Procedure

Procedure Calling

- Steps required
  1. Place parameters in registers
  2. Transfer control to procedure
  3. Acquire storage for procedure
  4. Perform procedure's operations
  5. Place result in register for caller
  6. Return to place of call

Register Usage

- $a0 – $a3: arguments (reg's 4 – 7)
- $v0, $v1: result values (reg's 2 and 3)
- $t0 – $t9: temporaries
  - Can be overwritten by callee
- $s0 – $s7: saved
  - Must be saved/restored by callee
- $gp: global pointer for static data (reg 28)
- $sp: stack pointer (reg 29)
- $fp: frame pointer (reg 30)
- $ra: return address (reg 31)

$t0 – $t7 are reg's 8 – 15
$t8 – $t9 are reg's 24 – 25
$s0 – $s7 are reg's 16 – 23

Procedure Call Instructions

- Procedure call:

  `jal ProcedureLabel  # jump and link`

  - Address of following instruction put in $ra
  - Jumps to target address

- Procedure return:

  `jr $ra  # jump register`

  - Copies $ra to program counter

Program Counter: register that contains the address of current instruction

Leaf Procedure Example

Leaf Procedure: procedures that do not make calls to other procedures

- C code:

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

  - Arguments g, h, i, j in $a0, $a1, $a2, $a3
  - f in $s0 (hence, need to save $s0 on stack)
  - Result in $v0

Leaf Procedure Example

- MIPS code:

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

Save $s0 on stack
(f : $s0)

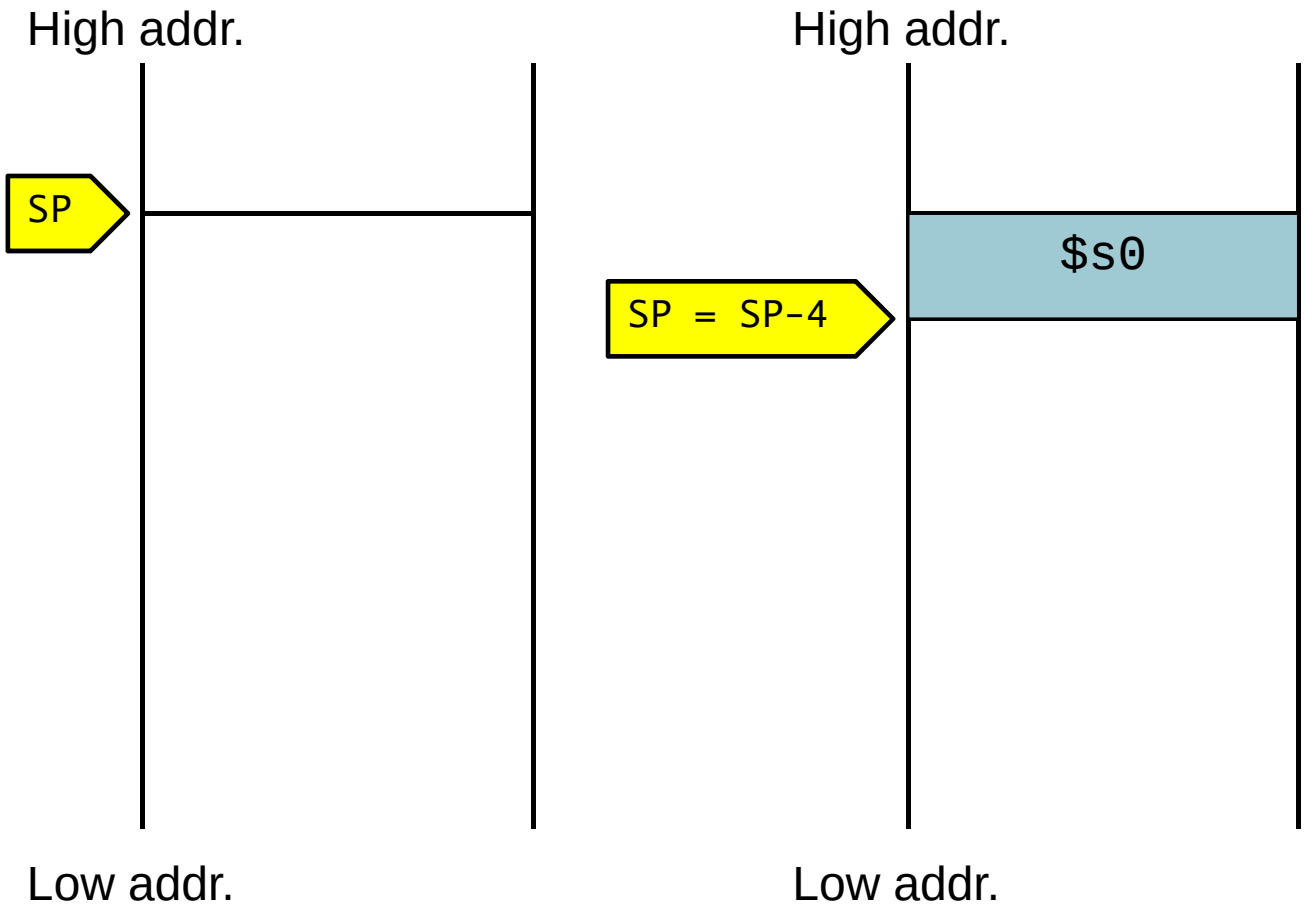
Procedure body

Result $s0→$v0

Restore $s0

Return

Stack Pointer



Non-Leaf Procedures

- Procedures that call other procedures
- For nested call, caller needs to save on the stack:
  - Its return address (caller)
  - Any arguments and temporaries needed after the call
- Restore from the stack after the call

Non-Leaf Procedure Example

- C code:

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

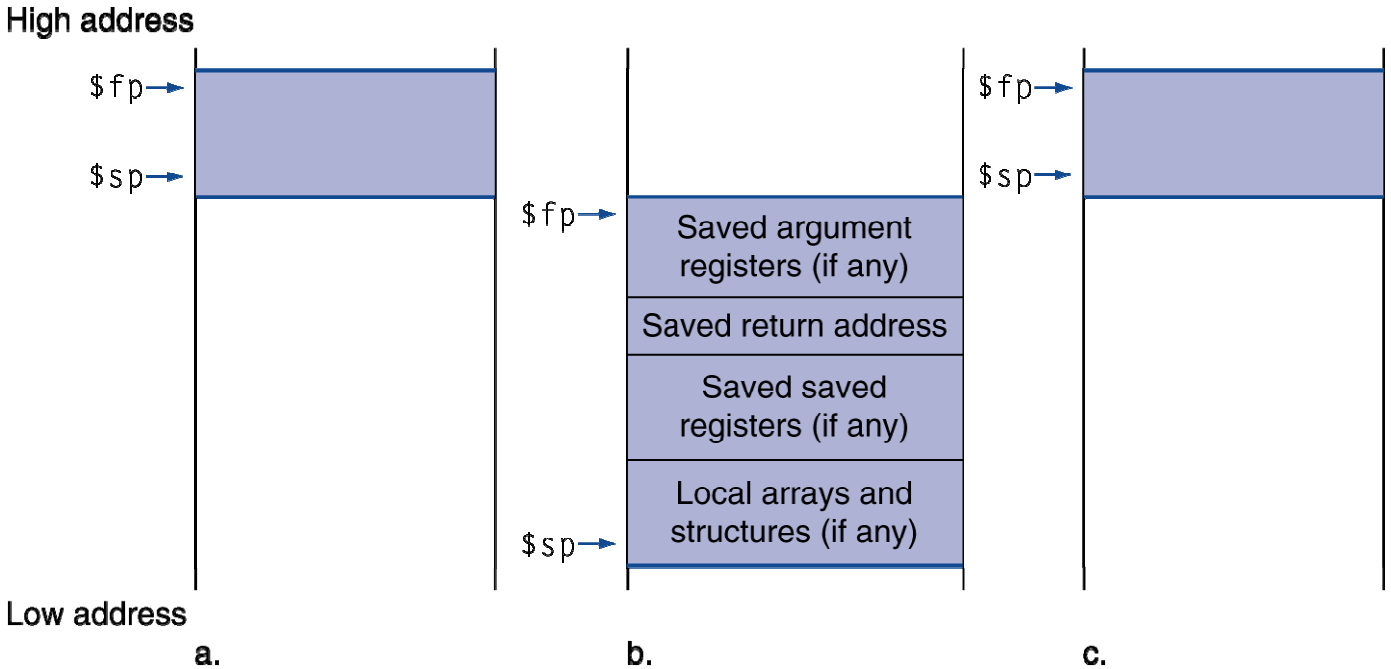
  - Argument n in $a0
  - Result in $v0

Non-Leaf Procedure Example

- MIPS code:

```
fact:
    addi $sp, $sp, -8      # adjust stack for 2 items
    sw   $ra, 4($sp)       # save return address
    sw   $a0, 0($sp)       # save argument
    slti $t0, $a0, 1       # test for n < 1
    beq  $t0, $zero, L1
    addi $v0, $zero, 1     # if so, result is 1
    addi $sp, $sp, 8       #   pop 2 items from stack
    jr   $ra               #   and return
L1: addi $a0, $a0, -1      # else decrement n
    jal  fact              # recursive call: fact(n-1)
    lw   $a0, 0($sp)       # restore original n
    lw   $ra, 4($sp)       #   and return address
    addi $sp, $sp, 8       # pop 2 items from stack
    mul  $v0, $a0, $v0     # multiply to get result
    jr   $ra               # and return
```

Local Data on the Stack



- Stack also store local variables that do not fit in registers
  - E.g., local array or structures

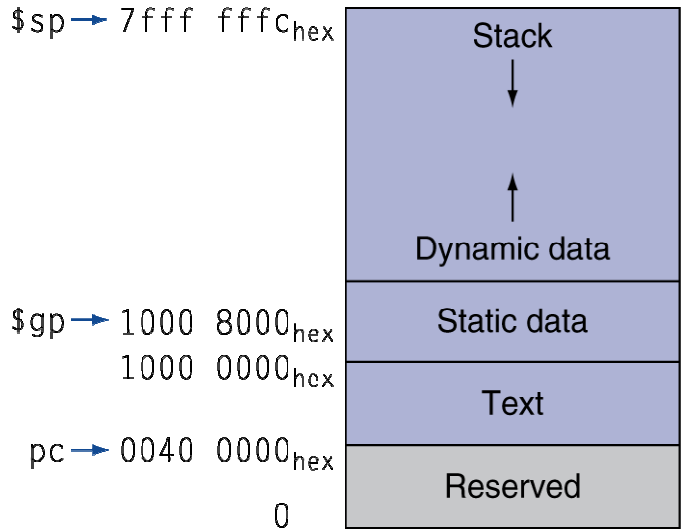
Frame Pointer

- Procedure frame
  - Also called activation record
  - The stack segment containing a procedure's saved registers and local variables

- Frame Pointer (`$fp`)
  - Point to the first word of the frame of a procedure
  - Offers a stable base register within a procedure for local memory-references

Memory Layout

- Text: program code
- Static data: global variables
  - e.g., static variables in C, constant arrays and strings
- Dynamic data: heap
  - E.g., malloc in C, new in Java
- Stack: automatic storage



Heap vs. Stack???

Stack Memory vs. Heap Memory

- Stack Memory
  - local variables
  - function parameters
  - not automatically initialized
  - variables on the stack disappear when the function exits
- Heap Memory
  - memory allocated by new, malloc
- Static data
  - global variables
  - static local variables