Recursion

- Demonstrate how to write a recursive function in mips

Function Review

Every time a function/subroutine is called, a stack frame is created for the function The stack frame contains areas for - Arguments - Saved registers - Return address - Local data - Some padding -

Function Call Conventions

1. Caller right before call
2. Callee upon entry
3. Calle right before exit
4. Caller upon return

```
// MIPS vs C++ (Mips conventions shown)
main(){

```

1

- Save a, t, & ra registers and the Frame Pointer
- Copy val of stack pointer into Frame Pointer
- Place arguments in a registers
- Call the function

```
vat = calculate(a, b, c); }
```

4

- Save v registers
- Restore the values saved to the stack
- Restore the stack pointer

```
int calculate(int a, int b, int c){
        // 2
        ...
        ....
        .....
        // 3
        // Store s registers
        // Restore s registers
        // Place val in v register
        return ans;
}
```

Recursive Functions

- A function that can call itself
- Base condition so its not infinite loop
- Change of state is needed

Will keep calling itself with different parameters, until a terminating condition is met

```
n = 3 (a0)                           // Stack Growth
res = 6 (v0)                         //    |  |
ra to main                           //    |  |
(call)                               //    |  |
n = 2 (a0)                           //    |  |
res = 2(v0)                          //  \‾‾‾‾‾‾/
ra to fact(3)                        //   \    /
(call)                               //    \  /
n = 1 (a0)                           //     \/
res = 1 (v0)
ra to fact(2)
```

Demonstrates why we need to save the a registers. They, as well as ra, are very important for recursive function

The code below demonstrates how the function call should be preformed for a recursive function. The full code can be viewed sqrt.asm

```
        .text
        .glovl main

main:
        addi $sp, $sp, $sp, -12                  # allocate 3 words
        sw $a0, 0($sp)                           # store values in stack
        sw $fp, 4($sp)                          # frame pointer - 4
        sw $ra, 8($sp)                          # reg address - 8
        or $fp, $sp, $0                         # fp = stack pointer

        li $a0, 10                              # param to function = 10
        li $s2, 1                               # exit condition number
        jal fact                                # function call

        or $s1, $v0, $0                         # save return value

        lw $ra, 8($sp)                          # restore stack
        lw $fp, 4($sp)
        lw $a0, 0($sp)
```