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

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

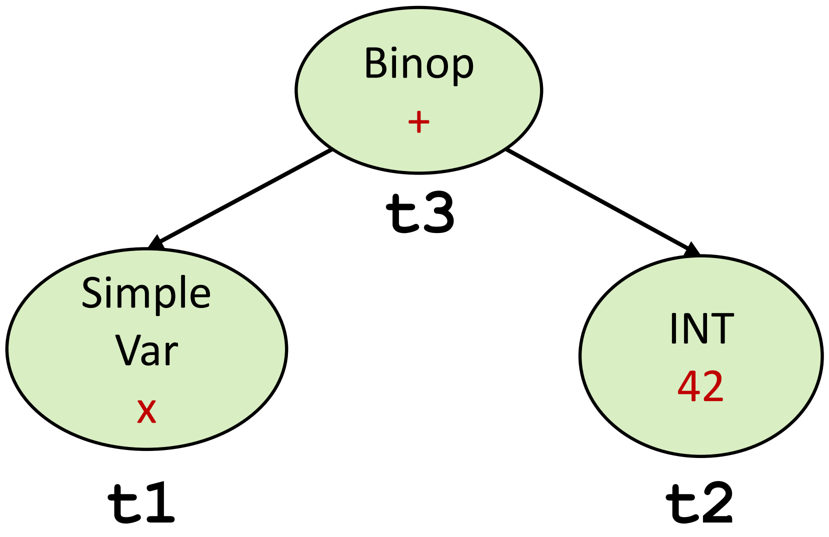
Translating Expressions

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

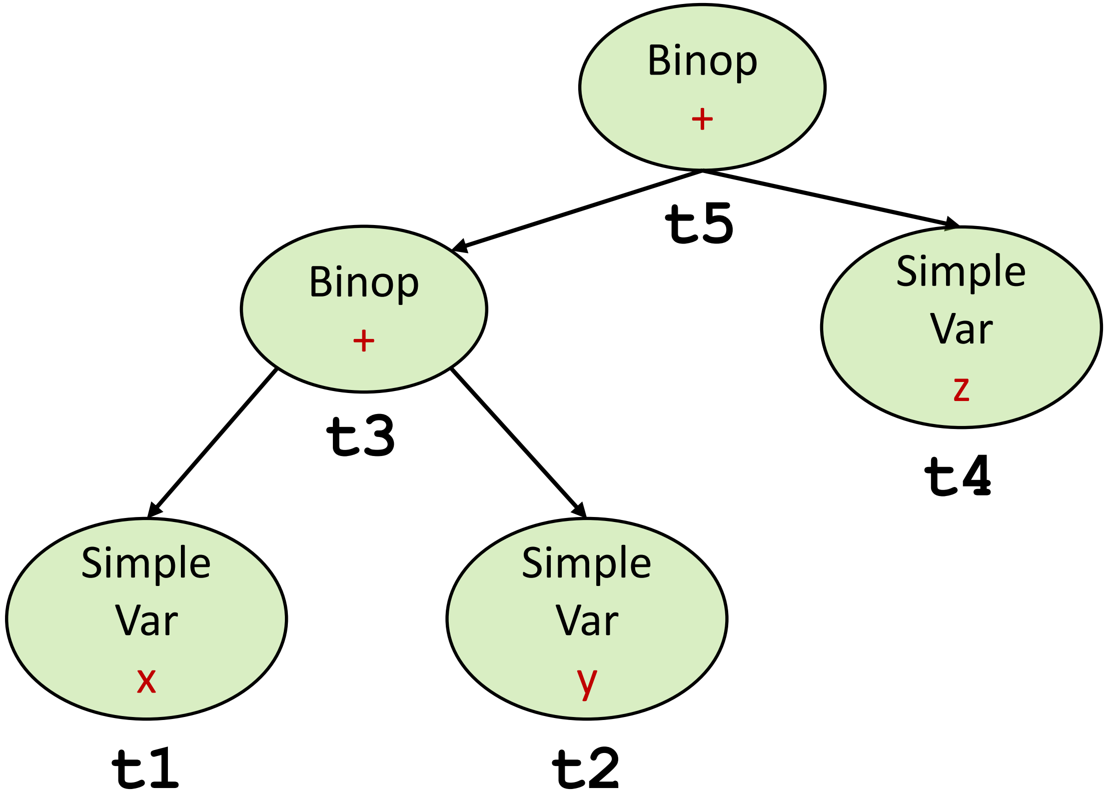
For x + 42:




```
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 $op\ e$:

$T_c(e)$

```
...
t1 = ...
t2 = op t1
```

$T_r(e)$

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

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

$T_r(e)$

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

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

$T_c(s_1)$
```
...
...
```

```
branch end_label
false_label:
```

$T_c(s_2)$
```
...
...
```

```
end_label:
```

Translating Statements

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

$T_c(e)$, $T_r(e)$, $T_c(s)$

```
cond_label:
...
t1 = ...
compare t1, 0
branch_eq end_label
...
...
branch cond_label
end_label:
```

Translating Statements

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

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

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

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

Translating Statements

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

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

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

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

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

Translating Statements

For $return\ e$:

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

```
return t1
```

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:
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
```

Alternative Representation

For z = x + 42 the generated code is:

```
t1 = x
t2 = 42
t3 = add t1, t2z
z = t3
```

Alternative Representation

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

```
t1 = load (bp + 8)
t2 = 42
t3 = add t1, t2
store (bp - 4), t3
```