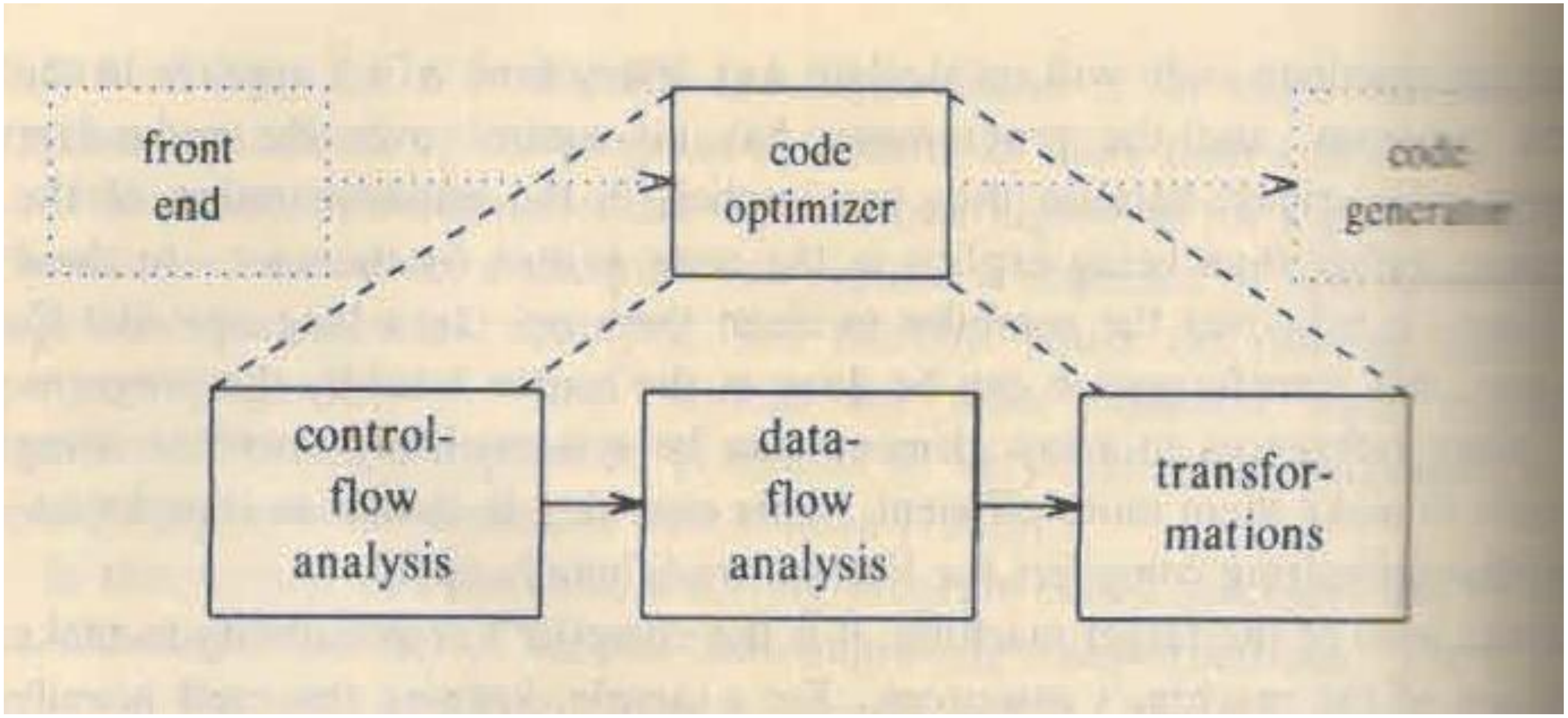
Code Optimization

Introduction



Organization of code optimizer

Sources of optimization

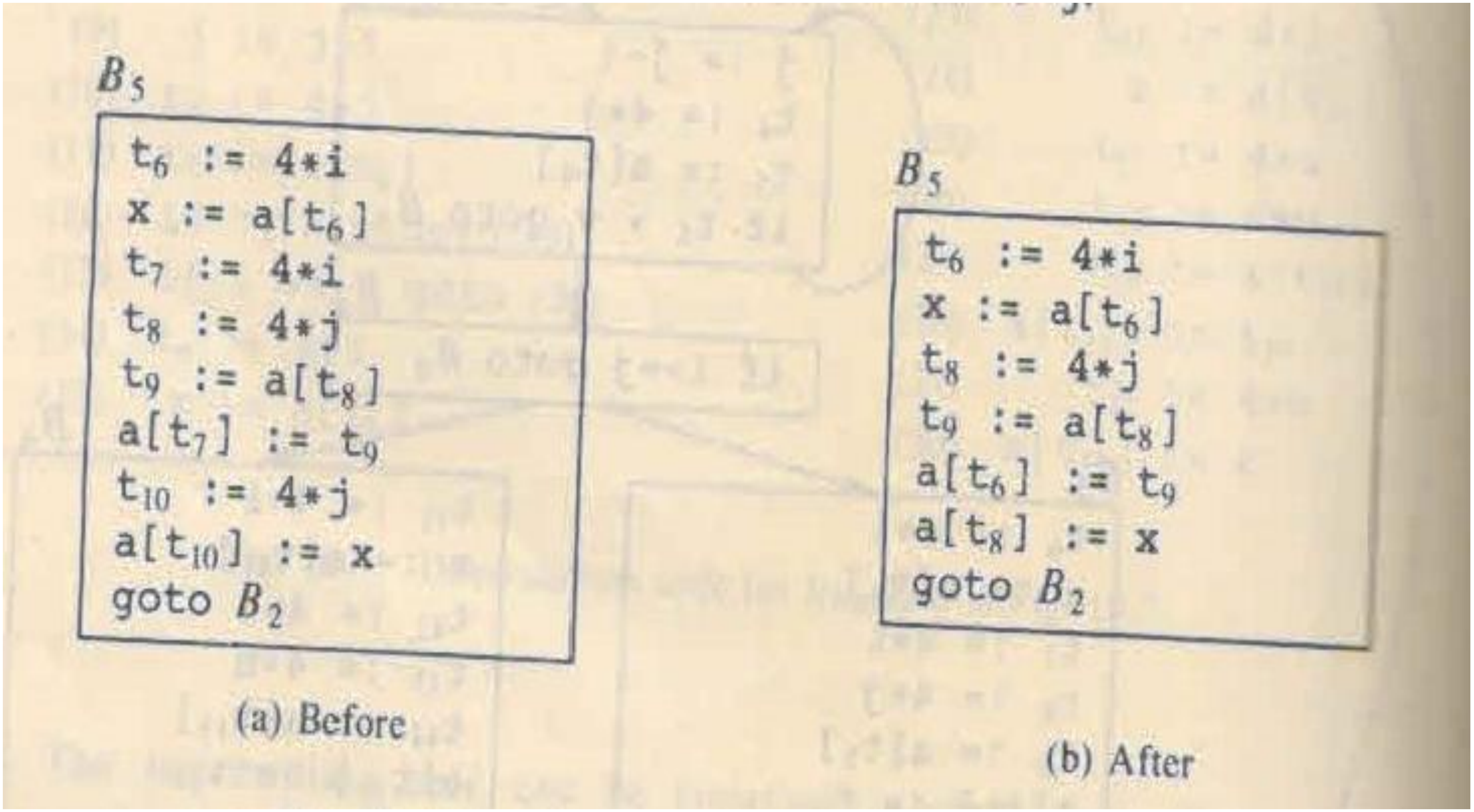
- A transformation of a program is called local if it can be performed by looking only at the statement in a basic block; otherwise it is called global. Local transformations are usually performed first.

Function preserving transformation

- Common sub-expression elimination
- Copy propagation
- Dead-code elimination
- Loop optimization
- Code motion
- Induction variable and Reduce in strength.

Common sub-expression s

- Example : E has previously computed and its value is not changed. Use previously computed value.



B_5

```
t6 := 4*i
x := a[t6]
t7 := 4*i
t8 := 4*j
t9 := a[t8]
a[t7] := t9
t10 := 4*j
a[t10] := x
goto B2
```

(a) Before

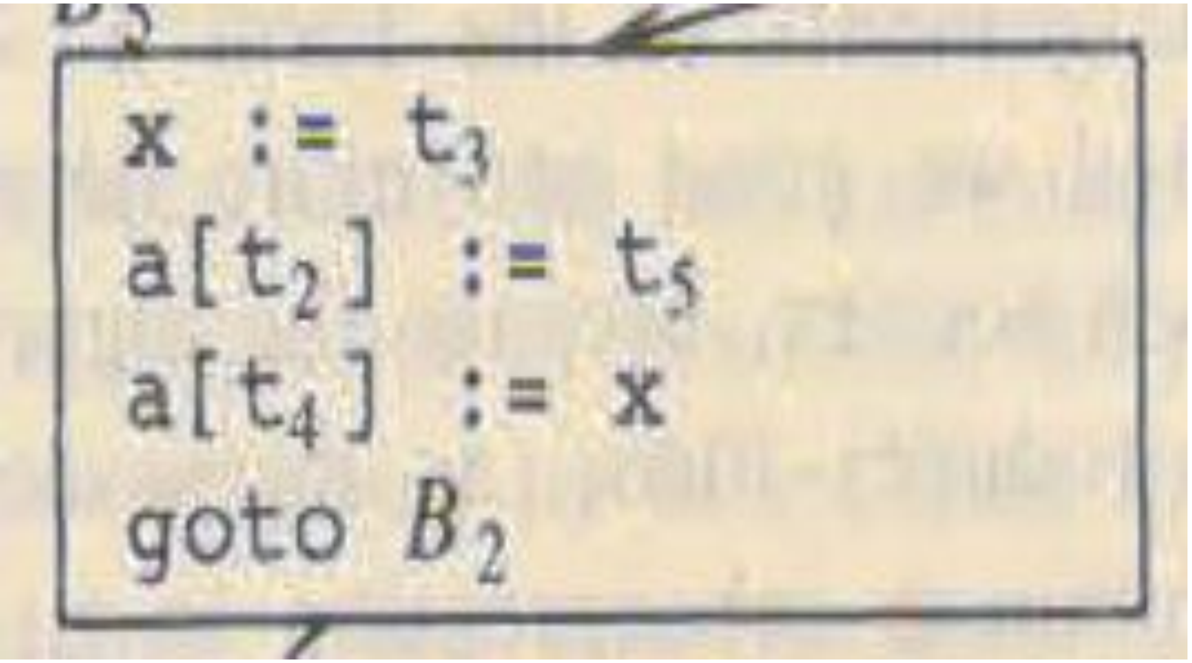
B_5

```
t6 := 4*i
x := a[t6]
t8 := 4*j
t9 := a[t8]
a[t6] := t9
a[t8] := x
goto B2
```

(b) After

Copy propagation

- The idea behind copy propagation transformation is to use g for f, wherever possible after the copy statement f := g.



```
x := t3
a[t2] := t5
a[t4] := x
goto B2
```

```
X := t3
A[t2] := t5
A[t4] := t3
goto B2
```

Dead-Code Elimination

- A variable is live at a point in a program if its value can be used subsequently; otherwise, it is dead at that point. Although, the programmer is unlikely to introduce any dead code intentionally, it may appear as the result of previous transformations.

Loop optimization

- The running time of a program may be improved if we decrease the number of instructions in the inner loop, even if we increase amount of code outside that loop.

Code Motion

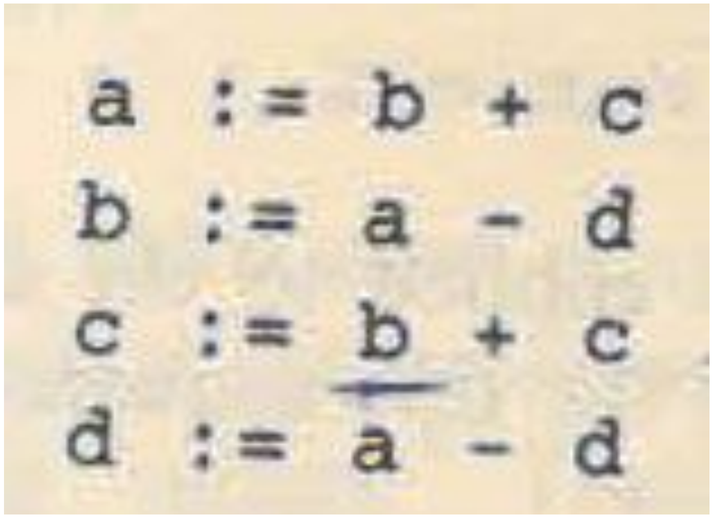
- Decreases the amount of code in a loop. This transformation takes an expression that yields the same result independent of the number of times a loop is executed and place the expression before loop.

Induction variable and reduction in Strength

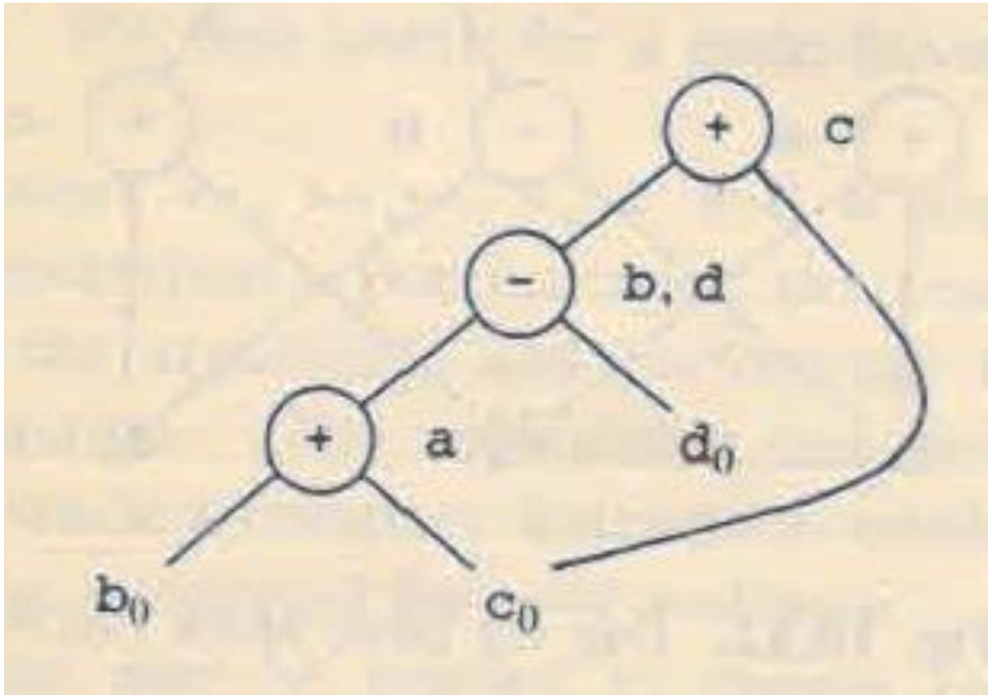
- **induction variable** is a **variable** that gets increased or decreased by a fixed amount on every iteration of a loop or is a linear function of another **induction variable**.
- When there are two or more induction variables in a loop, it may be possible to get rid of all but one, by the process of induction-variable elimination.

Optimization of Basic blocks (DAG)

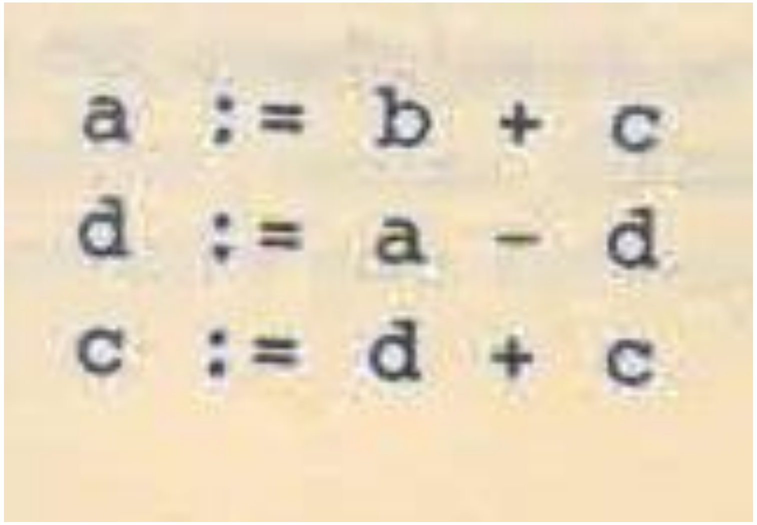
- Common sub-expressions can be detected by noticing, as new node m is about to be added, whether there is an existing node n with the same children, in the same order, and with the same operator. If so, n computes the same value as m and may be used in its place.



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



Optimization of Basic blocks (DAG)



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

DAG process miss the fact that the expression computed by the first and fourth statements in sequence. However, algebraic identities applied to DAG my expose the equivalence.

Loops in Flow Graph

Assignment