Basic Blocks and Control Flow Graphs
Basic Blocks

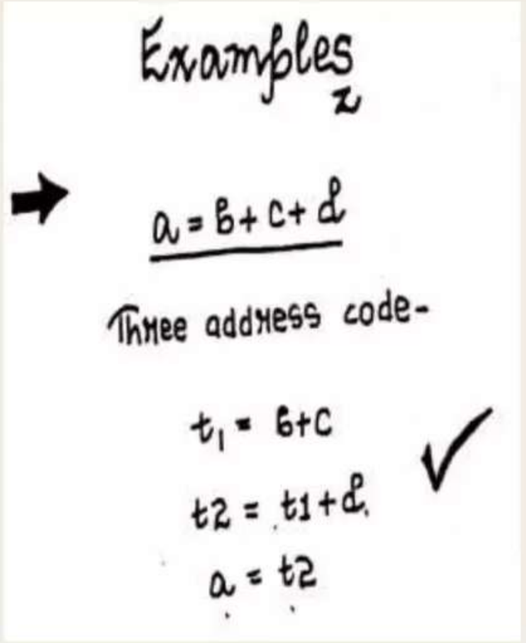
Introduction:

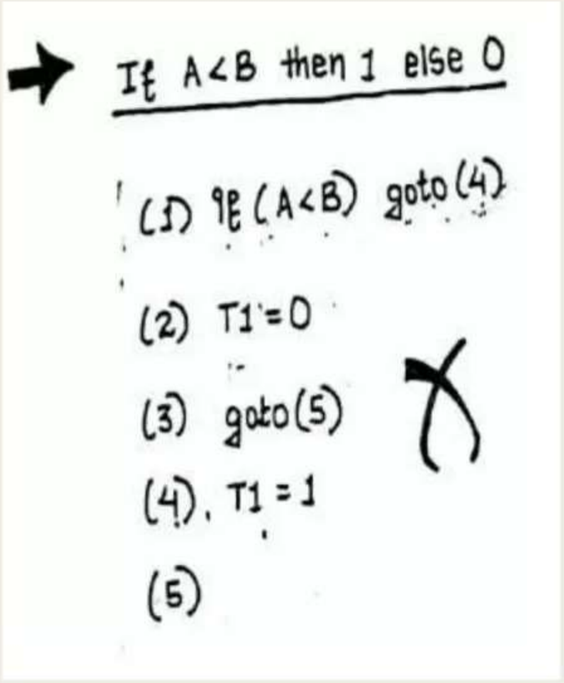
- A *basic block* is a sequence of consecutive instructions which are always executed in sequence without halt or possibilty of branching.
- The basic block doesnot have any jump statements among them.

Basic Blocks and Control Flow Graphs

Basic Blocks

- When the first instruction is executed , all the instructions in the same basic block will be executed in their sequence of appearance without losing the flow control from the program.





Basic Blocks Construction Algorithm

- *Input*: A sequence of three-address statements.

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

Basic Blocks Construction Algorithm

1. **Determine the set of *leaders*,**

    a. The first statement is the leader.

    b. Any statement that is the target of a conditional or goto is a leader.

    c. Any statement that immediately follows conditional or goto is a leader.

2. For each leader, its basic block consist of the leader and all statements up to but not including the next leader or the end of the program

Basic Blocks Construction Example

Consider the following three address code statements .Compute the basic blocks.

1) PROD = 0
2) I = 1
3) T2=addr(A)-4
4) T4 =addr(B)-4
5) T1 =4*I
6) T3= T2[T1]
7) T5 = T4[T1]
8) T6 =T3*T5
9) PROD =PROD + T6
10) I =I+1
11) IF I<= 20 GOTO (5)

Basic Blocks Construction Example

Solution:

- Because first statement is a leader ,so –

  PROD =0 is a leader.

- Because the target statement of conditional or unconditional goto statement is a leader, so-

  T1 =4*I is also a leader

Basic Blocks Construction Example

So the given code can be portioned in to 2 blocks as :

B1:

```
PROD = 0
I = 1
T2=addr(A)-4
T4 =addr(B)-4
```

B2:

```
T1 =4*I
T3= T2[T1]
T5 = T4[T1]
T6 =T3*T5
PROD =PROD + T6
I =I+1
IF I<= 20 GOTO B2
```

Basic Blocks and Control Flow Graphs
Flow Graphs

Introduction:

- A *flow graph* is a graphical representation of a sequence of instructions with control flow edges.
- A flow graph can be defined at the intermediate code level or target code level.
- The nodes of flow graphs are the basic blocks and flow-of-control to immediately follow node connected by directed arrow.

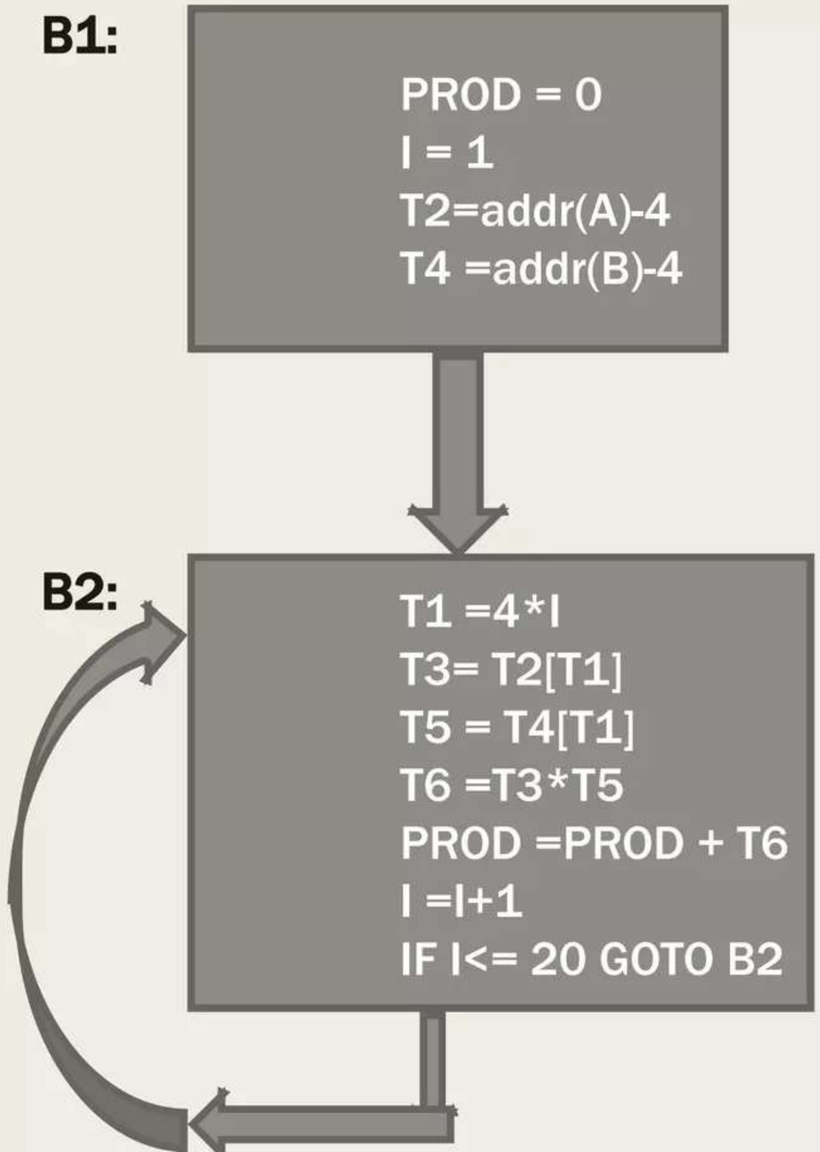
Flow Graphs

Points to remember:

- The basic blocks are the nodes to the flow graph .
- The block whose leader is the first statement is called initial block.
- There is a directed edge from block B1 to B2 if B2 immediately follows B1 in the given sequence
- Then we say that B1 is the predecessor of B2.

Flow Graphs

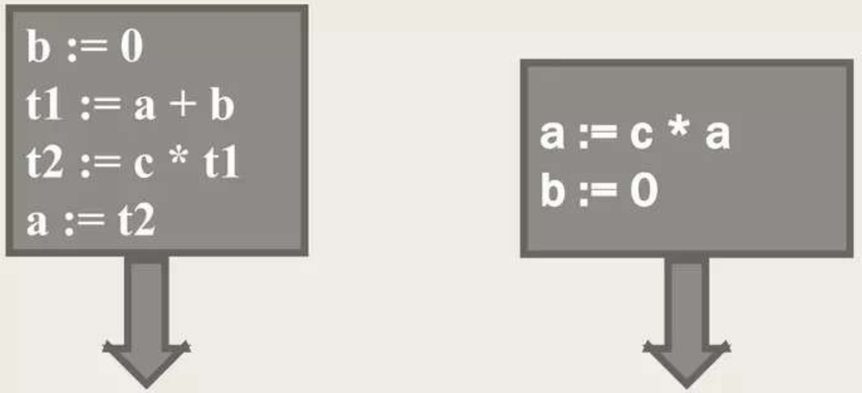
The flow graph for the above three address code is given below:



Equivalence of Basic Blocks

- Two basic blocks are (semantically) *equivalent* if they compute the same set of expressions

Transformation:



a := c*a
b := 0

a := c*a
b := 0

- Blocks are equivalent, assuming **t1** and **t2** are *dead*: no longer used (no longer *live*)

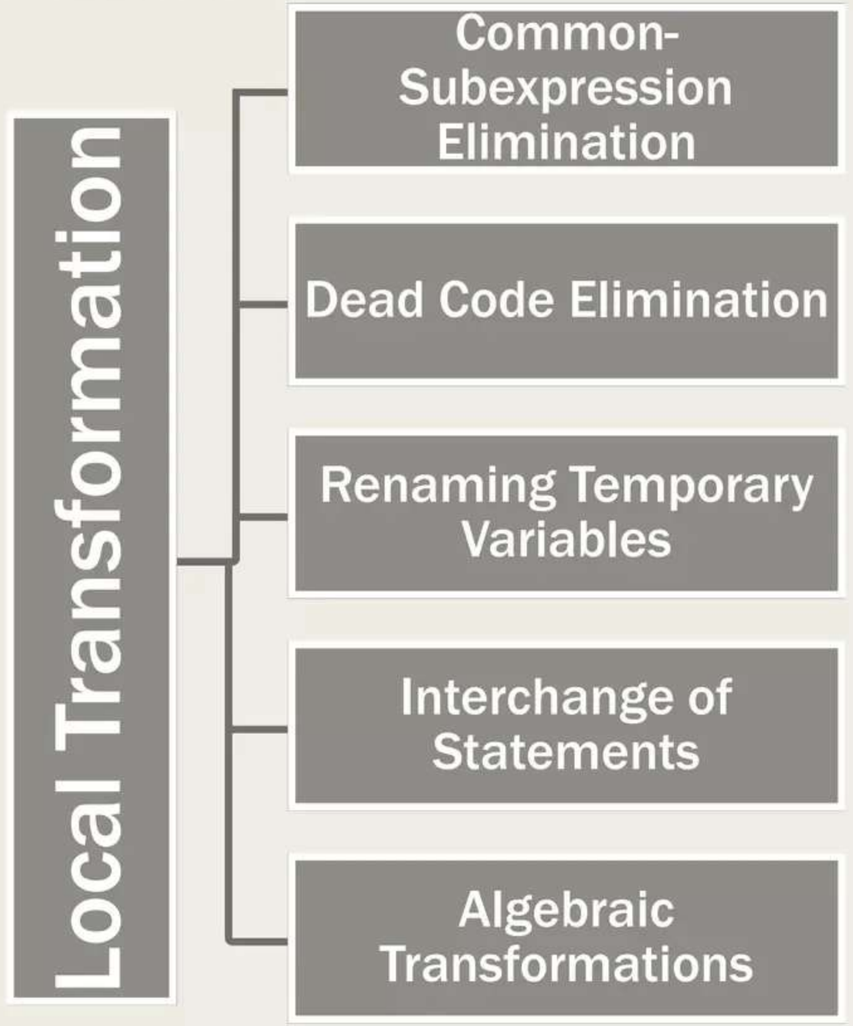
Transformations on Basic Blocks

- 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

Transformations on Basic Blocks

Some local transformation are:



Common-Subexpression Elimination

- Remove redundant computations
- 2nd and 4th:compute same expression in fig:1(a)
- Look at 1st and 3rd in fig:1(b) :b is redefine in 2nd therefore different in3rd, not the same expression

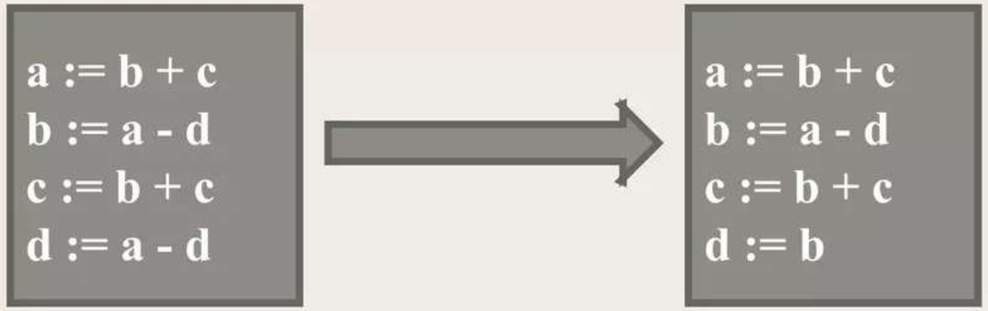
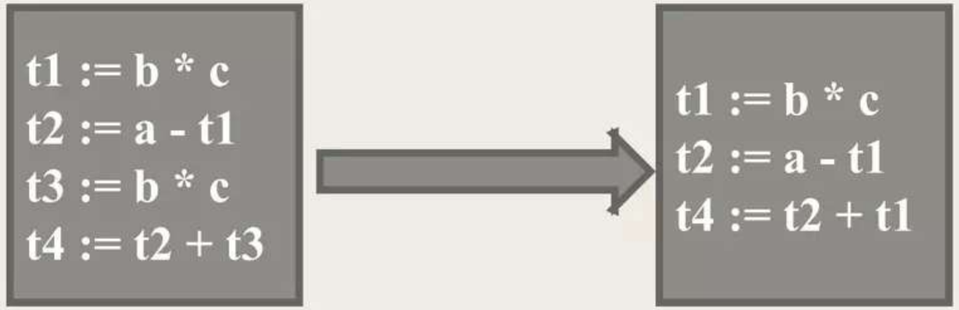
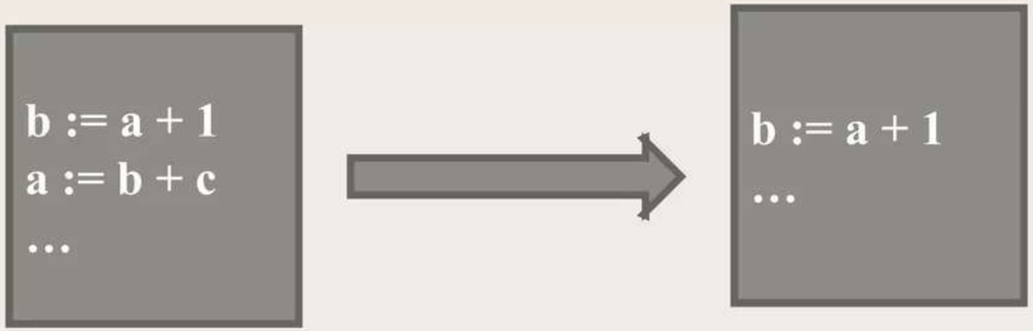


fig:1(a) fig:1(b)

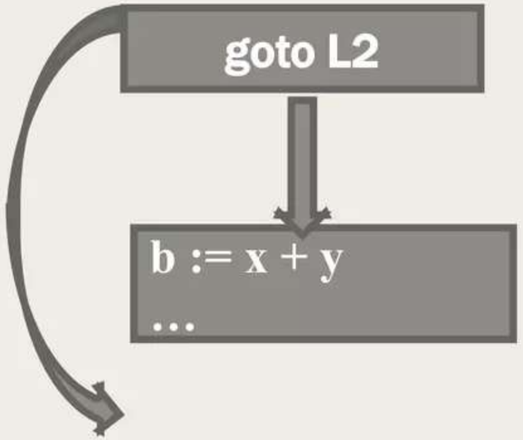


Dead Code Elimination

- Remove unused statements



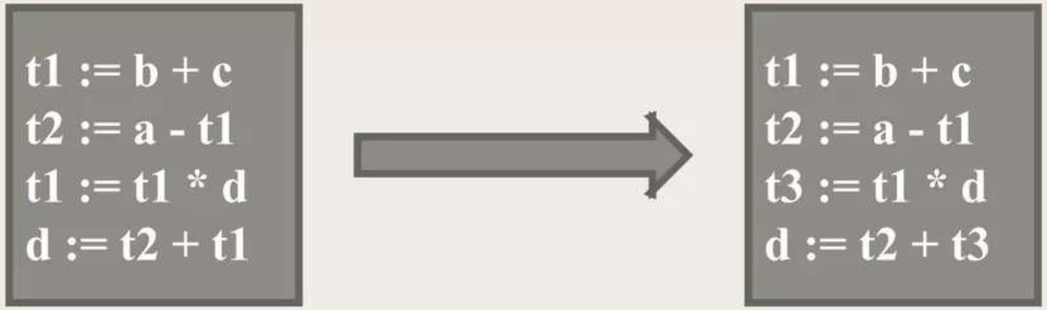
Assuming **a** is *dead* (not used)



Remove unreachable code

Renaming Temporary Variables

- Temporary variables that are dead at the end of a block can be safely renamed.
- The basic block is transforms into an equivalent block in which each statement that defines a temporary defines a new temporary.
- Such a basic block is called *normal-form block* or *simple block*.

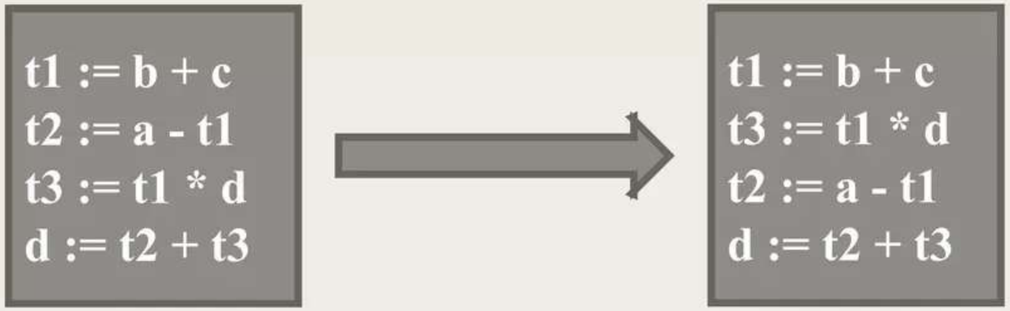


Normal-form block

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

Interchange of Statements

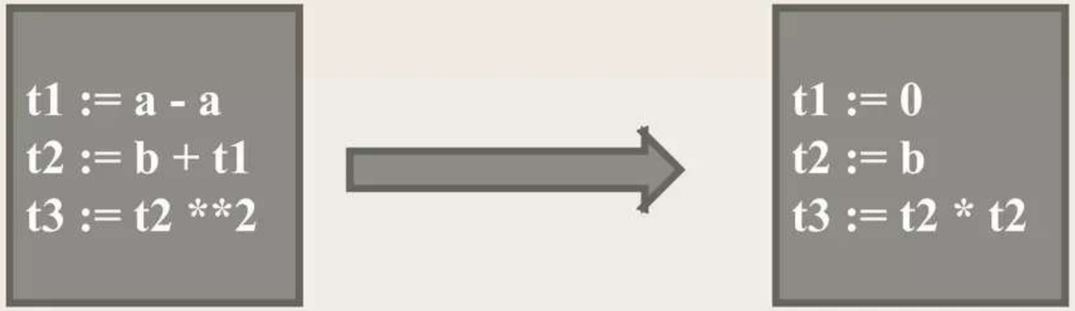
- Independent statements can be reordered without effecting the value of block to make its optimal use.



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

Algebraic Transformations

- Change arithmetic operations to transform blocks to algebraic equivalent forms.
- Simplify expression or replace expensive expressions by cheaper ones.



- In statement 3,usually require a function call
- Transforms to simple and equivalent statement

!! Queries ??

Q & A

☺Thank You ☺