Intermediate Representation

TEACHING ASSISTANT: DAVID TRABISH

Intermediate Representation

- Allows **language** and **machine** independent optimizations
- Translated from the AST
- Translated to machine code

IR Language

- Temporary variables (IR registers)
  - t1, t2, … (unlimited)
- Instructions
  - Assignments
    - t1 = c (assign constant value)
    - t1 = x (read from memory x)
    - x = t1 (write to memory x)
  - add, sub, call, return, …
- Labels
  - label_1:

IR Example

```
int foo(int x, int y) {
  int z = x + y;
  int w = z + 1;
  return w;
}
```

```
t1 = x
t2 = y
t3 = add t1, t2
z = t3
t4 = z
t5 = 1
t6 = add t4, t5
w = t6
t7 = w
return t7
```

Translating Expressions

- For leaf node
  - Generate code, and store in a new register t_{new}
- For Internal nodes
  - Process first child, store result in t_{left}
  - Process second child, store result in t_{right}
  - Apply node operation on t_{left} and t_{right}
  - Store the result in t_{result}

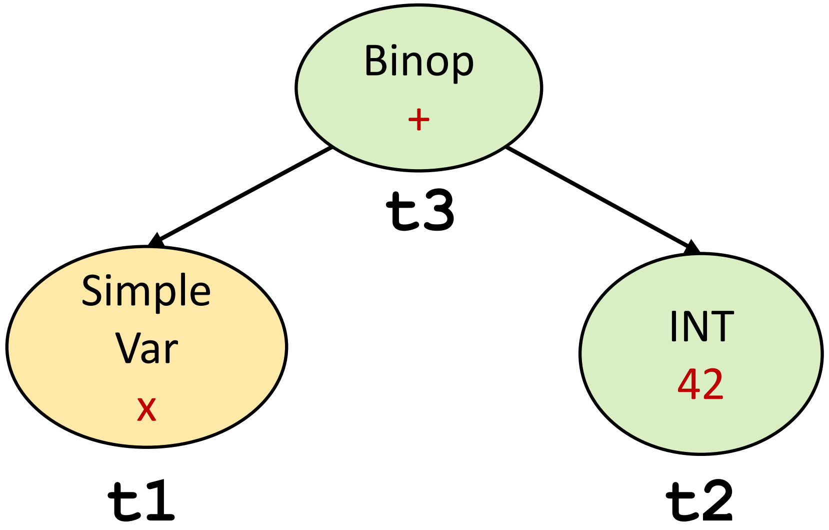
Translating Expressions

For an AST node e we define:

- $T_c(e)$
  - The generated instructions (code)
- $T_r(e)$
  - The register holding the result of the computation

Translating Expressions

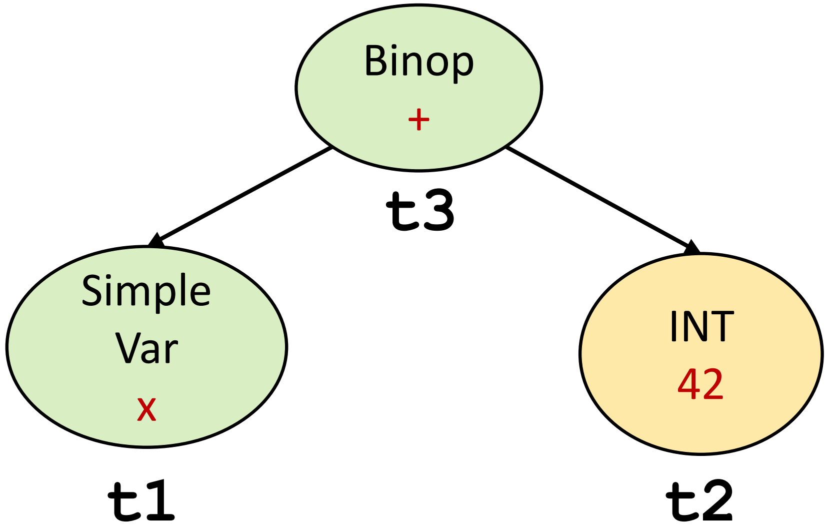
For x + 42:




```
t1 = x
```

Translating Expressions

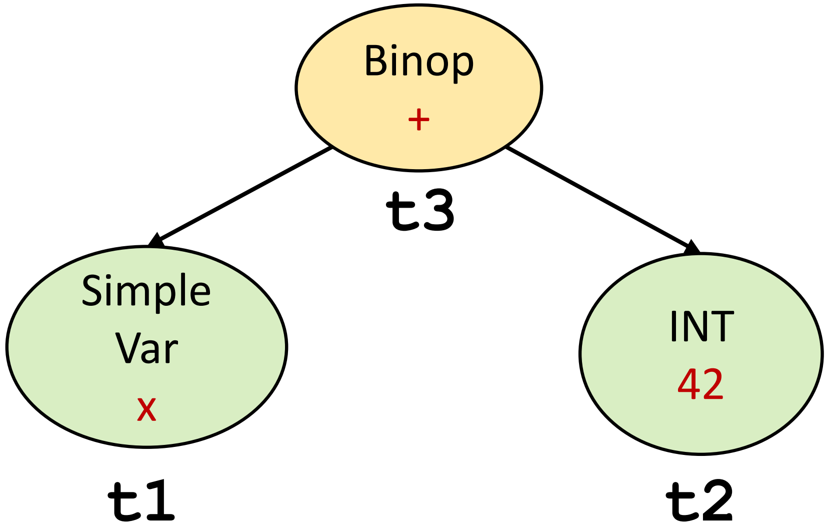
For x + 42:




```
t1 = x
t2 = 42
```

Translating Expressions

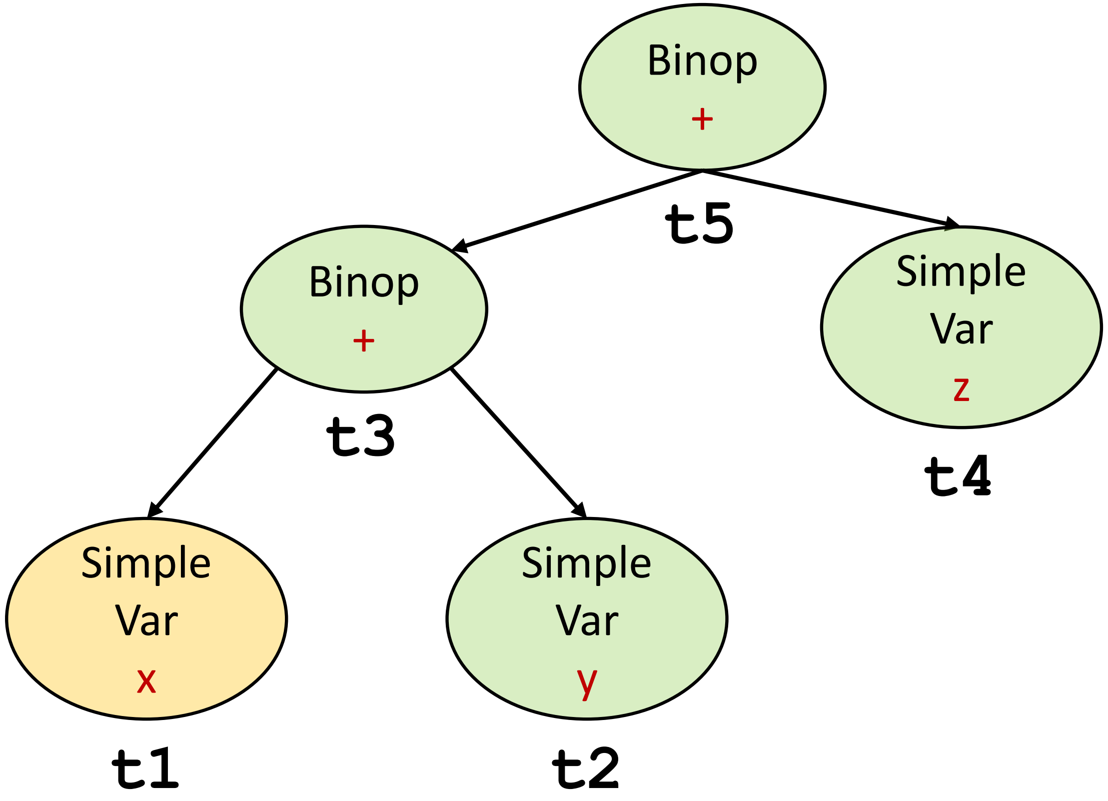
For x + 42:




```
t1 = x
t2 = 42
t3 = add t1, t2
```

Translating Expressions

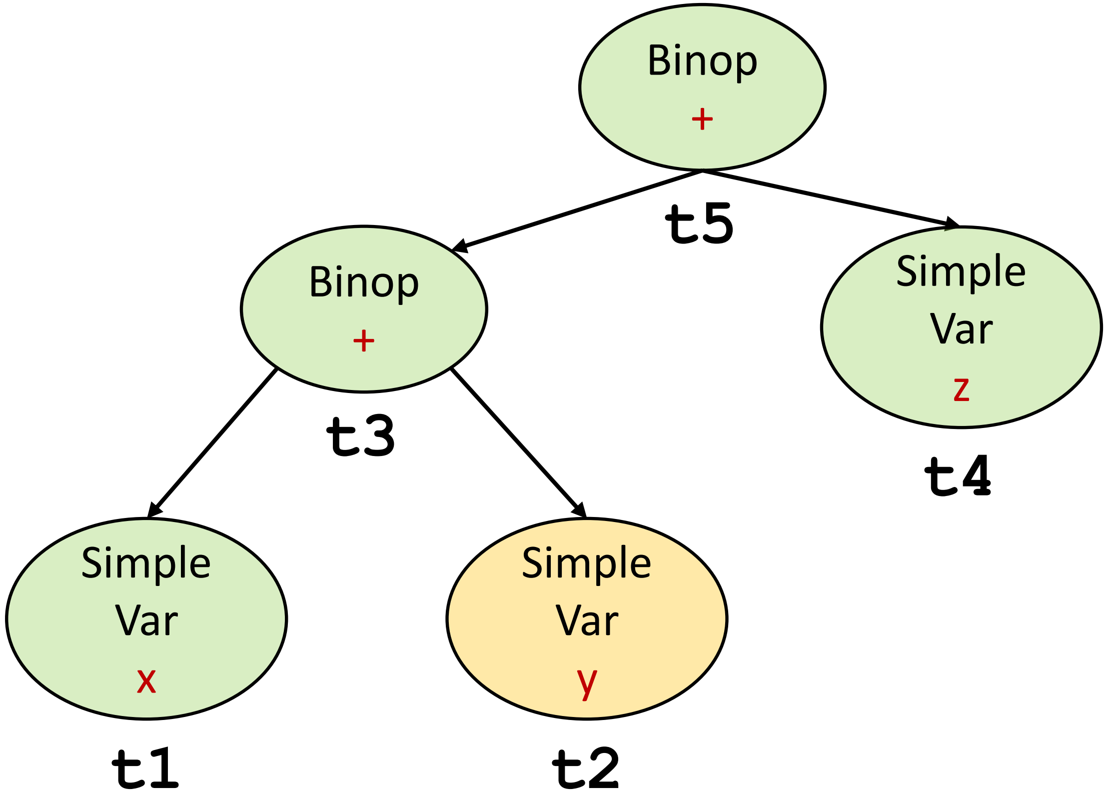
For x + y + z:




```
t1 = x
```

Translating Expressions

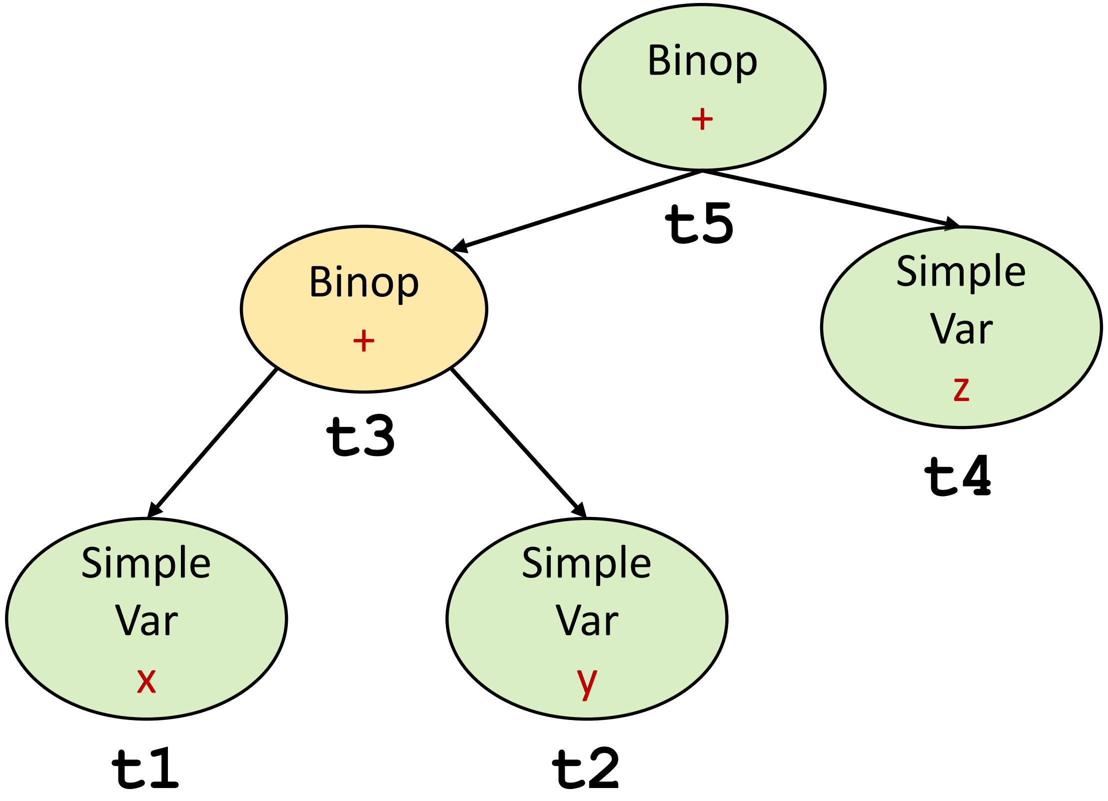
For x + y + z:




```
t1 = x
t2 = y
```

Translating Expressions

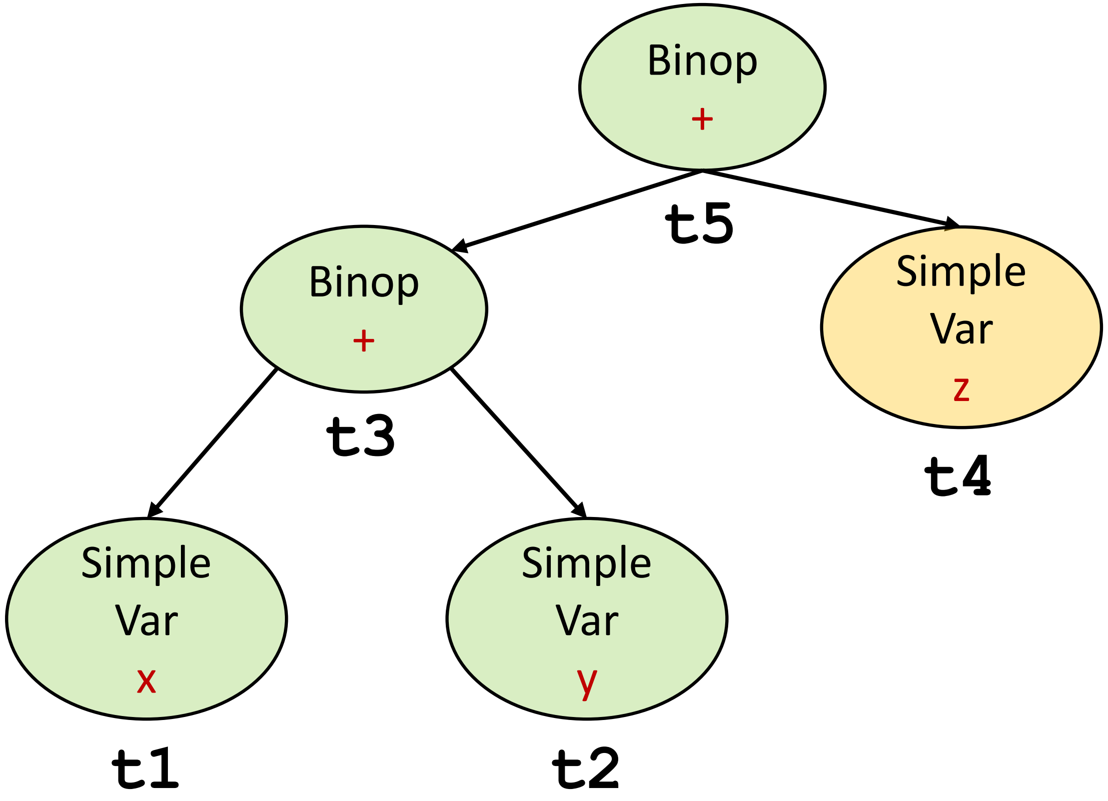
For x + y + z:




```
t1 = x
t2 = y
t3 = add t1, t2
```

Translating Expressions

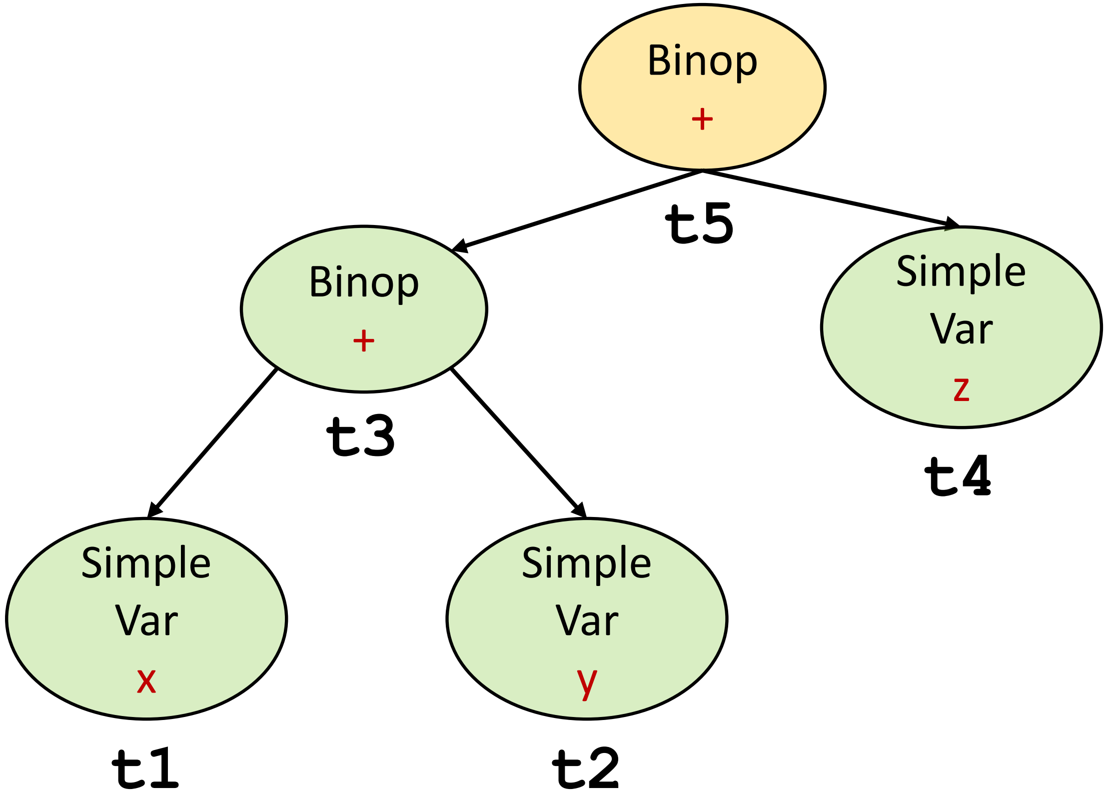
For x + y + z:




```
t1 = x
t2 = y
t3 = add t1, t2
t4 = z
```

Translating Expressions

For x + y + z:




```
t1 = x
t2 = y
t3 = add t1, t2
t4 = z
t5 = add t3, t4
```

Translating Expressions

For $e_1 \; or \; e_2$:

$T_c(e_1)$

```
...
t1 = ...
t3 = 1
compare t1, t3
branch_eq end_label
```

$T_r(e_1)$

$T_c(e_2)$

```
...
t2 = ...
t3 = or t1, t2
end_label:
```

$T_r(e_2)$

Translating Expressions

For $e_1 \; and \; e_2$:

$T_c(e_1)$
```
...
t1 = ...
```
$T_r(e_1)$
```
t3 = 0
compare t1, t3
branch_eq end_label
```
$T_c(e_2)$
```
...
t2 = ...
```
$T_r(e_2)$
```
t3 = and t1, t2
end_label:
```

Translating Expressions

For $e_1 == e_2$:

$T_c(e_1)$ $\left\{ \begin{array}{l} \texttt{...} \\ \texttt{t1 = ...} \end{array} \right.$

$T_c(e_2)$ $\left\{ \begin{array}{l} \texttt{...} \\ \texttt{t2 = ...} \end{array} \right.$

```
t3 = 1
compare t1, t2
branch_eq end_label
t3 = 0
end_label:
```

Translating Expressions

For a == b + 1:

```
t1 = a
t2 = b
t3 = 1
t4 = add t2, t3
t5 = 1
compare t1, t4
branch_eq end_label
t5 = 0
end_label:
```

Translating Expressions

For $e_1[e_2]$:

$T_c(e_1)$

```
...
t1 = ...
```

$T_r(e_1)$

$T_c(e_2)$

```
...
t2 = ...
t3 = array_access t1, t2
```

$T_r(e_2)$

Translating Expressions

For x[z+1]:

```
t1 = x
t2 = z
t3 = 1
t4 = add t2, t3
t5 = array_access t1, t4
```

Translating Expressions

For $e.f$:

$T_c(e)$

```
...
t1 = ...
```

$T_r(e)$

```
t2 = field_access t1, f
```

Translating Expressions

For x[3].foo:

```
t1 = x
t2 = 3
t3 = array_access t1, t2
t4 = field_access t3, foo
```

Translating Basic Block

For $s_1; s_2; \ldots$:

$$T_c(s_1)$$
$$T_c(s_2)$$
$$\ldots$$

Translating Statements

For $if\ (e)\ then\ \{s\}$:

$T_c(e)$
```
...
t1 = ...
```
$T_r(e)$ → `t1`
```
compare t1, 0
branch_eq end_label
```
$T_c(s)$
```
...
...
end_label:
```

Translating Statements

For if (x * y) then { z = 0; }:

```
t1 = x
t2 = y
t3 = mul t1, t2
compare t3, 0
branch_eq end_label
t4 = 0
z = t4
end_label:
```

Translating Statements

For $if\ (e)\ then\ \{s_1\}\ else\ \{s_2\}$:

$T_c(e)$
```
...
t1 = ...
```
$T_r(e_1)$
```
compare t1, 0
branch_eq false_label
```
$T_c(s_1)$
```
...
...
branch end_label
false_label:
```
$T_c(s_2)$
```
...
...
end_label:
```

Translating Statements

For if (w) then { z = 0; } else { z = 100; }:

```
t1 = w
compare t1, 0
branch_eq false_label
t2 = 0
z = t2
branch end_label
false_label:
t3 = 100
z = t3
end_label:
```

Translating Statements

For $while\ (e)\ \{s\}$:

```
cond_label:
```

$T_c(e)$

```
...
t1 = ...
```

$T_r(e)$

```
compare t1, 0
branch_eq end_label
```

$T_c(s)$

```
...
...
```

```
branch cond_label
end_label:
```

Translating Statements

For while (z / x) { }:

```
cond_label:
t1 = z
t2 = x
t3 = div t1, t2
compare t3, 0
branch_eq end_label
branch cond_label
end_label:
```

Translating Statements

For $f(e_1, e_2, \dots)$:

$T_c(e_1)$
```
...
t1 = ...
```

$T_c(e_2)$
```
...
t2 = ...
```

```
...
t0 = call f(t1, t2, ...)
```

Translating Statements

For func(2, x + 1):

```
t1 = 2
t2 = x
t3 = 1
t4 = add t2, t3
t5 = call func(t1, t4)
```

Translating Statements

For $o.f(e_1, e_2, \dots)$:

$T_c(o)$ $\left\{ \begin{array}{l} \texttt{...} \\ \texttt{t1} \end{array} \right.$

$T_c(e_1)$ $\left\{ \begin{array}{l} \texttt{...} \\ \texttt{t2 = ...} \end{array} \right.$

$T_c(e_2)$ $\left\{ \begin{array}{l} \texttt{...} \\ \texttt{t3 = ...} \end{array} \right.$

```
...
t0 = virtual_call t1.f(t2, t3, ...)
```

Translating Statements

For obj.bar(2, x + 1):

```
t1 = obj
t2 = 2
t3 = x
t4 = 1
t5 = add t3, t4
t6 = virtual_call t1.func(t2, t5)
```

Translating Statements

For $return\ e$:

$$T_c(e) \left\{ \begin{array}{l} \texttt{...} \\ \texttt{t1 = ...} \end{array} \right.$$

```
return t1
```

Translating Statements

For return w * 3:

```
t1 = w
t2 = 3
t3 = mul t1, t2
return t3
```

Example

```
x = 42;
while (x > 0) {
  x = x - 1;
}
```

$T_c($

```
x = 42;
while (x > 0) {
  x = x - 1;
}
```

$)$

Example

```
x = 42;
while (x > 0) {
  x = x - 1;
}
```

$T_c($`x = 42`$)$

$T_c($

```
  while (x > 0) {
    x = x - 1;
  }
```

$)$

Example

```
x = 42;
while (x > 0) {
  x = x - 1;
}
```

```
t1 = 42
x = t1
```
$T_c($

```
  while (x > 0) {
    x = x - 1;
  }
```
$)$

Example

```
x = 42;
while (x > 0) {
  x = x - 1;
}
```

```
t1 = 42
x = t1
cond_label:
```
$T_c($`x > 0`$)$

`compare` $T_r($`x > 0`$)$`, 0`

`branch_eq end_label`

$T_c($`x = x - 1`$)$

```
branch cond_label
end_label:
```

Example

```
x = 42;
while (x > 0) {
  x = x - 1;
}
```

```
t1 = 42
x = t1
cond_label:
t2 = x
t3 = 0
t4 = 0
compare t2, t3
branch_gt cmp_label:
t4 = 1
cmp_label:
compare T_r(x > 0), 0
branch_eq end_label
T_c(x = x - 1)
branch cond_label
end_label:
```

Example

```
x = 42;
while (x > 0) {
  x = x - 1;
}
```

```
t1 = 42
x = t1
cond_label:
t2 = x
t3 = 0
t4 = 0
compare t2, t3
branch_gt cmp_label:
t4 = 1
cmp_label:
compare t4, 0
branch_eq end_label
```

$T_c($`x = x - 1`$)$

```
branch cond_label
end_label:
```

Example

```
x = 42;
while (x > 0) {
  x = x - 1;
}
```

```
t1 = 42
x = t1
cond_label:
t2 = x
t3 = 0
t4 = 0
compare t2, t3
branch_gt cmp_label:
t4 = 1
cmp_label:
compare t4, 0
branch_eq end_label
t5 = x
t6 = 1
t7 = sub t5, t6
x = t7
branch cond_label
end_label:
```

Implementation

- Classes for IR Instructions
- AST Visitor
  - Define visitor for each node type
  - Should return
    - List of generated instructions
    - Result register (for expressions)

Alternative Representation

For z = x + 42 the generated code is:

```
t1 = x
t2 = 42
t3 = add t1, t2
z = t3
```

Alternative Representation

We can take a more low level approach:
(assuming that x is first parameter and z first local variable)

```
t1 = add fp, 8
t2 = load t1
t3 = 42
t4 = add t2, t3
t5 = sub fp, 4
store t5, t3
```