Nested subroutine call

\[ f(x, y) = \sqrt{x \cdot y} \]
Handling recursive procedure calls

Example. Compute factorial (n)

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

(Plan) Put n in $a0. Result should be available in $v0.

{Structure of the fact procedure}

```assembly
fact:    subi $sp, $sp, 8
sw $ra, 4($sp) {why?}
sw $a0, 0($sp)
```
The growth of the stack as the recursion unfolds

Calling program

- \( a_0 = n \) (3)
- \( \text{jal fact (4000)} \)
- read fact(n) from v0

Procedure fact

- push ra
- push a0
- if \( n < 1 \) then \( \{v0 = 1\}\)
  - Return to ra
- \( a_0 = n - 1 \)
- \( \text{jal fact (4000)} \)
- \( v0 = \text{old a0} \times \text{fact(n-1)} \)
- return

Stack:

- \( a0 = 1 \)
- ra = 4024
- \( a0 = 2 \)
- ra = 4024
- \( a0 = 3 \)
- ra = 1004

Result:

- \( n = 3 \)
- a0
- Result
- v0
Now test if \( n < 1 \) (i.e. \( n = 0 \)). In that case return 0 to \( v0 \).

```
slti $t0, $a0, 1       # if \( n \geq 1 \) then goto L1
beq $t0, $zero, L1
addi $v0, $zero, 1    # return 1 to \( v0 \)
addi $sp, $sp, 8      # pop 2 items from stack
jr $ra                 # return
L1: addi $a0, $a0, -1  # decrement \( n \)
    jal fact          # call fact with \( (n-1) \)
```

Now, we need to compute \( n \times \text{fact}(n-1) \)

```
lw $a0, 0($sp)          # restore argument \( n \)
lw $ra, 4($sp)          # restore return address
addi $sp, $sp, 8        # pop 2 items
mult $v0, $a0, $v0      # return \( n \times \text{fact}(n-1) \)
jr $ra                  # return to caller
```
Run time environment of a MIPS program

Stack pointer

Temporary local variables

Return address

Saved argument registers beyond a0-a3

Frame

Low address

Growth of stack

High address
A translation hierarchy

HLL program
\downarrow
COMPILER
\downarrow
Assembly language program
\downarrow
ASSEMBLER
\downarrow
Machine language module
\downarrow
LINKER ← Library routine
\downarrow
Executable machine language program
\downarrow
LOADER
\downarrow
Memory
What are Assembler directives?

Instructions that are not executed, but they tell the assembler about how to interpret something. Here are some examples:

. text

{Program instructions here}

. data

{Data begins here}

. byte 84, 104, 101

. asciiz “The quick brown fox”

. float f1, . . . , fn

. word w1, . . . , wn

. space n {reserve n bytes of space}
How does an assembler work?

In a two-pass assembler

PASS 1: Symbol table generation
PASS 2: Code generation

Follow the example in the class.