

# Chapter 2

---

## Instructions: Language of the Computer



# 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	Save \$s0 on stack
sw	\$s0,	0(\$sp)		
add	\$t0,	\$a0,	\$a1	Procedure body
add	\$t1,	\$a2,	\$a3	
sub	\$s0,	\$t0,	\$t1	
add	\$v0,	\$s0,	\$zero	Result
lw	\$s0,	0(\$sp)		Restore \$s0
addi	\$sp,	\$sp,	4	
jr	\$ra			Return



# 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 f;
    else return n * fact(n - 1);
}
```

- Argument n in \$a0
- Result in \$v0





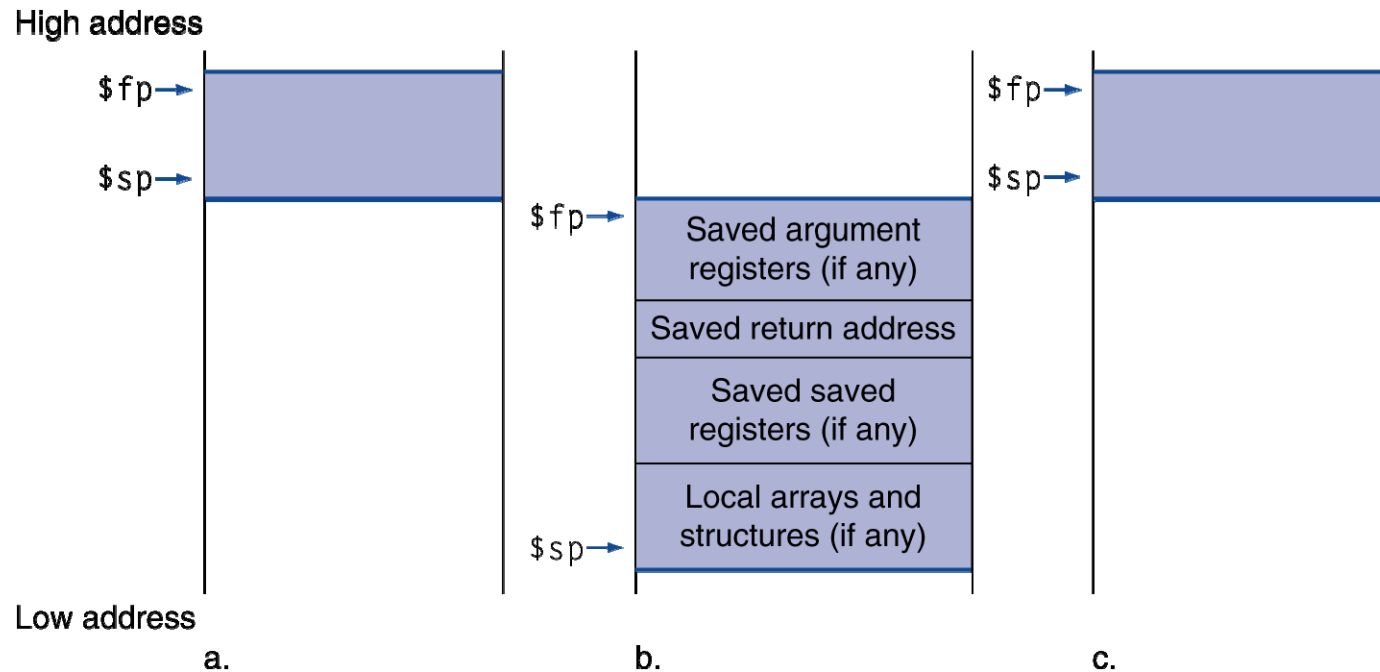
# Non-Leaf Procedure Example

- 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
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



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

