

# Functions, part 2

## Lecture B15



Lecture notes, section B15

2002-02-05

CSE1303 Part B lecture notes

1

## Last time

- 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

2

## In this lecture

- Accessing function parameters
- Returning from functions
- Recursion

2002-02-05

CSE1303 Part B lecture notes

3

## Function calling convention

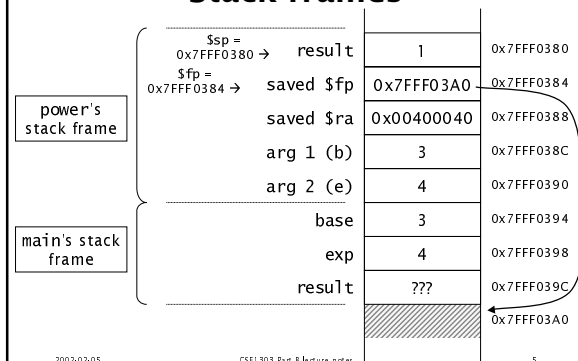
- When calling a function, caller:
  1. saves temporary registers by pushing their values on stack
  2. pushes arguments on stack
  3. calls the function with jal instruction
- On function entry, callee:
  1. saves \$ra by pushing its value on stack
  2. saves \$fp by pushing its value on stack
  3.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

2002-02-05

CSE1303 Part B lecture notes

4

## Stack frames

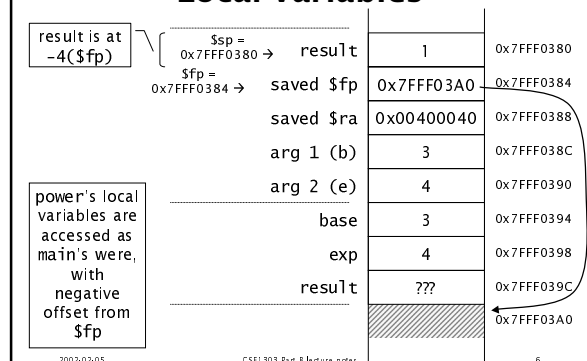


2002-02-05

CSE1303 Part B lecture notes

5

## Local variables

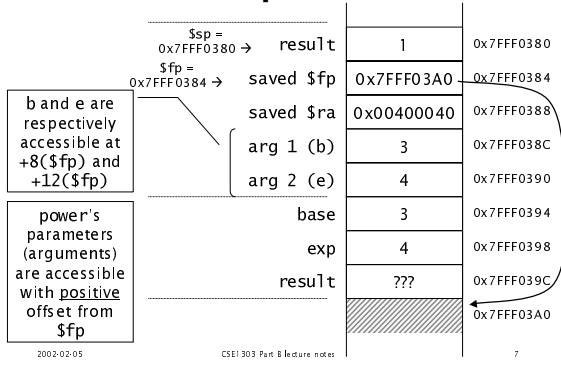


2002-02-05

CSE1303 Part B lecture notes

6

## Function parameters



## 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;
}
    
```

```

# ... Continued from
# last lecture.
loop: # Stop if e <= 0.
      lw $t0, 12($fp) # e
      ble $t0, 0, end

      # result = result * b
      lw $t1, -4($fp) # result
      mul $t0, $t0, $t1 # b
      sw $t0, -4($fp) # result

      # e--
      lw $t0, $t0, 12($fp) # e
      sub $t0, $t0, 1
      sw $t0, $t0, 12($fp) # e

      # Repeat loop.
      j loop

end: # Now ready to return.
     # Continued ...
    
```

2002-02-05 CSE1 303 Part B lecture notes 8

## 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;
}
    
```

```

# ... Continued from
# last lecture.
loop: # Stop if e <= 0.
      lw $t0, 12($fp) # e
      ble $t0, 0, end

      # result = result * b
      lw $t1, -4($fp) # result
      mul $t0, $t0, $t1 # b
      sw $t0, -4($fp) # result

      # e--
      lw $t0, $t0, 12($fp) # e
      sub $t0, $t0, 1
      sw $t0, $t0, 12($fp) # e

      # Repeat loop.
      j loop

end: # Now ready to return.
     # Continued ...
    
```

2002-02-05 CSE1 303 Part B lecture notes 9

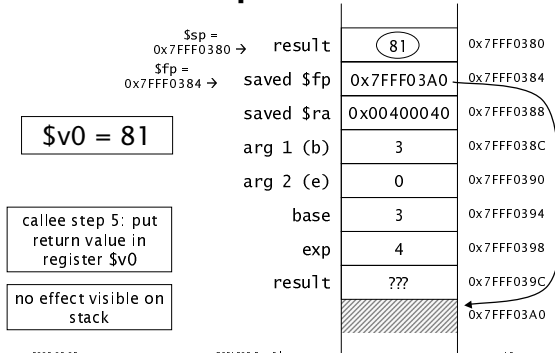
## Function return

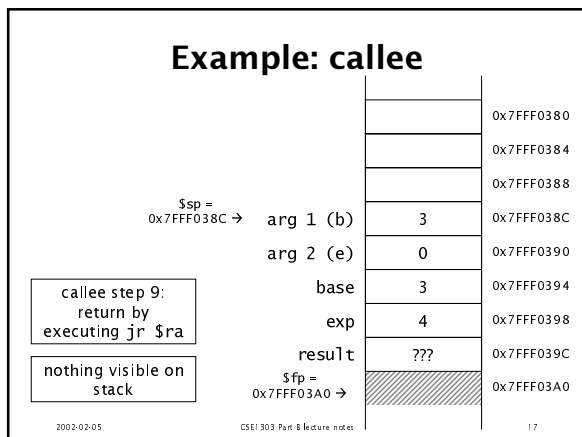
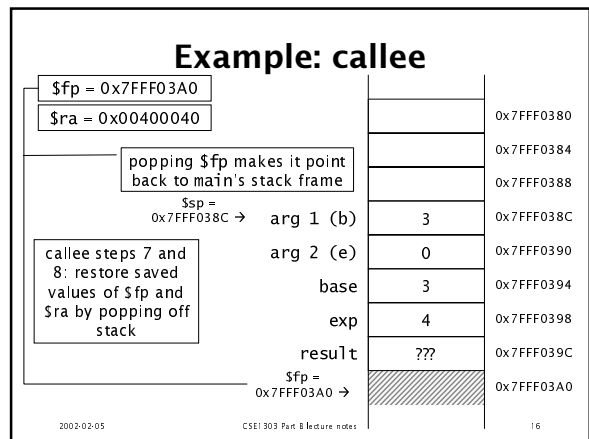
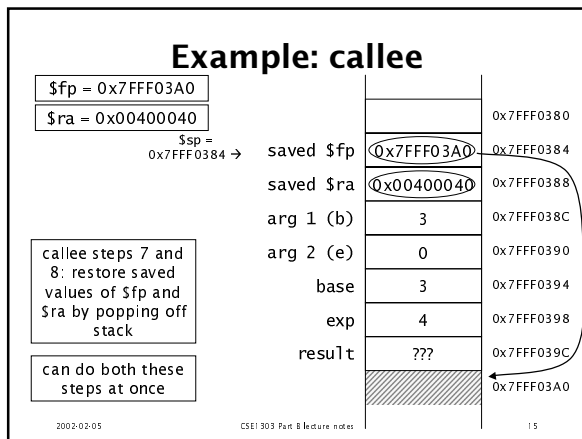
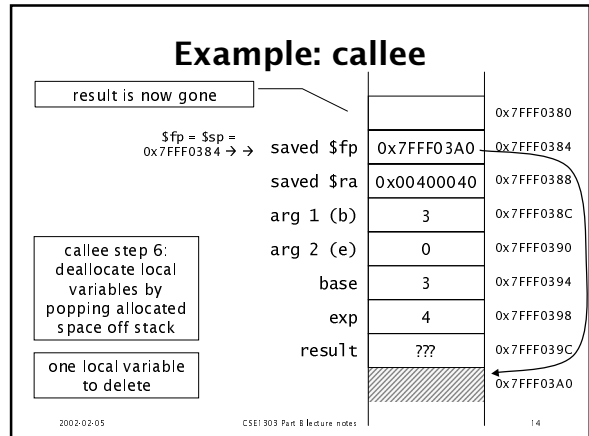
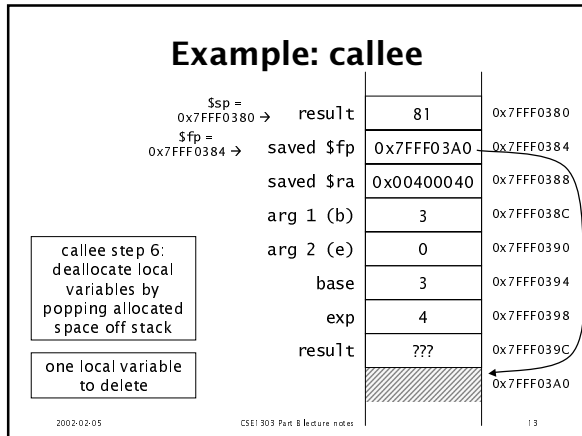
- When returning from a function, stack must be restored to its initial state
  - Achieved by undoing steps made during calling of function, in reverse order
- 2002-02-05 CSE1 303 Part B lecture notes 10

## Function return convention

- On function exit, callee:
    - chooses return value by setting register \$v0, if necessary
    - deallocates local variables by popping allocated space
    - restores \$fp by popping its saved value off stack
    - restores \$ra by popping its saved value off stack
    - returns with jr \$ra
  - On return from function, caller:
    - clears function arguments by popping allocated space
    - restores saved temporary registers by popping their values off stack
    - uses the return value found in \$v0, if necessary
- 2002-02-05 CSE1 303 Part B lecture notes 11

## Example: callee





### Example: caller

```

/* C program which calls a
function. */
#include <stdio.h>
#include <stdlib.h>

int power(int b, int e);

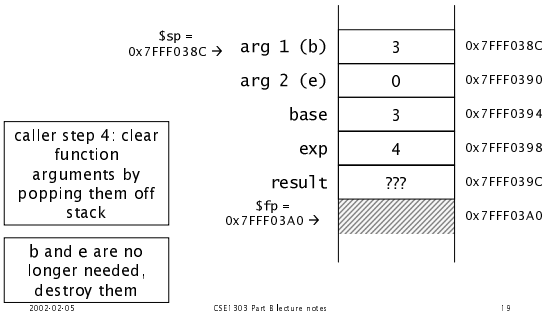
int main()
{
    int base;
    int exp;
    int result;

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

now back in caller, about to  
assign return value of  
function to main's local  
variable result

2002-02-05 CSE1 303 Part B lecture notes 18

### Example: caller

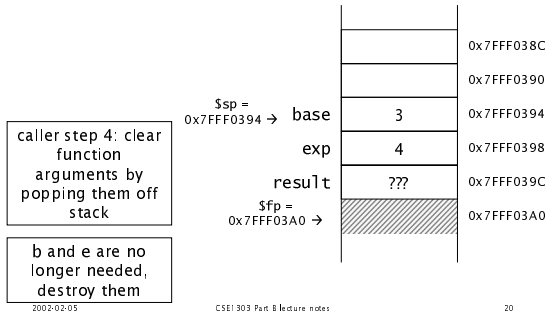


2002-02-05

CSE1303 Part B lecture notes

19

### Example: caller

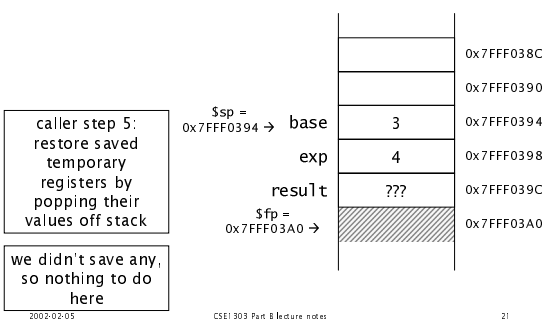


2002-02-05

CSE1303 Part B lecture notes

20

### Example: caller

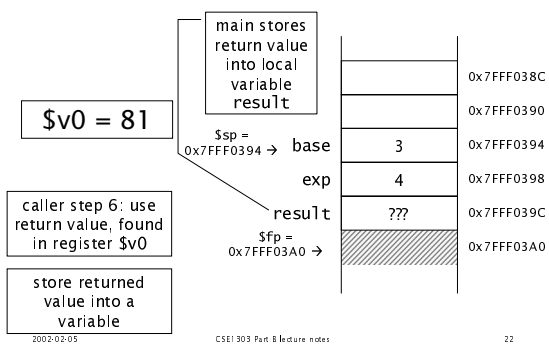


2002-02-05

CSE1303 Part B lecture notes

21

### Example: caller

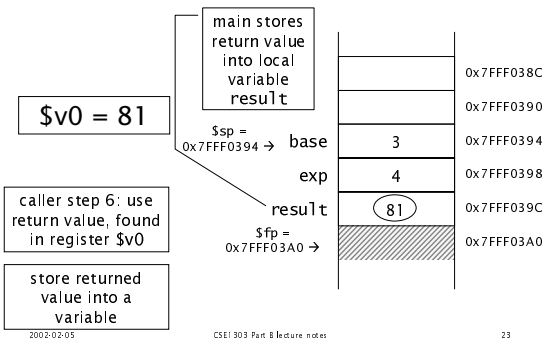


2002-02-05

CSE1303 Part B lecture notes

22

### Example: caller

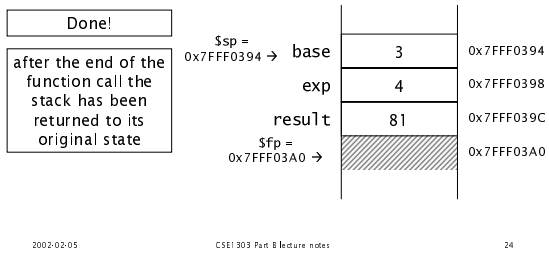


2002-02-05

CSE1303 Part B lecture notes

23

### Example: caller



2002-02-05

CSE1303 Part B lecture notes

24

## 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 function, continued
# from last lecture
# Last lecture, we
# finished with this ...
# jal power
# Remove arguments, they
# are no longer needed.
# 2 * 4 = 8 bytes.
addu $sp, $sp, 8
# Store return value
# in result.
sw $v0, -4($fp) # result
# Print result.
li $v0, 1
lw $a0, -4($fp) # result
syscall
li $v0, 10 # exit
syscall
```

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);
}
```

```
# ... main function, continued
# from last lecture
# Last lecture, we
# finished with this ...
# jal power
# Remove arguments, they
# are no longer needed.
# 2 * 4 = 8 bytes.
addu $sp, $sp, 8
# Store return value
# in result.
sw $v0, -4($fp) # result
# Print result.
li $v0, 1
lw $a0, -4($fp) # result
syscall
li $v0, 10 # exit
syscall
```

2002-02-05

CSE1 303 Part B lecture notes

26

## Function calling convention

▪ In summary, caller:

1. saves temporary registers by pushing their values on stack
2. pushes arguments on stack
3. calls the function with jal instruction
  - (function runs until it returns, then:)
4. clears function arguments by popping allocated space
5. restores saved temporary registers by popping their values off stack
6. uses the return value found in \$v0

2002-02-05

CSE1 303 Part B lecture notes

27

## Function calling convention

▪ In summary, 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
  - (body of function goes here, then:)
5. chooses return value by setting register \$v0
6. deallocates local variables by popping allocated space
7. restores \$fp by popping its saved value
8. restores \$ra by popping its saved value
9. returns with jr \$ra

2002-02-05

CSE1 303 Part B lecture notes

28

## Recursion

- Function calling convention works exactly the same for recursive functions
  - don't need to do anything special
- Each invocation of the function has its own stack frame
  - local variables and parameters
    - with their current values
  - return address
    - where to return to

2002-02-05

CSE1 303 Part B lecture notes

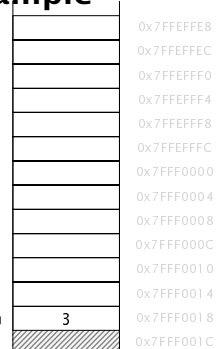
29

## Recursion example

```
/* Main program to call
factorial function. */
#include <stdlib.h>
#include <stdio.h>
int factorial(int p);
int main()
{
    int n;
    scanf("%d", &n);
    printf("%d", factorial(n));
    exit(0);
}
```

\$sp = 0x7FFF0018 → n

\$fp = 0x7FFF001C →



2002-02-05

CSE1 303 Part B lecture notes

30

## Recursion example

```

/* Main program to call
factorial function. */
#include <stdlib.h>
#include <stdio.h>

int factorial(int p);

int main()
{
    int n;
    scanf("%d", &n);
    printf("%d", factorial(n));
    exit(0);
}

```

\$sp = 0x7FFF0018 → arg 1 (p)  
 argument is at 0(\$sp)

\$fp = 0x7FFF001C → n

	0x7FFEF8
	0x7FFEFEC
	0x7FFEF0
	0x7FFEF4
	0x7FFEF8
	0x7FFEFCC
	0x7FFF000
	0x7FFF004
	0x7FFF008
	0x7FFF00C
	0x7FFF010
	0x7FFF014
	0x7FFF018
	0x7FFF01C

2002-02-05

CSEI 303 Part B lecture notes

31

## Recursion example: caller

```

/* Main program to call
factorial function. */
#include <stdlib.h>
#include <stdio.h>

int factorial(int p);

int main()
{
    int n;
    scanf("%d", &n);
    printf("%d", factorial(n));
    exit(0);
}

```

```

main:    .text
        # 1 * 4 = 4 bytes local.
        move $fp, $sp
        subu $sp, $sp, 4
        li $v0, 5
        syscall
        sw $v0, -4($fp) # n
        # call factorial.
        # 1 * 4 = 4 bytes arg.
        subu $sp, $sp, 4
        lw $t0, -4($fp) # n
        sw $t0, 0($sp) # arg 1
        jal factorial
        # Clear argument.
        addu $sp, $sp, 4
        move $a0, $v0 # returned
        li $v0, 1 # print int
        syscall
        li $v0, 10 # exit
        syscall

```

2002-02-05

CSEI 303 Part B lecture notes

32

## Recursion example: caller

```

/* Main program to call
factorial function. */
#include <stdlib.h>
#include <stdio.h>

int factorial(int p);

int main()
{
    int n;
    scanf("%d", &n);
    printf("%d", factorial(n));
    exit(0);
}

```

```

main:    .text
        # 1 * 4 = 4 bytes local.
        move $fp, $sp
        subu $sp, $sp, 4
        li $v0, 5
        syscall
        sw $v0, -4($fp) # n
        # call factorial.
        # 1 * 4 = 4 bytes arg.
        subu $sp, $sp, 4
        lw $t0, -4($fp) # n
        sw $t0, 0($sp) # arg 1
        jal factorial
        # Clear argument.
        addu $sp, $sp, 4
        move $a0, $v0 # returned
        li $v0, 1 # print int
        syscall
        li $v0, 10 # exit
        syscall

```

2002-02-05

CSEI 303 Part B lecture notes

33

## Recursion example: callee

```

/* The factorial function. */
int factorial(int p)
{
    int result;
    if (p <= 1)
    {
        /* Base case. */
        result = 1;
    }
    else
    {
        /* Recursive case. */
        result =
            factorial(p-1) * p;
    }
    return result;
}

```

2002-02-05

CSEI 303 Part B lecture notes

34

## Recursion example: callee

```

/* The factorial function. */
int factorial(int p)
{
    int result;
    ...
}

```

result is at -4(\$fp)

\$sp = 0x7FFF0008 → result

\$fp = 0x7FFF000C → saved \$fp

p is at 8(\$fp)

saved \$ra

	0x7FFEF8
	0x7FFEFEC
	0x7FFEF0
	0x7FFEF4
	0x7FFEF8
	0x7FFEFCC
	0x7FFF000
	0x7FFF004
	0x7FFF008
	0x7FFF00C
	0x00400028
	0x7FFF010
	0x7FFF014
	0x7FFF018
	0x7FFF01C

2002-02-05

CSEI 303 Part B lecture notes

35

## Recursion example: callee

```

/* The factorial function. */
int factorial(int p)
{
    int result;
    if (p <= 1)
    {
        /* Base case. */
        result = 1;
    }
    else
    {
        /* Recursive case. */
        result =
            factorial(p-1) * p;
    }
    return result;
}

```

```

factorial: # Function entry.
        subu $sp, $sp, 8
        sw $ra, 4($sp)
        sw $fp, 0($sp)
        move $fp, $sp
        # 1 * 4 = 4 bytes local.
        subu $sp, $sp, 4
        # if (p <= 1) ...
        lw $t0, 8($fp) # p
        bgt $t0, 1, rec
        # result = 1
        li $t0, 1
        sw $t0, -4($fp) # result
        j end
        # Continued ...

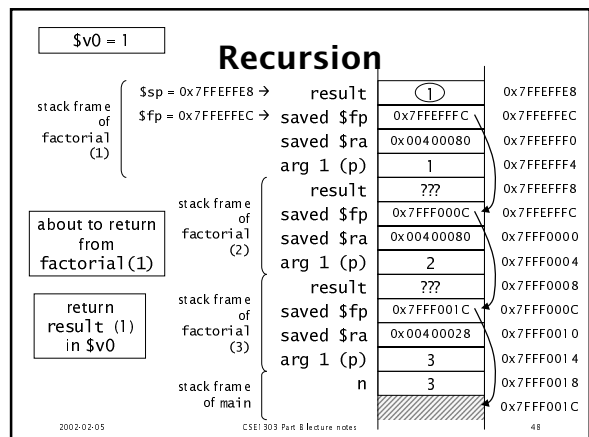
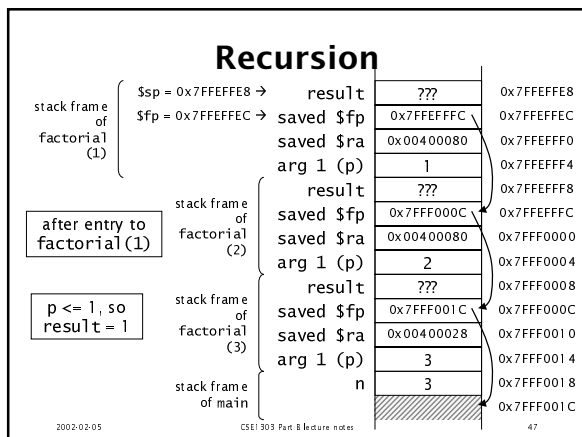
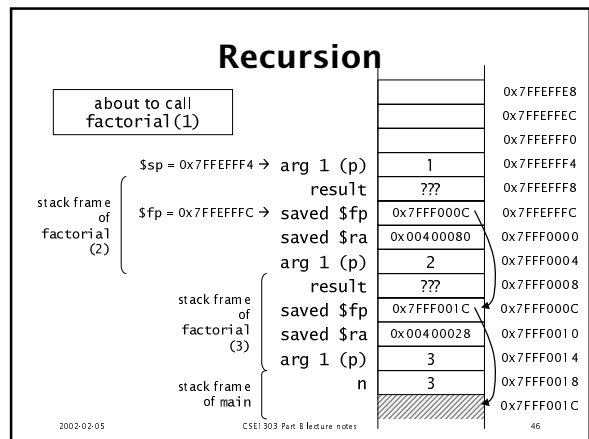
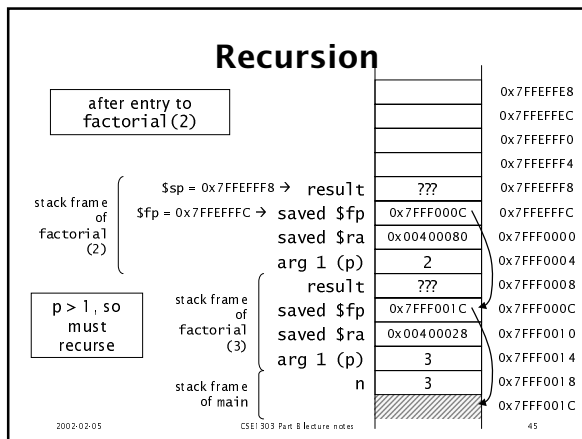
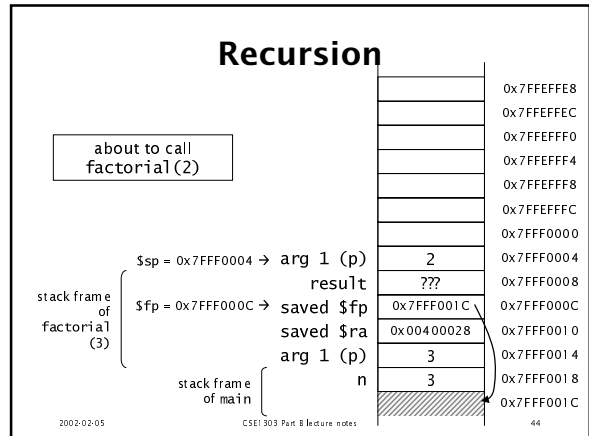
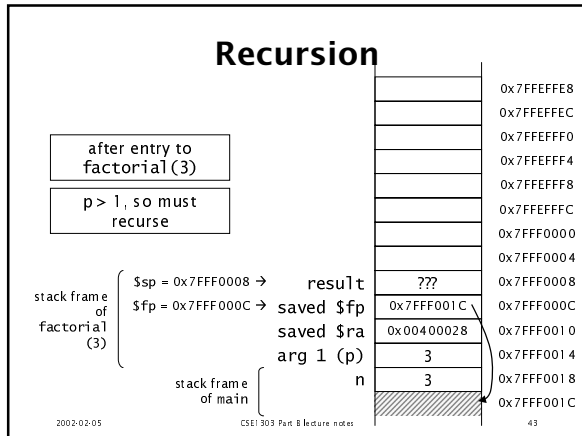
```

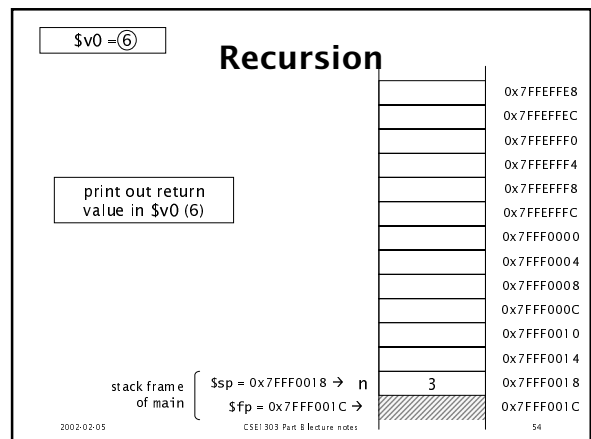
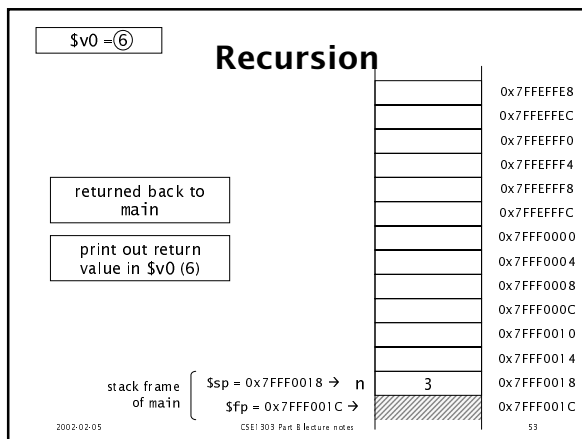
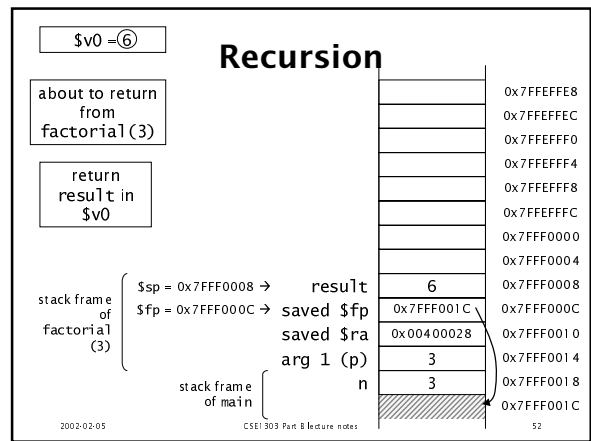
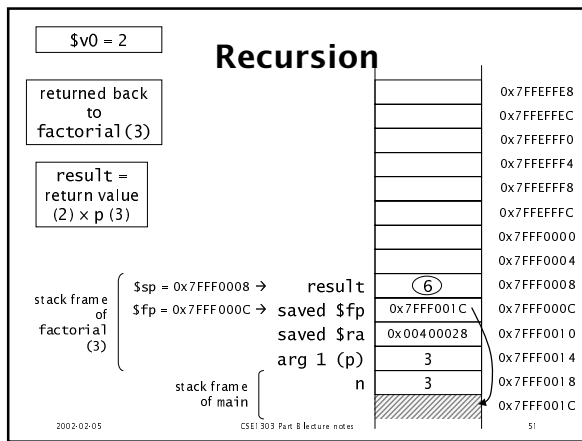
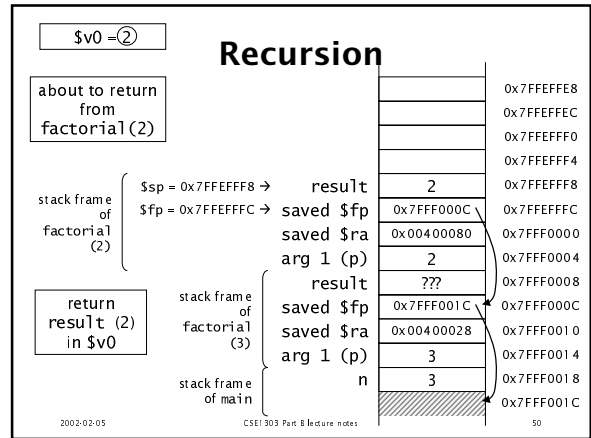
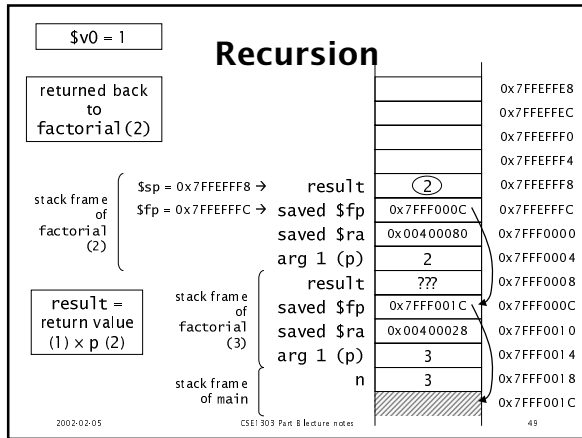
2002-02-05

CSEI 303 Part B lecture notes

36







## Covered in this lecture

- Accessing function parameters
- Returning from functions
- Recursion

2002-02-05

CSE1303 Part B lecture notes

55

## Going further

- Official MIPS stack frame convention
  - doesn't use \$fp at all!
  - slightly more efficient than CSE1303 convention
  - can be generated by compilers
  - hard for humans to write/understand

2002-02-05

CSE1303 Part B lecture notes

56

## Next time

- Analysis of translation
  - efficiency
- How to write better C code



Reading:  
Lecture notes section B1.6

2002-02-05

CSE1303 Part B lecture notes

57

## Copyright

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

2002-02-05

CSE1303 Part B lecture notes

58