Lecture 5: Procedure Calls

- Today's topics:
  - Procedure calls and register saving conventions

Registers

- The 32 MIPS registers are partitioned as follows:
  - Register 0 : $zero always stores the constant 0
  - Regs 2-3 : $v0, $v1 return values of a procedure
  - Regs 4-7 : $a0-$a3 input arguments to a procedure
  - Regs 8-15 : $t0-$t7 temporaries
  - Regs 16-23: $s0-$s7 variables
  - Regs 24-25: $t8-$t9 more temporaries
  - Reg 28 : $gp global pointer
  - Reg 29 : $sp stack pointer
  - Reg 30 : $fp frame pointer
  - Reg 31 : $ra return address

Procedures

- **Each procedure (function, subroutine) maintains a scratchpad of register values.**

– When another procedure is called (the callee), the new procedure takes over the scratchpad
– Values in the register file may have to be saved so we can safely return them to the caller

1. Parameters (arguments) are placed where the callee can see them
2. Control is transferred to the callee
3. Acquire storage resources for the callee
4. Execute the procedure
5. Place result value where **the caller** can access it
6. Return control to caller

Jump-and-Link

- A special register (storage not part of the register file) maintains the address of the instruction currently being executed – this is the ***program counter*** **(PC)**

- The procedure (function) call is executed by invoking the **jump-and-link (jal)** instruction – the current PC (actually, PC+4) is saved in the register **$ra** and we jump to the procedure's address (the PC is accordingly set to this address)

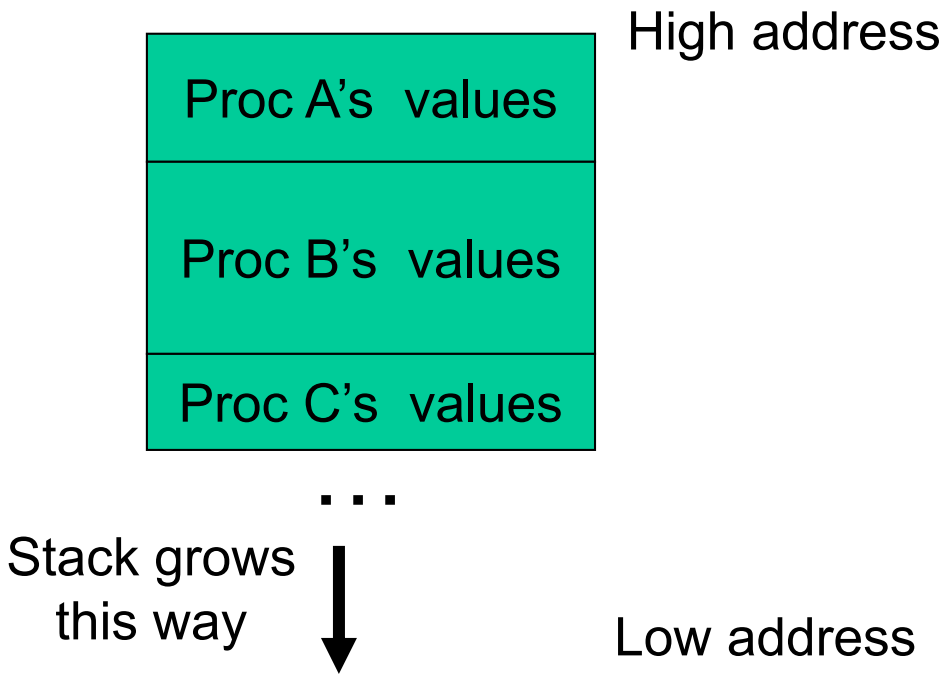
```
jal    NewProcedureAddress
```

- Since jal may over-write a relevant value in $ra, it must be saved somewhere (in memory?) before invoking the jal instruction

- How do we return the control back to the caller after completing the callee procedure?

The Stack

The register scratchpad for a procedure seems volatile – it seems to disappear every time we switch procedures – a procedure's values are therefore backed up in memory on a stack




```
Proc  A

        call  Proc B
              …
              call Proc C
                 …
              return
        return
return
```

Storage Management on a Call/Return

- A new procedure must create space for all its variables on the stack
- **Before/after executing the jal, the caller/callee must save relevant values in $s0-$s7, $a0-$a3, $ra, temps into the stack space**
- Arguments are copied into $a0-$a3; the jal is executed
- **After the callee creates stack space, it updates the value of $sp**
- Once the callee finishes, it copies the return value into $v0, frees up stack space, and $sp is incremented
- **On return, the caller/callee brings in stack values, ra, temps into registers**
- The responsibility for copies between stack and registers may fall upon either the caller or the callee

Example 1 (pg. 98)

```
int  leaf_example (int g, int h, int i, int j)
{
    int f ;
    f = (g + h) – (i + j);
    return f;
}
```

```
leaf_example:
   addi     $sp,  $sp,  -12
   sw       $t1, 8($sp)
   sw       $t0, 4($sp)
   sw       $s0, 0($sp)
   add      $t0, $a0, $a1
   add      $t1, $a2, $a3
   sub      $s0, $t0, $t1
   add      $v0, $s0, $zero
   lw       $s0, 0($sp)
   lw       $t0, 4($sp)
   lw       $t1, 8($sp)
   addi     $sp, $sp, 12
   jr       $ra
```

Notes:
Assume g,h,i, and j are in $a0~3.
In this example, the callee took care of saving the registers it needs.

The caller took care of saving its $ra and $a0-$a3.

Could have avoided using the stack altogether.

Example 2 (pg. 101)

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

Notes:
The caller saves $a0 and $ra
in its stack space.
Temp register $t0 is never saved.

```
fact:
    slti    $t0, $a0, 1
    beq     $t0, $zero, L1
      addi  $v0, $zero, 1
      jr    $ra
L1:
    addi    $sp, $sp, -8
    sw      $ra, 4($sp)
    sw      $a0, 0($sp)
    addi    $a0, $a0, -1
    jal     fact
    lw      $a0, 0($sp)
    lw      $ra, 4($sp)
    addi    $sp, $sp, 8
    mul     $v0, $a0, $v0
    jr      $ra
```

Example 2 (pg. 101)

n = 3

```
int   fact  (int n)
{                  3
    if (n < 1)  return (1);
       3
        else return (n * fact(n-1));
}                          3          2  2
```

```
int   fact  (int n)
{                  2
    if (n < 1)  return (1);
       2
        else return (n * fact(n-1));
}                          2          1  1
```

```
int   fact  (int n)
{                  1
    if (n < 1)  return (1);
       1
        else return (n * fact(n-1));
}                          1          0  1
```

```
int   fact  (int n)
{                  0
    if (n < 1)  return (1);  1
       0
        else return (n * fact(n-1));
}
```

Example 2 (pg. 101)

```
fact:
    slti    $t0, $a0, 1
    beq     $t0, $zero, L1
      addi  $v0, $zero, 1
      jr    $ra
L1:
    addi    $sp, $sp, -8
    sw      $ra, 4($sp)
    sw      $a0, 0($sp)
    addi    $a0, $a0, -1
    jal     fact
    lw      $a0, 0($sp)
    lw      $ra, 4($sp)
    addi    $sp, $sp, 8
    mul     $v0, $a0, $v0
    jr      $ra
```

3 0 3 2 3 6 3 2 — ra for caller — ra for caller

```
fact:
    slti    $t0, $a0, 1
    beq     $t0, $zero, L1
      addi  $v0, $zero, 1
      jr    $ra
L1:
    addi    $sp, $sp, -8
    sw      $ra, 4($sp)
    sw      $a0, 0($sp)
    addi    $a0, $a0, -1
    jal     fact
    lw      $a0, 0($sp)
    lw      $ra, 4($sp)
    addi    $sp, $sp, 8
    mul     $v0, $a0, $v0
    jr      $ra
```

2 0 2 1 2 2 1 — ra for "jal fact" — ra for "jal fact"

```
fact:
    slti    $t0, $a0, 1
    beq     $t0, $zero, L1
      addi  $v0, $zero, 1
      jr    $ra
L1:
    addi    $sp, $sp, -8
    sw      $ra, 4($sp)
    sw      $a0, 0($sp)
    addi    $a0, $a0, -1
    jal     fact
    lw      $a0, 0($sp)
    lw      $ra, 4($sp)
    addi    $sp, $sp, 8
    mul     $v0, $a0, $v0
    jr      $ra
```

1 0 1 0 1 1 1 1 — ra for "jal fact" — ra for "jal fact" — ra for "jal fact"

Example 2 (pg. 101)

```
fact:
    slti    $t0, $a0, 1
    beq     $t0, $zero, L1
      addi  $v0, $zero, 1
      jr    $ra
L1:
    addi    $sp, $sp, -8
    sw      $ra, 4($sp)
    sw      $a0, 0($sp)
    addi    $a0, $a0, -1
    jal     fact
    lw      $a0, 0($sp)
    lw      $ra, 4($sp)
    addi    $sp, $sp, 8
    mul     $v0, $a0, $v0
    jr      $ra
```

0
1
1

Saving Conventions

- Caller saves: Temp registers $t0-$t9 (the callee won't bother saving these, so save them if you care), $ra (it's about to get over-written), $a0-$a3 (so you can put in new arguments)

- Callee saves: $s0-$s7 (these typically contain "valuable" data)