Three-Address Code

- Or "**TAC**"
- The IR that you will be using for the final programming project.
- High-level assembly where each operation has at most three operands.
- Uses explicit runtime stack for function calls.
- Uses vtables for dynamic dispatch.

Sample TAC Code

```
int x;
int y;

int x2 = x * x;
int y2 = y * y;
int r2 = x2 + y2;
```

Sample TAC Code

```
int x;
int y;

int x2 = x * x;
int y2 = y * y;
int r2 = x2 + y2;
```

```
x2 = x * x;
y2 = y * y;
r2 = x2 + y2;
```

Sample TAC Code

```
int a;
int b;
int c;
int d;

a = b + c + d;
b = a * a + b * b;
```

Sample TAC Code

```
int a;
int b;
int c;
int d;

a = b + c + d;
b = a * a + b * b;
```

```
_t0 = b + c;
a = _t0 + d;
_t1 = a * a;
_t2 = b * b;
b = _t1 + _t2;
```

Sample TAC Code

```
int a;
int b;
int c;
int d;

a = b + c + d;
b = a * a + b * b;
```

```
_t0 = b + c;
a = _t0 + d;
_t1 = a * a;
_t2 = b * b;
b = _t1 + _t2;
```

Temporary Variables

- The “three” in “three-address code” refers to the number of operands in any instruction.
- Evaluating an expression with more than three subexpressions requires the introduction of temporary variables.
- This is actually a lot easier than you might think; we'll see how to do it later on.

Sample TAC Code

```
int a;
int b;

a = 5 + 2 * b;
```

Sample TAC Code

```
int a;
int b;

a = 5 + 2 * b;
```

```
_t0 = 5;
_t1 = 2 * b;
a = _t0 + _t1;
```

Sample TAC Code

```
int a;
int b;

a = 5 + 2 * b;
```

TAC allows for instructions with two operands.

```
_t0 = 5;
_t1 = 2 * b;
a = _t0 + _t1;
```

Simple TAC Instructions

- **Variable assignment** allows assignments of the form
  - `var = constant;`
  - var_1 = var_2;
  - var_1 = var_2 **op** var_3;
  - var_1 = constant **op** var_2;
  - var_1 = var_2 **op** constant;
  - var = $constant_1$ **op** $constant_2$;
- Permitted operators are **+**, **-**, *****, **/**, **%**.
- How would you compile **y = -x**; ?

Simple TAC Instructions

- **Variable assignment** allows assignments of the form
  - `var = constant;`
  - var_1 = var_2;
  - var_1 = var_2 **op** var_3;
  - var_1 = constant **op** var_2;
  - var_1 = var_2 **op** constant;
  - var = $constant_1$ **op** $constant_2$;
- Permitted operators are **+**, **-**, *****, **/**, **%**.
- How would you compile **y = -x;** ?

```
y = 0 - x;          y = -1 * x;
```

One More with `bool`s

```
int x;
int y;
bool b1;
bool b2;
bool b3;

b1 = x + x < y
b2 = x + x == y
b3 = x + x > y
```

One More with `bool`s

```
int x;
int y;
bool b1;
bool b2;
bool b3;

b1 = x + x < y
b2 = x + x == y
b3 = x + x > y
```

```
_t0 = x + x;
_t1 = y;
b1 = _t0 < _t1;

_t2 = x + x;
_t3 = y;
b2 = _t2 == _t3;

_t4 = x + x;
_t5 = y;
b3 = _t5 < _t4;
```

TAC with `bool`s

- Boolean variables are represented as integers that have zero or nonzero values.
- In addition to the arithmetic operator, TAC supports `<`, `==`, `||`, and `&&`.
- How might you compile `b = (x <= y)` ?

TAC with `bool`s

- Boolean variables are represented as integers that have zero or nonzero values.
- In addition to the arithmetic operator, TAC supports `<`, `==`, `||`, and `&&`.
- How might you compile `b = (x <= y)` ?

```
_t0 = x < y;
_t1 = x == y;
b = _t0 || _t1;
```

Control Flow Statements

```
int x;
int y;
int z;

if (x < y)
   z = x;
else
   z = y;

z = z * z;
```

Control Flow Statements

```
int x;
int y;
int z;

if (x < y)
    z = x;
else
    z = y;

z = z * z;
```

```
    _t0 = x < y;
    IfZ _t0 Goto _L0;
    z = x;
    Goto _L1;
_L0:
    z = y;
_L1:
    z = z * z;
```

Control Flow Statements

```
int x;
int y;
int z;

if (x < y)
    z = x;
else
    z = y;

z = z * z;
```

```
    _t0 = x < y;
    IfZ _t0 Goto _L0;
    z = x;
    Goto _L1;
_L0:
    z = y;
_L1:
    z = z * z;
```

Control Flow Statements

```
int x;
int y;
int z;

if (x < y)
    z = x;
else
    z = y;

z = z * z;
```

```
    _t0 = x < y;
    IfZ _t0 Goto _L0;
    z = x;
    Goto _L1;
_L0:
    z = y;
_L1:
    z = z * z;
```

Labels

- TAC allows for **named labels** indicating particular points in the code that can be jumped to.
- There are two control flow instructions:
  - **`Goto` *`label`*`;`**
  - **`IfZ` *`value`* `Goto` *`label`*`;`**
- Note that **`IfZ`** is always paired with **`Goto`**.

Control Flow Statements

```
int x;
int y;

while (x < y) {
   x = x * 2;
}

y = x;
```

Control Flow Statements

```
int x;
int y;

while (x < y) {
   x = x * 2;
}

y = x;
```

```
_L0:
     _t0 = x < y;
     IfZ _t0 Goto _L1;
     x = x * 2;
     Goto _L0;
_L1:
     y = x;
```