

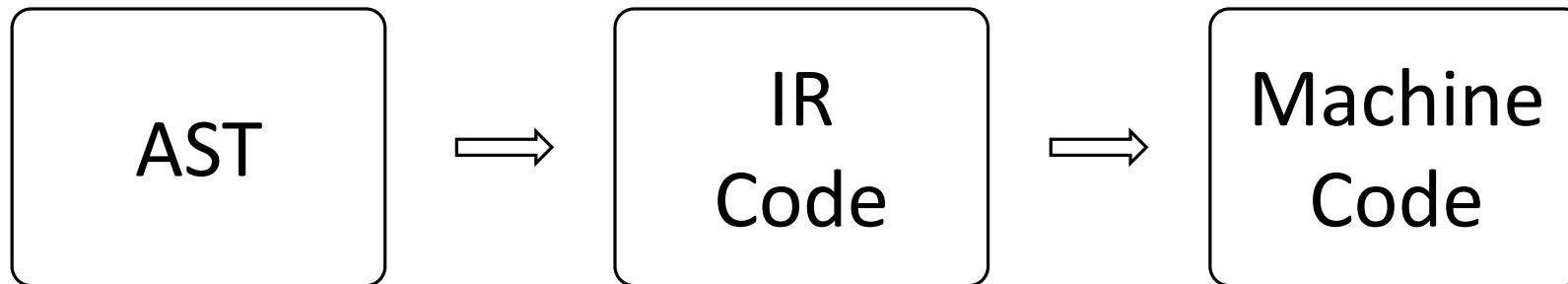
Intermediate Representation

TEACHING ASSISTANT: DAVID TRABISH



Intermediate Representation

- Generic representation of instructions
 - Allows **language** and **machine** independent optimizations
 - Not executable



IR Language

- Temporary variables (IR registers)
 - t1, t2, ... (unlimited)
- Instructions
 - assignments, add, sub, call, return, ...
- Labels
 - label_1:

IR Instructions

Constant assignment:

- **<register> = <constant>**
 - $t1 = 7$

IR Instructions

Read from memory:

- **<register> = <variable_name>**
 - $t1 = x$

Write to memory:

- **<variable_name> = <register>**
 - $y = t2$

IR Instructions

Arithmetic operations:

- **<register> = op <register> <register> ...**
 - t4 = add t1 t2
 - t0 = sub t0 t1

IR Instructions

Branches:

- **br <label>**
 - br some_label
- **beq <register> [<constant> | <register>] <label>**
 - beq t1 0 label_1
 - beq t2 t3 label_7

IR Instructions

Functions:

- **call** <function_name> <args>
 - call bar
 - call foo t1 t2
- **<register> = call** <function_name> <args>
 - t8 = call foo t7
- **return** <register>
 - return t3

IR Instructions

Arrays:

- **<register> = new_array <register>**
 - t0 = new_array t1
- **<register> = array_access <register> <register>**
 - t0 = array_access t1 t2
- **array_set <register> <register> <register>**
 - array_set t0 t1 t2

IR Instructions

Classes:

- `<register> = new_class <type_name>`
 - `t0 = new_array Base`
- `<register> = field_access <register> <field_name>`
 - `t0 = field_access t1 name`
- `field_set <register> <field_name> <register>`
 - `field_set t0 name t2`
- `virtual_call <register> <method_name> <args>`
 - `virtual_call t1 foo`
- `<register> = virtual_call <register> <method_name> <args>`
 - `t0 = virtual_call t1 foo t20, t21`

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 AST to IR

- Input: **AST**
- Output: **List of IR instructions**
- Done using **AST visitor**

Translating Expressions

Basic algorithm:

visit(node):

$instlist_1, reg_1 = visit(node.child_1)$

...

$instlist_n, reg_n = visit(node.child_n)$

$instlist = instlist_1 + \dots + instlist_n$

$t_{new} = compute(t_1, \dots, t_n)$

return instlist, t_{new}

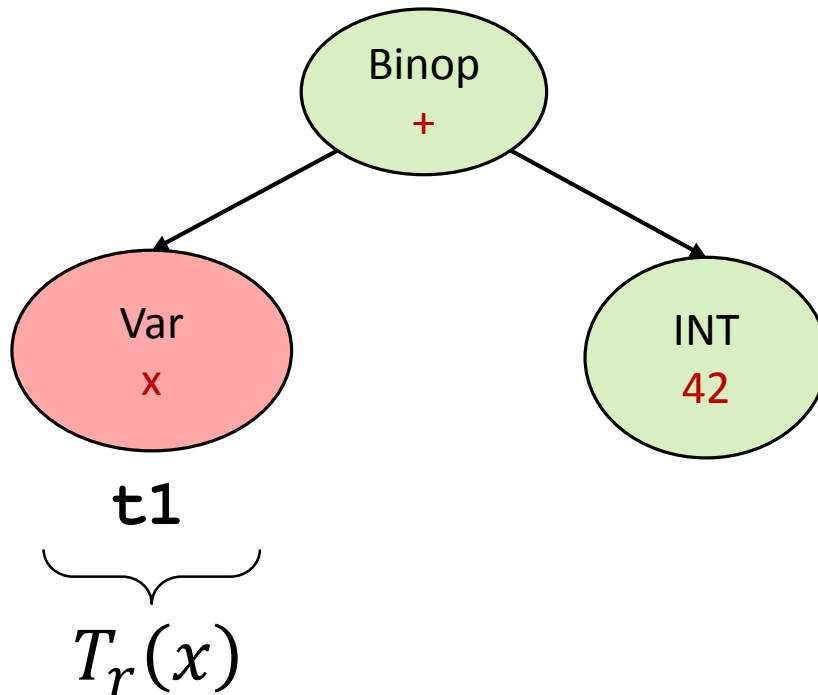
Translating Expressions

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

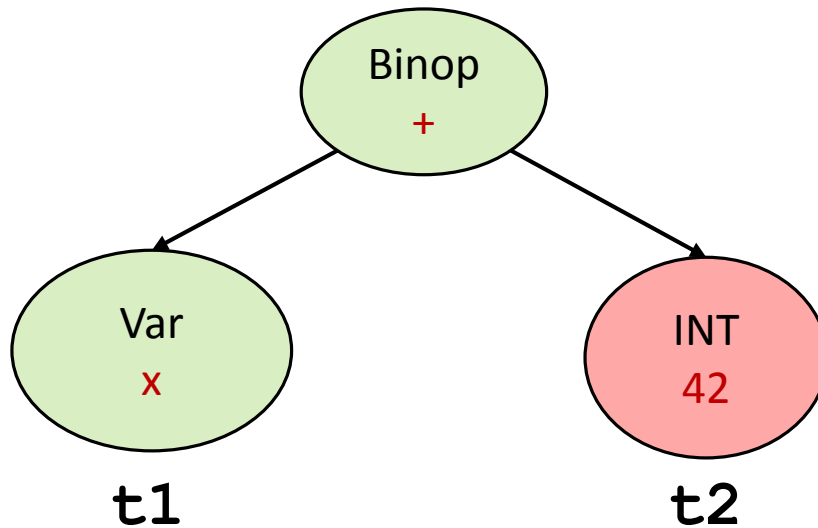
For $x + 42$:



$$\underbrace{\mathbf{t1} = \mathbf{x}}_{T_c(x)}$$

Translating Expressions

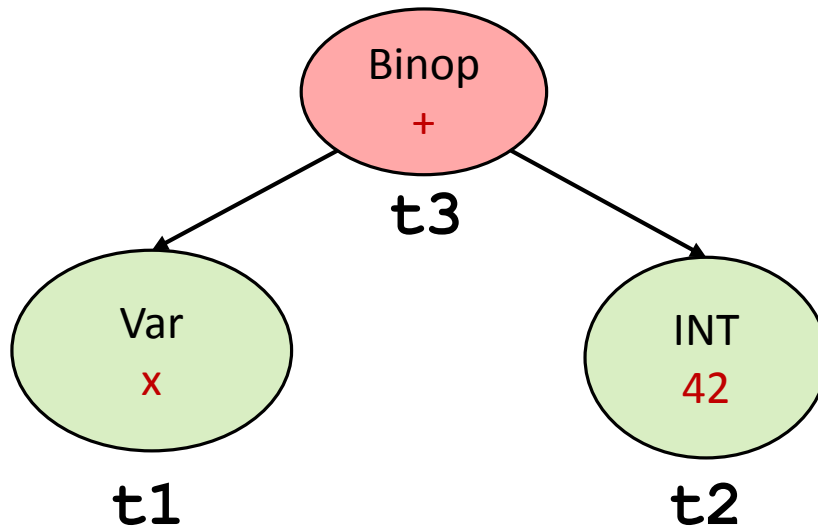
For $x + 42$:



t1 = x
t2 = 42

Translating Expressions

For $x + 42$:



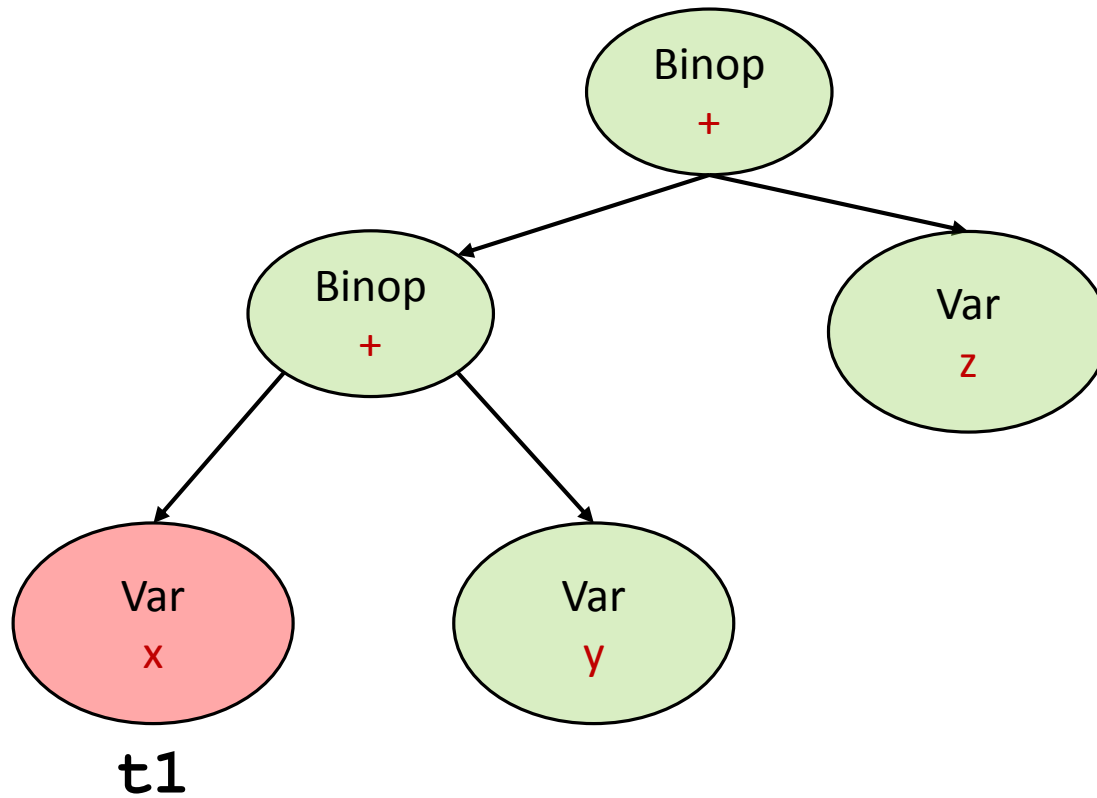
t1 = x

t2 = 42

t3 = add t1, t2

Translating Expressions

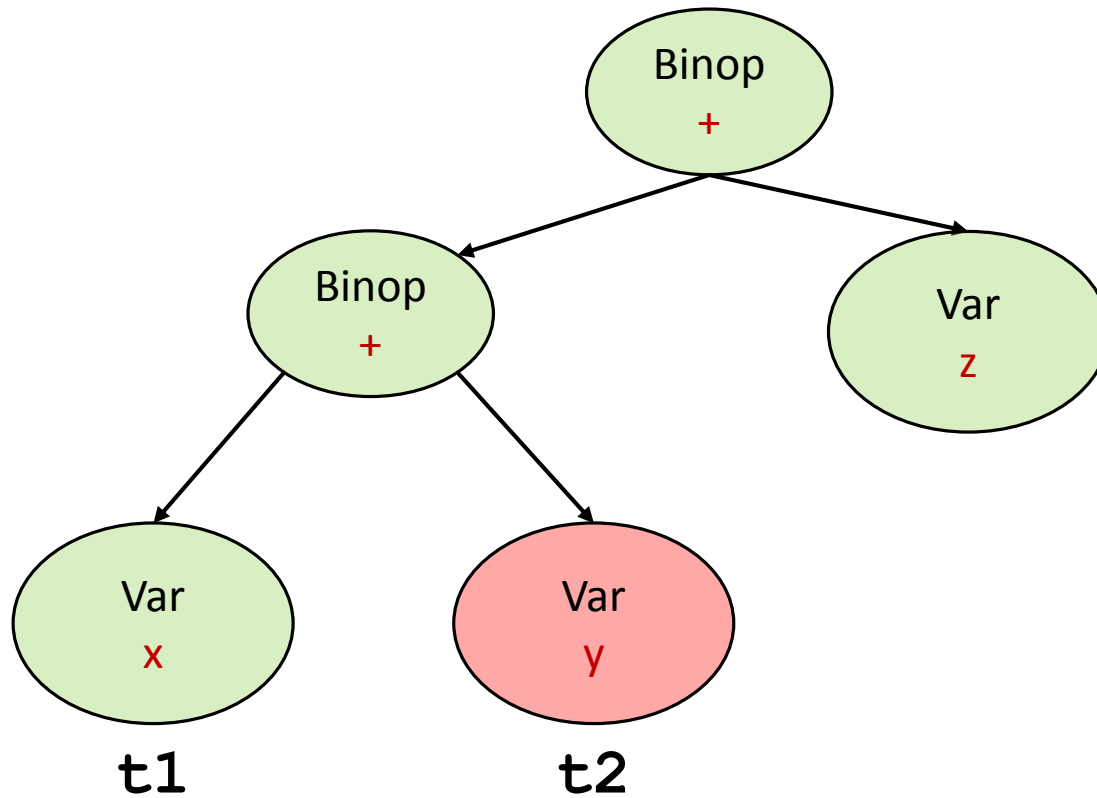
For $x + y + z$:



t1 = x

Translating Expressions

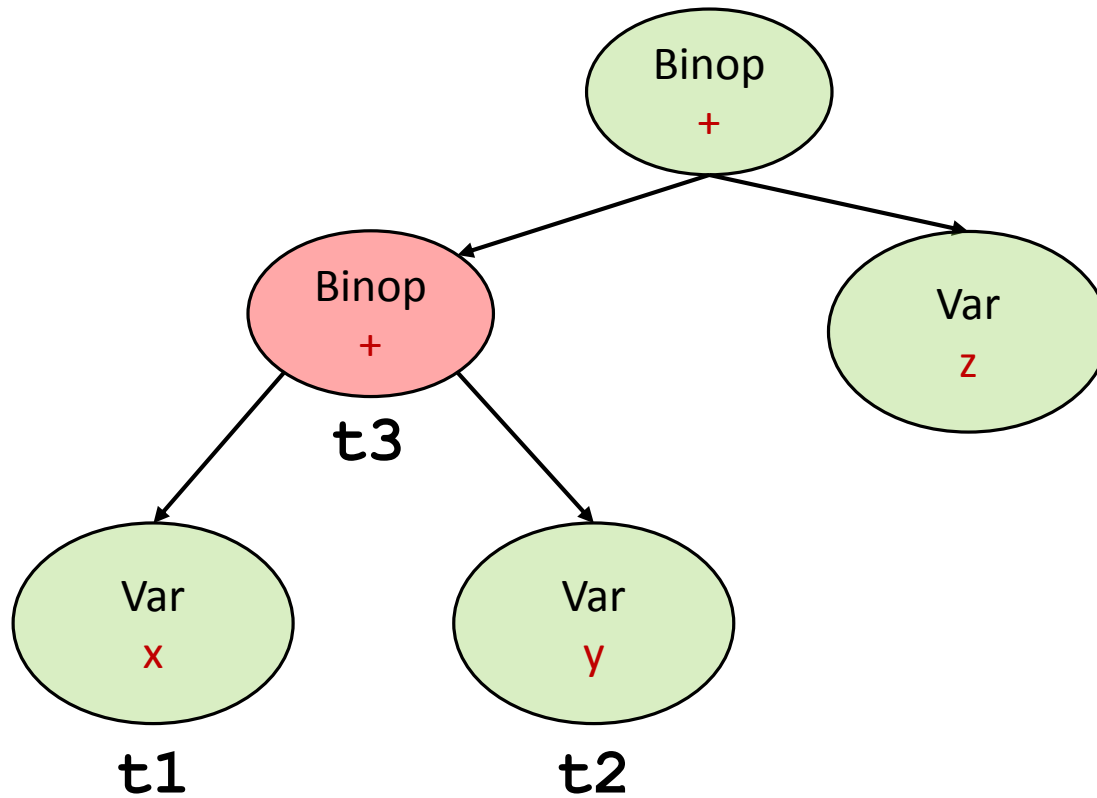
For $x + y + z$:



t1 = x
t2 = y

Translating Expressions

For $x + y + z$:



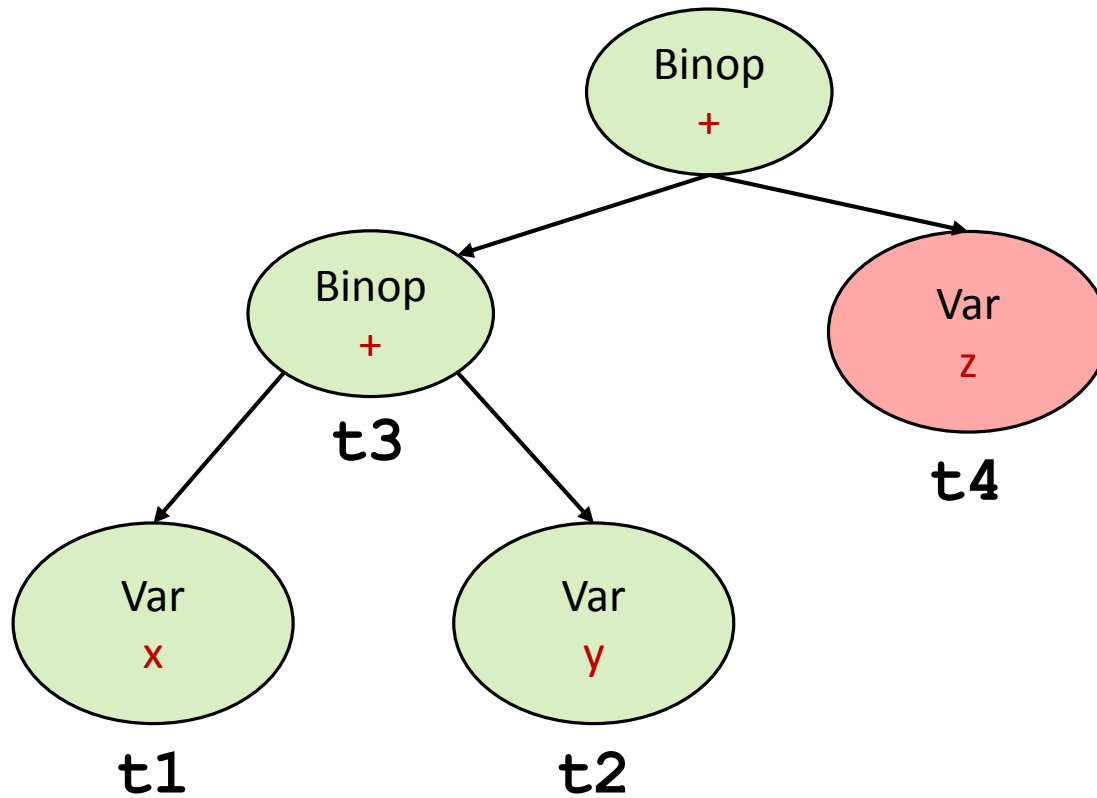
t1 = **x**

t2 = **y**

t3 = add **t1**, **t2**

Translating Expressions

For $x + y + z$:



t1 = x

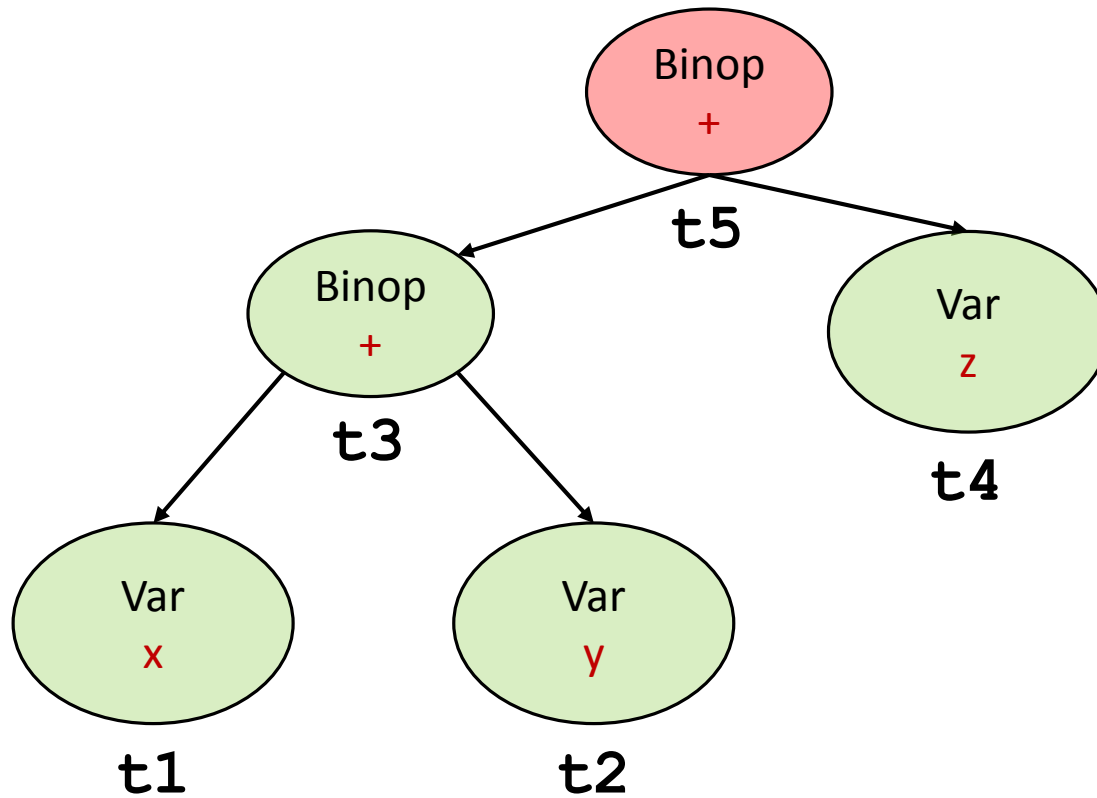
t2 = y

t3 = add t1, t2

t4 = z

Translating Expressions

For $x + y + z$:



t1 = **x**

t2 = **y**

t3 = add **t1**, **t2**

t4 = **z**

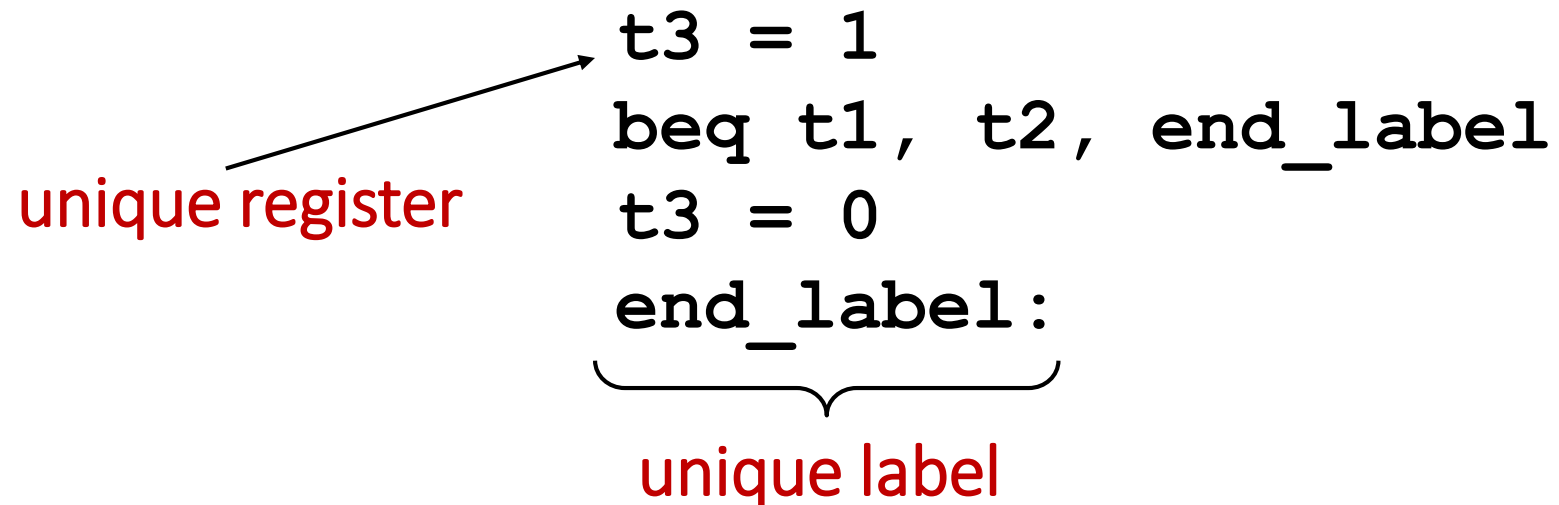
t5 = add **t3**, **t4**

Translating Expressions

For $e_1 == e_2$:

$T_c(e_1) \{ \begin{array}{l} \dots \\ t1 = \dots \end{array}$

$T_c(e_2) \{ \begin{array}{l} \dots \\ t2 = \dots \end{array}$



```
t3 = 1
beq t1, t2, end_label
t3 = 0
end_label:
```

unique register

unique label

Translating Expressions

For `a == b + 1`:

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


Translating Expressions

For e_1 and e_2 :

$T_c(e_1)$ { \dots
 $t1 = \dots$
 $t3 = 0$
 $T_r(e_1)$ $\text{beq } t1, t3, \text{end_label}$

$T_c(e_2)$ { \dots
 $t2 = \dots$
 $t3 = \text{and } t1, t2$
 $T_r(e_2)$ end_label:

Translating Expressions

For e_1 or e_2 :

$T_c(e_1)$ { \dots
 $t1 = \dots$
 $t3 = 1$
 $T_r(e_1)$ $\text{beq } t1, t3, \text{end_label}$

$T_c(e_2)$ { \dots
 $t2 = \dots$
 $t3 = \text{or } t1, t2$
 $T_r(e_2)$ end_label:

Translating Expressions

For *new type*[*e*]:

$T_c(e)$ $\{ \overset{\dots}{t1} = \dots$
 $T_r(e_1)$ \swarrow
 $t2 = \text{new_array } t1$

Translating Expressions

For `new int[k+1]`:

```
t1 = k
```

```
t2 = 1
```


```
t3 = add t1, t2
```

```
t4 = new_array t3
```

Translating Expressions

For $e_1[e_2]$:

$T_c(e_1) \{ \overset{\cdot \cdot \cdot}{t1} = \dots$

$T_r(e_1)$ 

$T_c(e_2) \{ \overset{\cdot \cdot \cdot}{t2} = \dots$

$T_r(e_2)$  $t3 = \text{array_access } t1, t2$

Translating Expressions

For `x[z+1]`:

```
t1 = x
```

```
t2 = z
```

```
t3 = 1
```

```
t4 = add t2, t3
```

```
t5 = array_access t1, t4
```

Translating Expressions

For *new type*:

```
t1 = new_class type
```

Translating Expressions

For **new Point**:

```
t1 = new_class Point
```


Translating Expressions

For $e.f$:

$$\begin{array}{l} T_c(e) \quad \{ \begin{array}{l} \dots \\ t1 = \dots \end{array} \\ T_r(e) \quad \swarrow \quad t2 = \text{field_access } t1, f \end{array}$$

Translating Expressions

For `x[3].foo`:

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

Translating Basic Block

For $s_1; s_2; \dots$:

$T_c(s_1)$

$T_c(s_2)$

...

Translating Statements

For *if* (*e*) {*s*}:

$T_c(e) \{ \begin{array}{l} \dots \\ t1 = \dots \end{array}$
 $T_r(e) \quad \text{beq } t1, 0, \text{end_label}$
 $T_c(s) \{ \begin{array}{l} \dots \\ \dots \end{array}$
 end_label:

Translating Statements

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

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

Translating Statements

For *if* (*e*) {*s*₁} *else* {*s*₂}:

$T_c(e) \{ \begin{array}{l} \dots \\ t1 = \dots \end{array}$
 \nearrow `beq t1, 0, false_label`

$T_r(e_1)$

$T_c(s_1) \{ \begin{array}{l} \dots \\ \dots \\ \text{br } \text{end_label} \\ \text{false_label:} \end{array}$

$T_c(s_2) \{ \begin{array}{l} \dots \\ \dots \\ \text{end_label:} \end{array}$

Translating Statements

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

```
t1 = w
beq t1, 0, false_label
t2 = 0
z = t2
br end_label
false_label:
t3 = 100
z = t3
end_label:
```

Translating Statements

For *while* (*e*) {*s*} :

$T_c(e)$ $\left\{ \begin{array}{l} \dots \\ t1 = \dots \\ \text{beq } t1, 0, \text{end_label} \end{array} \right.$

$T_r(e)$ \nearrow

$T_c(s)$ $\left\{ \begin{array}{l} \dots \\ \dots \\ \text{br } \text{cond_label} \\ \text{end_label:} \end{array} \right.$

cond_label:

Translating Statements

For `while (z / x) { }`:

```
cond_label:
t1 = z
t2 = x
t3 = div t1, t2
beq t3, 0, end_label
br cond_label
end_label:
```

Translating Statements

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

$T_c(e_1) \{ \begin{array}{l} \dots \\ \text{t1} = \dots \end{array}$

$T_c(e_2) \{ \begin{array}{l} \dots \\ \text{t2} = \dots \end{array}$

\dots
call $f, \text{t1}, \text{t2}, \dots$

Translating Statements

For `func(2, x + 1)`:

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

Translating Statements

For *return e*:

$$T_c(e) \{ \begin{array}{l} \dots \\ t1 = \dots \\ \text{return } t1 \end{array}$$

Translating Statements

For `return w * 3`:

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

Translating Statements

For $e_1[e_2] = e_3$:

$$T_c(e_1) \{ \ddots$$

$$T_c(e_2) \{ \ddots = \dots$$

$$T_c(e_3) \{ \ddots = \dots$$

array_set t1, t2, t3

Translating Statements

For `arr[0] = x+1`:

```
t1 = arr
t2 = 0
t3 = x
t4 = 1
t5 = add t3, t4
field_set t1, t2, t5
```

Translating Statements

For $o.f = e$:

$$T_c(o) \{ \dots$$

$$T_c(e) \{ t2 = \dots$$

field_set t1, f, t3

Translating Statements

For `obj.flag = 7`:

```
t1 = obj  
t2 = 7  
field_set t1, flag, t2
```

Translating Statements

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

$$T_c(o) \{ \begin{array}{l} \dots \\ t1 \end{array}$$

$$T_c(e_1) \{ \begin{array}{l} \dots \\ t2 = \dots \end{array}$$

$$T_c(e_2) \{ \begin{array}{l} \dots \\ t3 = \dots \end{array}$$

\dots
virtual_call t1 f t2, t3, \dots

Translating Statements

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

```
t1 = obj
t2 = 2
t3 = x
t4 = 1
t5 = add t3, t4
virtual_call t1, func, t2, t5
```

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;  
}
```

```
 $T_c(\mathbf{x} = 42)$   
 $T_c(\mathbf{while} \ (\mathbf{x} > 0) \ \{$   
     $\mathbf{x} = \mathbf{x} - 1;$   
     $\}$   
 $)$ 
```

Example

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

```
t1 = 42  
x = t1  
Tc(  
    while (x > 0) {  
        x = x - 1;  
    }  
)
```

Example

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

```
t1 = 42  
x = t1  
cond_label:  
Tc(x > 0)  
beq Tr(x > 0), 0, end_label  
Tc(x = x - 1)  
br 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  
bgt t2, t3, cmp_label:  
t4 = 1  
cmp_label:  
beq  $T_r(\mathbf{x} > 0)$ , 0, end_label  
 $T_c(\mathbf{x} = \mathbf{x} - 1)$   
br 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  
bgt t2, t3, cmp_label:  
t4 = 1  
cmp_label:  
beq t4, 0, end_label  
 $T_c(x = x - 1)$   
br 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  
bgt t2, t3, cmp_label:  
t4 = 1  
cmp_label:  
beq t4, 0, end_label  
t5 = x  
t6 = 1  
t7 = sub t5, t6  
x = t7  
br cond_label  
end_label:
```

Implementation

- Classes for IR Instructions
- AST Visitor
 - Define visitor for each node type
 - Returns
 - List of generated instructions
 - Result register (for expression nodes)