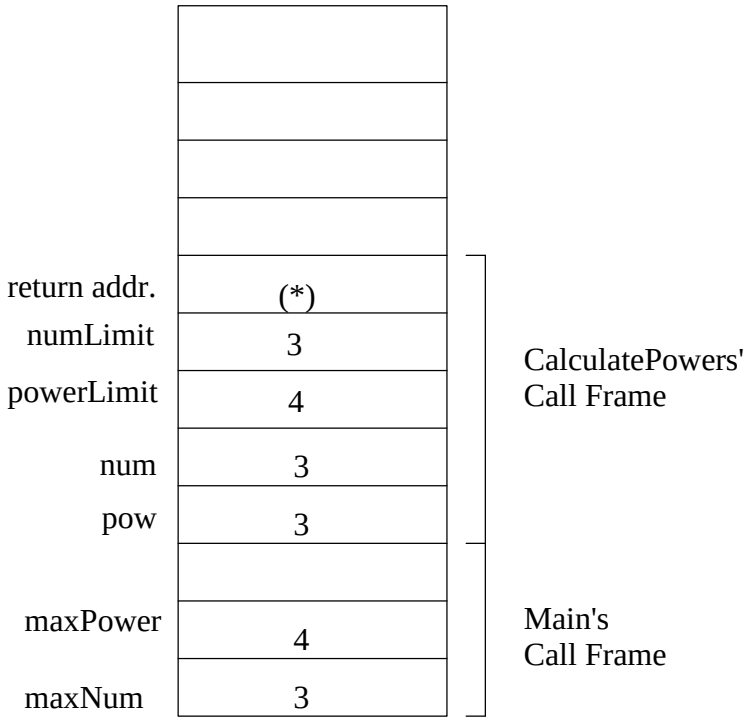
High-level Language Programmer's View

main:	**CalculatePowers**(**In:** integer numLimit, integer powerLimit)	integer **Power**(**In:** integer n, integer e)

maxNum = 3
maxPower = 4
 	
CalculatePowers(maxNum, maxPower)
(*)
. . .
end main
	integer num, pow
	
	for num := 1 to numLimit do
	for pow := 1 to powerLimit do
	
	print num " raised to " pow " power is "
	Power(num, pow)
	end for pow
	end for num
	
	end CalculatePowers
		integer result
		if e = 0 then
		result = 1
		else if e = 1 then
		result = n
		else
		result = Power(n, e - 1)* n
		end if
		return result
		end Power

HLL View of Run-time Stack



Compiler uses registers to avoid accessing memory as much as possible. Registers can be used for local variables, parameters, return address, function-return value.

```
AL code for subprogram "caller"

  <code using some registers>

call subprogram

  <wants used registers to be unchanged>
```

When a subprogram is called, some of the register values might need to be saved ("spilled") on the stack to free up some registers for the subprogram to use.

Standard conventions for spilling registers:
1) caller save - before the call, caller saves the register values it needs after execution returns from the subprogram
2) callee save - subprogram saves and restores any register it uses in its code
3) some combination of caller and callee saved (USED BY MIPS)

Reg. #	Convention Name	Role in Procedure Calls	Comments
$0	$zero	value zero	
$1	$at	Used by assembler to implement psuedoinstructions	DON'T USE
$2, $3	$v0, $v1	Results of a function	
$4 - $7	$a0 - $a3	First 4 arguments to a procedure	
$8 - $15, $24, $25	$t0 - $t9	Temporary registers (not preserved across call)	Caller-saved registers - subprogram can use them as scratch registers, but it must also save any needed values before calling another subprogram.
$16 - $23	$s0 - $s7	Saved temporary (preserved across call)	Callee-saved registers - it can rely on an subprogram it calls not to change them (so a subprogram wishing to use these registers must save them on entry and restore them before it exits)
$26, $27	$k0, $k1	Reserved for the Operating System Kernel	DON'T USE
$28	$gp	Pointer to global area	
$29	$sp	Stack pointer	Points to first free memory location above stack
$30	$fp/$s8	Frame pointer (if needed) or another saved register	$fp not used so use as $s8
$31	$ra	Return address (used by a procedure call)	Receives return addr. on *jal* call to procedure

Using MIPS Calling Convention

Caller Code

. . .

1) save on stack any $t0 - $t9 and $a0 - $a3 that are needed upon return
2) place arguments to be passed in $a0 - $a3 with additional parameters pushed onto the stack
3) jal ProcName # saves return address in $ra
4) restore any saved registers $t0 - $t9 and $a0 - $a3 from stack

Callee Code

. . .

1) allocate memory for frame by subtracting frame size from $sp
2) save callee-saved registers ($s0 - $s7) if more registers than $t0 - $t9 and $a0 - $a3 are needed
3) save $ra if another procedure is to be called

. . . code for the callee

4) for functions, place result to be returned in $v0 - $v1
5) restore any callee-saved registers ($s0 - $s7) from step (2) above
6) restore $ra if it was saved on the stack in step (3)
7) pop stack frame by adding frame size to $sp
8) return to caller by "jr $ra" instruction

main:	**CalculatePowers**(**In:** integer numLimit, integer powerLimit)	integer **Power**(**In:** integer n, integer e)
maxNum = 3		integer result
maxPower = 4	integer num, pow	if e = 0 then
		result = 1
CalculatePowers(maxNum, maxPower)	for num := 1 to numLimit do	else if e = 1 then
(*)	for pow := 1 to powerLimit do	result = n
. . .		else
end main	print num " raised to " pow " power is "	result = Power(n, e - 1)* n
	Power(num, pow)	end if
	end for pow	return result
	end for num	**end Power**
	end CalculatePowers	

a) Using the MIPS register conventions, what registers would be used to pass each of the following parameters to CalculatePowers:

maxNum	maxPower

b) Using the MIPS register conventions, which of these parameters ("numLimit", "powerLimit", or both of them) should be moved into s-registers?
(**NOTE: Use an s-register for any value you still need after you come back from a subprogram/function/procedure call, e.g., call to "Power")**

c) Using the MIPS register conventions, what registers should be used for each of the local variables:

num	pow

d) Using the MIPS register conventions, what registers would be used to pass each of the following parameters to Power:

num	pow

e) Using the MIPS register conventions, which of these parameters ("n", "e", or both of them) should be moved into s-registers?

f) Using the MIPS register conventions, what register should be used for the local variable:

result

g) Write the code for main, CalculatePowers, and Power in MIPS assembly language.