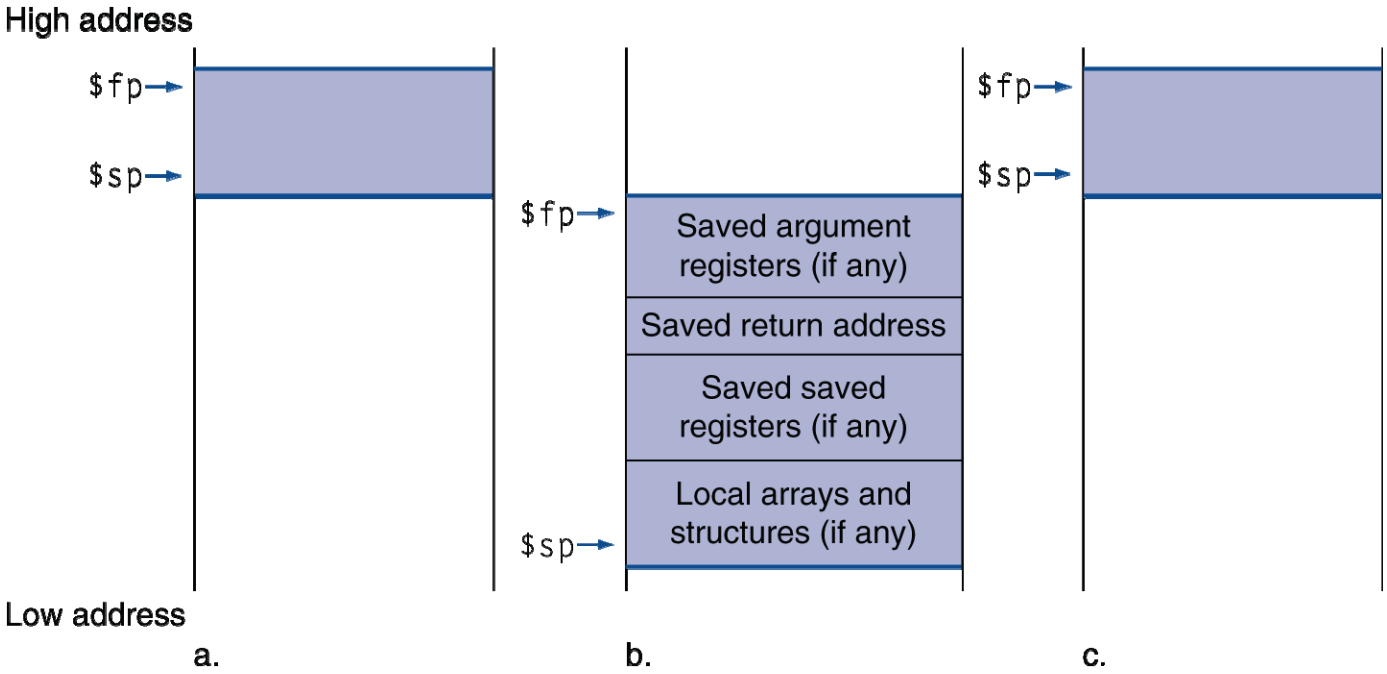
MIPS
Calling Convention

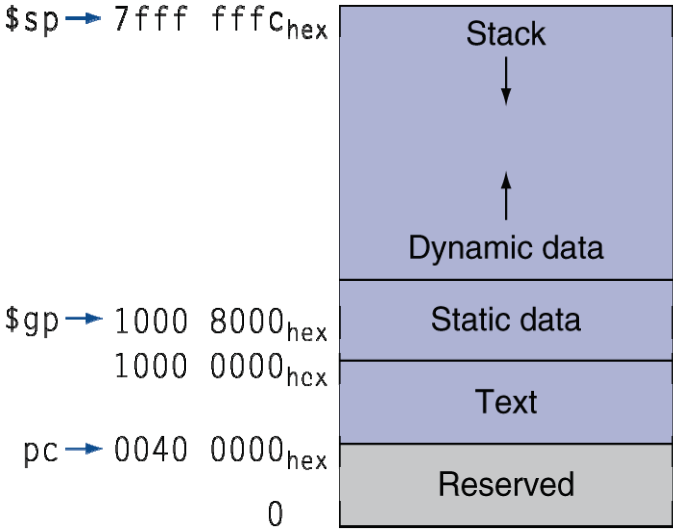
Local Data on the Stack



- Local data allocated by callee
  - e.g., C automatic variables
- Procedure frame (activation record)
  - Used by some compilers to manage stack storage

Memory Layout

- Text: program code
- Static data: global variables
  - e.g., static variables in C, constant arrays and strings
  - $gp initialized to address allowing ±offsets into this segment
- Dynamic data: heap
  - E.g., malloc in C, new in Java
- Stack: automatic storage



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)

Procedure Call Instructions

- Procedure call: jump and link

  `jal ProcedureLabel`

  – Address of following instruction put in $ra

  – Jumps to target address

- Procedure return: jump register

  `jr $ra`

  – Copies $ra to program counter

  – Can also be used for computed jumps

    - e.g., for case/switch statements

Leaf Procedure Example

- C code:

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

  - Arguments g, …, j in $a0, …, $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

Procedure body

Result

Restore $s0

Return

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

Non-Leaf Procedures

- Procedures that call other procedures
- For nested call, caller needs to save on the stack:
  - Its return address
  - 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:
00:      addi $sp, $sp, -8      # adjust stack for 2 items
04:      sw   $ra, 4($sp)       # save return address
08:      sw   $a0, 0($sp)       # save argument
12:      slti $t0, $a0, 1       # test for n < 1
16:      beq  $t0, $zero, L1
20:      addi $v0, $zero, 1     # if so, result is 1
24:      addi $sp, $sp, 8       #   pop 2 items from stack
28:      jr   $ra               #   and return
32:   L1: addi $a0, $a0, -1      # else decrement n
36:      jal  fact              # recursive call
40:      lw   $a0, 0($sp)       # restore original n
44:      lw   $ra, 4($sp)       #   and return address
48:      addi $sp, $sp, 8       # pop 2 items from stack
52:      mul  $v0, $a0, $v0     # multiply to get result
56:      jr   $ra               # and return
```