MIPS32 AL – More On Functions
(quick review on recursion)

- Toward a general "definition" of recursion:

  A term used to describe the…

  characteristic feature *of a* ***method*** *of dealing with a* ***subject***…

  whereby…

  part of the method *involves* ***other subjects*** *of the* ***same kind*** *as the* ***subject under consideration***
- Examples of "dealing with a subject" *recursively*:
  - Defining *tree* in terms of *(sub-)tree*s
  - Evaluating *n!* using *(n - 1)!*, *(n - 2)!*, ...
  - Implementing *function* with call(s) to *function being implemented*

    (function's implementation has direct or indirect call(s) to function being implemented)

MIPS32 AL – More On Functions
(quick review on recursion)

- An important role played by recursion in computer science:
  - (recall that computer science is about ***problem solving***)
  - *Divide-and-conquer* problem-solving strategy
    - Solve given problem by solving smaller problems of the same type
- C++ implementation (as function) of associated algorithm:
  - Function body has call(s) to same function → *recursive* function
- Recursive function is usually *less efficient* (resource-wise)
  - Compared to it's *iterative* counterpart
    - (any difficulties involved in obtaining the iterative version aside)
  - Due to *overhead associated with function calls*
- Recursion indispensable for certain important problems
  - (*e.g.*: traversing/processing *non-linear* data structures like trees)
  - Mightily difficult (if not impossible) to deal with iteratively
  - Where recursion finds its niche

MIPS32 AL – More On Functions
(quick review on recursion)

- For recursion to be useful/successful problem-solving tool, insofar as it means using the *divide-and-conquer* strategy, certain conditions must apply:
  - Problem is decomposable into *smaller problems* of the *same type*
  - At least a *base case* exists
    - Also called *anchor case, stopping case, …*
  - Each recursive step *makes progress* toward a base case
  - Base case(s) will *eventually be reached*
- Fatal error can result if method is not properly applied
  - *Infinite recursion*
  - *Stack overflow*

MIPS32 AL – More On Functions
(quick review on recursion)

- Main hurdle students face when applying recursion:
  - Express problem in terms of smaller problems of the same type
- Key to success:
  - Think divide-and-conquer
    - Do only a small part yourself
  - Have faith on others to (together) do the rest
- A simple problem we'll solve/implement recursively
  - Sum numbers from 1 to N (for N >= 1)
- 3 other relatively simple problems (for practice):
  - Flip contents of an array: {1, 2, 3, 4, 5} becomes {5, 4, 3, 2, 1}
  - Search if an array contains a value that matches a given value
  - Determine if an array contains any duplicates
- (You wish they were always so simple!)

MIPS32 AL – More On Functions
(a recursive function example)

- Sum from 1 to N (for N >= 1)

```
int SumToN(int N) // N >= 1 & not too big
{
   if (N < 2)
        return 1;
   else
        return N + SumToN(N – 1);
}
```

- **SumToN** is both caller and callee
- How should we implement in MIPS assembly?

MIPS32 AL – More On Functions
(a recursive function example)

- Sum from 1 to N (for N >= 1)

```
int SumToN(int N) // N >= 1 & not too big
{
   if (N < 2)
        return 1;
   else
        return N + SumToN(N – 1);
}
```

- **SumToN** is both caller and callee
- How should we implement it in MIPS assembly?
- Good news: no new things to be learned
  - We implement it just like any other (non-leaf) function

MIPS32 AL – More On F^{n}_{s}
(a recursive function e.g.)

- Sum from 1 to N

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

```
           .data
str1:      .asciiz "Desired N: "
str2:      .asciiz "Sum to N = "
           .text
           .globl main
main:
           addiu $sp, $sp, -32
           sw $ra, 28 ($sp)
           sw $fp, 24 ($sp)
           addiu $fp, $sp, 32

           la $a0, str1  # "Desired N: "
           li $v0, 4
           syscall
           li $v0, 5
           syscall
           j quitTest
repeat:    move $a0, $v0
           jal SumToN
           move $t0, $v0
           la $a0, str2  # "Sum to N = "
           li $v0, 4
           syscall
           move $a0, $t0
           li $v0, 1
           syscall
           li $a0, '\n'
           li $v0, 11
           syscall
           la $a0, str1  # "Desired N: "
           li $v0, 4
           syscall
           li $v0, 5
           syscall
quitTest:  bgtz $v0, repeat

           lw $fp, 24($sp)
           lw $ra, 28($sp)
           addiu $sp, $sp, 32
           li $v0, 10
           syscall
```


MIPS32 AL – More On F^{n}_{s}
(a recursive function e.g.)

- Let's trace ***sum from 1 to 3***

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 3

```
           .data
str1:      .asciiz "Desired N: "
str2:      .asciiz "Sum to N = "
           .text
           .globl main
main:
           addiu $sp, $sp, -32
           sw $ra, 28 ($sp)
           sw $fp, 24 ($sp)
           addiu $fp, $sp, 32

           la $a0, str1
           li $v0, 4
           syscall
           li $v0, 5
           syscall
           j quitTest
repeat:    move $a0, $v0
           jal SumToN
           move $t0, $v0
           la $a0, str2
           li $v0, 4
           syscall
           move $a0, $t0
           li $v0, 1
           syscall
           li $a0, '\n'
           li $v0, 11
           syscall
           la $a0, str1
           li $v0, 4
           syscall
           li $v0, 5
           syscall
quitTest:  bgtz $v0, repeat

           lw $fp, 24($sp)
           lw $ra, 28($sp)
           addiu $sp, $sp, 32
           li $v0, 10
           syscall
```

MIPS32 AL – More On F^n_s
(a recursive function e.g.)

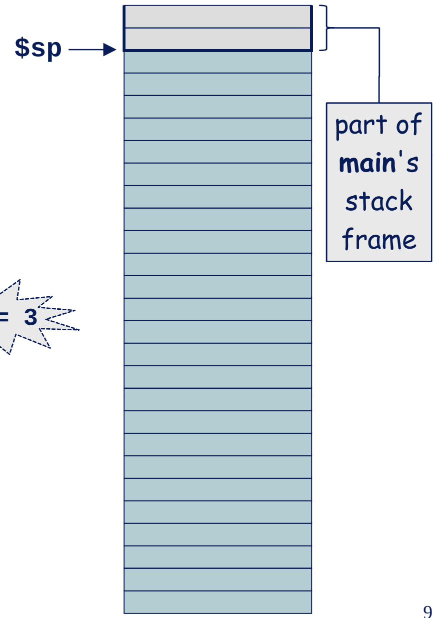
- Let's trace ***sum from 1 to 3***

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 3



MIPS32 AL – More On F^n_s
(a recursive function e.g.)

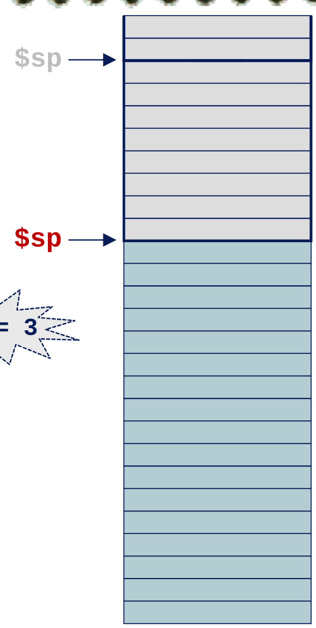
- SumToN(3)

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 3



MIPS32 AL – More On F^n_s

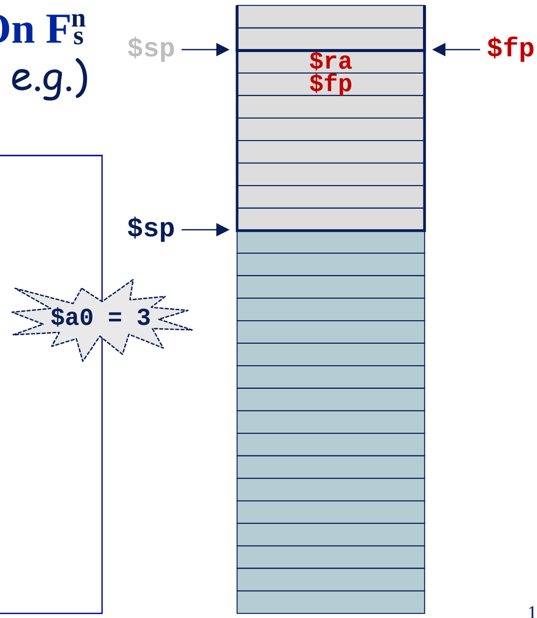
(a recursive function e.g.)

- SumToN(3)

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```



MIPS32 AL – More On F^n_s

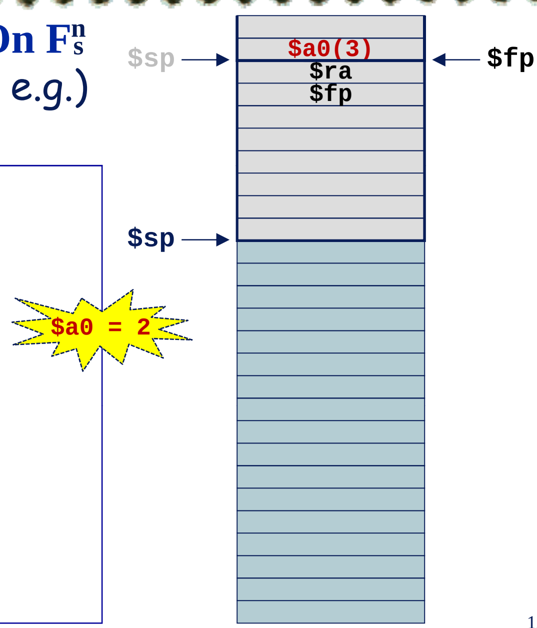
(a recursive function e.g.)

- SumToN(3)

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```



MIPS32 AL – More On F^n_s

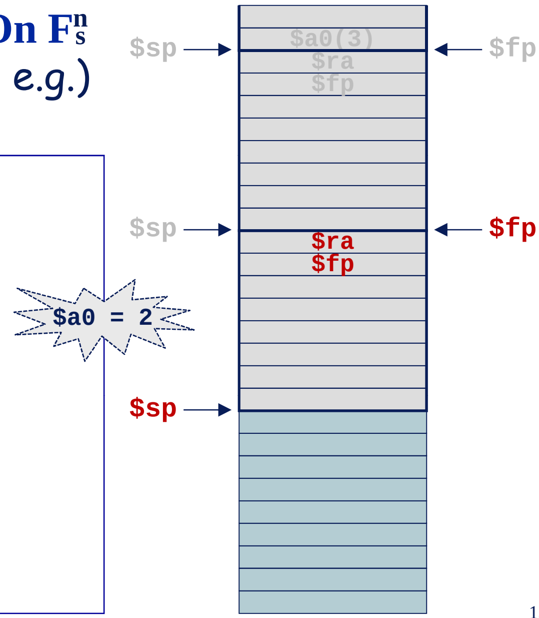
(a recursive function e.g.)

- SumToN(2)

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```



MIPS32 AL – More On F^n_s

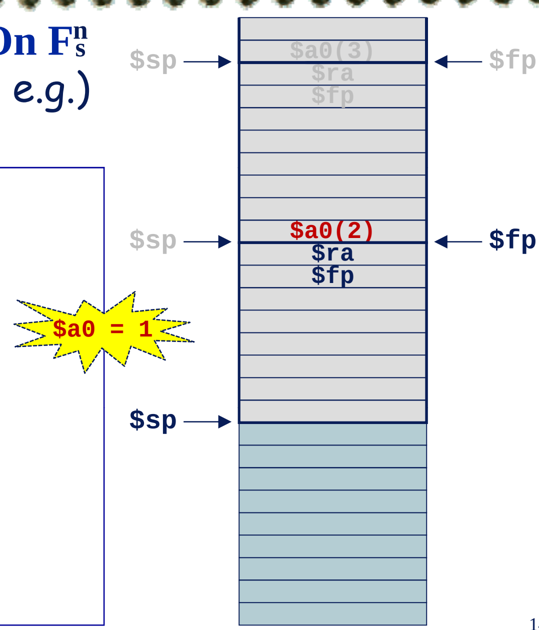
(a recursive function e.g.)

- SumToN(2)

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```



MIPS32 AL – More On F^n_s
(a recursive function e.g.)

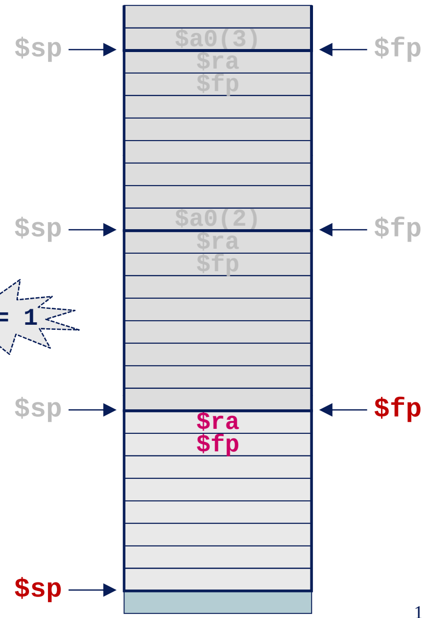
- SumToN(1)

```
SumToN:
         addiu $sp, $sp, -32
         sw $ra, 28 ($sp)
         sw $fp, 24 ($sp)
         addiu $fp, $sp, 32

         slti $t0, $a0, 2
         beq $t0, $0, recur
         addi $v0, $0, 1
         j done
recur:   sw $a0, 0($fp)
         addi $a0, $a0, -1
         jal  SumToN
         lw $a0, 0($fp)
         add $v0, $v0, $a0

done:    lw $fp, 24($sp)
         lw $ra, 28($sp)
         addiu $sp, $sp, 32
         jr $ra
```

$a0 = 1



MIPS32 AL – More On F^n_s
(a recursive function e.g.)

- SumToN(1)

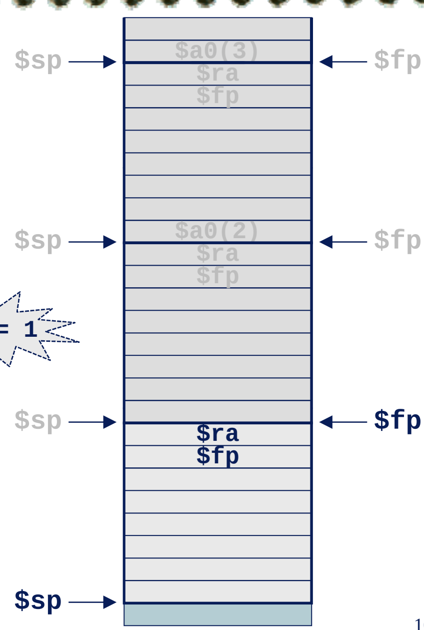
$v0 = 1

```
SumToN:
         addiu $sp, $sp, -32
         sw $ra, 28 ($sp)
         sw $fp, 24 ($sp)
         addiu $fp, $sp, 32

         slti $t0, $a0, 2
         beq $t0, $0, recur
         addi $v0, $0, 1
         j done
recur:   sw $a0, 0($fp)
         addi $a0, $a0, -1
         jal  SumToN
         lw $a0, 0($fp)
         add $v0, $v0, $a0

done:    lw $fp, 24($sp)
         lw $ra, 28($sp)
         addiu $sp, $sp, 32
         jr $ra
```

$a0 = 1



MIPS32 AL – More On F^{n}_{s}

(a recursive function e.g.)

- SumToN(1)

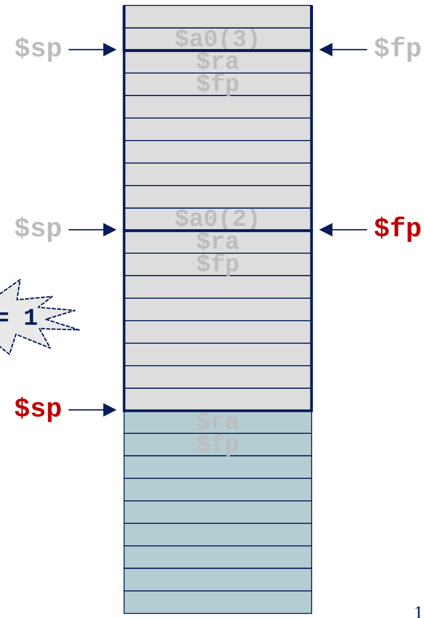
$v0 = 1

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 1



MIPS32 AL – More On F^{n}_{s}

(a recursive function e.g.)

- SumToN(2)

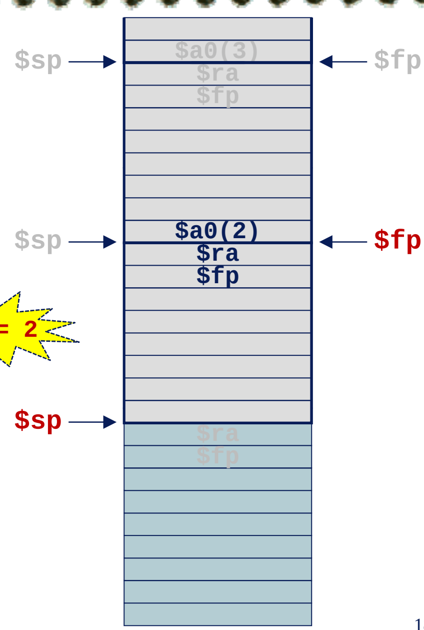
$v0 = 3

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 2



MIPS32 AL – More On F^n_s

(a recursive function e.g.)

- SumToN(2)

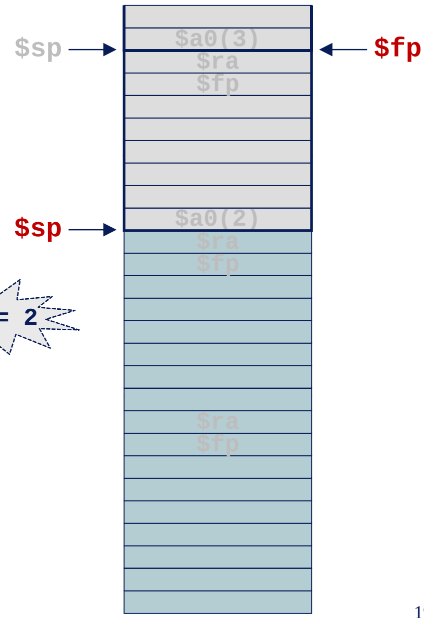
$v0 = 3

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 2



MIPS32 AL – More On F^n_s

(a recursive function e.g.)

- SumToN(3)

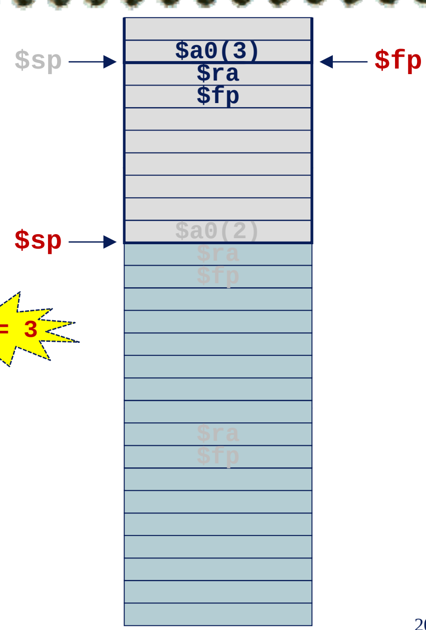
$v0 = 6

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 3



MIPS32 AL – More On F$^{n}_{s}$

(a recursive function e.g.)

- SumToN(3)

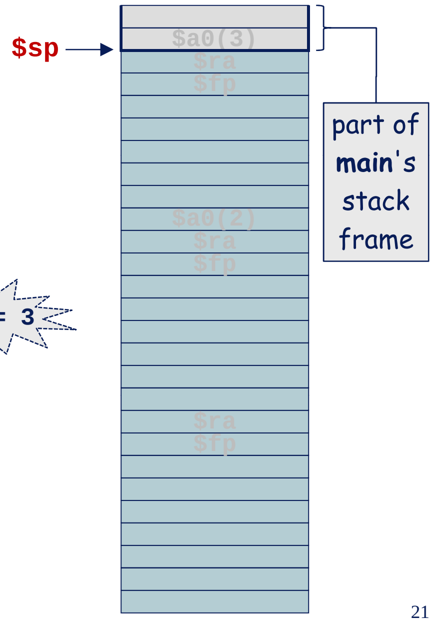
$v0 = 6

```
SumToN:
         addiu $sp, $sp, -32
         sw $ra, 28 ($sp)
         sw $fp, 24 ($sp)
         addiu $fp, $sp, 32

         slti $t0, $a0, 2
         beq $t0, $0, recur
         addi $v0, $0, 1
         j done
recur:   sw $a0, 0($fp)
         addi $a0, $a0, -1
         jal  SumToN
         lw $a0, 0($fp)
         add $v0, $v0, $a0

done:    lw $fp, 24($sp)
         lw $ra, 28($sp)
         addiu $sp, $sp, 32
         jr $ra
```

$a0 = 3



MIPS32 AL – More On F$^{n}_{s}$

(a recursive function e.g.)

- Let's trace *sum f*

$v0 = 6

```
SumToN:
         addiu $sp, $sp, -32
         sw $ra, 28 ($sp)
         sw $fp, 24 ($sp)
         addiu $fp, $sp, 32

         slti $t0, $a0, 2
         beq $t0, $0, recur
         addi $v0, $0, 1
         j done
recur:   sw $a0, 0($fp)
         addi $a0, $a0, -1
         jal  SumToN
         lw $a0, 0($fp)
         add $v0, $v0, $a0

done:    lw $fp, 24($sp)
         lw $ra, 28($sp)
         addiu $sp, $sp, 32
         jr $ra
```

```
             .data
str1:        .asciiz "Desired N: "
str2:        .asciiz "Sum to N = "
             .text
             .globl main
main:
             addiu $sp, $sp, -32
             sw $ra, 28 ($sp)
             sw $fp, 24 ($sp)
             addiu $fp, $sp, 32

             la $a0, str1
             li $v0, 4
             syscall
             li $v0, 5
             syscall
             j quitTest
repeat:      move $a0, $v0
             jal SumToN
             move $t0, $v0
             la $a0, str2
             li $v0, 4
             syscall
             move $a0, $t0
             li $v0, 1
             syscall
             li $a0, '\n'
             li $v0, 11
             syscall
             la $a0, str1
             li $v0, 4
             syscall
             li $v0, 5
             syscall
quitTest:    bgtz $v0, repeat

             lw $fp, 24($sp)
             lw $ra, 28($sp)
             addiu $sp, $sp, 32
             li $v0, 10
             syscall
```

$a0 = 3

$t0 = 6

MIPS32 AL – More On Functions
(reentrant function)

- A ***reentrant*** function is one that…
  - can be used by more than one task concurrently…
  - without fear of data corruption
- A ***non-reentrant*** function is one that…
  - cannot be shared by more than one task unless…
  - mutual exclusion to the function is ensured…
  - by using locking techniques
- A function is ***reentrant*** if, …
  - while it's being run and its execution is interrupted for a while, …
  - it can be re-activated (by itself or another routine) and…
  - still give the same result as if its execution hasn't been interrupted
- (often referred to as "pure code")

MIPS32 AL – More On Functions
(reentrant function)

- Some conditions a function must meet to be reentrant:
  - Never modifies itself (*i.e.*, no self-modifying code)
    - Code for function should not change during function's execution
  - Any variables changed by function must be allocated to each particular "instance" of function call
    - If function REENT is called by 3 different functions, then REENT's "volatile" data must be stored in 3 different/separate areas of memory
  - Must not call any non-reentrant functions
- To be reentrant, for our purpose, a function should…
  - Use *no global variables* (those allocated in *data segment*)
  - Use only local variables allocated in *stack segment*
- ✍ Important when function must be concurrently accessed
  - Multitasking/time-sharing/multi-threading/… environments