MIPS32 AL – More On Functions
(leaf-function situation we've seen so far)

```
            .data
someInts:   .word 1, 2, 3, 1, 2
max of1:    .asciiz "\nmax(1, 2, 3) = "
max_of2:    .asciiz "\nmax(2, 3, 1) = "
max_of3:    .asciiz "\nmax(3, 1, 2) = "
            .text
            .globl main
main:
            la $s0, someInts

            lw $a0, 0($s0)
            lw $a1, 4($s0)
            lw $a2, 8($s0)
            jal maxOf3Ints
            move $t1, $v0
            la $a0, max_of1
            li $v0, 4
            syscall
            move $a0, $t1
            li $v0, 1
            syscall

            lw $a0, 4($s0)
            lw $a1, 8($s0)
            lw $a2, 12($s0)
            jal maxOf3Ints
            move $t1, $v0
            la $a0, max_of2
            li $v0, 4
            syscall
```

(continued on the right)

```
            move $a0, $t1
            li $v0, 1
            syscall

            lw $a0, 8($s0)
            lw $a1, 12($s0)
            lw $a2, 16($s0)
            jal maxOf3Ints
            move $t1, $v0
            la $a0, max_of3
            li $v0, 4
            syscall
            move $a0, $t1
            li $v0, 1
            syscall

            li $v0, 10
            syscall

maxOf3Ints: move $t1, $a0
            bge $t1, $a1, nextChk
            move $t1, $a1
nextChk:    bge $t1, $a2, endChk
            move $t1, $a2
endChk:     move $v0, $t1
            jr $ra
```

What if **maxOf3Ints** calls another function?

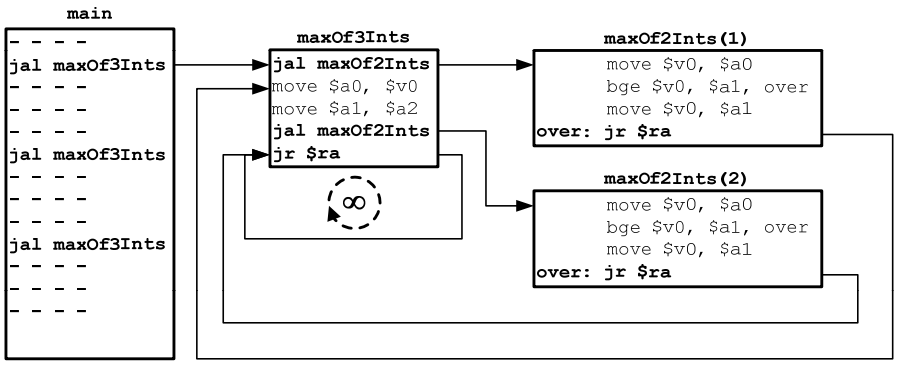
MIPS32 AL – More On Functions
(following C++ lead – will the cow come home?)

```
            .data
someInts:   .word 1, 2, 3, 1, 2
max of1:    .asciiz "\nmax(1, 2, 3) = "
max_of2:    .asciiz "\nmax(2, 3, 1) = "
max_of3:    .asciiz "\nmax(3, 1, 2) = "
            .text
            .globl main
main:
            la $s0, someInts

            lw $a0, 0($s0)
            lw $a1, 4($s0)
            lw $a2, 8($s0)
            jal maxOf3Ints
            move $t1, $v0
            la $a0, max_of1
            li $v0, 4
            syscall
            move $a0, $t1
            li $v0, 1
            syscall

            lw $a0, 4($s0)
            lw $a1, 8($s0)
            lw $a2, 12($s0)
            jal maxOf3Ints
            move $t1, $v0
            la $a0, max_of2
            li $v0, 4
            syscall
```

(continued on the right)

```
            move $a0, $t1
            li $v0, 1
            syscall

            lw $a0, 8($s0)
            lw $a1, 12($s0)
            lw $a2, 16($s0)
            jal maxOf3Ints
            move $t1, $v0
            la $a0, max_of3
            li $v0, 4
            syscall
            move $a0, $t1
            li $v0, 1
            syscall

            li $v0, 10
            syscall

maxOf3Ints: jal maxOf2Ints
            move $a0, $v0
            move $a1, $a2
            jal maxOf2Ints
            jr $ra

maxOf2Ints: move $v0, $a0
            bge $v0, $a1, over
            move $v0, $a1
over:       jr $ra
```

MIPS32 AL – More On Functions
(following C++ lead – cow is never seen again)



- In HLL (*e.g.*: C++), compiler takes care of necessary bookkeeping
- In AL, **bookkeeping responsibility rests with programmer**
  - Remember: there's *only ONE set of registers* available
  - Specifically: there's *only ONE* **`$ra`** that must be *shared by all function calls*

MIPS32 AL – More On Functions
(how 2 keep up w/ function-call bookkeeping)

- Function calls/returns in most programming languages:
  - Follow strict LIFO (last-in, first-out) order
- So, ***stack*** is "best suited function-call bookkeeping device"
  - Main reason every machine has runtime (system, call, …) stack
- Some machines have memory stack (special hardware) as part of architecture (*e.g.*, VAX)
  - Hardware supports stack management
  - Have special instructions (*e.g.*, push and pop)
- MIPS does not support stack management in hardware
  - Stack has to be implemented via software convention
    - Use of *stack segment* governed by agreed upon *social contract*
  - Normal data transfer instructions (*e.g.*, **`lw`** and **`sw`**) are used
    - Like (what we did before) using stack segment for local variables

MIPS32 AL – More On Functions

(for case in hand, we've known enough to fix)

```
            .data
someInts:   .word 1, 2, 3, 1, 2
max_of1:    .asciiz "\nmax(1, 2, 3) = "
max_of2:    .asciiz "\nmax(2, 3, 1) = "
max_of3:    .asciiz "\nmax(3, 1, 2) = "
            .text
            .globl main
main:
            la $s0, someInts

            lw $a0, 0($s0)
            lw $a1, 4($s0)
            lw $a2, 8($s0)
            jal maxOf3Ints
            move $t1, $v0
            la $a0, max_of1
            li $v0, 4
            syscall
            move $a0, $t1
            li $v0, 1
            syscall

            lw $a0, 4($s0)
            lw $a1, 8($s0)
            lw $a2, 12($s0)
            jal maxOf3Ints
            move $t1, $v0
            la $a0, max_of2
            li $v0, 4
            syscall
```

(continued on the right)

```
            move $a0, $t1
            li $v0, 1
            syscall

            lw $a0, 8($s0)
            lw $a1, 12($s0)
            lw $a2, 16($s0)
            jal maxOf3Ints
            move $t1, $v0
            la $a0, max_of3
            li $v0, 4
            syscall
            move $a0, $t1
            li $v0, 1
            syscall

            li $v0, 10
            syscall

maxOf3Ints: addiu $sp, $sp, -4
            sw $ra, 0($sp)
            jal maxOf2Ints
            move $a0, $v0
            move $a1, $a2
            jal maxOf2Ints
            lw $ra, 0($sp)
            addiu $sp, $sp, 4
            jr $ra

maxOf2Ints: move $v0, $a0
            bge $v0, $a1, over
            move $v0, $a1
over:       jr $ra
```


MIPS32 AL – More On Functions

(but a generally-usable *convention* is needed)

- Should support …
  - Any function calling any function any # of times from anywhere
    - Including function calling "itself" (*recursion*), directly or indirectly
  - Each function should also be able to pass any # of arguments
- Should ensure that …
  - Every function returns to where it should when it completes
- Should enable/facilitate …
  - Establishment and preservation of each function's *environment*
    - So that each function can successfully/correctly perform what is intended
  - A function's environment in general:
    - Information about (describing state of) function during execution
    - Arguments, values of local variables, which statement being executed, …
  - A function's environment in MIPS assembly program:
    - Values of all registers (whose contents are subject to change) that function references

MIPS32 AL – More On Functions
(a function-call convention for non-leaf funcs)

Part of Convention: MIPS Register Usage Convention

Name	*Reg. Number*	*Usage*	*Preserved Across Call*
`$zero`	`0`	constant 0	(NA)
`$v0-$v1`	`2-3`	function results	N
`$a0-$a3`	`4-7`	function arguments	N
`$t0-$t7`	`8-15`	temporaries (caller-saved)	N
`$s0-$s7`	`16-23`	saved (callee-saved)	Y
`$t8-$t9`	`24-25`	temporaries (caller-saved)	N
`$k0-$k1`	`26-27`	reserved for OS kernel	N
`$gp`	`28`	global pointer	Y
`$sp`	`29`	stack pointer	Y
`$fp`	`30`	frame pointer	Y
`$ra`	`31`	return address	Y

MIPS32 AL – More On Functions
(a function-call convention for non-leaf funcs)

- What ***callee*** (called function) must do
  - prolog or preamble:
    - Allocate stack space (create ***stack frame*** or ***activation record***)
      - Subtract *frame size* from stack pointer (**`$sp`**): can use **`subu`** or **`addiu`**
      - *Minimum frame size is 32 bytes* (per MIPS software architecture)
      - *Total bytes must be divisible by 8* (aligned for floating-point numbers)
      - Always allocate: 2 words for **`$ra`** and **`$fp`**, and 4 words for **`$a0-$a3`**
      - May-need-space items: callee-saved registers (**`$s0-$s7`**), caller-saved registers (**`$t0-$t9`**), local variables, and outgoing arguments
    - Save values of any callee-saved register that may be used
      - Always save value of **`$ra`** and **`$fp`**
      - Save values of any **`$s0-$s7`** that callee intends to use
    - Set **`$fp`** to "old" value of **`$sp`**
  - body:
    - Perform task (save/restore caller-saved registers accordingly)
  - epilog:
    - Place return result in proper location (*e.g.*, **`$v0`**) for caller
    - Restore values (previously saved) of callee-saved registers
    - Deallocate stack space (restore **`$sp`** to its "old" value)
    - Transfer control back to caller: **`jr $ra`**

MIPS32 AL – More On Functions

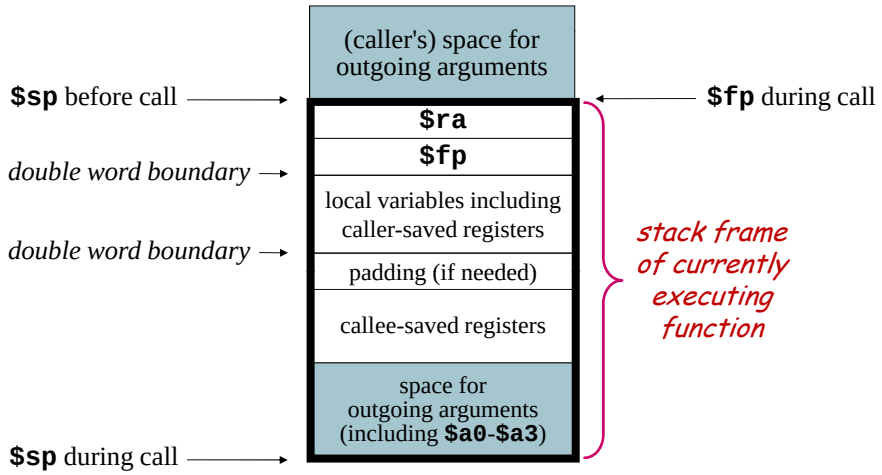
(a function-call convention for non-leaf $func_s^n$)

- What ***caller*** (calling function) must do
  - Save values of any *caller-saved* registers (**`$t0-$t9`**, **`...`**)
    - Only those that caller wants preserved
  - Pass arguments to callee
    - Up to 4 arguments via **`$a0`**, **`$a1`**, **`$a2`** and **`$a3`**
    - Any additional arguments via stack (pushed onto stack)
  - Transfer control to callee
    - Use **`jal`** (we won't be using **`jalr`**)
  - Wait for callee's return
  - Retrieve value returned by callee (if any)
  - Restore values (previously saved) of any caller-saved registers

MIPS32 AL – More On Functions

(a function-call convention for non-leaf functions)

who shall reside where

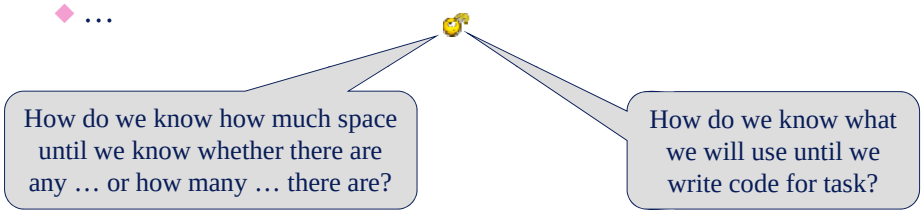


MIPS32 AL – More On Functions
(a function-call convention for non-leaf $func_s^n$)

- Additional points to note:
  - 4 words allocated for **`$a0`**-**`$a3`** are not used by current function
    - Reserved for use by any function current function may call
    - (Thus, these are slots allocated by caller but are used by callee!)
  - Caller must allocate *outgoing arguments section* based on the function it calls that has the *maximum* number of arguments
    - Subject to the *minimum* requirement of 4 arguments
  - So that *local variables section starts on double word boundary* AND *total frame size is multiple of 8*:
    - Insert (if necessary) 1-word *padding* after *callee-saved registers section*
    - Round up (if necessary) size of *local variables section* to next multiple of 8
  - Allocation of stack frame (by callee) done *only once* per function
    - Once **`$sp`** has been modified, *don't modify it again* for the rest of function
  - Values of argument registers **`$a0`**-**`$a3`** and return-value registers **`$v0`**-**`$v1`** are not required to be preserved across function calls
    - Function can thus change their values without saving/restoring them
  - Program begins with OS calling **`main`**, so **`main`** is 1st callee
    - Function is callee (gets called by another) 1st, then caller (calls another)

MIPS32 AL – More On Functions
(a function-call convention for non-leaf $func_s^n$)

- What ***callee*** (called function) must do
  - Allocate stack space (create ***stack frame*** or ***activation record***)
    - Subtract *frame size* from stack pointer (**`$sp`**): can use **`subu`** or **`addiu`**
    - *Minimum frame size is 32 bytes* (per MIPS software architecture)
    - *Total bytes must be divisible by 8* (aligned for floating-point numbers)
    - Always allocate: 2 words for **`$ra`** and **`$fp`**, and 4 words for **`$a0`**-**`$a3`**
    - May-need-space items: callee-saved registers (**`$s0`**-**`$s7`**), caller-saved registers (**`$t0`**-**`$t9`**), local variables, and outgoing arguments
  - …



(plan ahead as best we can & make adjustments along the way)

MIPS32 AL – More On Functions
(e.g. 1: stack frame analysis/design)

```
int f(int arg1, int arg2)
{
    int a[50], temp;
    ...
    a[3] = 5;
    ...
    g(a[1], a[2], a[3], a[4], a[5]);
    ...
    int h234 = h(a[2], a[3], a[4]);
    ...
    return a[3];
}
```

2 local variables

assume we need to use **$s0** and **$s1**, preserve **$t0**, and are short of registers (thus need memory space) for **temp** and **h234**

non-leaf function w/ maximum of 5 outgoing arguments

another local variable

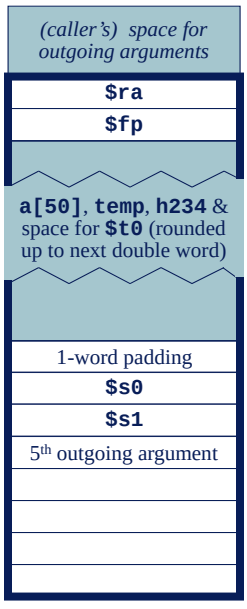
MIPS32 AL – More On Functions
(e.g. 1: stack frame construction & teardown)

```
f:
   addiu $sp, $sp, -256      # (5+2+1+54+2)*4
   sw $ra, 252($sp)
   sw $fp, 248($sp)
   addiu $fp, $sp, 256
   sw $s0, -232($fp)
   sw $s1, -236($fp)
   ...
   addi $t0, $0, 5           # $t0 = 5
   sw $t0, -212($fp)         # a[3] = 5
   ...
   lw $s1, -236($fp)
   lw $s0, -232($fp)
   lw $fp, 248($sp)
   lw $ra, 252($sp)
   addiu $sp, $sp, 256
   jr $ra
```

prolog

body

epilog



MIPS32 AL – More On Functions
(maxOf3Ints_maxOf2Ints e.g. revisited)

```
            .data
someInts:   .word 1, 2, 3, 1, 2
max_of1:    .asciiz "\nmax(1, 2, 3) = "
max_of2:    .asciiz "\nmax(2, 3, 1) = "
max_of3:    .asciiz "\nmax(3, 1, 2) = "
            .text
            .globl main
main:
            addiu $sp, $sp, -32
            sw $ra, 28($sp)
            sw $fp, 24($sp)
            addiu, $fp, $sp, 32
            sw $s0, -16($fp)

            la $s0, someInts
            lw $a0, 0($s0)
            lw $a1, 4($s0)
            lw $a2, 8($s0)
            jal maxOf3Ints
            move $t1, $v0
            la $a0, max_of1
            li $v0, 4
            syscall
            move $a0, $t1
            li $v0, 1
            syscall
            lw $a0, 4($s0)
            lw $a1, 8($s0)
            lw $a2, 12($s0)
            jal maxOf3Ints
```

(continued on the right)

```
            move $t1, $v0
            la $a0, max_of2
            li $v0, 4
            syscall
            move $a0, $t1
            li $v0, 1
            syscall
            lw $a0, 8($s0)
            lw $a1, 12($s0)
            lw $a2, 16($s0)
            jal maxOf3Ints
            move $t1, $v0
            la $a0, max_of3
            li $v0, 4
            syscall
            move $a0, $t1
            li $v0, 1
            syscall

            lw $s0, -16($fp)
            lw $fp, 24($sp)
            lw $ra, 28($sp)
            addiu $sp, $sp, 32
            li $v0, 10
            syscall
maxOf3Ints:
            addiu $sp, $sp, -32
            sw $ra, 28($sp)
            sw $fp, 24($sp)
            addiu $fp, $sp, 32
```

(continued on the right)

```
            jal maxOf2Ints
            move $a0, $v0
            move $a1, $a2
            jal maxOf2Ints

            lw $fp, 24($sp)
            lw $ra, 28($sp)
            addiu $sp, $sp, 32
            jr $ra
maxOf2Ints: move $v0, $a0
            bge $v0, $a1, over
            move $v0, $a1
over:       jr $ra
```


MIPS32 AL – More On Functions
(function-call convention adapted for leaf $func_s^n$)

just discussed, for non-leaf functions

- Case 1 ⇒ no need for any stack space
  - No local variables + no need to change callee-saved registers
    - `maxOf2Ints` of **015_MIPS32AssemblyLanguageMoreOnFunctions01**
  - Another *e.g.* (for better comparison with other cases to see next)

```
int g(int x, int y)
{
   return x + y;
}
```

```
g: add $v0, $a0, $a1      # put result (sum of arguments) in $v0
   jr $ra                 # return
```

  - No need for any stack frame

MIPS32 AL – More On Functions
(function-call convention adapted for leaf $func_s^n$)

- Case 2a ⇒ need stack space (<= 32 bytes)
  - For local variables

```
int g(int x, int y)
{
   int a[3];
   ... // calculations
   return a[0];
}
```

```
g: addiu $sp, $sp, -32    # push min-sized stack frame
   sw $ra, 28($sp)
   sw $fp, 24($sp)
   addi $fp, $sp, 32
   ...                    # calculations
   lw $v0, -32($fp)       # put result in $v0
   lw $fp, 24($sp)
   lw $ra, 28($sp)
   addiu $sp, $sp, 32     # pop stack frame
   jr $ra                 # return
```

  - Minimum frame size
    - Space requirement of 6 (= 3+1+2) words is less than 32 bytes
  - Calculations should not change any callee-saved registers

MIPS32 AL – More On Functions
(function-call convention adapted for leaf $func_s^n$)

- Case 2b ⇒ need stack space (> 32 bytes)
  - For local variables

```
int g(int x, int y)
{
   int a[15];
   ... // calculations
   return a[0];
}
```

```
g: addiu $sp, $sp, -72    # push stack frame
   sw $ra, 68($sp)
   sw $fp, 64($sp)
   addi $fp, $sp, 72
   ...                    #calculations
   lw $v0, -72($fp)       # put result in $v0
   lw $fp, 64($sp)
   lw $ra, 68($sp)
   addiu $sp, $sp, 72     # pop stack frame
   jr $ra                 # return
```

  - Stack frame of 18 (= 15+1+2) words
    - *Local variables section* padded so that it's divisible by 8
  - Calculations should not change any callee-saved registers

MIPS32 AL – More On Functions
(function-call convention adapted for leaf $func_s^n$)

- Case 3a ⇒ need stack space (<= 32 bytes)
  - For local variables and callee-saved registers

```
int g(int x, int y)
{
   int a[3];
   ... // calculations
   return a[0];
}
```

```
g: addiu $sp, $sp, -32    # push min-size stack frame
   sw $ra, 28($sp)
   sw $fp, 24($sp)
   addi $fp, $sp, 32
   sw $s0, -32($fp)
   ...                    # calculations ($s0 used)
   lw $v0, -24($fp)       # put result in $v0
   lw $s0, -32($fp)
   lw $fp, 24($sp)
   lw $ra, 28($sp)
   addiu $sp, $sp, 32     # pop stack frame
   jr $ra                 # return
```

  - Stack frame of 8 (= 3+1+1+1+2) words
    - *Local variables section* padded so that it's divisible by 8
    - *Callee-saved registers section* padded so that it's divisible by 8

MIPS32 AL – More On Functions
(function-call convention adapted for leaf $func_s^n$)

- Case 3b ⇒ need stack space (> 32 bytes)
  - For local variables and callee-saved registers

```
int g(int x, int y)
{
   int a[25];
   ... // calculations
   return a[0];
}
```

```
g: addiu $sp, $sp, -128   # push stack frame
   sw $ra, 124($sp)
   sw $fp, 120($sp)
   addi $fp, $sp, 128
   sw $s0, -128($fp)
   sw $s1, -124($fp)
   sw $s2, -120($fp)
   ...                    # calc ($s0, $s1, $s2 used)
   lw $v0, -112($fp)      # put result in $v0
   lw $s2, -120($fp)
   lw $s1, -124($fp)
   lw $s0, -128($fp)
   lw $fp, 120($sp)
   lw $ra, 124($sp)
   addiu $sp, $sp, 128    # pop stack frame
   jr $ra                 # return
```

  - Stack frame of 32 (= 25+1+3+1+2) words
    - *Local variables section* padded so that it's divisible by 8
    - *Callee-saved registers section* padded so that it's divisible by 8

MIPS32 AL – More On Functions
(stack frame design card)

Function:			
No stack frame needed ☐ ***Total bytes (>=32):***		leaf	non leaf
$ra and $fp (8)			
local var section	to-DW filler		
	local var		
	$t0-$t9		
padding (0 or 4)			
callee-saved reg section	$s0-$s7		
arguments section	add'l arg		
	$a0-$a3 (16)		