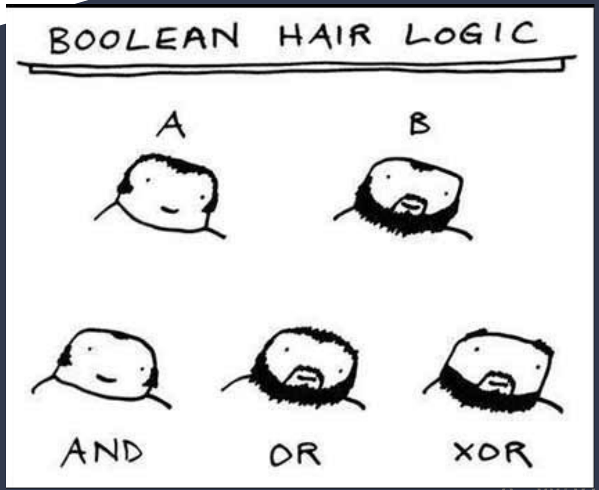
CS 2340 – Computer Architecture

9 Procedures (Leaf/Non-Leaf), Stack

Dr. Alice Wang



Review

Last Lecture

- MIPS has 3 instruction formats
  - R-, I- and J- type
- Practiced being an Assembler and converted assembly code to machine code and back

Today

- Procedures (Leaf/Non-leaf)
  - Stack

By end of today's lecture you will have all of the skills to do the Term Project!

Procedures

pro·ce·dure

/prəˈsējər/

noun

noun: **procedure**; plural noun: **procedures**

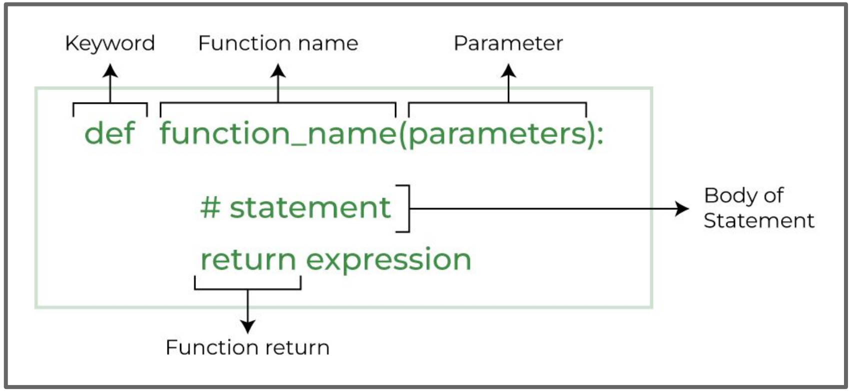
an established or official way of doing something.
"the police are now reviewing procedures"

Similar: course of action | line of action | plan of action | policy | series of steps

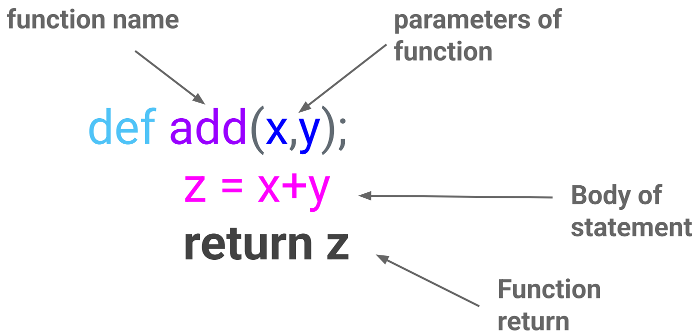
- a series of actions conducted in a certain order or manner.
  "the standard procedure for informing new employees about conditions of work"
- ~~a surgical operation.~~
  ~~"the procedure is carried out under general anesthesia"~~
- COMPUTING
  another term for subroutine.

Procedures enable modularity

- Most important advance in computer science
- Procedures enable structured programming
- Enable very complex programs
- They enable programmers to develop and test parts of a program in isolation
- Procedures help define interfaces between system components



Python Example

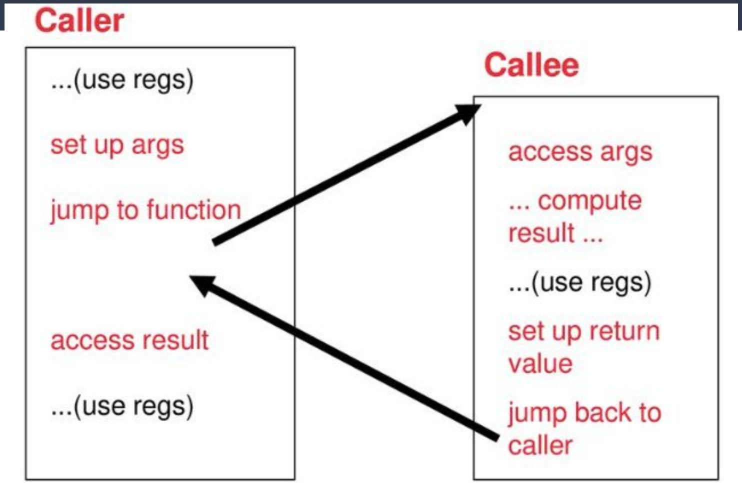


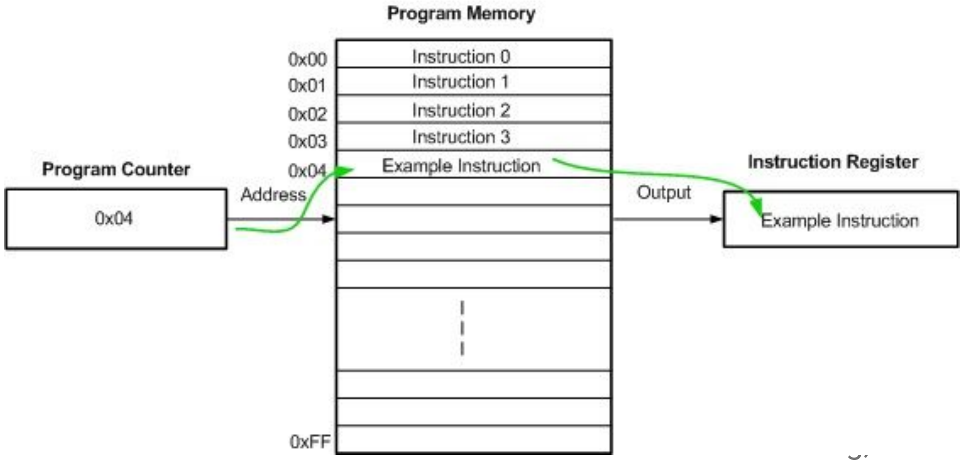
Procedure terminology

Caller - The program that calls or instigates a procedure

Callee - A procedure that executes a series of instructions based on parameters provided by the caller

Program counter (PC) - The register that contains the address of the current instruction being executed





Procedure Call and Return Instructions

- Procedure call: jump and link
  - **`jal ProcedureLabel`**
  - Address of following instruction put in `$ra`
  - Jumps to target address `ProcedureLabel`
- Procedure return: jump register
  - **`jr $ra`**
  - Copies `$ra` to program counter
  - Can also be used for computed jumps to any register
    - e.g., for case/switch statements

Procedure Calling

Steps required

1. Place parameters in passed registers - **`$a0~$a3`**
2. Transfer control to procedure - **`jal`**
3. Acquire storage for procedure - **`addi $sp, $sp, -4`**
4. Perform procedure's operations
5. Place result in register for caller - **`$v0,$v1`**
6. Restore variables and return to place of call - **`jr`**

First: passing arguments and return values through registers

Registers

- Can be overwritten by callee
  - $a0 – $a3: arguments
  - $v0, $v1: result values
  - $t0 – $t9: temporaries
  - …
- Must be saved/restored by callee
  - $s0 – $s7: saved
  - $gp: global pointer for static data
  - $sp: stack pointer
  - $fp: frame pointer

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	No

When you write a procedure <u>you</u> must follow the last column and preserve certain registers across a call, it's not done automatically

Preserved across procedure calls

"Not preserved across procedure calls"

- Register <u>may change</u> value if a procedure is called

"Preserved across procedure calls"

- Assume the register value <u>will not be changed</u> by a procedure
- Store the register value on the stack before you use it
- Restore the register from the stack before jumping back

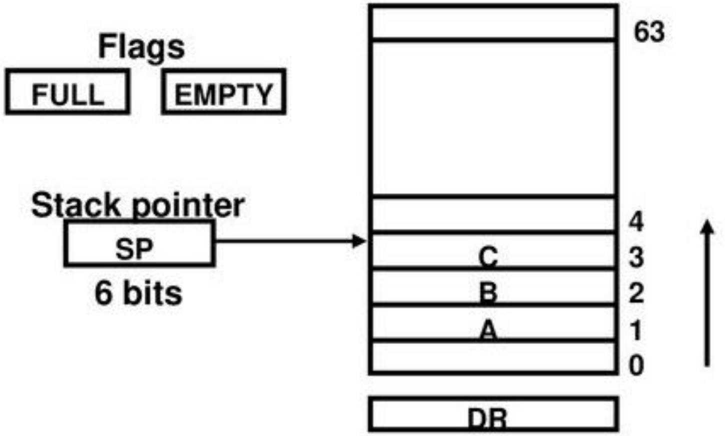
Procedure terminology - Stacks

Stack - A data structure of memory organized as last-in-first-out queue.

Stack pointer - A register that contains the memory address of the last data element added to the stack

Push/Pop - push adds an item to the top of the stack, pop removes the item from the top.

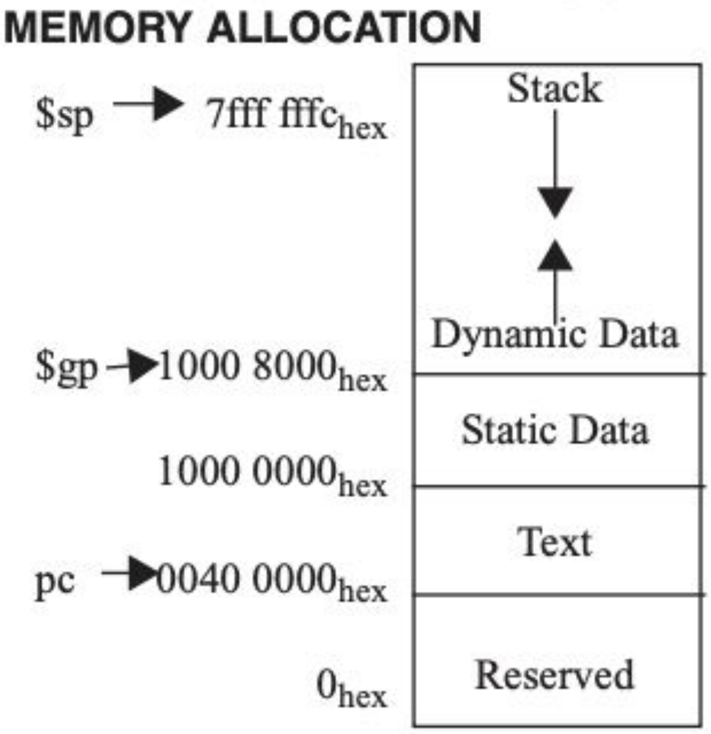
Stack



MIPS Memory Layout

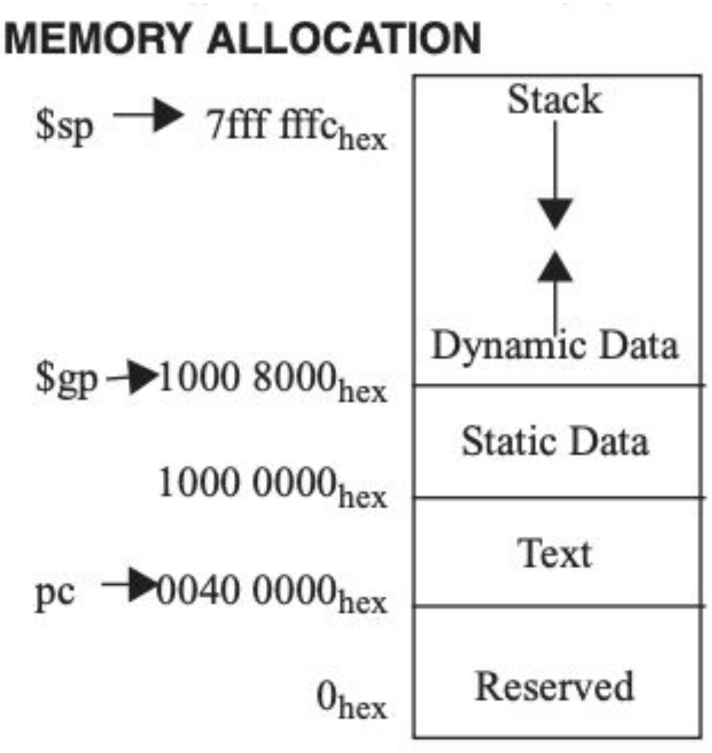
From bottom to top

- Text: program code (.text)
- Static data: global variables
  - static variables in C, constant arrays and strings (.data)
- Dynamic data: heap
  - malloc in C, new in Java, SysAlloc syscall in MIPS
- Stack: last-in first-out
  - allocated by procedures

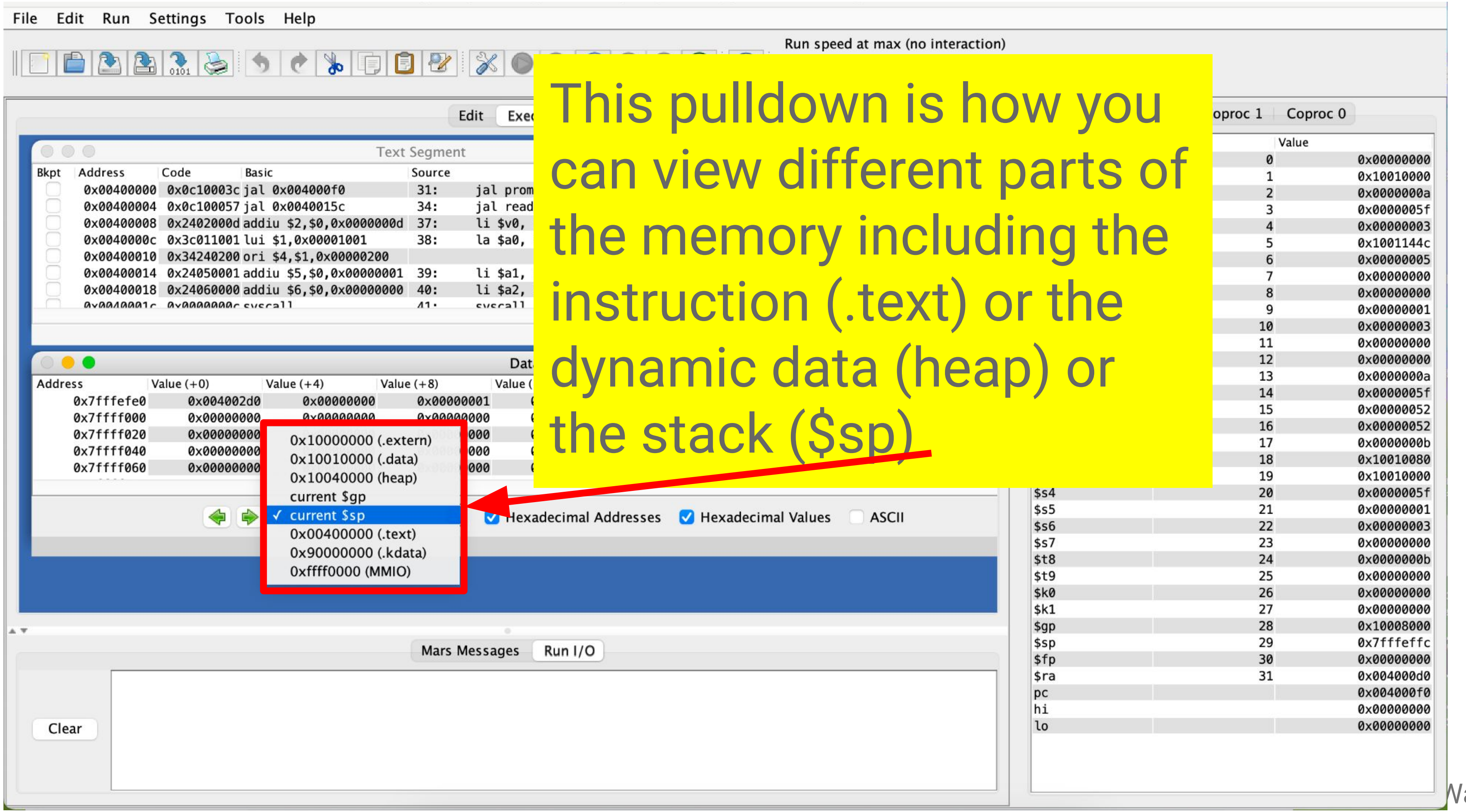


Memory Layout

- Forgetting to free space using free() can case a “memory leak”
- Freeing space too early can turn to “dangling pointers”
- Java has automatic memory allocation to avoid these bugs
- Stack overflow - stack pointer exceeds the stack bound

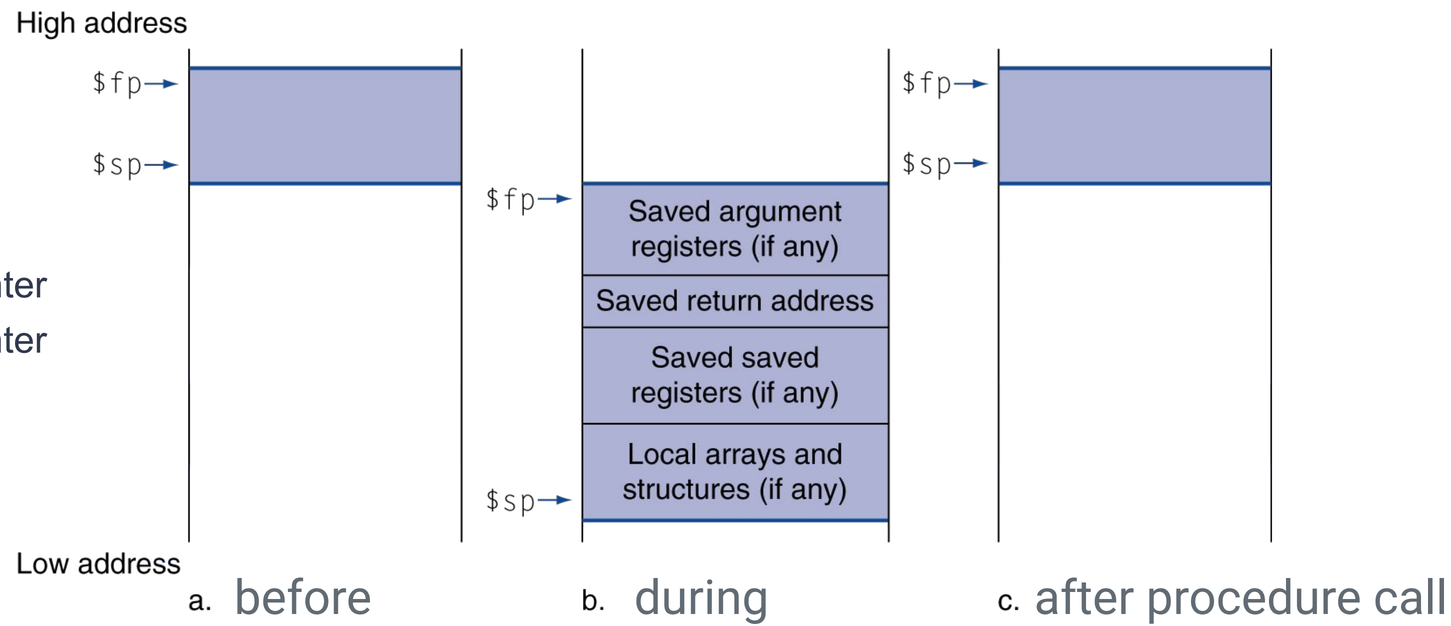


How can you view the Stack? (Mars Demo)



This pulldown is how you can view different parts of the memory including the instruction (.text) or the dynamic data (heap) or the stack ($sp)

Procedures are enabled by a stack



$sp: stack pointer

$fp: frame pointer

- Stack is a local data storage in memory for the callee
  - Data that doesn't fit in registers can be stored on the Stack

Leaf Procedure example – not using the Stack

Python Example

function name

parameters of function

```
def Addfunction(x,y):
    z = x+y
    return z
```

Body of statement

Function return

MIPS Example

function name

parameters of function

```
Addfunction:
    add $t0, $a0, $a1
    move $v0, $t0
    jr $ra
```

Body of statement

Function return

Leaf Procedure - does not call other procedures

Doesn’t use any registers that need to be preserved across a call, doesn’t use stack

Leaf Procedure example – using the Stack

Python Example

```
def Addfunction(x,y):
    z = x+y
    return z
```

- Use **$s1** in the body of the statement, then I need to store **$s1** on the stack beforehand
- **$s1** needs to be preserved across a call

MIPS Example

```
Addfunction:
    addi $sp, $sp, -4
    sw $s1, 0($sp)

    add $s1, $a0, $a1
    move $v0, $s1

    lw $s1, 0($sp)
    addi $sp, $sp, 4
    jr $ra
```

Allocate storage on stack: -4*# of elements

Store data on the stack before it is used

Body of statement

Restore data from the stack

Free storage on stack: 4*# of elements

Leaf Procedure Example

Python code:

```
def Leaf_example (g, h, i, j):
	f = (g + h) - (i + j)
	return f
```

Step 1: Arguments g, h, i, j placed in $a0, $a1, $a2, $a3

result f passed back through $v0

Leaf Procedure – My Turn

Python code:

```
def Leaf_example (g, h, i, j):
    f = (g+h) - (i+j)
    return f
```

Used $s2 to calculate f, thus need to save $s2 on the Stack

Restore $s2 before jumping back

MIPS code:

```
Leaf_example:                    # Step 2.Transfer Control
   addi $sp, $sp, -4             # Step 3. Acquire storage
   sw   $s2, 0($sp)              #  save $s0 on stack

   add  $t0, $a0, $a1            # Step 4. Perform
   add  $t1, $a2, $a3            #  procedure's operation
   sub  $s2, $t0, $t1

   add  $v0, $s0, $0             # Step 5. Place result
                                 #  in register for caller

   lw   $s2, 0($sp)              # Step 6. Restore $s0 and
   addi $sp, $sp, 4              #  return to place of
   jr   $ra                      #  call
```

Leaf Procedure – My Turn

MIPS code:

```
Leaf_example:               # Step 2.Transfer Control
  addi $sp, $sp, -4         # Step 3. Acquire storage
  sw   $s0, 0($sp)          #   save $s0 on stack

  add  $t0, $a0, $a1        # Step 4. Perform
  add  $t1, $a2, $a3        #   procedure’s operation
  sub  $s0, $t0, $t1

  add  $v0, $s0, $0         # Step 5. Place result
                            #   in register for caller

  lw   $s0, 0($sp)          # Step 6. Restore $s0 and
  addi $sp, $sp, 4          #   return to place of call
  jr   $ra
```

Registers

$v0	0
$a0	3
$a1	4
$a2	5
$a3	6
$t0	0
$t1	0
$s0	50
$ra	rtn_addr

Stack

$sp →		20
		1C
		18
		14
		10
		0C

Leaf Procedure – My Turn

MIPS code:

```
Leaf_example:              # Step 2.Transfer Control
  addi $sp, $sp, -4        # Step 3. Acquire storage
  sw   $s0, 0($sp)         #   save $s0 on stack

  add  $t0, $a0, $a1       # Step 4. Perform
  add  $t1, $a2, $a3       #   procedure's operation
  sub  $s0, $t0, $t1

  add  $v0, $s0, $0        # Step 5. Place result
                           #   in register for caller

  lw   $s0, 0($sp)         # Step 6. Restore $s0 and
  addi $sp, $sp, 4         #   return to place of call
  jr   $ra
```

Registers

$v0	0
$a0	3
$a1	4
$a2	5
$a3	6
$t0	**3+4=7**
$t1	**5+6=11**
$s0	**7-11= -4**
$ra	rtn_addr

Stack

$sp →

	20
50	1C
	18
	14
	10
	0C

Leaf Procedure – My Turn

MIPS code:

```
Leaf_example:               # Step 2.Transfer Control
  addi $sp, $sp, -4         # Step 3. Acquire storage
  sw   $s0, 0($sp)          #   save $s0 on stack

  add  $t0, $a0, $a1        # Step 4. Perform
  add  $t1, $a2, $a3        #   procedure's operation
  sub  $s0, $t0, $t1

  add  $v0, $s0, $ 0        # Step 5. Place result
                            #   in register for caller

  lw   $s0, 0($sp)          # Step 6. Restore $s0 and
  addi $sp, $sp, 4          #   return to place of call
  jr   $ra
```

Registers

$v0	**-4**
$a0	3
$a1	4
$a2	5
$a3	6
$t0	7
$t1	11
$s0	-4
$ra	rtn_addr

Stack

		20
$sp →	50	1C
		18
		14
		10
		0C

Leaf Procedure – My Turn

MIPS code:

```
Leaf_example:               # Step 2.Transfer Control
  addi $sp, $sp, -4         # Step 3. Acquire storage
  sw   $s0, 0($sp)          #   save $s0 on stack

  add  $t0, $a0, $a1        # Step 4. Perform
  add  $t1, $a2, $a3        #   procedure's operation
  sub  $s0, $t0, $t1

  add  $v0, $s0, $0         # Step 5. Place result
                            #   in register for caller

  lw   $s0, 0($sp)          # Step 6. Restore $s0 and
  addi $sp, $sp, 4          #   return to place of call
  jr   $ra
```

Registers

$v0	-4
$a0	3
$a1	4
$a2	5
$a3	6
$t0	7
$t1	11
$s0	**50**
$ra	rtn_addr

Stack

$sp →

	20
50	1C
	18
	14
	10
	0C

Leaf Procedure – Your Turn

MIPS code:

```
Leaf_example:                    # Step 2.Transfer Control
    <Store data on stack>        # Step 3. Acquire storage

    add  $s2, $a0, $a1
    add  $s1, $a2, $a3
    sub  $s0, $s1, $s2

    add  $v0, $s0, $0            # Step 5. Place result
                                 #   in register for caller
    <Restore data on stack>      # Step 6. Restore data and
    addi $sp, $sp, 4             #   return to place of call
    jr   $ra
```

Based on the program how many bytes do we need to allocate on the stack?

Leaf Procedure – Your Turn

MIPS code:

```
Leaf_example:                 # Step 2.Transfer Control
    addi $sp, $sp, -12        # Step 3. Acquire storage
    sw   $s0, 0($sp)          #   save $s0 on stack
    sw   $s1, 4($sp)          #   save $s1 on stack
    sw   $s2, 8($sp)          #   save $s2 on stack

    add  $s2, $a0, $a1
    add  $s1, $a2, $a3
    sub  $s0, $s1, $s2

    add  $v0, $s0, $0         # Step 5. Place result
                              #   in register for caller
    lw   $s0, 0($sp)          # Step 6. Restore $s0 and
    addi $sp, $sp, 4          #   return to place of call
    jr   $ra
```

What's wrong with this statement?

Leaf Procedure – Your Turn

MIPS code:

```
Leaf_example:                # Step 2.Transfer Control
    addi $sp, $sp, -12       # Step 3. Acquire storage
    sw   $s0, 0($sp)         #   save $s0 on stack
    sw   $s1, 4($sp)         #   save $s1 on stack
    sw   $s2, 8($sp)         #   save $s2 on stack

    add  $s2, $a0, $a1
    add  $s1, $a2, $a3
    sub  $s0, $s1, $s2

    add  $v0, $s0, $0        # Step 5. Place result
                             # in register for caller
    lw   $s2, 8($sp)         # Restore $s2
    lw   $s1, 4($sp)         # Restore $s1
    lw   $s0, 0($sp)         # Restore $s0
    addi $sp, $sp, 12
    jr   $ra                 # Step 6. Return to
                             # place of call
```

MIPS rules : We must restore all saved registers to original values before jumping back. We must have the stack pointer at the same place before jumping back

Non–Leaf Procedure

- A Non-Leaf procedure calls other procedures
- Non-leaf procedures <u>always</u> need to use the stack, at minimum to store the return address of the calling procedure ($ra).
- What happens if we don’t store the return address? …

Non-Leaf Procedure – Example (bad)

```
main:
        la $s0, A
        lw $a0, 0($s0)
        jal myproc1
        sw $v0, 4($s0)
        j exit

myproc1: #non-leaf
        addi $v0, $a0, 5
        move $a0, $v0
        jal myproc2
        jr $ra

myproc2: #leaf
        addi $v0, $a0, 10
        jr $ra
```

- *myproc1* is a non-leaf procedure because it calls myproc2
- $ra is set to the address of **`sw`** instruction

Non-Leaf Procedure – Example (bad)

```
main:
        la $s0, A
        lw $a0, 0($s0)
        jal myproc1
        sw $v0, 4($s0)
        j exit

myproc1: #non-leaf
        addi $v0, $a0, 5
        move $a0, $v0
        jal myproc2
        jr $ra

myproc2: #leaf
        addi $v0, $a0, 10
        jr $ra
```

- *myproc1* is a non-leaf procedure because it calls myproc2
- $ra is set to the address of **`sw`** instruction
- When *myproc2* is called $ra is set to the address of **`jr`** instruction

Non-Leaf Procedure - Example (bad)

```
main:
        la $s0, A
        lw $a0, 0($s0)
        jal myproc1
        sw $v0, 4($s0)
        j exit

myproc1: #non-leaf
        addi $v0, $a0, 5
        move $a0, $v0
        jal myproc2
        jr $ra

myproc2: #leaf
        addi $v0, $a0, 10
        jr $ra
```

- *myproc1* is a non-leaf procedure because it calls myproc2
- $ra is set to the address of **`sw`** instruction
- When *myproc2* is called $ra is set to the address of **`jr`** instruction
- Jump return from myproc2 is OK

Non-Leaf Procedure - Example (bad)

```
main:
        la $s0, A
        lw $a0, 0($s0)
        jal myproc1
        sw $v0, 4($s0)
        j exit

myproc1: #non-leaf
        addi $v0, $a0, 5
        move $a0, $v0
        jal myproc2
        jr $ra

myproc2: #leaf
        addi $v0, $a0, 10
        jr $ra
```

- *myproc1* is a non-leaf procedure because it calls myproc2
- $ra is set to the address of **`sw`** instruction
- When *myproc2* is called $ra is set to the address of `jr` instruction
- Jump return from myproc2 is OK
- Jump return from myproc1 is an infinite loop!

Non-Leaf Procedure – Example (good)

```
main:
        la $s0, A
        lw $a0, 0($s0)
        jal myproc1
        sw $v0, 4($s0)
        j exit

myproc1: # non-leaf
        addi $sp, $sp, -4
        sw   $ra, 0($sp)

        addi $v0, $a0, 5
        move $a0, $v0
        jal myproc2

        lw   $ra, 0($sp)
        addi $sp, $sp, 4

        jr $ra

myproc2: #leaf
        addi $v0, $a0, 10
        jr $ra
```

Solution is to store the $ra to the stack before calling myproc2

Store $ra to stack before calling myproc2

Restore $ra to stack after calling myproc2

Recursive Procedure

re·cur·sive

/rəˈkərsiv/

adjective

characterized by recurrence or repetition.

- MATHEMATICS · LINGUISTICS
  relating to or involving the repeated application of a rule, definition, or procedure to successive results.
  "this restriction ensures that the grammar is recursive"
- COMPUTING
  relating to or involving a program or routine of which a part requires the application of the whole, so that its explicit interpretation requires in general many successive executions.
  "a recursive subroutine"

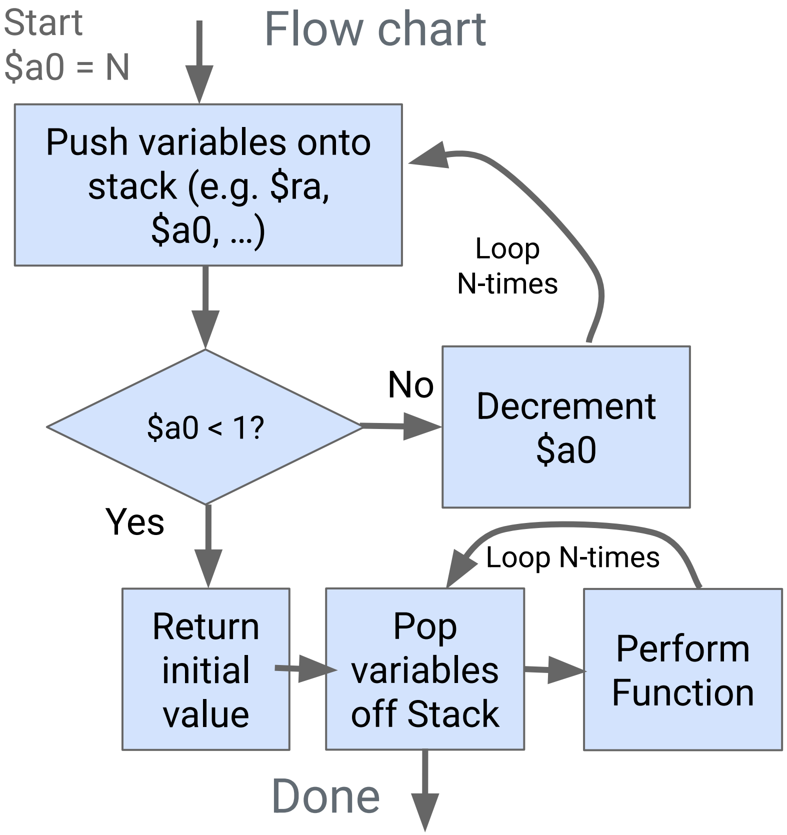
- Recursive Procedure or nested call is an example of a non-leaf procedure
  - One of the steps of the procedure involves invoking the procedure itself

Procedure Calling – Recursive

Steps required

1. Place parameters in passed registers - **`$a0~$a3`**
2. Transfer control to procedure - **`jal`**
3. Acquire storage for procedure - **`addi $sp, $sp, -4`**
4. Perform procedure's operations
   - Save return address (**`$ra`**)
   - Any arguments and temporaries needed after the call
   - Call the procedure itself
5. Place result in register for caller - **`$v0,$v1`**
6. Restore variables and return to place of call - **`jr`**
   - Restore from stack after the call
   - May return to place of call multiple times

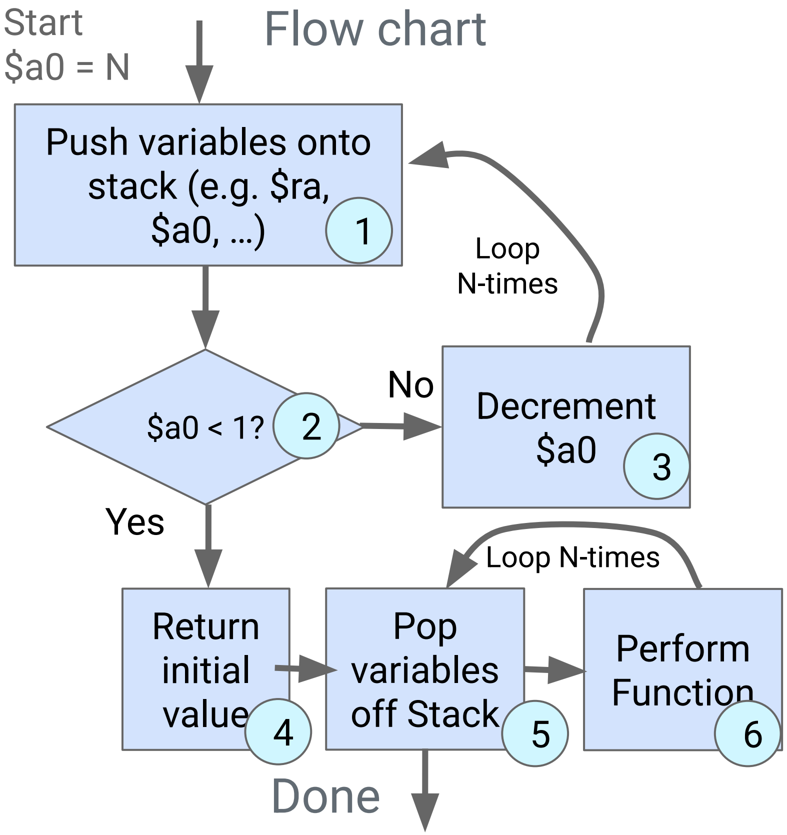
Recursive MIPS program



Loop 1: Push variables onto the Stack

Loop 2: Pop values and perform functions

Recursive MIPS program




```
NonLeafProc:
     addi $sp, $sp, -8        1
     sw   $ra, 4($sp)
     sw   $a0, 0($sp)

     slti $t0, $a0, 1          2
     beq  $t0, $zero, Loop1
     <Return initial value>    4

     addi $sp, $sp, 8
     jr   $ra

Loop1:
     addi $a0, $a0, -1        3
     jal  NonLeafProc

     lw   $a0, 0($sp)         5
     lw   $ra, 4($sp)
     addi $sp, $sp, 8
     <Perform Function>       6
     jr   $ra
```

Factorial C code – Recursive procedure

C code:

```
int fact (int n){
  if (n < 1) return 1;
  else return n * fact(n - 1);
}
```

Python code:

```
def fact (n):
  if n < 1:
return 1
  else: return n * fact(n-1)
```

n = 3, execute fact (3)

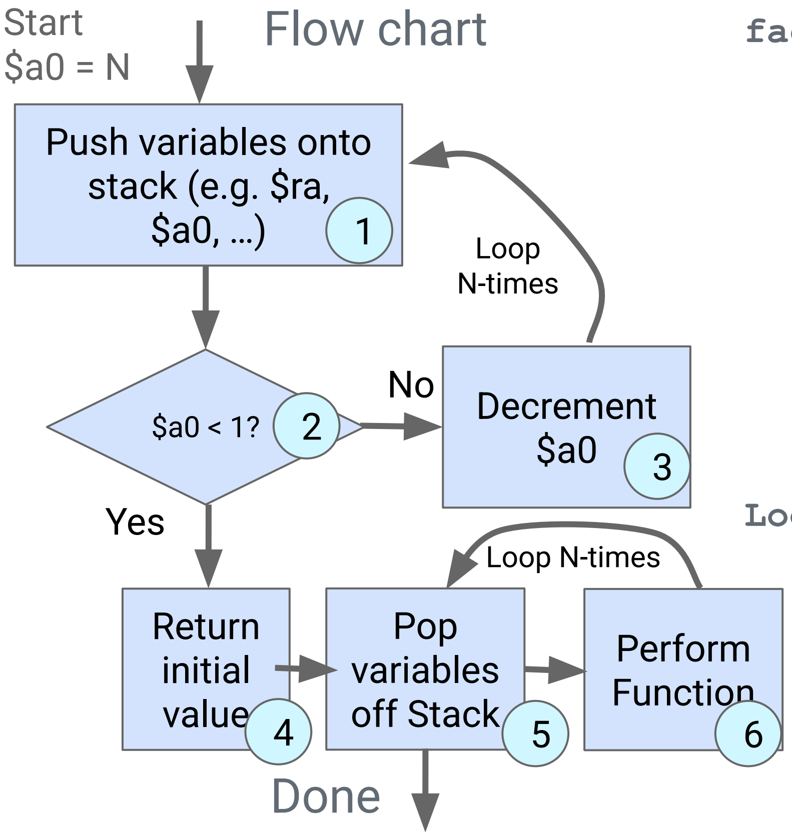
3 * fact (2)

3 * 2 * fact (1)

3 * 2 * 1 * fact (0)

3 * 2 * 1 * 1 = 6

Recursive MIPS program – Factorial Example




```
fact:
      addi $sp, $sp, -8
      sw   $ra, 4($sp)        1
      sw   $a0, 0($sp)

      slti $t0, $a0, 1          2
      beq  $t0, $zero, Loop1
      addi $v0, $zero, 1        4

      addi $sp, $sp, 8
      jr   $ra

Loop1:
      addi $a0, $a0, -1       3
      jal  fact

      lw   $a0, 0($sp)        5
      lw   $ra, 4($sp)
      addi $sp, $sp, 8
      mul  $v0, $a0, $v0      6
      jr   $ra
```

Recursive Procedure Example

C code:

```
int fact (int n){
  if (n < 1) return 1;
  else return n *
   fact(n - 1);
}
```

Implemented in MIPS with 2 procedures

MIPS code:

```
fact:                        # Transfer control
     addi $sp, $sp, -8       # Acquire storage
     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     # branch if !(n < 1)
     addi $v0, $zero, 1      # if n < 1 result is 1

     addi $sp, $sp, 8        # pop 2 items from stack
     jr   $ra                # and return

L1:  addi $a0, $a0, -1       # 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
```

Recursive Procedure Example

C code:

```
int fact (int n){
  if (n < 1) return 1;
  else return n *
   fact(n - 1);
}
```

Step 1: Put argument n in $a0

Expect result in $v0

MIPS code:

```
fact:                           # Transfer control
    addi $sp, $sp, -8           # Acquire storage
    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         # branch if !(n < 1)
    addi $v0, $zero, 1          # if n < 1 result is 1

    addi $sp, $sp, 8            # pop 2 items from stack
    jr   $ra                    # and return

L1: addi $a0, $a0, -1           # 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
```

Unrolling the Recursive Call

```
li $a0, 3                 # n = 3
jal fact                  # Calling fact
Done:...

fact:                     # Transfer control
    addi $sp, $sp, -8     # Acquire storage
    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   # branch if !(n < 1)
    addi $v0, $zero, 1    # if n < 1 result is 1

    addi $sp, $sp, 8      # pop 2 items from stack
    jr   $ra              # and return

L1: addi $a0, $a0, -1     # 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
```

Registers

$v0	
$a0	3
…	
$t0	
…	
$ra	Done

Stack

$sp →

	20
	1C
	18
	14
	10
	0C
	08
	04
	00

Unrolling the Recursive Call

```
li $a0, 3                  # n = 3
jal fact                   # Calling fact
Done:...

fact:                      # Transfer control
     addi $sp, $sp, -8     # Acquire storage
     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   # branch if !(n < 1)
     addi $v0, $zero, 1    # if n < 1 result is 1

     addi $sp, $sp, 8      # pop 2 items from stack
     jr   $ra              # and return

L1:  addi $a0, $a0, -1     # 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
```

Registers

$v0	
$a0	3
…	
$t0	
…	
$ra	0

Stack

	20
$ra = 0	1C
$a0 = 3	18
	14
	10
	0C
	08
	04
	00

$sp

Unrolling the Recursive Call

```
li $a0, 3                # n = 3
jal fact                 # Calling fact
Done:...

fact:                    # Transfer control
    addi $sp, $sp, -8    # Acquire storage
    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  # branch if !(n < 1)
    addi $v0, $zero, 1   # if n < 1 result is 1

    addi $sp, $sp, 8     # pop 2 items from stack
    jr   $ra             # and return

L1: addi $a0, $a0, -1    # 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
```

Registers

$v0	
$a0	3
...	
$t0	**0**
...	
$ra	0

Stack

	20
$ra = 0	1C
$a0 = 3	18
	14
	10
	0C
	08
	04
	00

$sp

Unrolling the Recursive Call

```
li $a0, 3                # n = 3
jal fact                 # Calling fact
Done:...

fact:                    # Transfer control
     addi $sp, $sp, -8   # Acquire storage
     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 # branch if !(n < 1)
     addi $v0, $zero, 1  # if n < 1 result is 1

     addi $sp, $sp, 8    # pop 2 items from stack
     jr   $ra            # and return

L1:  addi $a0, $a0, -1   # 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
```

Registers

$v0	
$a0	2
…	
$t0	0
…	
$ra	L1+8

Stack

$sp

	20
$ra = 0	1C
$a0 = 3	18
	14
	10
	0C
	08
	04
	00

Unrolling the Recursive Call

```
li $a0, 3                    # n = 3
jal fact                     # Calling fact
Done:...

fact:                        # Transfer control
     addi $sp, $sp, -8       # Acquire storage
     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     # branch if !(n < 1)
     addi $v0, $zero, 1      # if n < 1 result is 1

     addi $sp, $sp, 8        # pop 2 items from stack
     jr   $ra                # and return

L1:  addi $a0, $a0, -1       # 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
```

Registers

$v0	
$a0	2
…	
$t0	0
…	
$ra	L1+8

Stack

	20
$ra = 0	1C
$a0 = 3	18
$ra = L1+8	14
$a0 = 2	10
	0C
	08
	04
	00

$sp

Unrolling the Recursive Call

```
li $a0, 3                  # n = 3
jal fact                   # Calling fact
Done:...

fact:                      # Transfer control
    addi $sp, $sp, -8      # Acquire storage
    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    # branch if !(n < 1)
    addi $v0, $zero, 1     # if n < 1 result is 1

    addi $sp, $sp, 8       # pop 2 items from stack
    jr   $ra               # and return

L1: addi $a0, $a0, -1      # 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
```

Registers

$v0	
$a0	2
…	
$t0	**0**
…	
$ra	L1+8

Stack

	20
$ra = 0	1C
$a0 = 3	18
$ra = L1+8	14
$a0 = 2	10
	0C
	08
	04
	00

$sp

Unrolling the Recursive Call

```
li $a0, 3                   # n = 3
jal fact                    # Calling fact
Done:...

fact:                       # Transfer control
     addi $sp, $sp, -8      # Acquire storage
     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    # branch if !(n < 1)
     addi $v0, $zero, 1     # if n < 1 result is 1

     addi $sp, $sp, 8       # pop 2 items from stack
     jr   $ra               # and return

L1:  addi $a0, $a0, -1      # 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
```

Registers

$v0	
$a0	**1**
…	
$t0	0
…	
$ra	**L1+8**

Stack

		20
	$ra = 0	1C
	$a0 = 3	18
	$ra = L1+8	14
$sp →	$a0 = 2	10
		0C
		08
		04
		00

Unrolling the Recursive Call

```
li $a0, 3                # n = 3
jal fact                 # Calling fact
Done:...

fact:                    # Transfer control
    addi $sp, $sp, -8    # Acquire storage
    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  # branch if !(n < 1)
    addi $v0, $zero, 1   # if n < 1 result is 1

    addi $sp, $sp, 8     # pop 2 items from stack
    jr   $ra             # and return

L1: addi $a0, $a0, -1    # 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
```

Registers

$v0	
$a0	1
…	
$t0	0
…	
$ra	L1+8

Stack

	20
$ra = 0	1C
$a0 = 3	18
$ra = L1+8	14
$a0 = 2	10
$ra = L1+8	0C
$a0 = 1	08
	04
	00

$sp

Unrolling the Recursive Call

```
li $a0, 3                # n = 3
jal fact                 # Calling fact
Done:...

fact:                    # Transfer control
    addi $sp, $sp, -8    # Acquire storage
    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  # branch if !(n < 1)
    addi $v0, $zero, 1   # if n < 1 result is 1

    addi $sp, $sp, 8     # pop 2 items from stack
    jr   $ra             # and return

L1: addi $a0, $a0, -1    # 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
```

Registers

$v0	
$a0	1
…	
$t0	**0**
…	
$ra	L1+8

Stack

	20
$ra = 0	1C
$a0 = 3	18
$ra = L1+8	14
$a0 = 2	10
$ra = L1+8	0C
$a0 = 1	08
	04
	00

$sp

Unrolling the Recursive Call

```
li $a0, 3                   # n = 3
jal fact                    # Calling fact
Done:...

fact:                       # Transfer control
     addi $sp, $sp, -8      # Acquire storage
     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    # branch if !(n < 1)
     addi $v0, $zero, 1     # if n < 1 result is 1

     addi $sp, $sp, 8       # pop 2 items from stack
     jr   $ra               # and return

L1:  addi $a0, $a0, -1      # 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
```

Registers

$v0	
$a0	**0**
…	
$t0	0
…	
$ra	**L1+8**

Stack

	20
$ra = 0	1C
$a0 = 3	18
$ra = L1+8	14
$a0 = 2	10
$ra = L1+8	0C
$a0 = 1	08
	04
	00

$sp

Unrolling the Recursive Call

```
li $a0, 3                  # n = 3
jal fact                   # Calling fact
Done:...

fact:                      # Transfer control
    addi $sp, $sp, -8      # Acquire storage
    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    # branch if !(n < 1)
    addi $v0, $zero, 1     # if n < 1 result is 1

    addi $sp, $sp, 8       # pop 2 items from stack
    jr   $ra               # and return

L1: addi $a0, $a0, -1      # 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
```

Pushed onto the stack all of the data needed for the recursive call. Time to pop…

Registers

$v0	
$a0	0
…	
$t0	0
…	
$ra	L1+8

Stack

	20
$ra = 0	1C
$a0 = 3	18
$ra = L1+8	14
$a0 = 2	10
$ra = L1+8	0C
$a0 = 1	08
$ra = L1+8	04
$a0 = 0	00

$sp

Unrolling the Recursive Call

```
li $a0, 3                  # n = 3
jal fact                   # Calling fact
Done:...

fact:                      # Transfer control
    addi $sp, $sp, -8      # Acquire storage
    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    # branch if !(n < 1)
    addi $v0, $zero, 1     # if n < 1 result is 1

    addi $sp, $sp, 8       # pop 2 items from stack
    jr   $ra               # and return

L1: addi $a0, $a0, -1      # 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
```

Registers

$v0	1
$a0	0
…	
$t0	1
…	
$ra	L1+8

Stack

	20
$ra = 0	1C
$a0 = 3	18
$ra = L1+8	14
$a0 = 2	10
$ra = L1+8	0C
$a0 = 1	08
$ra = L1+8	04
$a0 = 0	00

$sp

Unrolling the Recursive Call

```
li $a0, 3                  # n = 3
jal fact                   # Calling fact
Done:...

fact:                      # Transfer control
    addi $sp, $sp, -8      # Acquire storage
    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    # branch if !(n < 1)
    addi $v0, $zero, 1     # if n < 1 result is 1

    addi $sp, $sp, 8       # pop 2 items from stack
    jr   $ra               # and return

L1: addi $a0, $a0, -1      # 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
```

Registers

$v0	1
$a0	0
…	
$t0	1
…	
$ra	**L1+8**

Stack

$sp

	20
$ra = 0	1C
$a0 = 3	18
$ra = L1+8	14
$a0 = 2	10
$ra = L1+8	0C
$a0 = 1	08
$ra = L1+8	04
$a0 = 0	00

Unrolling the Recursive Call

```
li $a0, 3                   # n = 3
jal fact                    # Calling fact
Done:...

fact:                       # Transfer control
    addi $sp, $sp, -8       # Acquire storage
    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     # branch if !(n < 1)
    addi $v0, $zero, 1      # if n < 1 result is 1

    addi $sp, $sp, 8        # pop 2 items from stack
    jr   $ra                # and return

L1: addi $a0, $a0, -1       # 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
```

Registers

$v0	**1=1*1**
$a0	**1**
…	
$t0	1
…	
$ra	**L1+8**

Stack

	20
$ra = 0	1C
$a0 = 3	18
$ra = L1+8	14
$a0 = 2	10
$ra = L1+8	0C
$a0 = 1	08
$ra = L1+8	04
$a0 = 0	00

$sp

Unrolling the Recursive Call

```
li $a0, 3                 # n = 3
jal fact                  # Calling fact
Done:...

fact:                     # Transfer control
    addi $sp, $sp, -8     # Acquire storage
    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   # branch if !(n < 1)
    addi $v0, $zero, 1    # if n < 1 result is 1

    addi $sp, $sp, 8      # pop 2 items from stack
    jr   $ra              # and return

L1: addi $a0, $a0, -1     # 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
```

Registers

$v0	**2=2*1**
$a0	**2**
…	
$t0	1
…	
$ra	**L1+8**

Stack

	20
$ra = 0	1C
$a0 = 3	18
$ra = L1+8	14
$a0 = 2	10
$ra = L1+8	0C
$a0 = 1	08
$ra = L1+8	04
$a0 = 0	00

$sp

Unrolling the Recursive Call

```
li $a0, 3                  # n = 3
jal fact                   # Calling fact
Done:...

fact:                      # Transfer control
    addi $sp, $sp, -8      # Acquire storage
    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    # branch if !(n < 1)
    addi $v0, $zero, 1     # if n < 1 result is 1

    addi $sp, $sp, 8       # pop 2 items from stack
    jr   $ra               # and return

L1: addi $a0, $a0, -1      # 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
```

$v0 = 6
$ra, $sp are restored

Registers

$v0	**6=3*2**
$a0	**3**
…	
$t0	1
…	
$ra	**Done**

Stack

$sp

	20
$ra = 0	1C
$a0 = 3	18
$ra = L1+8	14
$a0 = 2	10
$ra = L1+8	0C
$a0 = 1	08
$ra = L1+8	04
$a0 = 0	00

Leaf Procedure example – My turn

Execute the following procedure:

```
MyProc:
        addi $sp, $sp, -12
        sw $s0, 8($sp)
        sw $s1, 4($sp)
        sw $s2, 0($sp)

        and $s0, $a0, $a1
        srl $s1, $a2, 3
        slt $s2, $s1, $a2

        add  $v0, $s0, $0
        add  $v1, $s1, $0

        lw   $s2, 0($sp)
        lw   $s1, 4($sp)
        lw   $s0, 8($sp)
        addi $sp, $sp, 12
        jr   $ra
```

What is the final value of $v1?
What is the final value of $s0?

Registers (Initial)

Name	Number	Value
$v0	2	1
$v1	3	15
$a0	4	5
$a1	5	9
$a2	6	15
$a3	7	7
$t0	8	63
$t1	9	8
$t2	10	10
$t3	11	5
$t4	12	15
$t5	13	15
$t6	14	0
$t7	15	0
$s0	16	8
$s1	17	60
$s2	18	0

Leaf Procedure example – My turn

Execute the following procedure:

```
MyProc:
        addi $sp, $sp, -12
        sw $s0, 8($sp)
        sw $s1, 4($sp)
        sw $s2, 0($sp)

        and $s0, $a0, $a1
        srl $s1, $a2, 3
        slt $s2, $s1, $a2

        add  $v0, $s0, $0
        add  $v1, $s1, $0

        lw   $s2, 0($sp)
        lw   $s1, 4($sp)
        lw   $s0, 8($sp)
        addi $sp, $sp, 12
```

Allocate storage on stack
Store variables on stack

Stored onto stack (stack not shown)

Registers

Name	Number	Value
$v0	2	1
$v1	3	15
$a0	4	5
$a1	5	9
$a2	6	15
$a3	7	7
$t0	8	63
$t1	9	8
$t2	10	10
$t3	11	5
$t4	12	15
$t5	13	15
$t6	14	0
$t7	15	0
$s0	16	8
$s1	17	60
$s2	18	0

Leaf Procedure example – My turn

Execute the following procedure:

```
MyProc:
        addi $sp, $sp, -12
        sw $s0, 8($sp)
        sw $s1, 4($sp)
        sw $s2, 0($sp)

        and $s0, $a0, $a1
        srl $s1, $a2, 3
        slt $s2, $s1, $a2

        add  $v0, $s0, $0
        add  $v1, $s1, $0

        lw   $s2, 0($sp)
        lw   $s1, 4($sp)
        lw   $s0, 8($sp)
        addi $sp, $sp, 12
        jr   $ra
```

Allocate storage on stack
Store variables on stack

Execute Program:

$s0 = $a0 & $a1 = 1
$s1 = $a2 >> 3 = 1
$s2 = 1 if $s1 < $a2 = 0

Registers

Name	Number	Value
$v0	2	1
$v1	3	15
$a0	4	5
$a1	5	9
$a2	6	15
$a3	7	7
$t0	8	63
$t1	9	8
$t2	10	10
$t3	11	5
$t4	12	15
$t5	13	15
$t6	14	0
$t7	15	0
$s0	16	1
$s1	17	1
$s2	18	1

Leaf Procedure example – My turn

Execute the following procedure:

```
MyProc:
        addi $sp, $sp, -12
        sw $s0, 8($sp)
        sw $s1, 4($sp)
        sw $s2, 0($sp)

        and $s0, $a0, $a1
        srl $s1, $a2, 3
        slt $s2, $s1, $a2

        add  $v0, $s0, $0
        add  $v1, $s1, $0

        lw   $s2, 0($sp)
        lw   $s1, 4($sp)
        lw   $s0, 8($sp)
        addi $sp, $sp, 12
        jr   $ra
```

Allocate storage on stack
Store variables on stack

Execute Program:
$s0 = $a0 & $a1 = 1
$s1 = $a2 >> 3 = 1
$s2 = 1 if $s1 < $a2 = 0

Return results in $v0 and $v1

Registers

Name	Number	Value
$v0	2	1
$v1	3	1
$a0	4	5
$a1	5	9
$a2	6	15
$a3	7	7
$t0	8	63
$t1	9	8
$t2	10	10
$t3	11	5
$t4	12	15
$t5	13	15
$t6	14	0
$t7	15	0
$s0	16	1
$s1	17	1
$s2	18	1

Leaf Procedure example – My turn

Execute the following procedure:

```
MyProc:
        addi $sp, $sp, -12
        sw $s0, 8($sp)
        sw $s1, 4($sp)
        sw $s2, 0($sp)

        and $s0, $a0, $a1
        srl $s1, $a2, 3
        slt $s2, $s1, $a2

        add  $v0, $s0, $0
        add  $v1, $s1, $0

        lw   $s2, 0($sp)
        lw   $s1, 4($sp)
        lw   $s0, 8($sp)
        addi $sp, $sp, 12
        jr   $ra
```

Allocate storage on stack
Store variables on stack

Execute Program:

\$s0 = \$a0 & \$a1 = 1
\$s1 = \$a2 >> 3 = 1
\$s2 = 1 if \$s1 < \$a2 = 0

Return results in \$v0 and \$v1

Restore variables
Free Storage

Return to Caller

Registers

Name	Number	Value
\$v0	2	1
\$v1	3	1
\$a0	4	5
\$a1	5	9
\$a2	6	15
\$a3	7	7
\$t0	8	63
\$t1	9	8
\$t2	10	10
\$t3	11	5
\$t4	12	15
\$t5	13	15
\$t6	14	0
\$t7	15	0
[illegible]	[illegible]	8
[illegible]	[illegible]	60
[illegible]	[illegible]	0

Restored from stack (stack not shown)

Leaf Procedure example – My turn

Execute the following procedure:

```
MyProc:
        addi $sp, $sp, -12
        sw $s0, 8($sp)
        sw $s1, 4($sp)
        sw $s2, 0($sp)

        and $s0, $a0, $a1
        srl $s1, $a2, 3
        slt $s2, $s1, $a2

        add  $v0, $s0, $0
        add  $v1, $s1, $0

        lw   $s2, 0($sp)
        lw   $s1, 4($sp)
        lw   $s0, 8($sp)
        addi $sp, $sp, 12
        jr   $ra
```

What is the final value of \$v1? 1

What is the final value of \$s0? 8

(all saved registers must be restored to original value before returning MyProc)

Registers (final)

Name	Number	Value
\$v0	2	1
\$v1	3	1
\$a0	4	5
\$a1	5	9
\$a2	6	15
\$a3	7	7
\$t0	8	63
\$t1	9	8
\$t2	10	10
\$t3	11	5
\$t4	12	15
\$t5	13	15
\$t6	14	0
\$t7	15	0
\$s0	16	8
\$s1	17	60
\$s2	18	0

Procedure-Your Turn

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	5	10	8	2	3	10	5	8
0x10010020	2	15	34	56	23	68	75	0
0x10010040	0	0	0	0	0	0	0	0

0x10010000 (.data) ✓ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII

Execute the following procedure called SProc:

```
SProc:
        addi $sp, $sp, -8
        sw $s0, 4($sp)
        sw $s1, 0($sp)

        la $t0, NumArray
        sll $t1, $a0, 2
        sll $t2, $a1, 2
        add $t1, $t1, $t0
        add $t2, $t2, $t0

        lw $s0, 0($t1)
        lw $s1, 0($t2)
        sw $s1, 0($t1)
        sw $s0, 0($t2)

        lw $s1, 0($sp)
        lw $s0, 4($sp)
        addi $sp, $sp, 8
        jr $ra
```

NumArray base address is 0x10010008
argument i and j are stored in $a0, $a1

What is the final value of NumArray[9]?
What is the final value of $s1?
What is the final value of $t1?

Registers

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	11
$v1	3	15
$a0	4	1
$a1	5	9
$a2	6	15
$a3	7	7
$t0	8	-7
$t1	9	0
$t2	10	12
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	5
$s1	17	75
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	0x7fffeffc
$fp	30	0
$ra	31	0

Summary

- Procedures:
  - Caller, Callee, Program Counter
  - Stack, Stack pointer, Push/Pop
  - Leaf Procedure Example
  - Non-leaf Procedure, Recursive Procedure Example

Now you have all of the skills to do the Term Project!

Next

CPU Performance