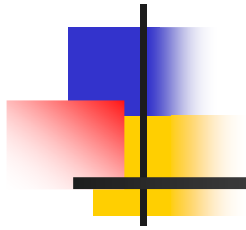


# Basic Blocks and Flow Graphs





## Example of Basic Block

---

**Following sequence of three-address statements forms a basic block:**

```
t1 := a * a
t2 := a * b
t3 := 2 * t2
t4 := t1 + t3
t5 := b * b
t6 := t4 + t5
```

It computes:  $a*a + 2*a*b + b*b$

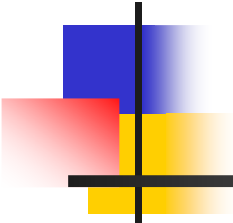
# Partition Algorithm for Basic Blocks

*Input:* A sequence of three-address statements

*Output:* A list of basic blocks with each three-address statement in exactly one block

1. Determine the set of *leaders*, the first statements of basic blocks
  - a) The first statement is the leader
  - b) Any statement that is the target of a goto is a leader
  - c) Any statement that immediately follows a goto is a leader
2. For each leader, its basic block consists of the leader and all statements up to but not including the next leader or the end of the program

## Partition into Basic Blocks



```
(1)  prod := 0
(2)  i := 1
(3)  t1 := 4 * i
(4)  t2 := a[t1]
(5)  t3 := 4 * i
(6)  t4 := b[t3]
(7)  t5 := t2 * t4
(8)  t6 := prod + t5
(9)  prod := t6
(10) t7 := i+1
(11) i := t7
(12) if( i <=20) goto (3)
```

# Partition into Basic Blocks



(1)  $\text{prod} := 0$

(2)  $i := 1$

**B1**

A leader by rule 1.a

A block by rule 2

(3)  $t1 := 4 * i$

(4)  $t2 := a[t1]$

(5)  $t3 := 4 * i$

(6)  $t4 := b[t3]$

(7)  $t5 := t2 * t4$

(8)  $t6 := \text{prod} + t5$

(9)  $\text{prod} := t6$

(10)  $t7 := i + 1$

(11)  $i := t7$

(12)  $\text{if}(i \leq 20) \text{goto}(3)$

**B2**

A leader by rule 1.b

A block by rule 2

A leader by rule 1.c

# Transformations on Basic Blocks

- A *code-improving transformation* is a code optimization to improve speed or reduce code size
- *Global transformations* are performed across basic blocks
- *Local transformations* are only performed on single basic blocks
- Transformations must be safe and preserve the meaning of the code
  - A local transformation is safe if the transformed basic block is guaranteed to be equivalent to its original form



# Local Transformations

---

## 2 Types

### 1. Structure Preserving Transformation

- a. Common Sub expression Elimination
- b. Dead code Elimination
- c. Renaming Temporary Variables
- d. Interchange of Statements

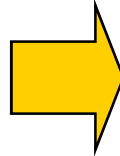
### 2. Algebraic Transformation



# Common-Sub Expression Elimination

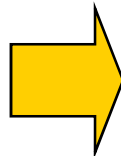
---

```
a := b + c  
b := a - d  
c := b + c  
d := a - d
```



```
a := b + c  
b := a - d  
c := b + c  
d := b
```

```
t1 := b * c  
t2 := a - t1  
t3 := b * c  
t4 := t2 + t3
```

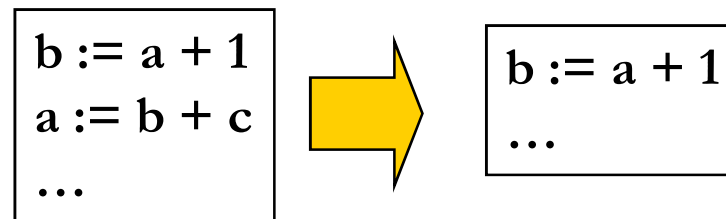


```
t1 := b * c  
t2 := a - t1  
t4 := t2 + t1
```

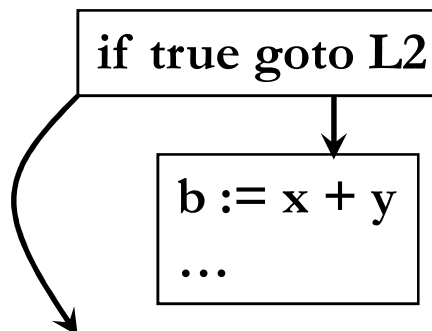


# Dead Code Elimination

- Remove unused statements



Assuming **a** is *dead* (not used)

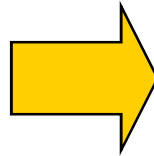


Remove unreachable code

# Renaming Temporary Variables

- Temporary variables that are dead at the end of a block can be safely renamed

```
t1 := b + c  
t2 := a - t1  
t1 := t1 * d  
d := t2 + t1
```



```
t1 := b + c  
t2 := a - t1  
t3 := t1 * d  
d := t2 + t3
```

Normal-form block

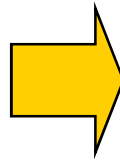


# Interchange of Statements

---

- Independent statements can be reordered

```
t1 := b + c  
t2 := a - t1  
t3 := t1 * d  
d := t2 + t3
```



```
t1 := b + c  
t3 := t1 * d  
t2 := a - t1  
d := t2 + t3
```

Note that normal-form blocks permit all statement interchanges that are possible

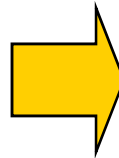


# Algebraic Transformations

---

- Change arithmetic operations to transform blocks to algebraic equivalent forms

```
t1 := a - a  
t2 := b + t1  
t3 := 2 * t2
```



```
t1 := 0  
t2 := b  
t3 := t2 << 1
```



## Algebraic Transformations Cont...

---

- Eliminate  **$x := x + 0$  or  $x := x * 1$**
- Modify  **$x := y ** 2 \rightarrow x := y * y$**



# Flow Graphs

---

- Flow of control information is added to the set of blocks making up a program by constructing a directed graph called a Flow graph.
- Nodes are the basic blocks.

There is a directed edge from B1 to B2 ie B2 can follow B1 if

  1. There is a conditional or unconditional jump from the last statement of B1 to the first statement of B2 or
  2. B2 immediately follows B1 in the order of the program, and B1 does not end in an unconditional jump.

# Representation of Flow Graphs

