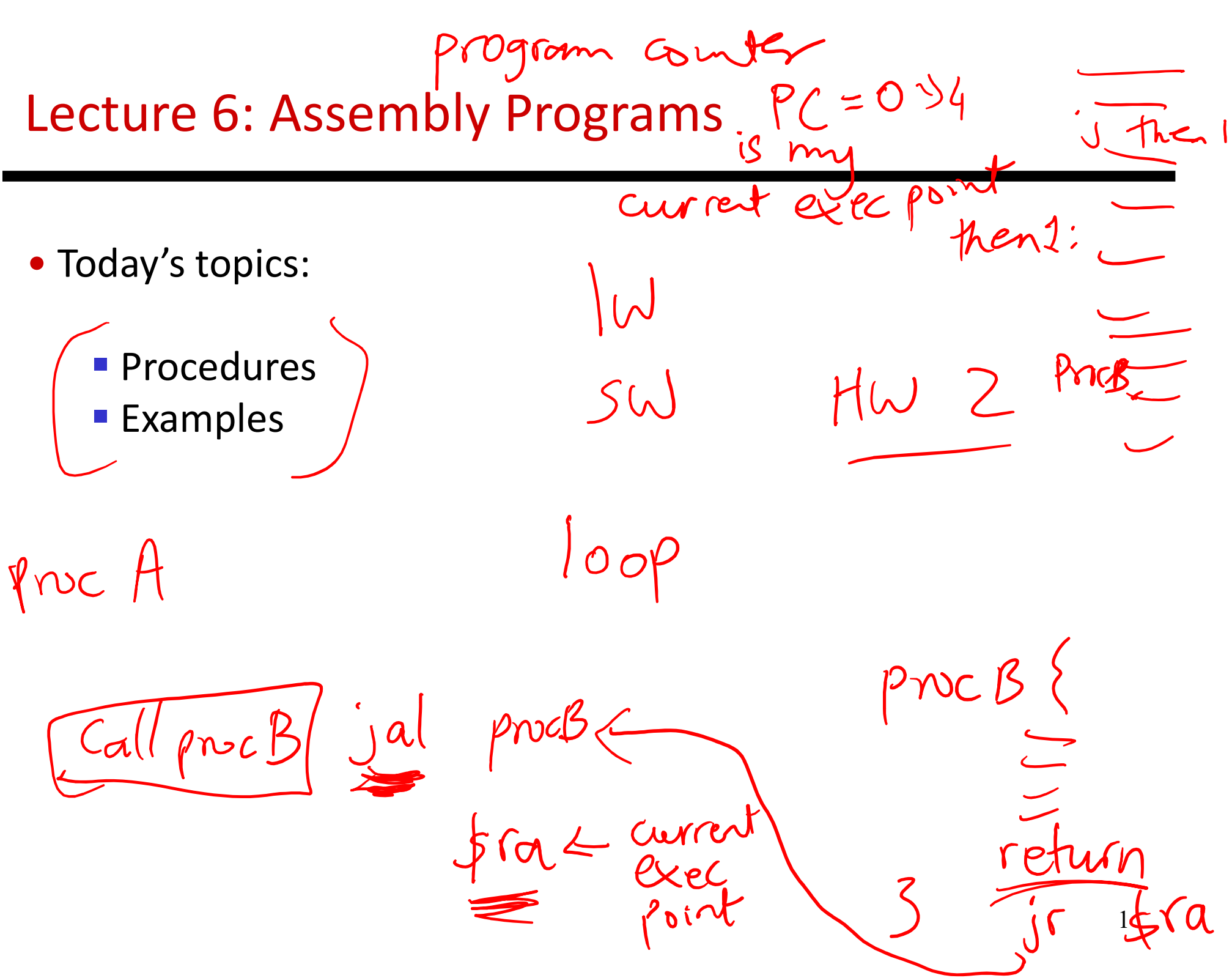
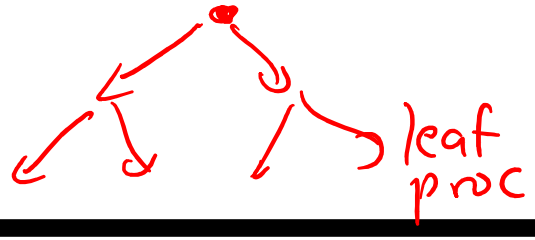
Lecture 6: Assembly Programs

- Today's topics:
  - Procedures
  - Examples



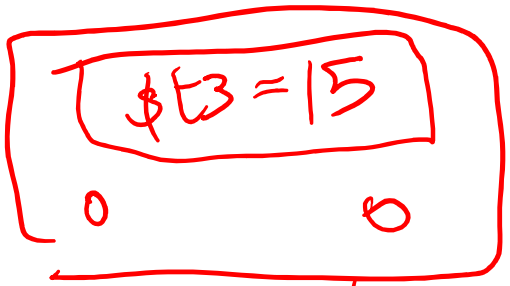
int g,h

Procedures

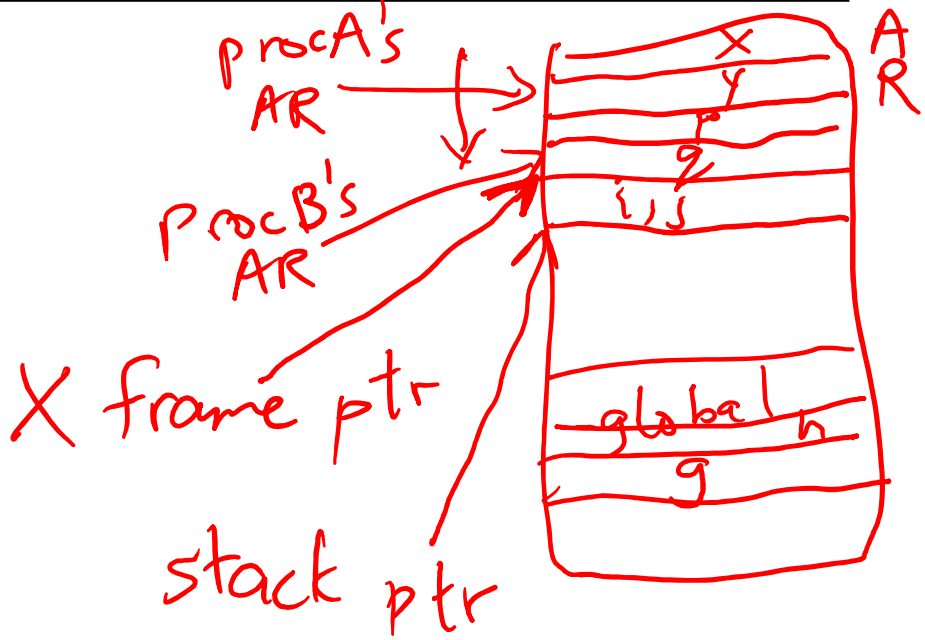


64 GB Mem

```
procA ( )
{ int x,y
  $t3 ← 15
  call procB
  call procC
}
```



Reg file



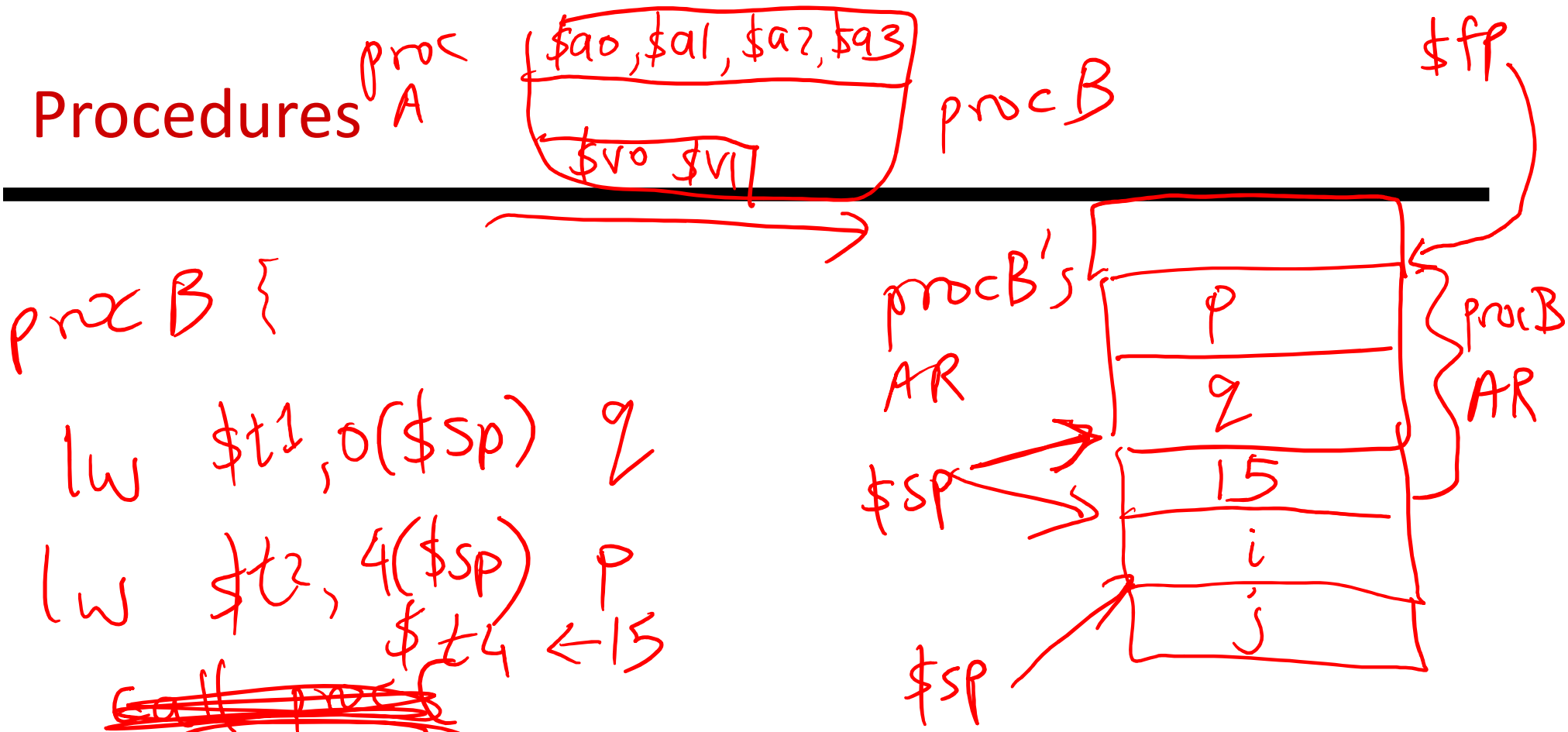

```
procB ( )
{ int p,q
  call procC
}
```

```
procC ( ) → leaf
{ int i,j
  addi $t1, $sp, 4
  ret
}
```

- Local variables, AR, $fp, $sp
- Scratchpad and saves/restores
- Arguments and returns
- jal and $ra

Mem Reg

Procedures



- Local variables, AR, $fp, $sp
- Scratchpad and saves/restores
- Arguments and returns
- jal and $ra

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 may have to be saved so we can safely return to the caller

  - parameters (arguments) are placed where the callee can see them
  - control is transferred to the callee
  - acquire storage resources for callee
  - execute the procedure
  - place result value where caller can access it
  - return control to caller

Jump-and-Link

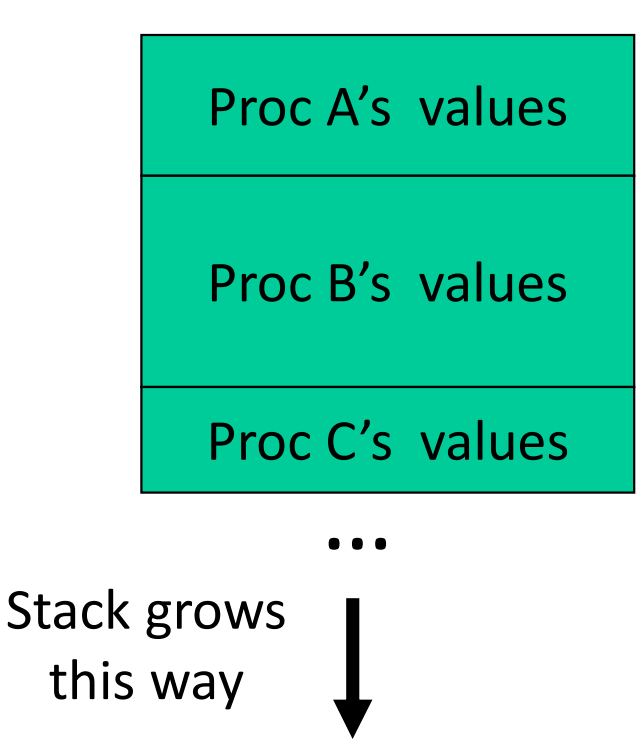
- A special register (storage not part of the register file) maintains the address of the instruction currently being executed – this is the *program counter* (PC)
- The procedure call is executed by invoking the jump-and-link (jal) instruction – the current PC (actually, PC+4) is saved in the register $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 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
```

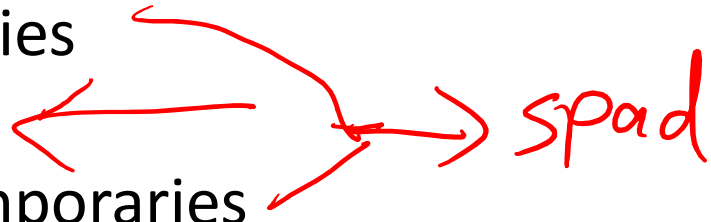
Saves and Restores

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, $fp, 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

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



Example 1 (pg. 98) [This example does not follow the conventions.]

```
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:
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.

Saving Conventions

- Caller saved: 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), $fp (if being used by the caller)

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

- Read the Notes on the class webpage on this topic

Example 2 (pg. 101)

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

```
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
```

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