

## 5. Instruction Set Architecture – from C to assembly – Part 2

---

EECS 370 – Introduction to Computer Organization – Winter 2009

**Prof. Todd Austin & Prof. Marios Papaefthymiou**

EECS Department  
University of Michigan, Ann Arbor, USA

© T. Austin & M. Papaefthymiou, 2009

The material in this presentation cannot be  
copied in any form without our written permission

### Instruction Set Architecture (ISA) Design Lectures

---

- ❑ Lecture 2: Storage types and addressing modes
- ❑ Lecture 3: LC-2K and MIPS architecture
- ❑ **Lecture 4 and 5: Converting C to assembly:**
  - **Data structures**
  - **Branch instructions**
  - **Calling functions / passing arguments**
- ❑ Lecture 6: Translation software; libraries, VMs
  
- ❑ Reminder: HW1 due in class (or under my office door by 1:30pm) on Tuesday 1/27!

## In part 1....

---

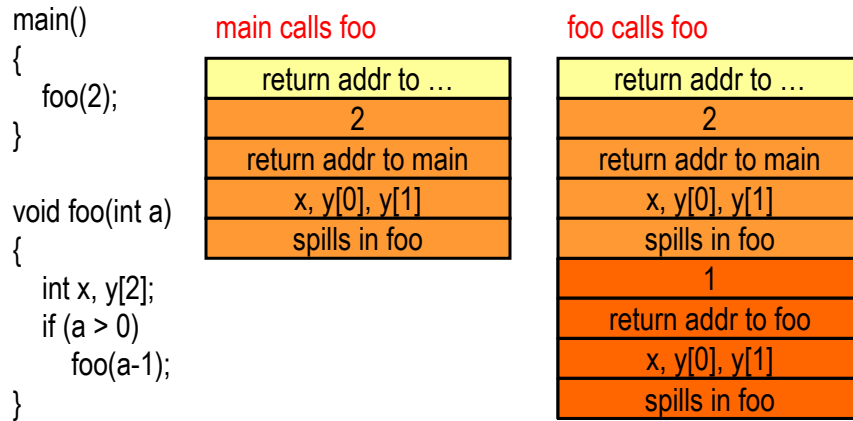
- ❑ Expression trees
- ❑ Structs
- ❑ Data layout in memory
- ❑ Branches
- ❑ Basic function calls – call stack
- ❑ Memory layout

## Call stack with recursion

---

- ❑ Each instance of the recursive function is given its own stack frame
  - Local variables in function are private to a particular invocation
  - Each has its own return address, spill space
  - Global and static variables are shared → there is 1 copy for all calls

## Recursive function example



## Saving registers during a call

- ❑ What happens to the values we have in registers when we make a function call?

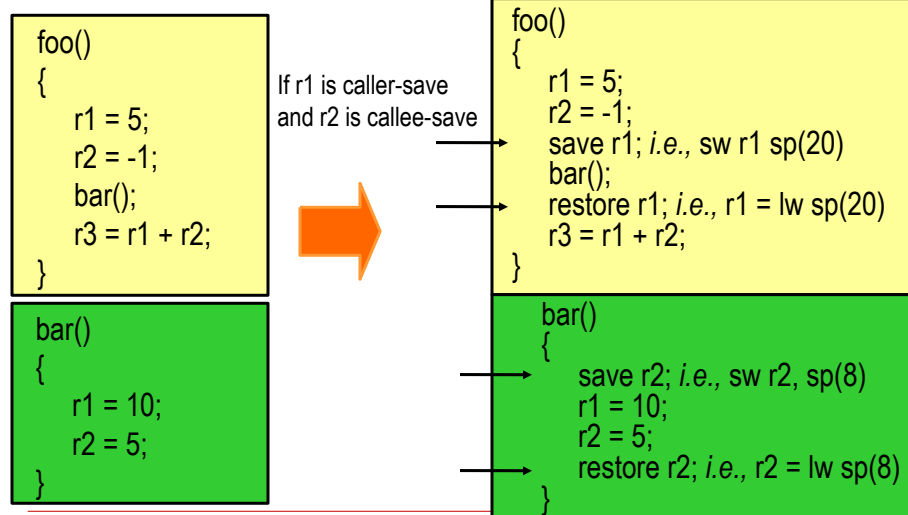
$$a = b * c + \text{sqrt}(d);$$

Options:

1. Caller can save registers **before** it makes the function call and restore the registers after the function returns (**caller-save register**).  
Where? The stack frame is used to store anything required to support function calls.
2. Callee can save registers **after** the function call and restore the registers just before it returns (**callee-save register**).  
Where? On the call stack.

What if the function called (i.e., the callee) doesn't use that register? No harm done, but wasted work!!!

## Caller / callee example

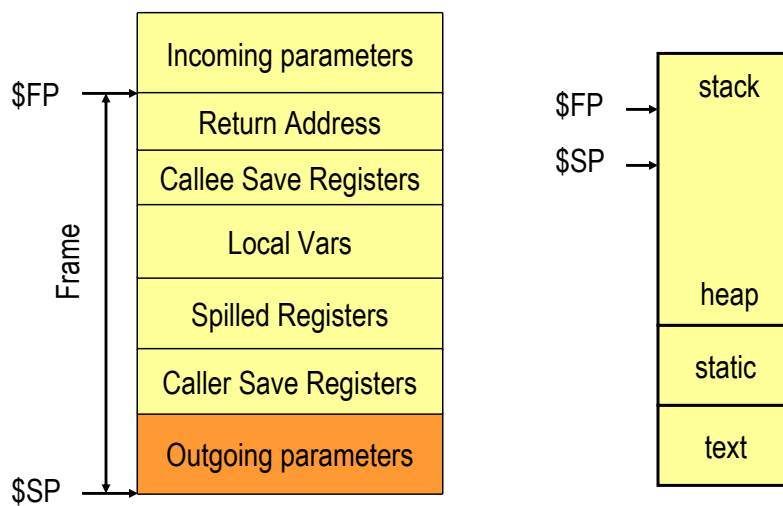


EECS 370: Introduction to  
Computer Organization

The University of Michigan  
© T. Austin & M. Papaefthymiou - 2009

7/24

## MIPS Stack Frame (updated)



EECS 370: Introduction to  
Computer Organization

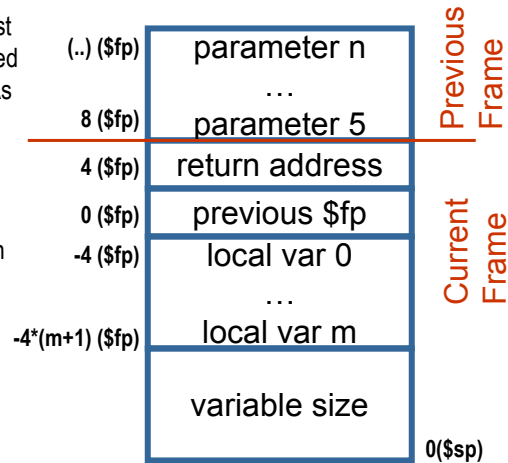
The University of Michigan  
© T. Austin & M. Papaefthymiou - 2009

8/24

## The MIPS stack – a live animation

Note 1: in MIPS, the first 4 parameters are passed via registers. Other ISAs have  $\neq$  conventions

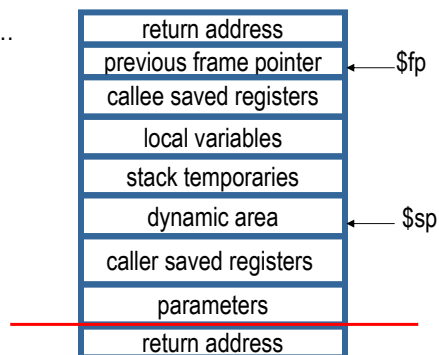
Note 2: why the last parameter is the first on the stack ?



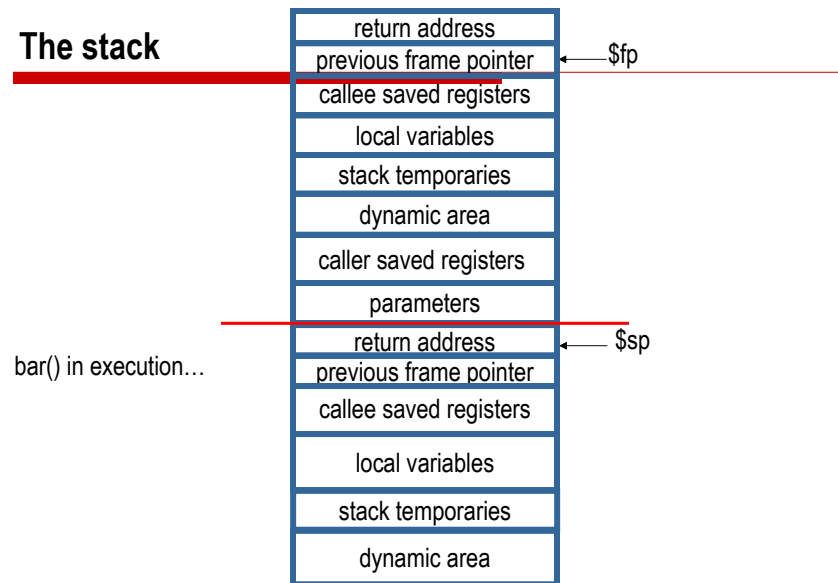
## The stack during program execution...

foo() is in execution...

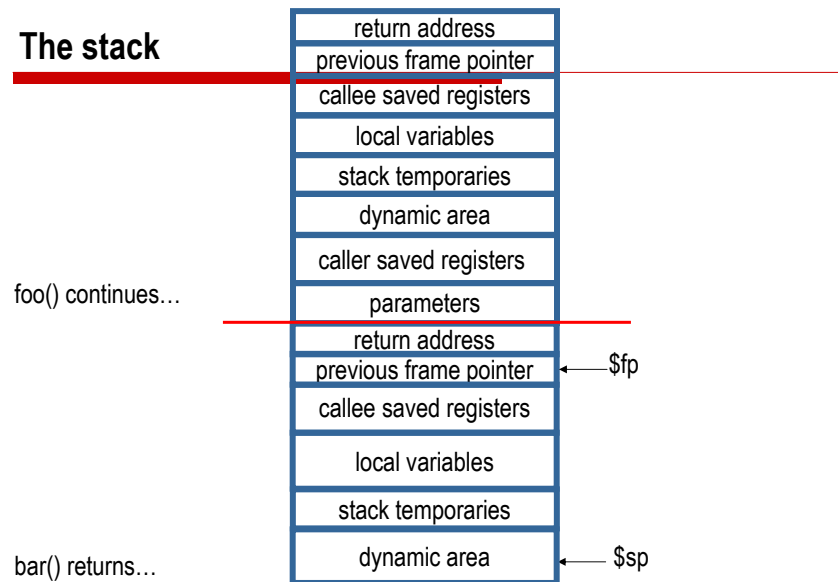
...foo() calls bar...



## The stack



## The stack



## Which is Better: Caller or Callee?

---

- ❑ Advantages of caller-saved
  - Only needs saving if it is “live” across a function call
  - In a leaf routine, caller saves can be used without overhead
  
- ❑ Advantages of callee-saved
  - Only needs saving at beginning of function (generally infrequent as outside of loops)
  - Only save it if you use it
  
- ❑ Each has its advantages. Neither is better.

## Provide Both Caller / Callee Regs

---

- ❑ Have some registers that are caller-saved, some that are callee-saved
  - Example: MIPS
    - 10 caller saved
    - 8 callee saved
  
- ❑ Choose registers for variables that minimize the number of dynamic saves/restores

## MIPS Register Convention

---

- \$0 constant 0
- \$1 reserved for assembler
- \$2,\$3 temporary and return value
- \$4 - \$7 **parameters**
- \$8 - \$15, \$24, \$25 **caller saved**
- \$16 - \$23 **callee saved**
- \$26, \$27 reserved for operating system
- \$28 global pointer
- \$29 stack pointer
- \$30 frame pointer
- \$31 return address

P&H pg. A-23

## Calling convention

---

- ❑ The register convention is just a convention
  - There is no difference in the hardware between caller and callee save registers
- ❑ The use of parameter passing registers is also a convention
- ❑ These conventional allows assembly code written by different people to work together
  - Need conventions about who saves registers and where arguments are passed.
- ❑ They collectively make up the ABI or ***application binary interface***

## Putting it all together (using activation records)

```

addi $4, $0, 1000    // param "hello world\n"
sw   $12, 24($sp)   // caller save $12
jal  _printf        // call printf()
                        // $2 holds return value (ignored)
lw   $12, 24($sp)   // restore $12 value
    
```

```

subi  $sp, $sp, 16    // allocate space for 2 locals +
sw    $16, 8($sp)    // callee save $16
sw    $31, 12($sp)   // save return address
...                                     // function body
addi  $2, $0, 0      // return value 0
lw    $16, 8($sp)    // restore $16
lw    $31, 12($sp)   // restore return address
addi  $sp, $sp, 16   // deallocate call frame
jr    $31            // return to calling function
    
```

## Caller-saved vs. callee saved – A full example

<pre> void main(){   int a,b,c,d;   .   c = 5; d = 6;   a = 2; b = 3;   foo();   d = a+b+c+d;   .   . }         </pre>	<pre> void foo(){   int a,b;   .   a = 2; b = 3;   bar();   a = a + b;   .   . }         </pre>	<pre> void bar(){   int a,b,c,d;   .   c = 0; d = 1;   a = 2; b = 3;   final();   a = a+b+c+d;   .   . }         </pre>	<pre> void final(){   int a,b,c;   .   a = 2; b = 3;   .   c = a+b;   .   . }         </pre>
--	---	---	--

Note: assume main does not have to save any callee reg. (that is really the case for start)

## Caller-saved vs. callee saved – A full example

### □ Questions:

1. How many regs. need to be stored/loaded in total if we use a **caller-save** convention ?
2. How many regs. need to be stored/loaded in total if we use a **callee-save** convention ?
3. How many regs. need to be stored/loaded in total if we use a mixed **caller/callee**-save convention with 3 callee-s. and 3 caller-s. registers ?
4. What if bar() is in a loop inside foo() and the loop is iterated 10 times ?

## Caller-saved vs. callee saved – Question 1

```
void main() {  
  .  
  .  
  .  
  [4 sw]  
  foo();  
  [4 lw]  
  .  
  .  
}
```

```
void foo() {  
  .  
  .  
  .  
  [2 sw]  
  bar();  
  [2 lw]  
  .  
  .  
  .  
}
```

```
void bar() {  
  .  
  .  
  .  
  [4 sw]  
  final();  
  [4 lw]  
  .  
  .  
  .  
}
```

```
void final() {  
  .  
  .  
  .  
  .  
  .  
  .  
  .  
  .  
}
```

Total: 10 sw / 10 lw

## Caller-saved vs. callee saved – Question 2

```
void main() {
.
.
.
foo();
.
.
.
}
```

```
void foo() {
[2 sw]
.
.
bar();
.
.
[2 lw]
}
```

```
void bar() {
[4 sw]
.
.
final();
.
.
[4 lw]
}
```

```
void final() {
[3 sw]
.
.
.
.
[3 lw]
}
```

Total: 9 sw / 9 lw

## Caller-saved vs. callee saved – Question 3

```
void main() {
.
.
.
[1 sw]
foo();
[1 lw]
.
.
}
```

```
void foo() {
[2 sw]
.
.
bar();
.
.
[2 lw]
}
```

```
void bar() {
[3 sw]
.
.
[1 sw]
final();
[1 lw]
.
.
[3 lw]
}
```

```
void final() {
.
.
.
.
}
```

1 caller r.  
3 callee r.

0 caller r.  
2 callee r.

1 caller r.  
3 callee r.

3 caller r.  
0 callee r.

Total: 7 sw / 7 lw

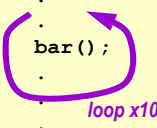
## Caller-saved vs. callee saved – Question 4

❑ Mixed 3 caller / 3 callee

```
void main() {
    .
    .
    .
    [1 sw]
    foo();
    [1 lw]
    .
    .
    .
}
```

1 caller r.  
3 callee r.

```
void foo() {
    [2 sw]
    .
    .
    .
    bar();
    .
    .
    [2 lw]
}
```



2 callee r.

```
void bar() {
    [3 sw]
    .
    .
    .
    [1 sw]
    final();
    [1 lw]
    .
    .
    .
    [3 lw]
}
```

1 caller r.  
3 callee r.  
*x10*

```
void final() {
    .
    .
    .
    .
    .
    .
    .
    .
    .
}
```

3 caller r.  
*x10*

Mixed: 43 sw / 43 lw

Pure caller: (4+20+40+0) sw / lw - Pure callee (0+2+40+30) sw / lw

EECS 370: Introduction to  
Computer Organization

The University of Michigan  
© T. Austin & M. Papaefthymiou – 2009

23/24

## Class Problem

```
foo() {
    a = ...
    b = ...
    bar();
    ... = a;
    ... = b;
    for (1... 15) {
        c = ...
        d = ...
        ... = c;
        printf();
        ... = d;
    }
}
```

Assume that you have 2 caller and 2 callee save registers. Pick the best assignment for a, b, c, d. Assume each requires its own register.

EECS 370: Introduction to  
Computer Organization

The University of Michigan  
© T. Austin & M. Papaefthymiou – 2009

24/24