CS305
Computer Architecture

Function Call Support in the Instruction Set

Bhaskaran Raman
Room 406, KR Building
Department of CSE, IIT Bombay

http://www.cse.iitb.ac.in/~br

Recall: MIPS Instructions So Far

Arithmetic: add, addi, sub

Logical: and, or, nor

sll, srl

memory: lw, sw

branch: beq, bne, j

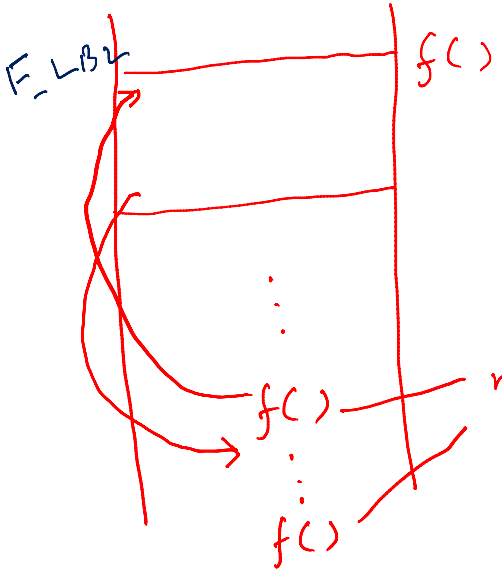
$0-$31

$s0-$s7

$t0-$t9

$zero

Basic Function Call Support: What is Needed?



many calls to the same function

function call, return

j is sufficient

remember the return address

jal F_LBL

part of MIPS set of regs

(1) $ra = PC+4

(2) transfer ctrl to F_LBL
PC = F_LBL

jr $ra

jump register

can be any of the 32 regs

Arguments and Return Values

among the 32 MIPS regs:

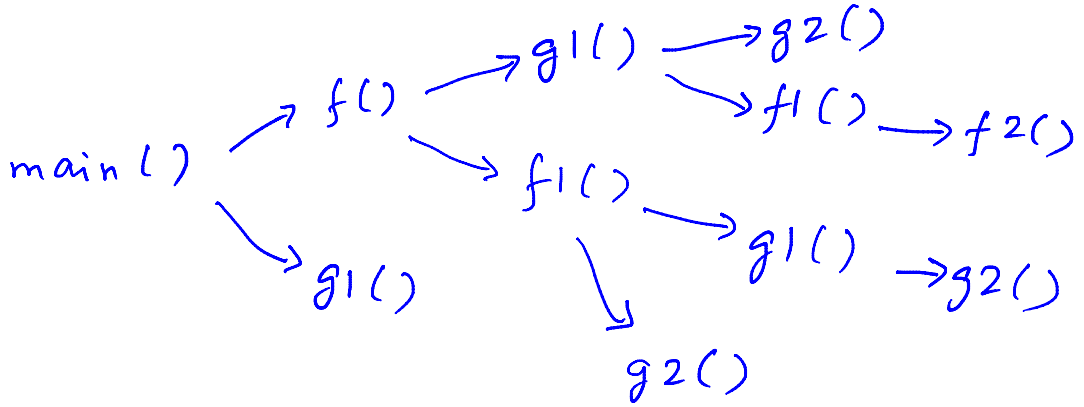
- $a0, $a1, $a2, $a3 – arguments
- $v0, $v1 – return values

Convention – not enforced by MIPS h/w

What About Nested Function Calls?

main() → f() → g()

$ra gets overwritten



Arbitrary nesting of function calls - where to store $ra?

Need for a Stack Data Structure

- before making a function call, store own return addr on to stack
- restore $ra from stack after function call

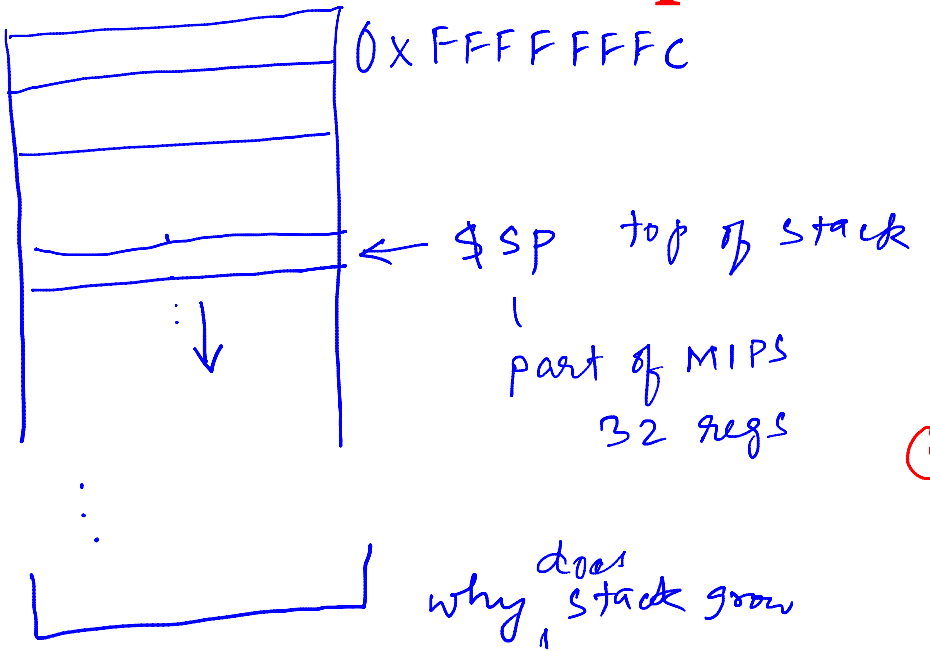
e.g. main → f() → g1()
f() → g2()

```
f():
    :
    save $ra onto stack
    jal g1
    restore $ra from stack
    :
    save $ra onto stack
    jal g2
    restore $ra from stack
    :
```

more efficient

```
f():
    save $ra onto stack
    :
    jal g1
    :
    jal g2
    :
    restore $ra from stack
```

Stack Implementation



Q: write code to push $ra onto stack

```
addi  $sp, $sp, -4
sw    0($sp), $ra
```

Q: write code to pop $ra from stack

```
lw    $ra, 0($sp)
addi  $sp, $sp, 4
```

Arguments and Return Values in Nested Calls

main() → f(...) → g(...)

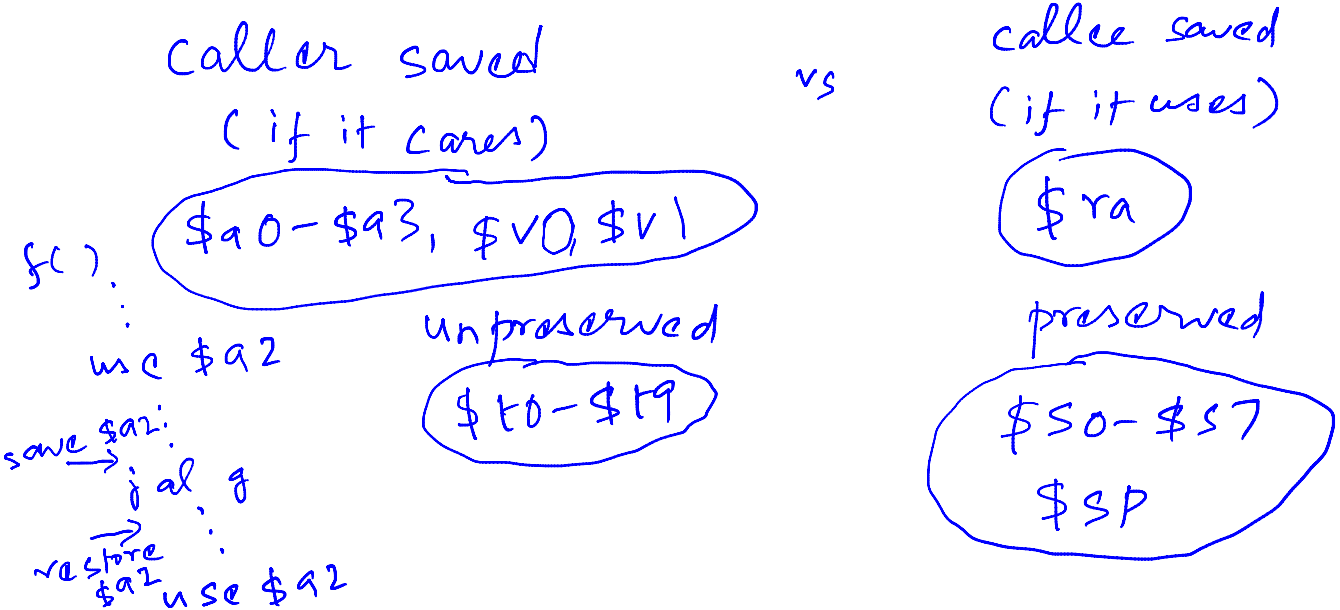
$a0-$a3, $v0, $v1 – caller has to save and restore if caller cares

What if arg/retval cannot be accommodated within given regs?

- use stack for these

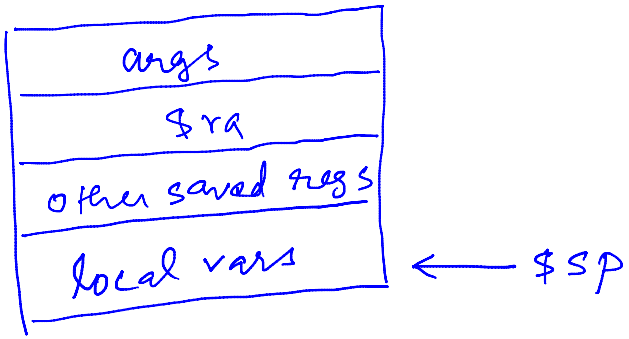
Caller vs Callee Saving Conventions

main() → f() → g()
caller callee caller callee

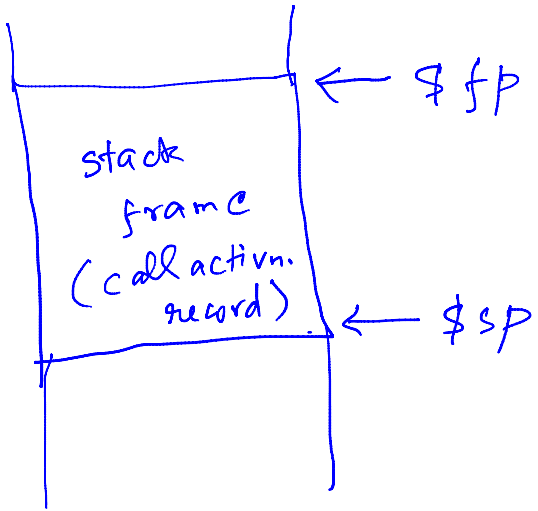


Stack Frame, or Call Activation Record

More arguments than 4 x 32 bit? Use stack
More return value than 2 x 32 bit? Use stack
Saving registers? Use stack
Local variables? Use stack



The Frame Pointer



← $fp – makes it easier for compiler

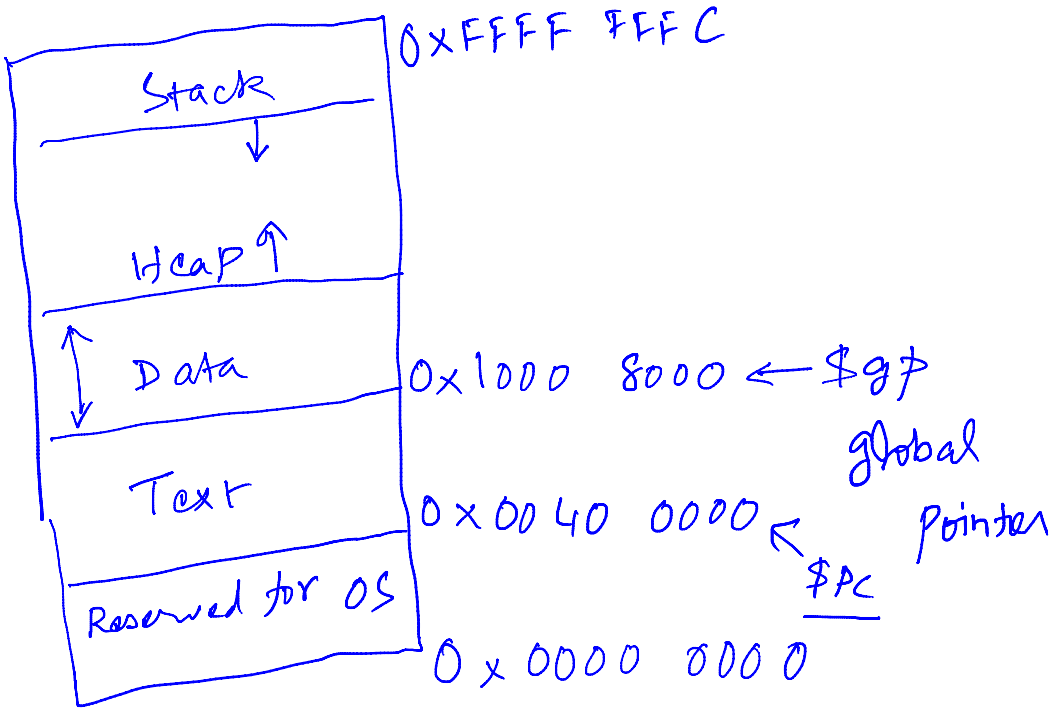
← $sp

```
f() {
    int x;
    :
L1 → x used here
    :
    for(int i=0; ... ) {
L2 →    x used here
    }
    :
} // End f()
```

Ref to x at L2 and L1
will be same w.r.t $fp
will be different w.r.t $sp

$fp – preserved or
callee saved

Memory Organization: Stack, Heap, Text



Recursion Example

```
int factorial(int n) {
  if(n == 0)
    return 1;
  else
    return n*factorial(n-1);
}
```

```
FACT:
addi   $sp, $sp, -8
sw     4($sp), $ra # $ra in stack
sw     0($sp), $a0 # $a0 in stack
bne    $a0, $zero, ELSE
addi   $v0, $zero, 1 # base case
j      EXIT
ELSE:
addi   $a0, $a0, -1
jal    FACT  # recursive call
lw     $a0, 0($sp) # restore $a0
mul    $v0, $v0, $a0
EXIT:
lw     $ra, 4($sp) # restore $ra
addi   $sp, $sp, 8 # restore $sp
jr     $ra   # return to caller
```

A Few More Details: Pseudo-Instructions, Assembler Temporary, Dealing with Bytes/Half-Words

li — addi
ori

beq $zero
beq bne
beq bnz

beq $s0, 10, EXIT

↓

assembler temporary

addi $at, $zero, 10
beq $s0, $at, EXIT

j FAR-AWAY

lui $at, something
ori $at, $at, something
jr $at

lb, lh
sb, sh

lbu, lhu
sbu? shu?

Summary So Far

- MIPS instruction set design
- Instruction encoding
- Instructions for function call support