdlib.h>
```

```
int power(int b, int e);
```

```
int main()
```

```
{
    int base;
    int exp;
    int result;

    scanf("%d", &base);
    scanf("%d", &exp);

    result = power(base, exp);
    printf("%d", result);
    exit(0);
}
```

```
main: .text
      # 3 * 4 = 12 bytes local
      move $fp, $sp
      subu $sp, $sp, 12

      li $v0, 5
      syscall
      sw $v0, -12($fp) # base

      li $v0, 5
      syscall
      sw $v0, -8($fp) # exp

      # Now we are up to the
      # function call ...
```

2002-02-05

CSE1 303 Part B lecture notes

27

## Example: caller

/\* C program which calls a function. \*/

```
#include <stdio.h>
#include <stdlib.h>
```

```
int power(int b, int e);
```

```
int main()
```

```
{
    int base;
    int exp;
    int result;

    scanf("%d", &base);
    scanf("%d", &exp);

    result = power(base, exp);
    printf("%d", result);
    exit(0);
}
```

```
main: .text
      # 3 * 4 = 12 bytes local
      move $fp, $sp
      subu $sp, $sp, 12

      li $v0, 5
      syscall
      sw $v0, -12($fp) # base

      li $v0, 5
      syscall
      sw $v0, -8($fp) # exp

      # Now we are up to the
      # function call ...
```

2002-02-05

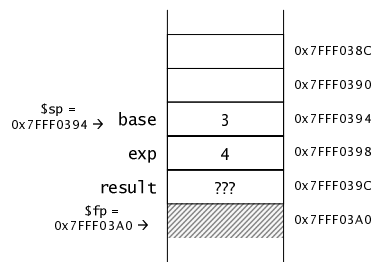
CSE1 303 Part B lecture notes

28

## Example: caller

caller step 1: save temporary registers by pushing their values on stack

not needed in this program



2002-02-05

CSE1 303 Part B lecture notes

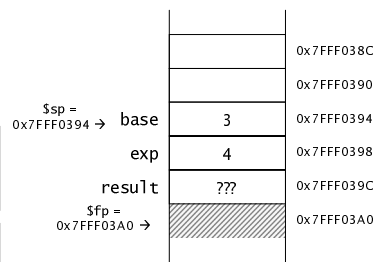
29

## Example: caller

caller step 2: push function arguments onto the stack

two arguments, called b and e

```
int power(int b, int e);
```



2002-02-05

CSE1 303 Part B lecture notes

30

### Example: caller

note offsets of arguments:  
b at 0(\$sp)  
e at 4(\$sp)

caller step 2: push function arguments onto the stack

two arguments, called b and e

```
int power(int b, int e);
```

\$sp = 0x7FFF038C →	arg 1 (b)	3		0x7FFF038C			
	arg 2 (e)	4		0x7FFF0390			
	base	3		0x7FFF0394			
	exp	4		0x7FFF0398			
	result	???		0x7FFF039C			
\$fp = 0x7FFF03A0 →				0x7FFF03A0			

2002-02-05CSEI 303 Part B lecture notes31

### Example: caller

caller step 3: call function with jal instruction

no effect visible on the stack

\$sp = 0x7FFF038C →	arg 1 (b)	3		0x7FFF038C			
	arg 2 (e)	4		0x7FFF0390			
	base	3		0x7FFF0394			
	exp	4		0x7FFF0398			
	result	???		0x7FFF039C			
\$fp = 0x7FFF03A0 →				0x7FFF03A0			

2002-02-05CSEI 303 Part B lecture notes32

### Example: caller

```
/* C program which calls a function. */
#include <stdio.h>
#include <stdlib.h>
int power(int b, int e);
int main()
{
    int base;
    int exp;
    int result;
    scanf("%d", &base);
    scanf("%d", &exp);
    result = power(base, exp);
    printf("%d", result);
    exit(0);
}
```

```
# ... continued from previous slide
# call function.
# push 2 * 4 = 8 bytes
# of arguments
subu $sp, $sp, 8
# arg 1 = base
lw $t0, -12($fp) # base
sw $t0, 0($sp) # arg 1
# arg 2 = exp
lw $t0, -8($fp) # exp
sw $t0, 4($sp) # arg 2
# call power function
jal power
# main function To Be
# Continued next lecture ...
```

2002-02-05CSEI 303 Part B lecture notes33

### Example: caller

```
/* C program which calls a function. */
#include <stdio.h>
#include <stdlib.h>
int power(int b, int e);
int main()
{
    int base;
    int exp;
    int result;
    scanf("%d", &base);
    scanf("%d", &exp);
    result = power(base, exp);
    printf("%d", result);
    exit(0);
}
```

```
# ... continued from previous slide
# call function.
# push 2 * 4 = 8 bytes
# of arguments
subu $sp, $sp, 8
# arg 1 = base
lw $t0, -12($fp) # base
sw $t0, 0($sp) # arg 1
# arg 2 = exp
lw $t0, -8($fp) # exp
sw $t0, 4($sp) # arg 2
# call power function
jal power
# main function To Be
# Continued next lecture ...
```

2002-02-05CSEI 303 Part B lecture notes34

### Example: callee

```
/* A function that gets called. */
int power(int b, int e)
{
    int result = 1;
    /* Keep going while exponent is positive. */
    while (e > 0)
    {
        /* Multiply by base. */
        result = result * b;
        /* less to multiply. */
        e--;
    }
    /* Return the result to the caller. */
    return result;
}
```

2002-02-05CSEI 303 Part B lecture notes35

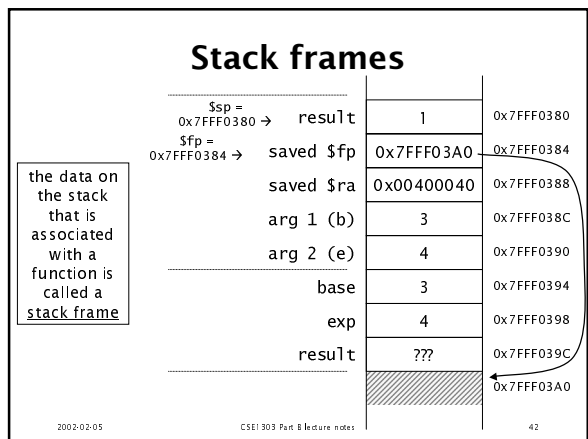
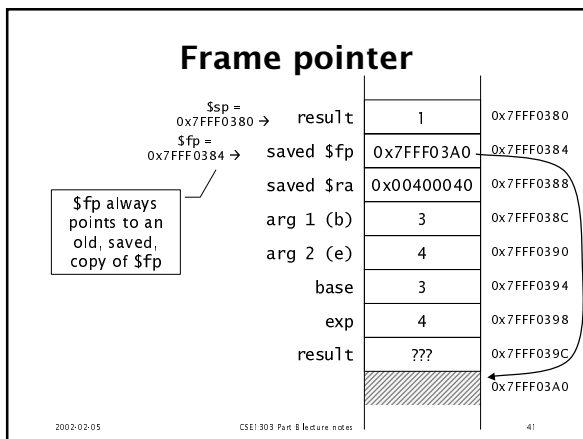
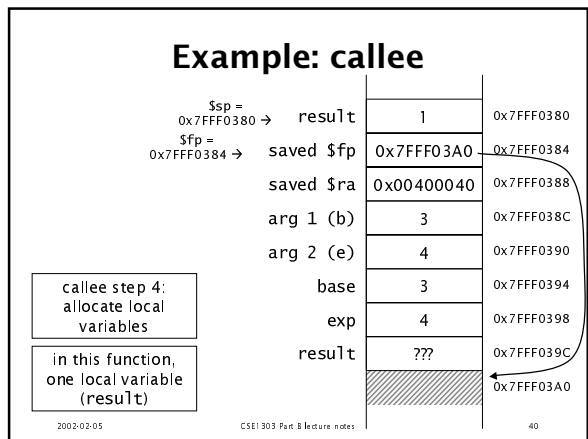
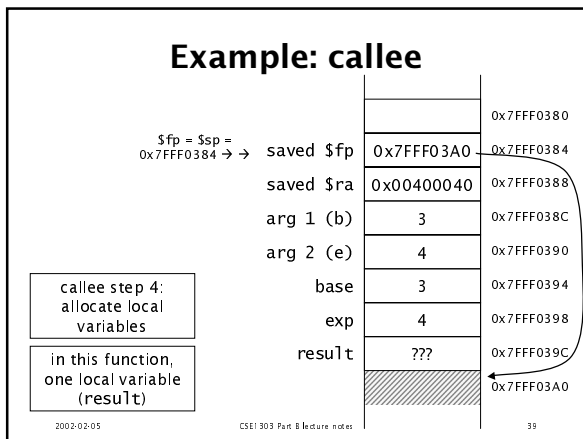
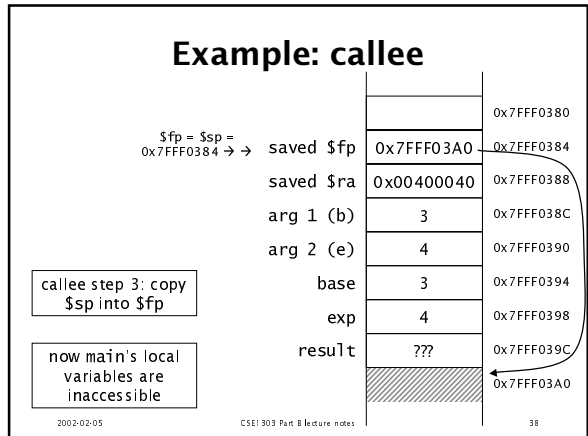
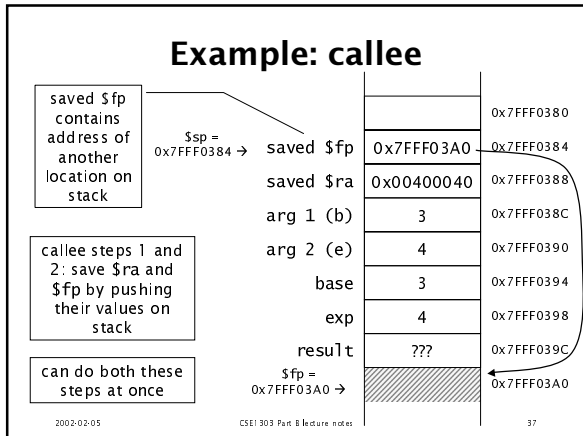
### Example: callee

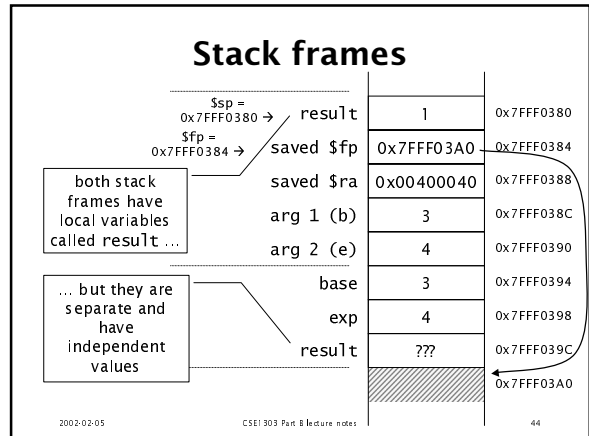
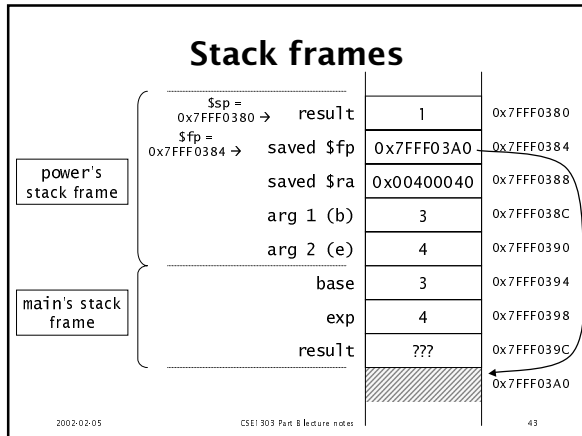
callee steps 1 and 2: save \$ra and \$fp by pushing their values on stack

can do both these steps at once

							0x7FFF0380
							0x7FFF0384
							0x7FFF0388
\$sp = 0x7FFF038C →	arg 1 (b)	3		0x7FFF038C			
	arg 2 (e)	4		0x7FFF0390			
	base	3		0x7FFF0394			
	exp	4		0x7FFF0398			
	result	???		0x7FFF039C			
\$fp = 0x7FFF03A0 →				0x7FFF03A0			

2002-02-05CSEI 303 Part B lecture notes36





### Example: callee

```

/* A function that gets called.
*/
int power(int b, int e)
{
    int result = 1;

    /* Keep going while exponent
    is positive. */
    while (e > 0)
    {
        /* Multiply by base. */
        result = result * b;

        /* less to multiply. */
        e--;
    }

    /* Return the result
    to the caller. */
    return result;
}

```

```

power: # Save $ra and $fp.
        subu $sp, $sp, 8
        sw $ra, 4($sp)
        sw $fp, 0($sp)

        # Copy $sp to $fp
        move $fp, $sp

        # Alloc local variables.
        # 1 * 4 = 4 bytes.
        subu $sp, $sp, 4

        # Initialize locals.
        li $t0, 1
        sw $t0, -4($fp) # result

        # Now we are inside
        # the function body.

        # Continued next lecture ...

```

2002-02-05 CSE1303 Part B lecture notes 45

### Example: callee

```

/* A function that gets called.
*/
int power(int b, int e)
{
    int result = 1;

    /* Keep going while exponent
    is positive. */
    while (e > 0)
    {
        /* Multiply by base. */
        result = result * b;

        /* less to multiply. */
        e--;
    }

    /* Return the result
    to the caller. */
    return result;
}

```

```

power: # Save $ra and $fp.
        subu $sp, $sp, 8
        sw $ra, 4($sp)
        sw $fp, 0($sp)

        # Copy $sp to $fp
        move $fp, $sp

        # Alloc local variables.
        # 1 * 4 = 4 bytes.
        subu $sp, $sp, 4

        # Initialize locals.
        li $t0, 1
        sw $t0, -4($fp) # result

        # Now we are inside
        # the function body.

        # Continued next lecture ...

```

2002-02-05 CSE1303 Part B lecture notes 46

- ### Covered in this lecture
- Function calling
    - jal and jr instructions
  - Calling convention
    - for making a function call
  - Structure of stack
    - stack frames
- 2002-02-05 CSE1303 Part B lecture notes 47

- ### Going further
- Relationship between \$t0-\$t9 and \$s0-\$s7
    - t-registers saved by caller
    - s-registers saved by callee
    - s-registers used to store frequently-accessed local variables, instead of stack
- 2002-02-05 CSE1303 Part B lecture notes 48

## Next time

- Function parameters
- Function return
- Recursion



Reading:  
Lecture notes section B1 5

2002-02-05

CSE1 303 Part B lecture notes

49

## Copyright

Copyright © 2001 Deborah Pickett.  
No part of this presentation may be  
duplicated without permission from  
the author.

2002-02-05

CSE1 303 Part B lecture notes

50