Compilers

Temporaries
• Idea: Keep temporaries in the AR

• The code generator must assign a location in the AR for each temporary
def fib(x) = if x = 1 then 0 else
  if x = 2 then 1 else
    fib(x - 1) + fib(x - 2)

2 temporaries
• Let $\text{NT}(e) = \# \text{ of temps needed to evaluate } e$

• $\text{NT}(e_1 + e_2)$
  – Needs at least as many temporaries as $\text{NT}(e_1)$
  – Needs at least as many temporaries as $\text{NT}(e_2) + 1$

• Space used for temporaries in $e_1$ can be reused for temporaries in $e_2$
\[ NT(e_1 + e_2) = \max(NT(e_1), 1 + NT(e_2)) \]
\[ NT(e_1 - e_2) = \max(NT(e_1), 1 + NT(e_2)) \]
\[ NT(%if e_1 = e_2 then e_3 else e_4%) = \max(NT(e_1), 1 + NT(e_2), NT(e_3), NT(e_4)) \]
\[ NT(id(e_1, \ldots, e_n)) = \max(NT(e_1), \ldots, NT(e_n)) \]
\[ NT(int) = 0 \]
\[ NT(id) = 0 \]
def fib(x) = if x = 1 then 0 else
  if x = 2 then 1 else
    fib(x - 1) + fib(x - 2)
• For a function definition $f(x_1,\ldots,x_n) = e$ the AR has $2 + n + \text{NT}(e)$ elements
  – Return address
  – Frame pointer
  – $n$ arguments
  – $\text{NT}(e)$ locations for intermediate results
### Temporaries

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Old FP</strong></td>
<td></td>
</tr>
<tr>
<td><strong>$x_n$</strong></td>
<td></td>
</tr>
<tr>
<td><strong>…</strong></td>
<td></td>
</tr>
<tr>
<td><strong>$x_1$</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Return Addr.</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Temp NT(e)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>…</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Temp 1</strong></td>
<td></td>
</tr>
</tbody>
</table>
For the `powerOfTwo()` function at right, what are the numbers of temporaries required to evaluate each sub-expression, and the total number of temporaries required for `powerOfTwo()`?

```python
def powerOfTwo(x):
    if x % 2 == 0:
        return powerOfTwo(x / 2)
    else:
        return x == 1
```

<table>
<thead>
<tr>
<th>$x % 2 == 0$</th>
<th>$powerOfTwo(x / 2)$</th>
<th>$x == 1$</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>
• Code generation must know how many temporaries are in use at each point

• Add a new argument to code generation – the position of the next available temporary

• The temporary area is used like a small, fixed-size stack
cgen(e₁ + e₂) =
\[
\begin{align*}
\text{cgen}(e₁) \\
\text{sw } $a₀ 0($sp) \\
\text{addiu } $sp $sp -4 \\
\text{cgen}(e₂) \\
\rightarrow \text{lw } $t₁ 4($sp) \\
\rightarrow \text{add } $a₀ $t₁ $a₀ \\
\rightarrow \text{addiu } $sp $sp 4
\end{align*}
\]
\[ \text{cgen}(e_1 + e_2, \text{nt}) = \]
\[ \text{cgen}(e_1, \text{nt}) \]
\[ \rightarrow \text{sw} \; \$a0 \; \text{nt}($fp) \]
\[ \text{cgen}(e_2, \text{nt} + 4) \]
\[ \text{lw} \; \$t1 \; \text{nt}($fp) \]
\[ \rightarrow \text{add} \; \$a0 \; \$t1 \; \$a0 \]