Today’s topics

Language Translation
Generating Code

Upcoming
Inheritance
(back to Chapter 5)

Reading
Great Ideas, Chapters 9
Adding Semantics

- Need to add semantic components to our rules
  - For every syntax rule, we will add a semantic rule
  - This will show the assembler code generated
  - The code, as interpreted by the machine will provide the meaning

- Revise R1

<table>
<thead>
<tr>
<th>Syntax Rule</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1: (&lt;n)&gt;j -&gt; w</td>
<td>M(&lt;n&gt;j) = w</td>
</tr>
</tbody>
</table>

- M ... Meaning of ... Name ... Memory location
- In other words, use same identifier/name in both Java and Assembler
Adding Semantics

- **Revise R2**
  Syntax Rule  Semantic Rule
  R2: \(<e>_i \rightarrow <n>_j\)  \(M(<e>_i) = M(<n>_j)\)
  code\(<e>_i\) = *nothing*

  - No code is generated!

- **Revise R3**
  Syntax Rule  Semantic Rules
  R3: \(<s>_k \rightarrow <n>_j = <e>_i;\)  code\(<s>_k\) = code\(<e>_i\)
  
  - Says code for statement is code to calculate expression \(<e>_i\) and code to copy it into memory associated with \(<n>_j\)
## Generating Code

- **Now have enough to demonstrate simplest case**
  - Use syntactic production to *control* process
  - Associated semantic rules are applied at each step
- **Use rules to generate code for** \( x = y; \)

<table>
<thead>
<tr>
<th>Derivation</th>
<th>Syntax Rule</th>
<th>Semantic Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>R3: s1 -&gt; n2 = e3; code(s1) = code(e3)</td>
<td>COPY AX, M(e3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COPY M(n2), AX</td>
</tr>
</tbody>
</table>

**MEANING:**

\[
\text{code(s1)} = \text{code(e3)}
\]

\[
\text{COPY AX, M(e3)}
\]

\[
\text{COPY M(n2), AX}
\]

n2 = e3;

<table>
<thead>
<tr>
<th>R1: n2 -&gt; x</th>
<th>M(n2) = x</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEANING:</td>
<td>code(s1) = code(e3)</td>
</tr>
<tr>
<td></td>
<td>COPY AX, M(e3)</td>
</tr>
<tr>
<td></td>
<td>COPY X, AX</td>
</tr>
</tbody>
</table>

the
# Generating Code

<table>
<thead>
<tr>
<th>Derivation</th>
<th>Syntax Rule</th>
<th>Semantic Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = e3;</td>
<td>R2: e3 -&gt; n4</td>
<td>M(e3) = M(n4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>code(e3) = nothing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MEANING: code(s1) = nothing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COPY AX, M(n4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COPY X, AX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x = n4;</th>
<th>R1: n4 -&gt; Y</th>
<th>M(n4) = Y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MEANING: code(s1) =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COPY AX, Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COPY X, AX</td>
</tr>
</tbody>
</table>
More Rules

✧ Revise R4

Syntax Rule

R4: $<e>_i \rightarrow ( <e>_j + <e>_k )$

Semantic Rules

$M( <e>_i ) = createName$

code( $<e>_i$ ) =

\[
\text{code}( <e>_j ) \quad \text{code}( <e>_k ) \\
\text{COPY AX, M(} <e>_j \text{)} \\
\text{ADD AX, M(} <e>_k \text{)} \\
\text{COPY M(} <e>_i \text{), AX}
\]

☐ Says code for $<e>_i$ is code to calculate expression $<e>_j$ followed by code to calculate expression $<e>_k$ and code to add them together and store that sum into memory associated with $<e>_i$
More Rules

❖ Revise R5

Syntax Rule
R5: \(<e>_i -> (<e>_j * <e>_k)\)

Semantic Rule
M(<e>_i) = createname

code(<e>_i) = code(<e>_j) code(<e>_k)

COPY AX, M(<e>_j)

MUL AX, M(<e>_k)

COPY M(<e>_i), AX

❖ Says code for <e>_i is code to calculate expression <e>_j followed by code to calculate expression <e>_k and code to multiply them together and store that sum into memory associated with <e>_i

❖ Basically, rules R4 and R5 are identical except that the + and ADD in one are replaced by the * and MUL in the other.
**Code for** \( Z = (X + Y) ; \)

<table>
<thead>
<tr>
<th>Derivation</th>
<th>Syntax Rule</th>
<th>Semantic Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>R3: s1 -&gt; n2 = e3;</td>
<td>code(s1) = code(e3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COPY AX, M(e3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COPY M(n2), AX</td>
</tr>
</tbody>
</table>

**MEANING:**

code(s1) = code(e3)

COPY AX, M(e3)

COPY M(n2), AX

---

n2 = e3;  
R1: n2 -> Z  
M(n2) = Z

**MEANING:**

code(s1) = code(e3)

COPY AX, M(e3)

COPY Z, AX
### Code for \( Z = (X + Y) ; \)

<table>
<thead>
<tr>
<th>Derivation</th>
<th>Syntax Rule</th>
<th>Semantic Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Z = e_3; )</td>
<td>R4: ( e_3 \rightarrow (e_4 + e_5) )</td>
<td>( M(e_3) = \text{CN1} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{code}(e_3) = \text{code}(e_4) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{code}(e_5) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{COPY AX, } M(e_4) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{ADD AX, } M(e_5) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{COPY } M(e_3), \ AX )</td>
</tr>
</tbody>
</table>

**MEANING:**

\[
\text{code}(s_1) = \text{code}(e_4) \\
\text{code}(e_5) \\
\text{COPY AX, } M(e_4) \\
\text{ADD AX, } M(e_5) \\
\text{COPY CN1, AX} \\
\text{COPY AX, } CN1 \\
\text{COPY Z, AX}
\]
### Code for \( Z = (X + Y) \); .3

\[
Z = (e4 + e5); \quad \text{R2: } e4 \rightarrow n6 \quad \text{M(e4) = M(n6)}
\]

\[
\text{code(e4) = } \text{nothing}
\]

**MEANING:**
- \( \text{code(s1) = nothing} \)
- \( \text{code(e5)} \)
  - \( \text{COPY AX, M(n6)} \)
  - \( \text{ADD AX, M(e5)} \)
  - \( \text{COPY CN1, AX} \)
  - \( \text{COPY AX, CN1} \)
  - \( \text{COPY Z, AX} \)

\[
Z = (n6 + e5); \quad \text{R1: } n6 \rightarrow x \quad \text{M(n6) = x}
\]

**MEANING:**
- \( \text{code(s1) = code(e5)} \)
  - \( \text{COPY AX, X} \)
  - \( \text{ADD AX, M(e5)} \)
  - \( \text{COPY CN1, AX} \)
  - \( \text{COPY AX, CN1} \)
  - \( \text{COPY Z, AX} \)
**Code for** \( Z = (X + Y); \)

<table>
<thead>
<tr>
<th>( Z = (X + e5); )</th>
<th>R2: ( e5 \rightarrow n7 )</th>
<th>( M(e5) = M(n7) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>code(e5) = nothing</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MEANING:**

<table>
<thead>
<tr>
<th>s1</th>
</tr>
</thead>
<tbody>
<tr>
<td>code(s1) = nothing</td>
</tr>
<tr>
<td>COPY AX, X</td>
</tr>
<tr>
<td>ADD AX, M(n7)</td>
</tr>
<tr>
<td>COPY CN1, AX</td>
</tr>
<tr>
<td>COPY AX, CN1</td>
</tr>
<tr>
<td>COPY Z, AX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( Z = (X + n7); )</th>
<th>R1: ( n7 \rightarrow Y )</th>
<th>( M(n7) = Y )</th>
</tr>
</thead>
</table>

**MEANING:**

<table>
<thead>
<tr>
<th>s1</th>
</tr>
</thead>
<tbody>
<tr>
<td>code(s1) =</td>
</tr>
<tr>
<td>COPY AX, X</td>
</tr>
<tr>
<td>ADD AX, Y</td>
</tr>
<tr>
<td>COPY CN1, AX</td>
</tr>
<tr>
<td>COPY AX, CN1</td>
</tr>
<tr>
<td>COPY Z, AX</td>
</tr>
</tbody>
</table>
Towards a Real Program

- More complicated statement:

```
U1 = (X + (Y * Z)) ;
```

- Done on pages 277-279 in text

- (Note that book uses `<i>j` where we used `<n>j`)

- Rules for Looping Sequence of statements

- Rules 6 and 7: A sequence of statements

<table>
<thead>
<tr>
<th>Syntax Rule</th>
<th>Semantic Rules</th>
</tr>
</thead>
</table>
| R6: `<q>_<i> -> `<s>_<j> | code(`<q>_<i>) = code(`<s>_<j>)
| `<q>_<k> | code(`<q>_<k>) |

<table>
<thead>
<tr>
<th>R7: <code>&lt;q&gt;_&lt;i&gt; -&gt; </code>&lt;s&gt;_&lt;j&gt;</th>
<th>code(<code>&lt;q&gt;_&lt;i&gt;) = code(</code>&lt;s&gt;_&lt;j&gt;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Says code for a sequence of statements is the code for the first statement followed by the code for the next statement, etc.</td>
<td></td>
</tr>
<tr>
<td>Notice the recursive nature of these statements.</td>
<td></td>
</tr>
</tbody>
</table>
More Complicated Statements

- **Rule 8: Compound Statement**

  Syntax Rule
  
  R8: $<c>_i \rightarrow \{$
  
  <q>_j
  
  $\}$

  Semantic Rules
  
  $\text{code}(<c>_i) = \text{code}(<q>_j)$

- **Rule 9: While Statement**

  Syntax Rule
  
  R9: $<s>_i \rightarrow$
  
  while ($<n>_j < <e>_k$)
  
  $<c>_h$

  Semantic Rules
  
  $\text{M}(<s>_i) = \text{createname}$
  
  $\text{M}'(<s>_i) = \text{createname}$
  
  $\text{code}(<s>_i) =$
  
  $\text{M}(<s>_i) \text{ code}(<e>_k) \text{ COPY AX, M(<n>_j)\text{ CMP AX, M(<e>_k)\text{ JNB M'(<s>_i)\text{ code(<c>_h)\text{ JMP M(<s>_i)\text{ M}'(<s>_i) \text{NO-OP}}}$
Final Thoughts

- Clean Up Translation
  - Some code generated can be removed
  - Modern compilers spend a lot of effort *optimizing*
- Important: Everything done by *simple substitution*
- Everything “adds up”
  - `code( { <s>₁;<s>₂;<s>₃ } )`
    - is
    - `code(<s>₁)`
    - `code(<s>₂)`
    - `code(<s>₃)`