

# 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:
  - $TR_r(e) = t_{new}$
  - $TR_c(e) = [t_{new} = e]$
- For internal node:
  - $TR_r(e_1 \text{ op } e_2) = t_{new}$
  - $TR_c(e_1 \text{ op } e_2) = TR(e_1); TR(e_2); t_{new} = \text{op } TR_r(e_1), TR_r(e_2)$
- $TR(e) = (TR_r(e), TR_c(e))$

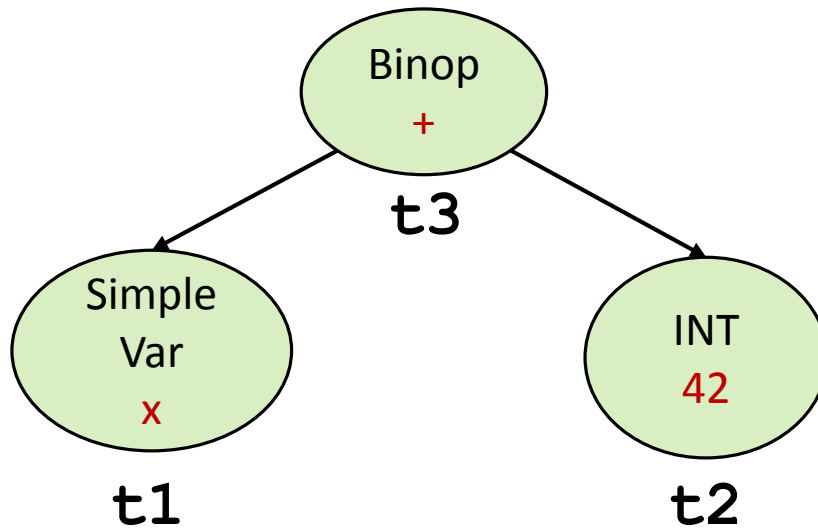
# Translating Expressions

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  $x + 42$ :



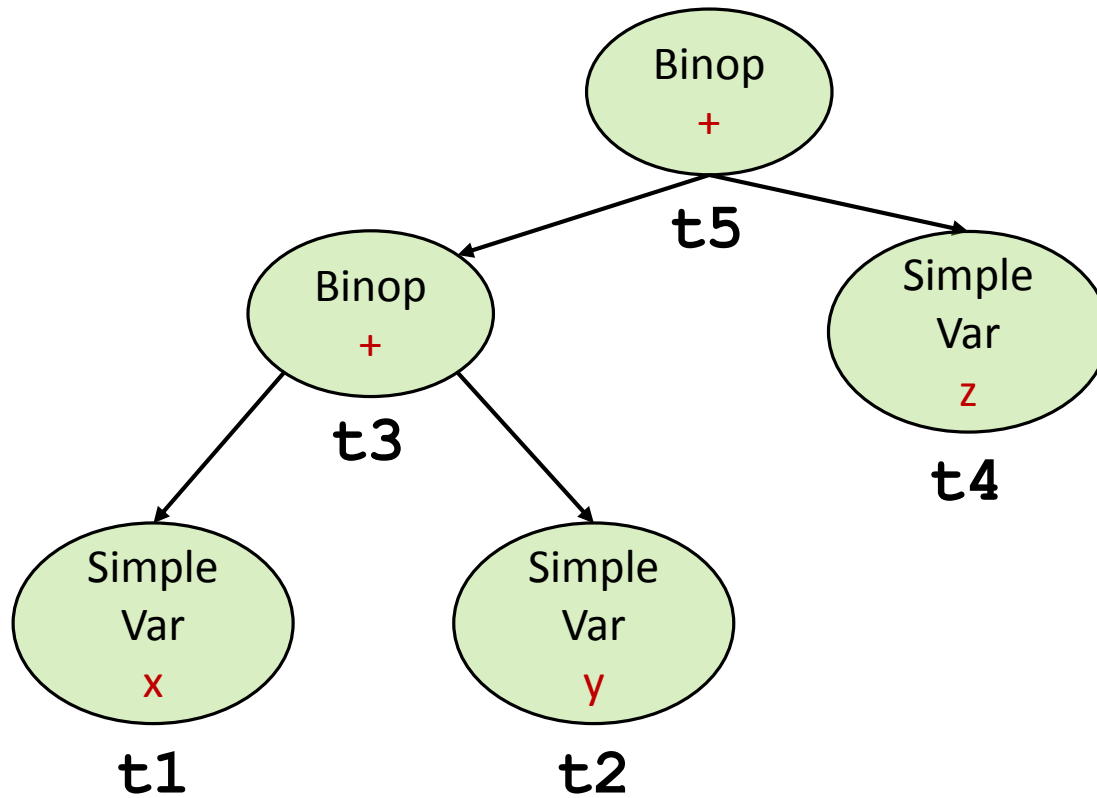
**t1 = x**

**t2 = 42**

**t3 = add t1, t2**

# Translating Expressions

For  $x + y + z$ :



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



# Translating Expressions

For *op e*:

$$\begin{array}{l} T_c(e) \\ T_r(e) \end{array} \left\{ \begin{array}{l} \dots \\ t1 = \dots \\ t2 = \text{op } t1 \end{array} \right.$$

# Translating Expressions

For  $e_1$  or  $e_2$ :

$T_c(e_1)$	{	...
		t1 = ...
		t3 = 1
$T_r(e_1)$		compare t1, t3
		branch_eq end_label
$T_c(e_2)$	{	...
		t2 = ...
		t3 = or t1, t2
$T_r(e_2)$		end_label:

# Translating Expressions


For  $e_1$  and  $e_2$ :

$T_c(e_1)$	{	...
		t1 = ...
		t3 = 0
$T_r(e_1)$		compare t1, t3
		branch_eq end_label
$T_c(e_2)$	{	...
		t2 = ...
		t3 = <b>and</b> t1, t2
$T_r(e_2)$		end_label:

# 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  $e.f$  :

$$\begin{array}{l} T_c(e) \\ T_r(e) \end{array} \left\{ \begin{array}{l} \dots \\ t1 = \dots \\ t2 = \text{field\_access } t1, f \end{array} \right.$$

# Translating Basic Block

For  $s_1; s_2; \dots$ :

$T_c(s_1)$

$T_c(s_2)$

...

# Translating Statements

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

$T_c(e)$  {  $\dots$   
t1 =  $\dots$   
compare t1, 0  
 $T_r(e)$  branch\_eq end\_label  
 $T_c(s)$  {  $\dots$   
 $\dots$   
end\_label:

# Translating Statements

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

$T_c(e)$  {  $\dots$   
t1 =  $\dots$   
           $\nearrow$  compare t1, 0  
              branch\_eq end\_label  
 $T_r(e_1)$   
 $T_c(s_1)$  {  $\dots$   
               $\dots$   
              branch end\_label  
              false\_label:  
 $T_c(s_2)$  {  $\dots$   
               $\dots$   
              end\_label:



# Translating Statements

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

```
cond_label:
    ...
    t1 = ...
    compare t1, 0
    branch_eq end_label
Tc(e) {
Tr(e) ↗
Tc(s) {
    ...
    branch cond_label
    end_label:
```

# Translating Function Call

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

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

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

$\dots$   
**call**  $f(t1, t2, \dots)$

# Translating Return

For *return e*:

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

# 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;  
}
```

```
 $T_c(\mathbf{x} = 42)$   
 $T_c($   
    while (x > 0) {  
        x = 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:  
t2 = x  
compare t2, 0  
branch_le 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  
compare t2, 0  
branch_le end_label  
t4 = x  
t5 = 1  
t6 = sub t4, t5  
x = t6  
branch cond_label  
end_label
```