**Plan for Learning about Stack Frames**

**Typical C stack frame**
- gcc calling convention described in A.5 and A.6 in appendix on assembly code and MIPS

**Stack frame the MiniJava compiler will generate**
- Need to match the Wisconsin C-- compiler to implement garbage collection

**General stack frame concept**
- Agreement amongst programmers, procedure call convention
- What about gcc for x86?
- Handling nested procedures
- Handling first class functions

**Structure of the MiniJava Compiler**

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Synthesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR code generation</td>
<td>PA7</td>
</tr>
<tr>
<td>tokens</td>
<td>Tree</td>
</tr>
<tr>
<td>“words”</td>
<td></td>
</tr>
<tr>
<td>syntactic analysis</td>
<td>PA8</td>
</tr>
<tr>
<td>“sentences”</td>
<td></td>
</tr>
<tr>
<td>semantic analysis</td>
<td></td>
</tr>
<tr>
<td>code generation</td>
<td>PA8</td>
</tr>
</tbody>
</table>

**Wisconsin C-- calling convention**

**Calling convention (contract between caller and callee)**
- $sp$ must be divisible by 4
- Caller should pass parameters in order on the stack
- Upon callee entry, the stack pointer $sp$ should be pointing at the first empty slot past the last parameter
- Upon callee exit, the stack pointer $sp$ should be pointing at the first parameter
- Upon callee exit, return value should be in $v0$

**Rules to follow for PA6 (to standardize frame usage)**
- $sp$ should always be pointing at next empty slot on the stack
- $sra$ and $sfp$ should be stored right after the parameters on stack, you can’t use any other callee-saved registers
- $sfp$ should be made to point at the first parameter, so that the address for the first parameter is $sfp-0$, the address for the second parameter is $sfp-4$, ...
- Locals should be stored in order, right after $sra$ and $sfp$

---

**Mapping out the stack frame for the funcCall1 example**

```c
int foo(int x, int y, int *z) {
    int a;
    a = x * y - *z;
    return a;
}
void main() {
    int x;
    x = 2;
    cout << foo(4, 5, &x);
    cout << "\n";
}
```

```assembly
.text
.globl main
main:
    sw $ra, 0($sp)       #PUSH
    subu $sp, $sp, 4
    sw $fp, 0($sp)       #PUSH
    subu $fp, $sp, 4
    addu $fp, $sp, 20
    li $t0, 2
    sw $t0, 0($sp)       #PUSH
    subu $sp, $sp, 4
    li $t0, 5
    sw $t0, $fp
    addu $fp, $sp, 4
    li $t0, 4
    sw $t0, 4($fp)
    addu $fp, $sp, 4
    li $t0, 0
    sw $t0, 8($fp)
    addu $fp, $sp, 4
    li $t0, 0
    sw $t0, ($fp)
    jal _foo
    move $sra, $ro
    move $sra, $fp
    move $sfp, -4($fp)
    move $sfp, $sp
jr $ra
```

---

**Wisconsin C-- Stack Frame Layout**

```
+---------------------+
| Static             |
+---------------------+
| int    | int    | int    | int    |
| __ra__ |
| __fp__ |
| __sra__ |
| __sfp__ |
+---------------------+
```

```
+-------------------+
| Calling convention|
+-------------------+
| sp must be divisible by 4 |
| caller should pass parameters in order on the stack |
| upon callee entry, the stack pointer $sp$ should be pointing at the first empty slot past the last parameter |
| upon callee exit, the stack pointer $sp$ should be pointing at the first parameter |
| upon callee exit, return value should be in $v0 |
| sp should always be pointing at next empty slot on the stack |
| sra and sfp should be stored right after the parameters on stack, you can’t use any other callee-saved registers |
| sfp should be made to point at the first parameter, so that the address for the first parameter is sfp-0, the address for the second parameter is sfp-4, |
| locals should be stored in order, right after sra and sfp |
```
Another example: where does each variable go?

```java
class A {
    public static void main(String[] a) {
        System.out.println(42);
    }
}
class B {
    int[] x;
    boolean mBool;
    public int foo(boolean p1, int p2, B b, int[] y) {
        boolean v1; int i; int j; return 0;
    }
    public B bar() {
        B b;
        b = new B;
        return b;
    }
    public boolean baz() {
        return mBool;
    }
}
```

Determining locations for vars

Local vars
- maintain counter for method that is initialized to 0
- store counter in a temporary variable
- **decrement** current counter by size of the local variable
- return the value in the temporary variable

Class members
- maintain counter for method that is initialized to 0
- store counter in a temporary variable
- **increment** current counter by size of the local variable
- return the value in the temporary variable