CS 241 Week 10 Tutorial Solutions

Code Generation: Pointers & Procedures

Fall 2018

1 Extending WLP4: Pre- and Post-Increment

1. If lvalue is an int, then so is factor. If it is an int*, then so is factor.

2. We need to modify the line which says add $3, $3, $11 to say:

   if (typeOf(lvalue) == int*) {
       add $3, $3, $4
   } else {
       add $3, $3, $11
   }

2 Extending WLP4: Pointers as Conditions

1. Recall that all of our conditions are under the test nonterminal, so we should add our new rule there:

   test → expr

2. We need to verify that expr has type int* for the new rule to be well-typed.

3. We want to put 1 in $3 if the condition is true, and 0 otherwise. Note that different sections may use different values for NULL. Here assume that it is 0, but it should be easy to swap out.

   genCode(test expr) {
       genCode(expr)
       add $5, $3, $0 ;Back up the value of expr
       lis $6
       .word 0 ;Our value for NULL
       add $3, $0, $0
       beq $5, $6, 1
       add $3, $11, $0
   }
3 Procedures in WLP4

We will first review the steps for calling a procedure, assuming the compiler puts the responsibility to save/restore registers (except $29, $31) on the callee. In reality, a compiler may use another approach to save the registers, so here is one of the possible solution:

In caller:
1. Push $31, the caller’s return address to the stack.
2. Push $29, the caller’s frame pointer to the stack.
3. Push the parameters of the call to the stack, from left to right.
4. Pass control to callee

In callee:
1. Set the frame pointer, the compiler will assume the top \( n \) items in the stack is the arguments.
2. Reserve spaces for other local variables in the procedure and set the stack pointer.
3. Push registers this procedure will overwrite to the stack.

By looking at the given memory blocks and the value stored in $29, we can figure out the frame for wain. The frame pointer points to address 0x00fffffc, so the first variable \( a \) (with offset 0) is in address 0x00fffffc, and the second variable \( b \) (with offset -4) is in address 0x00fffff8.

Finally the return address is stored in $30 and the frame pointer is stored in $29.

We will place these information into stack according to the steps, and get the following:

<table>
<thead>
<tr>
<th>address</th>
<th>content</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00ffffd0</td>
<td></td>
<td>saved register $5</td>
</tr>
<tr>
<td>0x00ffffd4</td>
<td>0x000000088</td>
<td>f: f</td>
</tr>
<tr>
<td>0x00ffffd8</td>
<td>0x0000001</td>
<td>f: e</td>
</tr>
<tr>
<td>0x00ffffdc</td>
<td>0x00000014</td>
<td>f: d</td>
</tr>
<tr>
<td>0x00ffffe0</td>
<td>0x0000002</td>
<td>f: c</td>
</tr>
<tr>
<td>0x00ffffe4</td>
<td>0x00fffff8</td>
<td>f: a</td>
</tr>
<tr>
<td>0x00ffffe8</td>
<td>0x0000001</td>
<td>f: b</td>
</tr>
<tr>
<td>0x00ffffec</td>
<td>0x00fffffc</td>
<td>f: a</td>
</tr>
<tr>
<td>0x00ffffff0</td>
<td>0x00ffffffc</td>
<td>fp for wain</td>
</tr>
<tr>
<td>0x00ffffff4</td>
<td>0x8123456c</td>
<td>ret for wain</td>
</tr>
<tr>
<td>0x00ffffff8</td>
<td>0x00000002</td>
<td>wain: b</td>
</tr>
<tr>
<td>0x00ffffffc</td>
<td>0x00000020</td>
<td>wain: a</td>
</tr>
<tr>
<td>0x01000000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The frame pointer points to the bottom of the frame, which is the address of the parameter \( a \). Finally, since the procedure is about to return, the return value should be ready in $3. Thus, we have:

$29 = 0x00ffffffec // fp for procedure f (could also be 0x00ffffffdc if you choose to positive offsets for parameters in non-wain procedures)
$30 = 0x00ffffffd4 // top of the stack
$03 = 0x00000009 // The return value should be computed.