Compiler Construction

Lecture 10 - Optimization

What Is Optimization?

- The process of automated translation of a program will invariably introduced ***inefficiencies***. Our goal in ***optimization*** is to remove as many of these inefficiencies as possible.
- Optimization can be ***local*** (optimizing basic blocks within a program) or ***global*** (across the entire program).
- Even after optimizing intermediate code, it may be necessary to optimize the final object code because of *inefficiencies introduced in final code generation.*

A Sample Program in JASON

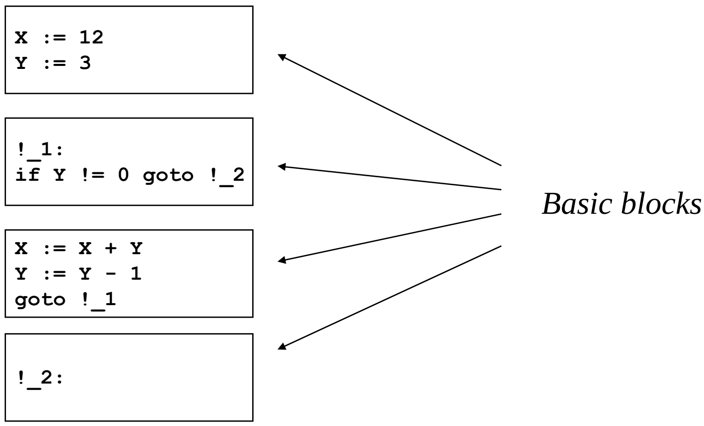
```
PROGRAM MySample;
INTEGER x, y;
BEGIN
  SET x = 12;
  SET y = 3;

  WHILE y ! 0 DO
    SET x = x + y;
    SET y = y - 1
  ENDWHILE;
END.
```

```
X := 12
Y := 3
!_1:
if y = 0 goto !_2
X := X + Y
Y := Y - 1
goto !_1
!_2:
```

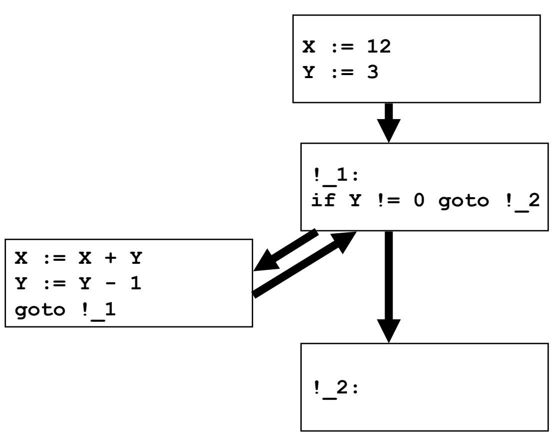
Basic Blocks

- A ***basic block*** is a sequence of instruction that will be performed in sequence, always going from the beginning of the block to the end of the block without jumping out of the block.
- There may be more than one basic block that transfers control to a given block and there may be more than one basic block to which we will transfer control as we leave a given block.

The Basic Blocks Of Our Sample Program



Flow Graphs



Principle Optimizations On Basic Blocks

- There are several different optimizations that we can (and will) perform on basic blocks. They include:
  - Common Sub-expression Elimination
  - Copy propagation
  - Dead-Code Elimination
  - Arithmetic Transformation

Common Subexpression Elimination

```
b := 4-2                       b := 4-2
$_1 := b / 2                   $_1 := b / 2
$_2 := a* $_1                  $_2 := a* $_1
$_3 := $_2 * b                 $_3 := $_2 * b
$_4 := $_3 + c                 $_4 := $_3 + c
$_5 := $_2 * b      =>         $_5 := $_3
$_6 := $_5 + c                 $_6 := $_5 + c
d := $_4 * $_6                 d := $_4 * $_6
```

Common Subexpression Elimination

```
b := 4-2
$_1 := b / 2
$_2 := a* $_1
$_3 := $_2 * b
b := $_3 + c
$_5 := $_2 * b
$_6 := $_5 + c
d := $_4 * $_6
```

We cannot use subexpression elimination here because `b`*'s value was changed*

Copy Propagation

```
b := 4-2
$_1 := b / 2
$_2 := a* $_1
$_3 := $_2 * b
$_4 := $_3 + c
$_5 := $_3
$_6 := $_5 + c
d := $_4 * $_6
```

⇒

```
b := 4-2
$_1 := b / 2
$_2 := a* $_1
$_3 := $_2 * b
$_4 := $_3 + c
$_5 := $_3
$_6 := $_3 + c
d := $_4 * $_6
```

Subexpression After Copy Propagation

```
b := 4-2
$_1 := b / 2
$_2 := a* $_1
$_3 := $_2 * b
$_4 := $_3 + c
$_5 := $_3
$_6 := $_3 + c
d := $_4 * $_6
```

```
b := 4-2
$_1 := b / 2
$_2 := a* $_1
$_3 := $_2 * b
$_4 := $_3 + c
$_5 := $_3
$_6 := $_4
d := $_4 * $_6
```

Copy Propagation After Subexpression

```
b := 4-2
$_1 := b / 2
$_2 := a* $_1
$_3 := $_2 * b
$_4 := $_3 + c
$_5 := $_3
$_6 := $_4
d := $_4 * $_6
```

```
b := 4-2
$_1 := b / 2
$_2 := a* $_1
$_3 := $_2 * b
$_4 := $_3 + c
$_5 := $_3
$_6 := $_4
d := $_4 * $_4
```

Dead-Code Elimination

```
b := 4-2
$_1 := b / 2
$_2 := a* $_1
$_3 := $_2 * b
$_4 := $_3 + c
$_5  := $_3
$_6 := $_4
d := $_4 * $_4
```

```
b := 4-2
$_1 := b / 2
$_2 := a* $_1
$_3 := $_2 * b
$_4 := $_3 + c

$_6 := $_4
d := $_4 * $_4
```

No references to $_5 after defining its value

Arithmetic Transformations

- We can use the laws of algebra to replace expressions that either do not need to be calculated or can be calculated more quickly by other means.
- These algebraic transformations include:
  - Constant Folding
  - Algebraic Simplification
  - Reduction In Strength

Constant Folding

```
b := 4-2
$_1 := b / 2
$_2 := a* $_1
$_3 := $_2 * b
$_4 := $_3 + c
$_6 := $_4
d := $_4 * $_4
```

```
b := 2
$_1 := b / 2
$_2 := a* $_1
$_3 := $_2 * b
$_4 := $_3 + c
d := $_4 * $_4
```

Copy Propagation & Dead-Code Elimination After Constant Folding

```
b := 2
$_1 := b / 2
$_2 := a* $_1
$_3 := $_2 * b
$_4 := $_3 + c
d := $_4 * $_4
```

```
$_1 := 2 / 2
$_2 := a* $_1
$_3 := $_2 * 2
$_4 := $_3 + c
d := $_4 * $_4
```

More Constant Folding

```
$_1 := 2 / 2
$_2 := a* $_1
$_3 := $_2 * 2
$_4 := $_3 + c
d := $_4 * $_4
```

```
$_1 := 1
$_2 := a* $_1
$_3 := $_2 * 2
$_4 := $_3 + c
d := $_4 * $_4
```

More Copy Propagation & Dead-Code Elimination

```
$_1 := 1
$_2 := a* $_1
$_3 := $_2 * 2
$_4 := $_3 + c
$_6 := $_4
d := $_4 * $_4
```

```
$_2 := a * 1
$_3 := $_2 * 2
$_4 := $_3 + c
d := $_4 * $_4
```

Algebraic Simplification

- We can simplify our expressions by using algebraic identities:

  $x + 0 = 0 + x = x$

  $x - 0 = x$

  $x \bullet 1 = 1 \bullet x = x$

  $x / 1 = x$

Applying Algebraic Simplification

```
$_2 := a * 1          $_2 := a
$_3 := $_2 * 2        $_3 := $_2 * 2
$_4 := $_3 + c        $_4 := $_3 + c
d := $_4 * $_4        d := $_4 * $_4
```

After Copy Propagation & Dead-Code Elimination

```
$_2 := a
$_3 := $_2 * 2
$_4 := $_3 + c
d := $_4 * $_4
```

```
$_3 := a * 2
$_4 := $_3 + c
d := $_4 * $_4
```


After Copy Propagation & Dead-Code Elimination

```
$_2 := a
$_3 := $_2 * 2
$_4 := $_3 + c
d := $_4 * $_4
```

```
$_3 := a * 2
$_4 := $_3 + c
d := $_4 * $_4
```

Reduction In Strength

- We can replace multiplication and division (or exponentiation) with addition and subtraction (or multiplication) which can usually be done much more quickly.
- We can use the identities:

  $x^2 = x \bullet x$

  $2 \bullet x = x + x$
- We can also use shifts to replace multiplication and division by powers of 2

Applying Reduction In Strength

```
$_3 := a * 2          ⇒   $_3 := a + a
$_4 := $_3 + c            $_4 := $_3 + c
d := $_4 * $_4            d := $_4 * $_4
```

Our End Result

```
b := 4-2
$_1 := b / 2
$_2 := a* $_1
$_3 := $_2 * b
$_4 := $_3 + c
$_5 := $_2 * b
$_6 := $_5 + c
d := $_4 * $_6
```

⇒

```
$_3 := a + a
$_4 := $_3 + c
d := $_4 * $_4
```