Administrative

• Lab #2 due Thurs. at 5 pm.
• HW #1 due today at 5 pm.
Function Call Steps

- Place parameters in accessible location
- Transfer control to function
- Acquire storage for procedure variables
- Perform calculations in function
- Place result in place accessible to caller
- Return control to caller
MIPS Function Calls

- Parameters
  - $a0–$a3
- Return values
  - $v0–$v1
- Where to return control to
  - $ra
MIPS Function Calls

- Transfer control to function
  - jal label
    - Jumps to label’s instruction
    - Stores return address in $ra (PC+4)

- Return control to caller
  - jr $ra
Other Register Conventions

• Caller-saved registers
  ✤ $t0–$t9, $a0–$a3
  ✤ No preservation assumed

• Callee-saved registers
  ✤ $s0–$s7
  ✤ If you use these, you must restore values before returning

• Stack pointer
  ✤ $sp
  ✤ Points to last location on stack
Examples

- Suppose $s0 = a$, $s1 = b$, $s2 = c$, $s3 = d$
- Write MIPS instructions for the following code:
  \[ b = \text{ABS}(d) \]
Address Space

- Each process has an address space
- The address space is divided into segments:
  - Text
    - Instructions
  - Initialized Data
    - Globals
  - Uninitialized Data or Heap
    - `new` allocates space here
  - Stack
    - local variables are given space here
MIPS Function Calls: Local Storage

- **Stack**
  - LIFO
  - $sp

- **Non-volatile registers**
  - Push onto stack at function call
  - Restore to registers before function return

- **Spill local register values onto stack if not enough registers for function operation**
Procedure Frame/Activation Record

- Segment of stack that contains procedure’s saved registers and local variables
- Frame pointer ($fp$) points to first word of procedure frame

![Diagram showing procedure frame with high and low addresses, frame pointer (fp), stack pointer (sp), arguments (arg 5, 6, ...), saved registers, and local variables.](image_url)
Procedure Frame/Activation Record

- Segment of stack that contains procedure’s saved registers and local variables
- Frame pointer ($fp$) points to first word of procedure frame
Function Call Example

```c
int CalculateTriangleArea(int b, int h)
{
    int area = b * h;
    area /= 2;
    return area;
}

int main()
{
    int b = 4;
    int h = 10;
    int val = CalculateTriangleArea(b, h);
}
```
About the code that follows...

- It was generated by a compiler, so it’s not like code one would write
- Some assemblers use $s8 to store the frame pointer (this code does)
- $gp (the “global pointer” register), when used, points to a pool of global data that can be commonly referenced by all functions.
  - Convention dictates you should always store it when you code a function (who knows why)
12:    int b = 4;
[ 12] 0x100010c0:  24 02 00 04  li v0,4
[ 12] 0x100010c4:  af c2 00 10  sw v0,16(s8)
13:    int h = 10;
[ 13] 0x100010c8:  24 02 00 0a  li v0,10
[ 13] 0x100010cc:  af c2 00 14  sw v0,20(s8)
14:    int val = CalculateTriangleArea(b, h);
[ 14] 0x100010d0:  8f c4 00 10  lw a0,16(s8)
[ 14] 0x100010d4:  8f c5 00 14  lw a1,20(s8)
[ 14] 0x100010d8:  8f 99 80 68  lw t9,-32664(gp)
[ 14] 0x100010dc:  03 20 f8 09  jalr t9
[ 14] 0x100010e0:  00 00 00 00  nop
[ 14] 0x100010e4:  af c2 00 18  sw v0,24(s8)
Callee Function:
Stack Setup

4: {
  _Z21CalculateTriangleAreaii:
[  4] 0x10001030:  27 bd ff d0  addiu sp,sp,-48
[  4] 0x10001034:  ff be 00 28  sd s8,40(sp)
[  4] 0x10001038:  ff bc 00 20  sd gp,32(sp)
[  4] 0x1000103c:  03 a0 f0 25  move s8,sp
[  4] 0x10001040:  3c 01 00 02  lui at,2
[  4] 0x10001044:  24 21 b3 b8  addiu at,at,-19528
[  4] 0x10001048:  00 39 e0 2d  daddu gp,at,t9
[  4] 0x1000104c:  af c4 00 10  sw a0,16(s8)
[  4] 0x10001050:  af c5 00 14  sw a1,20(s8)
Callee Function:
Do Work

5:   int area = b * h;
    0x10001054:  8f c3 00 10  lw v1,16(s8)
    0x10001058:  8f c2 00 14  lw v0,20(s8)
    0x1000105c:  00 62 00 18  mult v1,v0
    0x10001060:  00 00 10 12  mflo v0
    0x10001064:  af c2 00 18  sw v0,24(s8)
    0x10001068:  00 00 00 00  nop

6:   area /= 2;
    0x1000106c:  8f c3 00 18  lw v1,24(s8)
    0x10001070:  00 03 17 c3  sra v0,v1,31
    0x10001074:  00 02 17 c2  srl v0,v0,31
    0x10001078:  00 62 10 21  addu v0,v1,v0
    0x1000107c:  00 02 10 43  sra v0,v0,1
    0x10001080:  af c2 00 18  sw v0,24(s8)

Wacky stuff takes care of two’s complement. Just skip over it.
Callee Function: Return Value and Control

7:   return area;
[    7] 0x10001084:  8f c2 00 18  lw v0,24(s8)
8:   }
[    8] 0x10001088:  03 c0 e8 25  move sp,s8
[    8] 0x1000108c:  df be 00 28  ld s8,40(sp)
[    8] 0x10001090:  df bc 00 20  ld gp,32(sp)
[    8] 0x10001094:  27 bd 00 30  addiu sp,sp,48
[    8] 0x10001098:  03 e0 00 08  jr ra
[    8] 0x1000109c:  00 00 00 00  nop
```c
int pow(int base, int exponent)
// Assumes base and exponent are both >= 0
{
    int result = 1;
    for(int i = 0; i < exponent; i++){
        result *= base;
    }
    return result;
}
```
addiu $sp, $sp, -4
sw $ra, 0($sp)
li $v0, 1
add $t0, $0, $0
loop: bge $t0, $a1, done
mul $v0, $v0, $a0
addi $t0, $t0, 1
b loop
done: lw $ra, 0($sp)
addiu $sp, $sp, 4
jr $ra
Recursive Functions

int fact(int n)
{
    if(n < 1)
        return 1;
    else
        return (n*fact(n-1));
}
Recursive Functions

```c
int fact(int n)
{
    if(n < 1)
        return 1;
    else
        return (n*fact(n-1));
}
```

- Acquire storage for procedure variables
- Perform calculations in function
- Place result in place accessible to caller
- Return control to caller
addiu $sp, $sp, -8
sw $ra, 0($sp)
sw $a0, 4($sp)
li $t0, 1
blt $a0, $t0, lessThan
addi $a0, $a0, -1
jal fact
lw $a0, 4($sp)
mul $v0, $v0, $a0
lw $ra, 0($sp)
addiu $sp, $sp, 8
jr $ra
lessThan: li $v0, 1
lw $ra, 0($sp)
addiu $sp, $sp, 8
jr $ra
int pow(int base, int exponent)  
   // Assumes base and exponent are both >= 0  
{
    int result = 1;

    if (exponent == 0)  
       return result;  
    else{
       result = base * pow(base, exponent-1);
    }
    return result;
}
bne $a1, $0, else
add $v0, $s0, $s0
jr $sra
else: addiu $sp, $sp, -8
sw $ra, 0($sp)
sw $a0, 4($sp)
addi $a1, $a1, -1
jal pow
lw $a0, 4($sp)
mul $v0, $v0, $a0
lw $sra, 0($sp)
addiu $sp, $sp, 8
jr $sra
System Calls

• Used to interact with operating system
• For our purposes, use for I/O
  ❧ Print output to console

• syscall
  ❧ Place arguments to syscall in registers
  ❧ Put number specifying which syscall into $v0
  ❧ It’s like a function call with respect to register conventions

<table>
<thead>
<tr>
<th>syscall</th>
<th>num</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td>print_int</td>
<td>1</td>
<td>$a0=integer</td>
</tr>
<tr>
<td>print_string</td>
<td>4</td>
<td>$a0=string</td>
</tr>
<tr>
<td>read_int</td>
<td>5</td>
<td>result in $v0</td>
</tr>
<tr>
<td>read_string</td>
<td>8</td>
<td>$a0=buffer, $a1=length</td>
</tr>
</tbody>
</table>
Discussion

Given the following function header,

```c
int foo(int a, int b);
```

what will be on the stack before any of the calculations in `foo` are performed? Assume `foo()` calls some other function.
What will be on the stack on a call to `int foo(int a, int b, int c, int d, int e, int f)`?