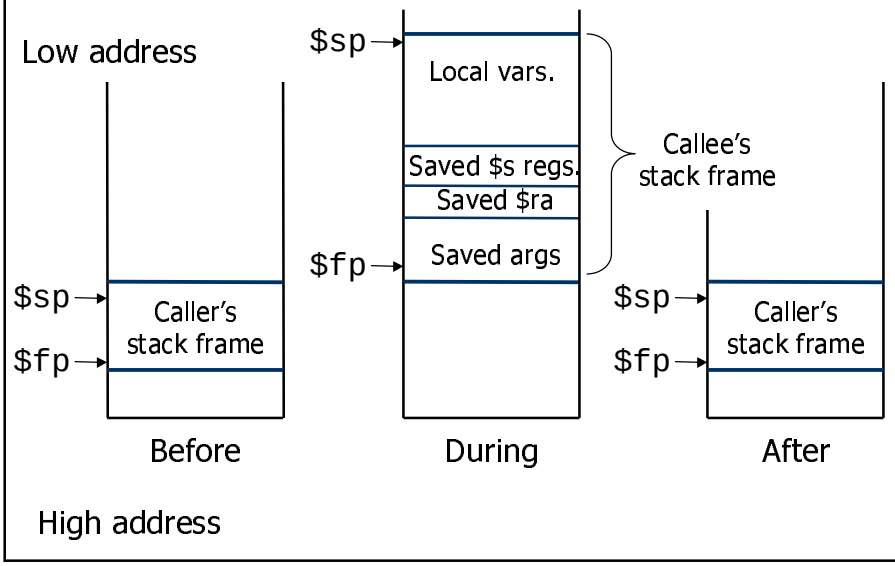
10/8: Lecture Topics

- C Brainteaser from last time
- More on Procedure Call
  - More than four arguments
  - A recursive example
- Starting a Program
- Exercise 3.2 from H+P

5+ Arguments

- Up to four arguments can be passed in registers $a0-$a3
- To pass more than four,
  - Push them on the stack
  - Set the frame pointer ($fp) to point to them
- The frame pointer provides a stable base register for the duration of the procedure

Stack Allocation



Recursive Procedure Call

- A recursive procedure calls itself
- With the stack, no more difficult than nested procedure call

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

Factorial Implementation

```
fact:  subi $sp, $sp, 8
       sw   $ra, 4($sp)
       sw   $a0, 0($sp)
       slt  $t0, $a0, 1
       beq  $t0, $zero, L1
       add  $v0, $zero, 1
       add  $sp, $sp, 8
       jr   $ra
L1:    sub  $a0, $a0, 1
       jal  fact
       lw   $a0, 0($sp)
       lw   $ra, 4($sp)
       add  $sp, $sp, 8
       mult $v0, $a0, $v0
       jr   $ra
```

Starting a Program

- Two phases from source code to execution
- Compile time
  - compiler
  - assembler
  - linker
- Run time
  - loader

Compile Time

- You're experts on compiling from source to assembly
- Two parts to translating from assembly to machine language:
  - Instruction encoding
  - Translating labels to addresses
- Label translations go in the *symbol table*

Modular Program Design

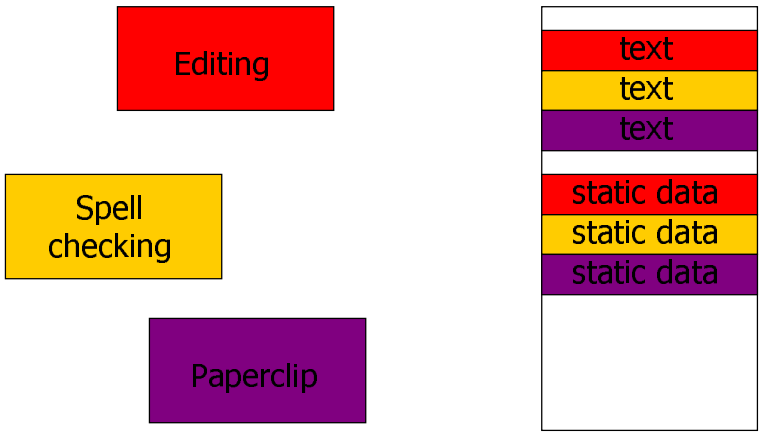
- Small projects may use only one file
  - Any time any one line changes, recompile and reassemble the whole thing
- For larger projects, recompilation time is significant
- Solution: split project into modules
  - compile and assemble modules separately
  - link the object files

The Linker

- The linker's job is to "stitch together" the object files:
  1. Place the data modules in memory
  2. Determine the addresses of data and labels
  3. Match up references between modules

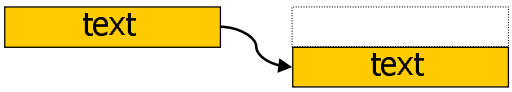
Placing Modules in Memory

- Link a word processor application:



Determining Addresses

- Some addresses change during memory layout
- Modules were compiled in isolation
- *Absolute* addresses must be *relocated*



Resolving References

- The editing module calls a routine from the spell checker
- That symbol is *unresolved* at compile time
- The linker matches unresolved symbols to locations in other modules

Libraries

- Some code is used so often, it is bundled into *libraries* for common access
- Libraries contain most of the code you use but didn't write: e.g., printf()
- Library code is (often) merged with yours at link time

The Executable

- End result of compiling, assembling, and linking: the *executable*
- Contains:
  - Header, listing the lengths of the other segments
  - Text segment
  - Static data segment
  - Potentially other segments, depending on architecture & OS conventions

Run Time

- We'll learn a lot more about this during the OS part of the course
- In a nutshell:
  - Some *dynamic linking* may occur
  - The segments are loaded into memory
  - The OS transfers control to the program and it runs

Exercise 3.2

- 25% Correct: computes the mode and the number of times the mode occurs
- 50% Part way there
  - Computes the number of times the A[4999] occurs
  - Some other computation involving looping over the array
- 25% Incorrect or no answer

Key Observations

```
       add  $a1, $a1, $a1
       add  $a1, $a1, $a1
       add  $v0, $zero, $zero
       add  $t0, $zero, $zero
outer: add  $t4, $a0, $t0
       lw   $t4, 0($t4)
       add  $t5, $zero, $zero
       add  $t1, $zero, $zero
inner: add  $t3, $a0, $t1
       lw   $t3, 0($t3)
       bne  $t3, $t4, skip
       addi $t5, $t5, 1
skip:  addi $t1, $t1, 4
       bne  $t1, $a1, inner
       slt  $t2, $t5, $v0
       bne  $t2, $zero, next
       add  $v0, $t5, $zero
       add  $v1, $t4, $zero
next:  addi $t0, $t0, 4
       bne  $t0, $a1, outer
```