CS 5470

Compiler Techniques and Principles

February 22, 2010 — LECTURE 16

More on translation into IR trees

(handling arrays and calls)

Array Variables

Array-valued variables are treated differently in different languages.

- *Pascal*: variable is actual contents of array

```
var a, b : array[1..12] of integer
begin
  a := b
end;
```

- *C*: variable is a pointer

```
int a[12], *b;
b = a;
```

- *MiniJava*?

Array Variables

- In MiniJava, array variables behave as pointers.
- New array values are created and initialized using `new int[n]`.
- There are *n* elements, each with initial value 0.

```
int[] a;
int[] b;
a = new int[12];
b = new int[12];
a = b;
```

- Variable *a* points to same 12 zeros as *b*, original 12 zeros allocated for *a* are discarded.

Array Indexing

- To index an array `a[i]`, compute the address of the i^{th} element of `a`: $(i - l) * s + a$.
  - l: lower bound of index range
  - s: size of each array element
  - a: base address of the array
- In Pascal, base address is the array variable.

  BINOP(PLUS, TEMP(*fp*), CONST(*k*))
- In MiniJava, base address is contents of a pointer variable. MEM(BINOP(PLUS, TEMP(*fp*), CONST(*k*)))

Array Indexing

- A memory-resident MiniJava array variable is represented as MEM(e) where the contents at address e is a pointer value p.
- The contents of addresses p, $p+W$, $p+2W$, ... (where W is word size, and all elements are one word long), are the first element, second element, third element, ...
- `a[i]`:

```
MEM(BINOP(PLUS, MEM(e),
                           BINOP(MUL, i, CONST(W))))
```

Array Creation

1. Determine how much space is needed.

   $(array\_length + 1)$ `*` $word\_size$

2. Call external function (`alloc`) to get space on heap, returns pointer to beginning of memory block.

3. Generate code for saving array length at offset 0.

   MOVE(p, CONST($array\_length$))

4. Generate code for initializing each array element to 0, starting at offset 4.

L-Values

- An *l-value* is the result of an expression that can occur on the left of an assignment (`x`, `p.y`, `a[i+2]`).

- A *r-value* is the result of an expression that can only occur on the right of an assignment (`a+3`, `f(x)`).

- An l-value occurring on left denotes a location that can be assigned to.

- An integer or pointer value is “scalar”, has only one component and occupies one word of memory.

Structured L-Values

- Handling *structured l-values* (C structs, Pascal arrays and records) requires some extra work.
- We must know the size of such “large” variables.
- Then the MEM class of the `Tree` language would need to be extended with a notion of size.

  MEM(BINOP(PLUS, TEMP(*fp*), CONST(*k*)), *S*)

- *S* indicates size of the object to be fetched or stored, depending on where MEM appears in MOVE.

Function Calls

- To translate a function call $f(a_1,\ldots,a_n)$,

$$\mathrm{CALL}(\mathrm{NAME}(l_f), [e_1,\ldots,e_n])$$

- For an O-O language, the implicit variable `this` must be made an explicit argument of the call.

- For $p.m(a_1,\ldots,a_n)$,

$$\mathrm{CALL}(\mathrm{NAME}(l_{c\$m}), [p, e_1,\ldots,e_n])$$

Calling External Functions

- To call an external function (such as `alloc`, written in C or assembly language) with *args*,

  Label `alloc` = new Label("alloc")

  CALL(NAME(`alloc`), *args*)

- The calling conventions for C (or other languages) functions may be different from those of MiniJava.

- Such target-machine details are encapsulated into an `externalCall` function in `Frame`.