Phases of a compiler

Intermediate Representation (IR): specification and generation

Figure 1.6, page 5 of text
Reminder:
Friday is a workshop day

- Come ready to work
- Ensure all team members are present
- Bring along a list of questions
function calls
Function calls

```c
int foo(int x, int y) {
    int temp;
    temp = 2 * x + 3 * y;
    return temp;
}

int main() {
    ... 
    int a = ...  
    int b = ...
    int c = foo(a * b, a + b);
    ... 
}
```

What happens during function call?
Function calls

- Basic form: id(e1,e2,...,ek)
Function calls

- Basic form: \( \text{id}(e_1, e_2, \ldots, e_k) \)
- General form: \( \text{assignable}(e_1, e_2, \ldots, e_k) \)

- If \( f \) is a function, \( g(4,5) \) yields a function, and \( r.h \) yields a function, then the following are legal:

\[
\begin{align*}
    f(3) & \quad g(4,5)(3) & \quad r.h(3)
\end{align*}
\]
How is function call carried out?

1. evaluate each of the argument expressions
2. mark the resulting values as parameters
3. invoke the function
How is function call carried out?

1. evaluate each of the argument expressions
   - use compiler-generated temporaries

2. mark the resulting values as parameters
   - use 'param' IR instruction

3. invoke the function
   - use 'call(f,n)' IR instruction:
     - f is a function
     - n is arity of function
     - # of parameters
examples

f(x+1)
examples

\[ f(x+1) \]

Remember that the function call has structure.
examples

$f(x+1)$

t1 = x + 1

Generate code for the argument expression
examples

\( f(x+1) \)

\( t1 = x + 1 \)

param \( t1 \)

Mark the result as a parameter of the function call
examples

\[ f(x+1) \]

\[ t1 = x + 1 \]

param t1

\[ t2 = \text{call}(f,1) \]

Call the function. The second argument of the call indicates the arity of the function (i.e. how many parameters it has)
examples

\( f(x+1) \)

\( t_1 = x + 1 \)

\text{param} t_1

\( t_2 = \text{call}(f,1) \)

\( f(x+1,2*y) \)
Examples

\[ f(x+1) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = \text{call}(f,1) \]

\[ f(x+1,2\times y) \]

Remember that the function call has structure.
examples

\( f(x+1) \)

\( t_1 = x + 1 \)

\textit{param} t_1

\( t_2 = \text{call}(f,1) \)

\( f(x+1, 2*y) \)

\( t_1 = x + 1 \)

\textit{Evaluate the first argument expression.}
examples

\[ f(x+1) \]
\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ t2 = \text{call}(f,1) \]

\[ f(x+1,2*y) \]
\[ t1 = x + 1 \]
\[ \text{param } t1 \]

Mark the result as a parameter.
Evaluate the second argument expression.

\[ f(x+1) \]
\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ t2 = \text{call}(f,1) \]

\[ f(x+1, 2*y) \]
\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ t2 = 2 \times y \]
examples

\[ f(x+1) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = \text{call}(f,1) \]

\[ f(x+1,2*y) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = 2 * y \]
\[ \text{param } t_2 \]

Mark the result as a parameter.
examples

\[ f(x+1) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = \text{call}(f,1) \]

\[ f(x+1,2y) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = 2 \times y \]
\[ \text{param } t_2 \]
\[ t_3 = \text{call}(f,2) \]

Call the function.
examples

\[ f(x+1) \]
\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ t2 = \text{call}(f,1) \]

\[ f(x+1,2*y) \]
\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ t2 = 2 * y \]
\[ \text{param } t2 \]
\[ t3 = \text{call}(f,2) \]

\[ f(g(3*z),h(a+b,a*b)) \]

A slightly more involved example.
exercise

What intermediate code do you come up with for this example?

\[ f(g(3z), h(a+b, a*b)) \]
As before, remember the structure...

Examples:

\[ f(x+1) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = \text{call}(f,1) \]

\[ f(x+1,2*y) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = 2 \times y \]
\[ \text{param } t_2 \]
\[ t_3 = \text{call}(f,2) \]

\[ f(g(3*z),h(a+b,a*b)) \]
examples

\[ f(x+1) \]
\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ t2 = \text{call}(f,1) \]

\[ f(x+1,2*y) \]
\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ t2 = 2 * y \]
\[ \text{param } t2 \]
\[ t3 = \text{call}(f,2) \]

\[ g(3*z) \]

...view this as a function call in isolation.
examples

\[ f(x+1) \]

\[ t_1 = x + 1 \]

param \( t_1 \)

\[ t_2 = \text{call}(f,1) \]

\[ f(x+1,2\cdot y) \]

\[ t_1 = x + 1 \]

param \( t_1 \)

\[ t_2 = 2 \cdot y \]

param \( t_2 \)

\[ t_3 = \text{call}(f,2) \]

\[ g(3\cdot z) \]

\[ t_1 = 3 \cdot z \]

...first compute the argument value...
examples

\[ f(x+1) \]
\[ t1 = x + 1 \]
\[ param \ t1 \]
\[ t2 = call(f,1) \]

\[ f(x+1,2*y) \]
\[ t1 = x + 1 \]
\[ param \ t1 \]
\[ t2 = 2 * y \]
\[ param \ t2 \]
\[ t3 = call(f,2) \]

\[ g(3*z) \]
\[ t1 = 3 * z \]
\[ param \ t1 \]
\[ t2 = call(g,1) \]

...then mark \( t1 \) as a parameter and call the function \( g \).
Examples

\[ f(x+1) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = \text{call}(f,1) \]

\[ f(x+1,2*y) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = 2 * y \]
\[ \text{param } t_2 \]
\[ t_3 = \text{call}(f,2) \]

\[ f(g(3*z),h(a+b,a*b)) \]
\[ t_1 = 3 * z \]
\[ \text{param } t_1 \]
\[ t_2 = \text{call}(g,1) \]

This translation will happen automatically due to the recursive structure of the function call for \( f \)...
Mark the result as a parameter.
examples

\[ f(x+1) \]
\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ t2 = \text{call}(f,1) \]

\[ f(x+1,2*y) \]
\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ t2 = 2 \times y \]
\[ \text{param } t2 \]
\[ t3 = \text{call}(f,2) \]

\[ f(g(3*z),h(a+b,a\times b)) \]
\[ t1 = 3 \times z \]
\[ \text{param } t1 \]
\[ t2 = \text{call}(g,1) \]
\[ \text{param } t2 \]
examples

\[ f(x+1) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = \text{call}(f,1) \]

\[ f(x+1,2*y) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = 2 \times y \]
\[ \text{param } t_2 \]
\[ t_3 = \text{call}(f,2) \]

\[ f(g(3*z),h(a+b,a*b)) \]
\[ t_1 = 3 \times z \]
\[ \text{param } t_1 \]
\[ t_2 = \text{call}(g,1) \]
\[ \text{param } t_2 \]
\[ t_3 = a + b \]

expression
examples

\[
\begin{align*}
\text{f}(x+1) & \quad \text{f}(x+1,2\times y) & \quad \text{f}(g(3z),h(a+b,a\times b)) \\
\text{t1} = x + 1 & \quad \text{t1} = x + 1 & \quad \text{t1} = 3 \times z \\
\text{param t1} & \quad \text{param t1} & \quad \text{param t1} \\
\text{t2} = \text{call(f,1)} & \quad \text{t2} = 2 \times y & \quad \text{t2} = \text{call(g,1)} \\
\text{param t2} & \quad \text{param t2} & \quad \text{param t2} \\
\text{t3} = \text{call(f,2)} & \quad \text{t3} = a + b & \quad \text{param t3}
\end{align*}
\]
examples

\[ f(x+1) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = \text{call}(f,1) \]

\[ f(x+1,2*y) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = 2 \times y \]
\[ \text{param } t_2 \]
\[ t_3 = \text{call}(f,2) \]

\[ f(g(3*z),h(a+b,a*b)) \]
\[ t_1 = 3 \times z \]
\[ \text{param } t_1 \]
\[ t_2 = \text{call}(g,1) \]
\[ \text{param } t_2 \]
\[ t_3 = a + b \]
\[ \text{param } t_3 \]
\[ t_4 = a \times b \]

expression
examples

\[ f(x+1) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = \text{call}(f,1) \]

\[ f(x+1, 2*y) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = 2 \times y \]
\[ \text{param } t_2 \]
\[ t_3 = \text{call}(f,2) \]

\[ f(g(3*z), h(a+b, a*b)) \]
\[ t_1 = 3 \times z \]
\[ \text{param } t_1 \]
\[ t_2 = \text{call}(g,1) \]
\[ \text{param } t_2 \]
\[ t_3 = a + b \]
\[ \text{param } t_3 \]
\[ t_4 = a \times b \]
\[ \text{param } t_4 \]
\[ t_5 = \text{call}(h,2) \]
examples

\[ f(x+1) \]
\[ t1 = x + 1 \]
\[ param \ t1 \]
\[ t2 = call(f,1) \]

\[ f(x+1,2*y) \]
\[ t1 = x + 1 \]
\[ param \ t1 \]
\[ t2 = 2 * y \]
\[ param \ t2 \]
\[ t3 = call(f,2) \]

\[ f(g(3*z),h(a+b,a*b)) \]
\[ t1 = 3 * z \]
\[ param \ t1 \]
\[ t2 = call(g,1) \]
\[ param \ t2 \]
\[ t3 = a + b \]
\[ param \ t3 \]
\[ t4 = a * b \]
\[ param \ t4 \]
\[ t5 = call(h,2) \]
\[ param \ t5 \]
\[ t6 = call(f,2) \]
Memory Organization
# Memory organization

<table>
<thead>
<tr>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>free memory</td>
</tr>
<tr>
<td>heap</td>
</tr>
<tr>
<td>BSS</td>
</tr>
<tr>
<td>data</td>
</tr>
<tr>
<td>code</td>
</tr>
</tbody>
</table>
Memory organization

- Stack
- Free memory
- Heap
- BSS
- Data
- Code

Machine language
Instructions of the program
Memory organization

- stack
- free memory
- heap
- BSS
- data
- code

initialized static variables, constants, string literals
Memory organization

- stack
- free memory
- heap
- BSS
- data
- code

uninitialized static variables
Memory organization

- Stack
- Free memory
- Heap
- BSS
- Data
- Code

Dynamically allocated memory (e.g. records, arrays)
Memory organization

- stack
- heap
- BSS
- data
- code

Heap grows towards stack
## Memory organization

<table>
<thead>
<tr>
<th>Stack</th>
<th>Free Memory</th>
<th>Heap</th>
<th>BSS</th>
<th>Data</th>
<th>Code</th>
</tr>
</thead>
</table>

'Free memory' denotes the unmapped memory between heap and stack.
Memory organization

<table>
<thead>
<tr>
<th><strong>stack</strong></th>
<th>stack is used for function invocation records (&quot;stack frames&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>free memory</strong></td>
<td></td>
</tr>
<tr>
<td><strong>heap</strong></td>
<td></td>
</tr>
<tr>
<td><strong>BSS</strong></td>
<td></td>
</tr>
<tr>
<td><strong>data</strong></td>
<td></td>
</tr>
<tr>
<td><strong>code</strong></td>
<td></td>
</tr>
</tbody>
</table>
Memory organization

- stack
- heap
- BSS
- data
- code

Stack grows towards heap
## Memory organization

<table>
<thead>
<tr>
<th>Memory Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack</td>
</tr>
<tr>
<td>free memory</td>
</tr>
<tr>
<td>heap</td>
</tr>
<tr>
<td>BSS</td>
</tr>
<tr>
<td>data</td>
</tr>
<tr>
<td>code</td>
</tr>
</tbody>
</table>

The size, layout and contents of both the code and static regions are determined at compile time.
Memory organization

<table>
<thead>
<tr>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>free memory</td>
</tr>
<tr>
<td>heap</td>
</tr>
<tr>
<td>BSS</td>
</tr>
<tr>
<td>data</td>
</tr>
<tr>
<td>code</td>
</tr>
</tbody>
</table>

These regions are handled dynamically (i.e. at runtime)
Memory organization

- Stack
- Free memory
- Heap
- BSS
- Data
- Code

Heap allocation: reserve & release
Memory organization

<table>
<thead>
<tr>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>free memory</td>
</tr>
<tr>
<td>heap</td>
</tr>
<tr>
<td>BSS</td>
</tr>
<tr>
<td>data</td>
</tr>
<tr>
<td>code</td>
</tr>
</tbody>
</table>

Stack allocation: function call
## Stack frame organization

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actual parameters</td>
<td>(arguments)</td>
</tr>
<tr>
<td>returned value</td>
<td></td>
</tr>
<tr>
<td>control link</td>
<td>(dynamic link)</td>
</tr>
<tr>
<td>access link</td>
<td>(static link)</td>
</tr>
<tr>
<td>saved machine status</td>
<td>(return address)</td>
</tr>
<tr>
<td>local data</td>
<td></td>
</tr>
<tr>
<td>temporaries</td>
<td></td>
</tr>
</tbody>
</table>