

Plan for Today

General stack frame concept

- agreement amongst programmers, procedure call convention

Stack frame the MiniJava compiler will generate

- Need to match the Wisconsin C-- compiler to implement garbage collection

Building the symbol table

- determining the memory locations for each local and class member variable
- the Frame class
- specifics for inMethodDecl

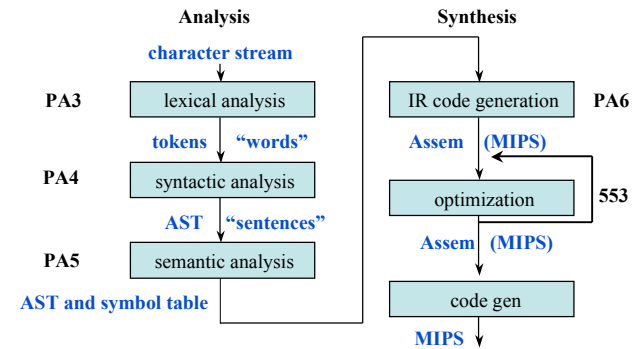
Suggested Exercise for figuring out stack frame information in x86 code

CS453 Lecture

Offsets and Frame Class

1

Structure of the MiniJava Compiler



CS453 Lecture

Offsets and Frame Class

2

Questions a calling convention must answer

Contract between caller and callee

- Where is the return value?
- Where is the stack pointer pointing upon entry to a function?
- Where are the parameters?
- Is the caller or callee responsible for popping the parameters?
- Does the stack pointer point at the top of the stack or the next empty slot?

Decisions needed for manipulating a frame/activation record

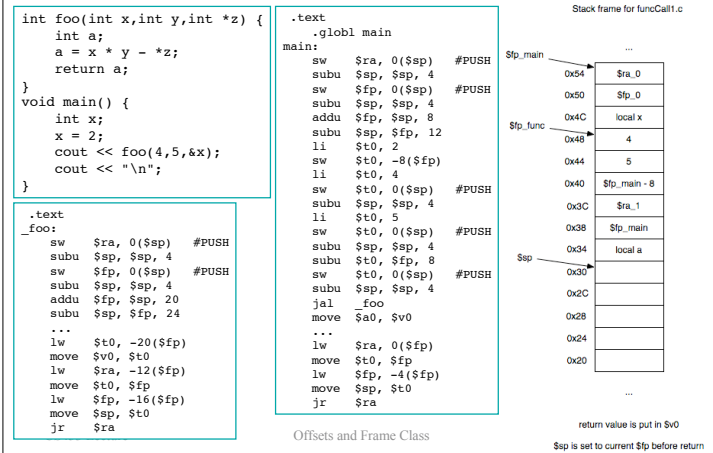
- Layout of callee-saved registers, caller-saved registers, locals, and temps
- Are parameters pushed by moving the stack pointer or is enough space set aside initially?

CS453 Lecture

Offsets and Frame Class

3

Mapping out the stack frame for the funcCall1 example



Offsets and Frame Class

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 PA2 (to standardize frame usage)

- \$sp should always be pointing at next empty slot on the stack
- \$ra and \$fp should be stored right after the parameters on stack, you can't use any other callee-saved registers
- \$fp should be made to point at the first parameter, so that the address for the first parameter is \$fp-0, the address for the second parameter is \$fp-4, ...
- locals should be stored in order, right after \$ra and \$fp

CS453 Lecture

Offsets and Frame Class

5

Another example: where does each variable go?

```
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;
    }
}
```

CS453 Lecture

Offsets and Frame Class

6

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

CS453 Lecture

Offsets and Frame Class

7

Interface to Frame

Three main responsibilities

- provide a factory interface for generating machine-specific frames
 - Frame newFrame(Label name, List<Boolean> formals)
- answer queries that are machine-specific, but not method specific
 - int wordSize()
 - Temp FP(), coming to an interface near you in PA7
- store method-specific information about frame layout
 - Label name
 - List<Access> formals
 - Access allocLocal(boolean escape)

CS453 Lecture

Offsets and Frame Class

8

When do parameters and/or locals escape?

Nesting of classes and methods

When the variable may have its address taken

```
int addressFunc(int x)
{
    int z;
    return blah(6x, &z);
}
```

When the language uses pass-by-reference

```
FORTRAN
subroutine
foo(x,y)
double precision x
double precision y
y=x*2
end subroutine
```

```
subroutine head()
double precision a, b
call foo(a,b)
end subroutine
```

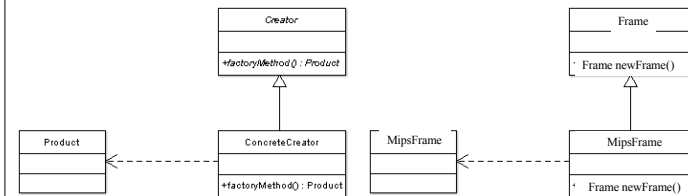
```
int foo(int x)
{
    int baz(int y)
    {
        return x+y;
    }
    int bar() {
        return baz(2);
    }
    return bar();
}
int main() {
    printf("%f\n", foo(3));
}
```

CS453 Lecture

Offsets and Frame Class

9

Frame class implements Factory design pattern



CS453 Lecture

Offsets and Frame Class

10

What, why, and where?

What?

- one instance of the MipsFrame will generate other instances

Why?

- need a Frame instance for each function and want to avoid calling the MipsFrame constructor everywhere

Where?

```
// FrameLayout.java
Frame.Frame frame = new Mips.MipsFrame();
BuildSymTable buildSTvisitor = new BuildSymTable(linesToNodes, frame);

// BuildSymTable::inMethodDecl
...
Frame.Frame methodFrame = mFrame.newFrame(
    new Temp.Label(mCurrentClass.getName()+"_"+node.getName().getText()),
    formalEscapeList);
```

CS453 Lecture

Offsets and Frame Class

11

inMethodDecl for BuildSymTable visitor

Steps needed in the inMethodDecl

- does the method name conflict
- create a formal escape list
- add an entry into the formal escape list for the implicit “this” parameter
- create a list of types for explicit formals
- for each explicit parameter add entry to formal escape list
- create method Signature(return type, formal types list)
- create Frame with newFrame
- create MethodSTE and insert it into current ST
- push method scope
- create LocalSTE for implicit “this” and insert it into current ST
- create LocalSTEs for each explicit formal and insert it into current ST

CS453 Lecture

Offsets and Frame Class

12

funcCall1.c

```
int foo(int x,int y,int *z) {
    int a;
    a = x * y - *z;
    return a;
}
int main() {
    int x;
    x = 2;
    x = foo(4,5,&x);
    return x;
}
```

Suggested Exercise: funcCall1.c for x86

```
foo:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   8(%ebp), %eax
    movl   %eax, %edx
    imull  12(%ebp), %edx
    movl   16(%ebp), %eax
    movl   (%eax), %eax
    movl   %edx, %ecx
    subl   %eax, %ecx
    movl   %ecx, %eax
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
    ret

main:
    leal   4(%esp), %ecx
    andl   $-16, %esp
    pushl  -4(%ecx)
    pushl  %ebp
    movl   %esp, %ebp
    pushl  %ecx
    subl   $28, %esp
    movl   $2, -8(%ebp)
    leal   -8(%ebp), %eax
    movl   %eax, 8(%esp)
    movl   $5, 4(%esp)
    movl   $4, (%esp)
    call   foo
    movl   %eax, -8(%ebp)
    movl   -8(%ebp), %eax

    addl   $28, %esp
    popl   %ecx
    popl   %ebp
    leal   -4(%ecx), %esp
    ret
```

What register holds the stack pointer?

frame pointer? the return value?

In instructions, where are source and dest?

How is the local variable "a" accessed?