Code Optimization

C.Naga Raju

B.Tech(CSE),M.Tech(CSE),PhD(CSE),MIEEE,MCSI,MISTE

Professor

Department of CSE

YSR Engineering College of YVU

Proddatur

Contents

- Introduction to code optimization
- Optimization techniques
- GATE Problems and Solutions(17)

Introduction

- Code optimization is the fifth phase in compiler design
- It rearranges the tree generated by parser **such that to** consume fewer resources and to produces more speed.
- The meaning of the code is not altered.
- It increases the execution speed of the program
- It reduces the storage space
- Optimization can be categorized into two types:

  1)machine dependent

  2) machine independent.

Machine-dependent optimization

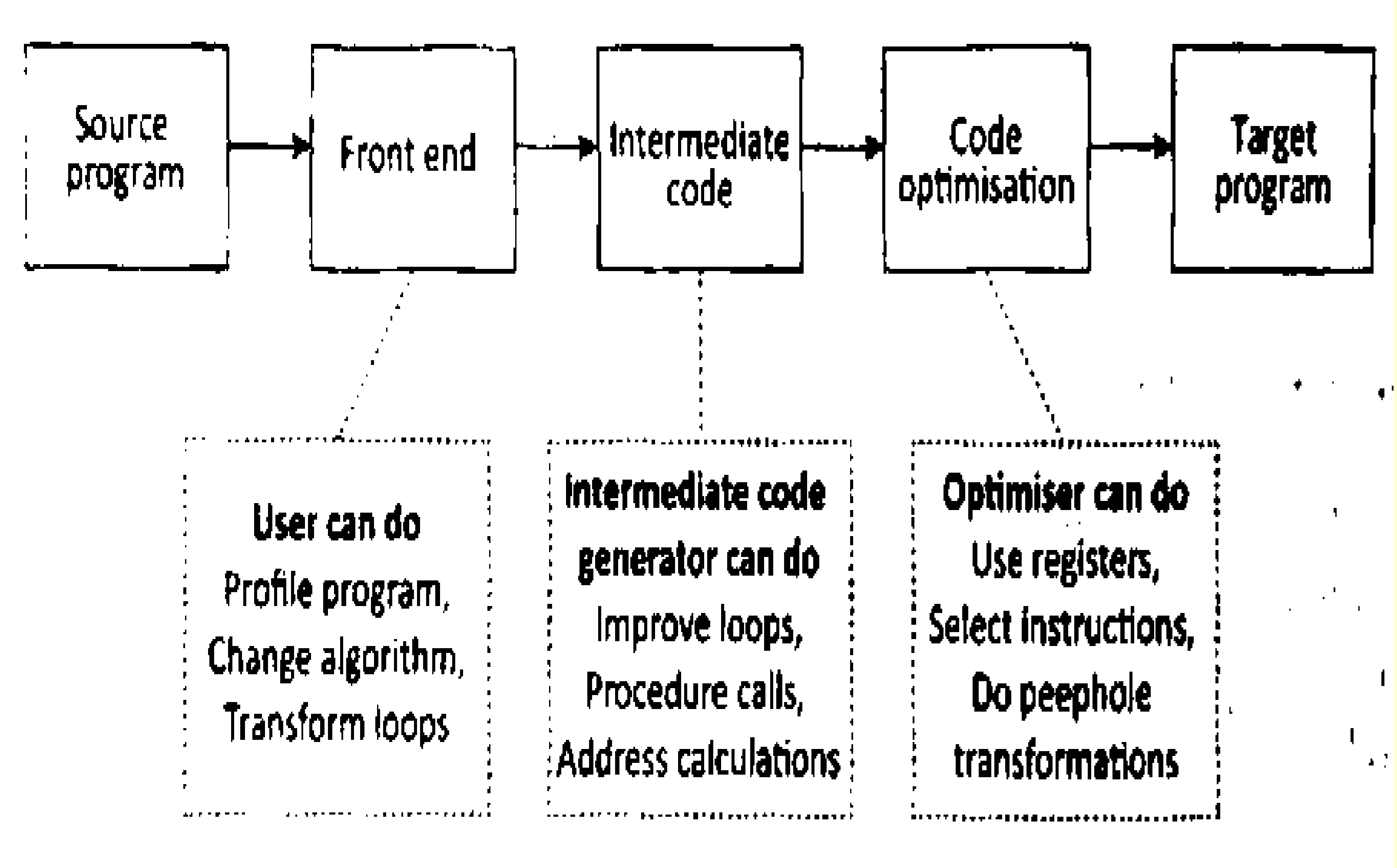
It has some special machine properties that can reduce the amount of code or increases the execution time

Ex: Register allocation and utilization of special machine instruction sequences.

Machine–independent optimization

It depends only on the algorithms or arithmetic operations in the language and not on the target machine.

Code optimization



Source program

Front end

Intermediate code

Code optimisation

Target program

User can do

Profile program,

Change algorithm,

Transform loops

Intermediate code generator can do

Improve loops,

Procedure calls,

Address calculations

Optimiser can do

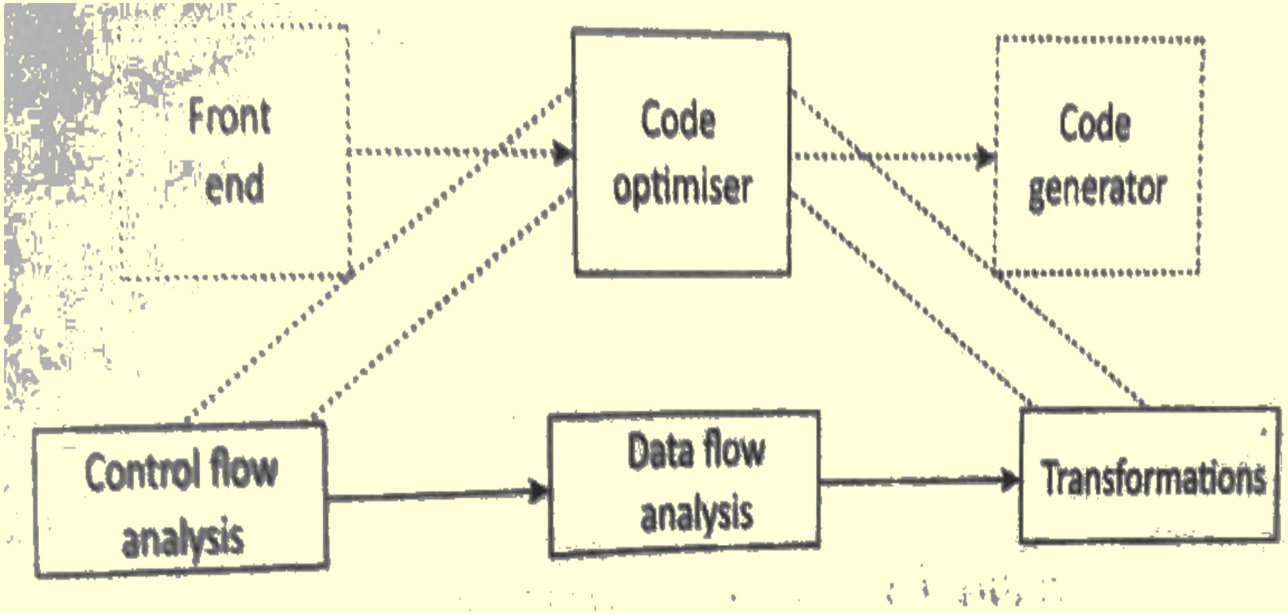
Use registers,

Select instructions,

Do peephole transformations

organization of optimized compiler

- The code optimized phase consists of control-flow, **data-flow analysis** and the application optimization.
- The code generator produces the target program from the optimized intermediate code.
- Intermediate code is independent from source and target so optimization can be improved



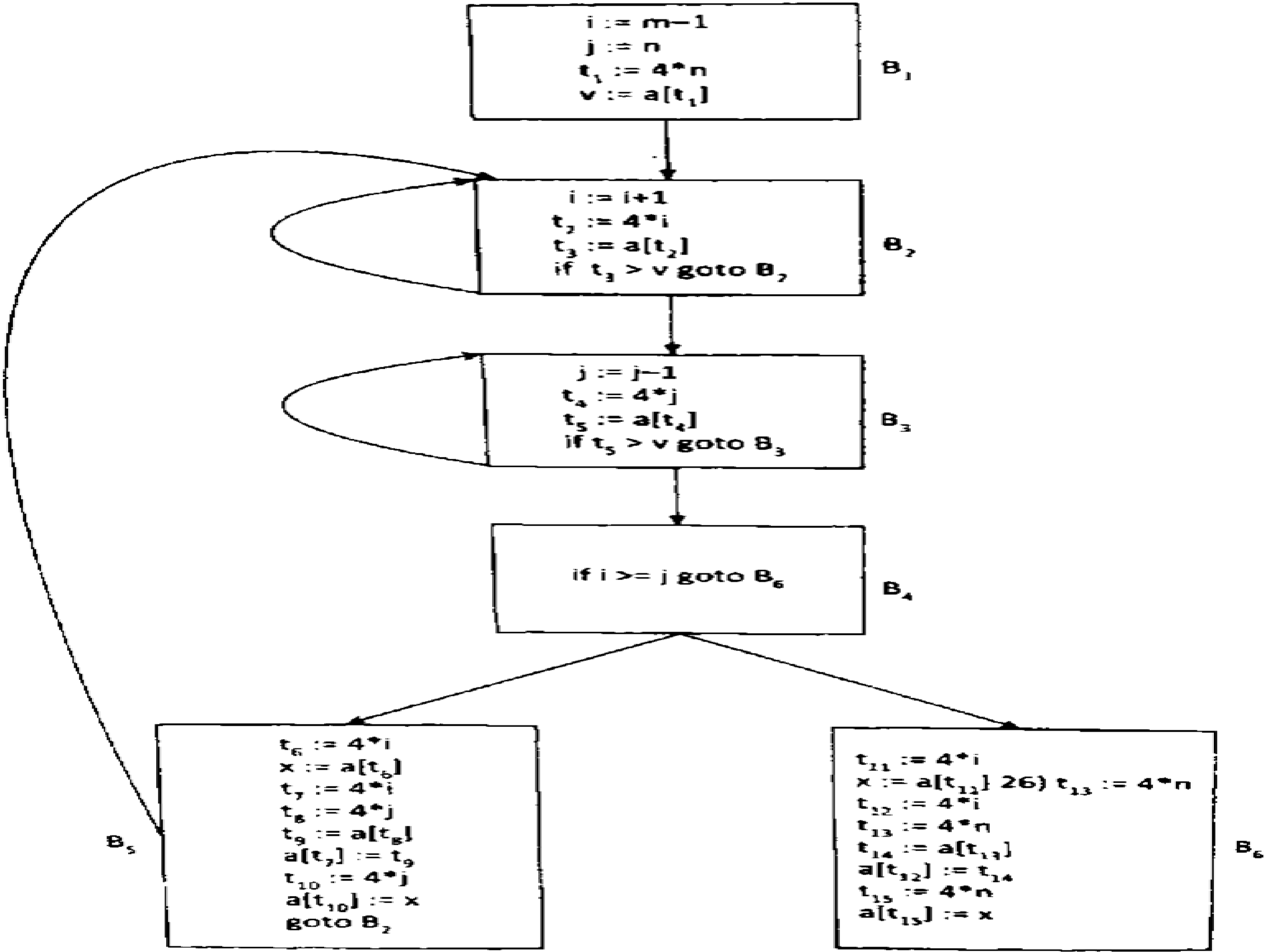
Example

consider the example of quick sort program:

```
   void quicksort
Int  m,n;
{
int i,j;
int v,x;
If(n<=m)
return;
i=m-1;   j=n;   v=a[n];
While(1)
{
do
i=i+1;
while(a[i]<v);
do
{
j=j+1;
while(a[j]<v)
If(i>=j)
break;
x=a[i];
a[i]=a[n];
a[j]=x;
}
a=a[i];   a[i]=a[n];    a[n]=x;
quicksort(m,j);
quicksort(i+1,n);
}
```

Three Address Code

```
1.  i   :=m-1
2.      j  :=n
3.  t1  :=4*n
4.  v  :=a[t1]
5.  i   :=i+1
6.  t2  :=4*I
7.  t3  :=a[t2]
8.  If t3  > v goto (5)
9.  j  :=j+1
10. t4  :=4*j
11.   t5  :=a[t4]
  12.   if  t5  >v goto (9)
13.   if i  >=j goto (23)
14.   t6  :=4*i
15.   x  :=a[t6]
16.   t7 :=4*i
17.    t8  :=4*j
18.   t9  :=a[t8]
19.  a[t7]  :=t9
20.  t10  :=4*j
21.  a[t10]  :=x
22.  goto  (5)
23.  t11  :=4*i
24. X  :=a[t11]
25.  t12  :=4*i
26.  t13   :=4*n
27.  t14  :=a[t13]
28.  a[t12]  :=t14
29.  t15  :=4*n
30.  a[t15]  :=x
```



$i := m-1$
$j := n$
$t_1 := 4*n$
$v := a[t_1]$

B_1

$i := i+1$
$t_2 := 4*i$
$t_3 := a[t_2]$
if $t_3 > v$ goto B_2

B_2

$j := j-1$
$t_4 := 4*j$
$t_5 := a[t_4]$
if $t_5 > v$ goto B_3

B_3

if $i >= j$ goto B_6

B_4

$t_6 := 4*i$
$x := a[t_6]$
$t_7 := 4*i$
$t_8 := 4*j$
$t_9 := a[t_8]$
$a[t_7] := t_9$
$t_{10} := 4*j$
$a[t_{10}] := x$
goto B_2

B_5

$t_{11} := 4*i$
$x := a[t_{11}\}$ 26) $t_{13} := 4*n$
$t_{12} := 4*i$
$t_{13} := 4*n$
$t_{14} := a[t_{13}]$
$a[t_{12}] := t_{14}$
$t_{15} := 4*n$
$a[t_{15}] := x$

B_6

Optimization depends on various factors

- Memory
- Algorithms
- Execution time
- programming language
- Function-preserving optimization
- Loop optimisations

Principle source of optimization

- Local optimization: if it can be performed by looking only at the statements in a block.
- Global optimization : if it can be performed by looking at on entire program.
- Many optimization Techniques can be performed at both local and global levels.
- The local transformations are usually performed first.

Function-Preserving optimization

- Function-Preserving optimization: It can improve the code optimization without changing the function it computes.

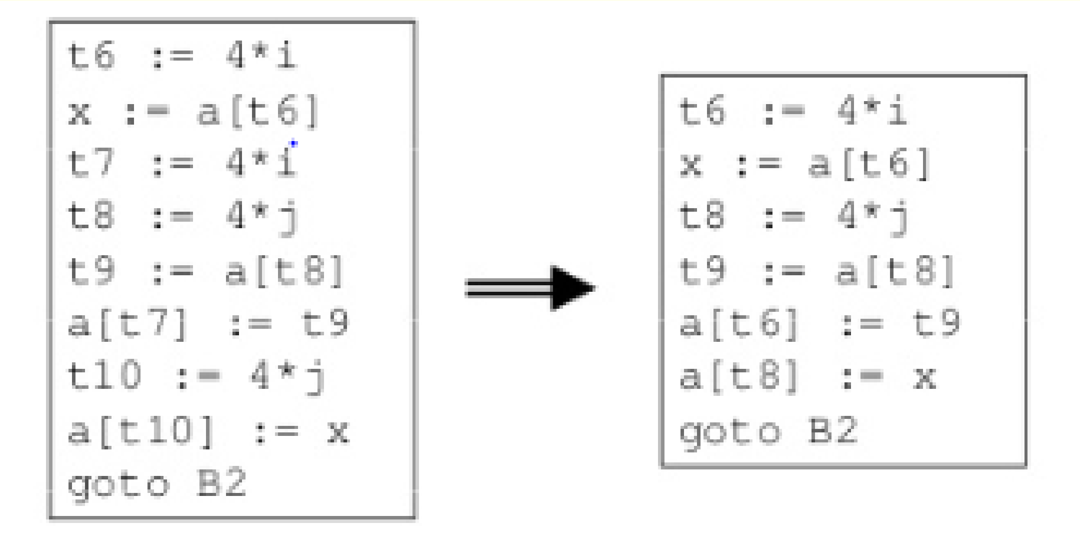
- Examples include
  - » Common sub-expression elimination
  - » Copy propagation
  - » Dead-code elimination
  - » Constant folding

Common sub-expression elimination

- If the expression is repeated more than one time in the code is called sub- expression or duplicate code.

- The assignments to t7 and t10 have common sub-expression 4*i and 4*j respectively on the right side.

- They are eliminated by using t6 and t8.

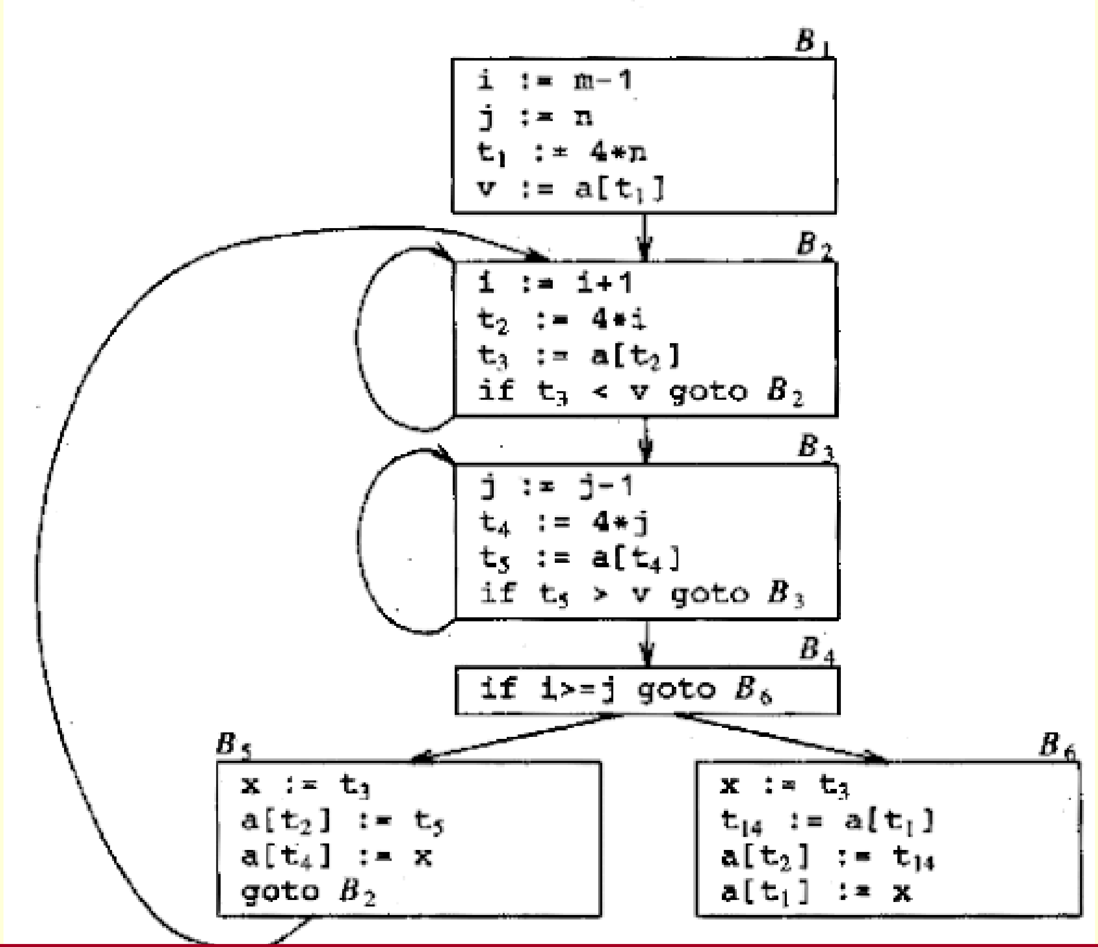


```
t6  := 4*i
x  := a[t6]
t7  := 4*i
t8  := 4*j
t9  := a[t8]
a[t7]  := t9
t10  := 4*j
a[t10]  := x
goto B2
```

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

After common sub-expression elimination

- The result of eliminating both global and local common sub-expression from blocks B5and B6in the flow graph.



```
B1
i := m-1
j := n
t1 := 4*n
v := a[t1]
```

```
B2
i := i+1
t2 := 4*i
t3 := a[t2]
if t3 < v goto B2
```

```
B3
j := j-1
t4 := 4*j
t5 := a[t4]
if t5 > v goto B3
```

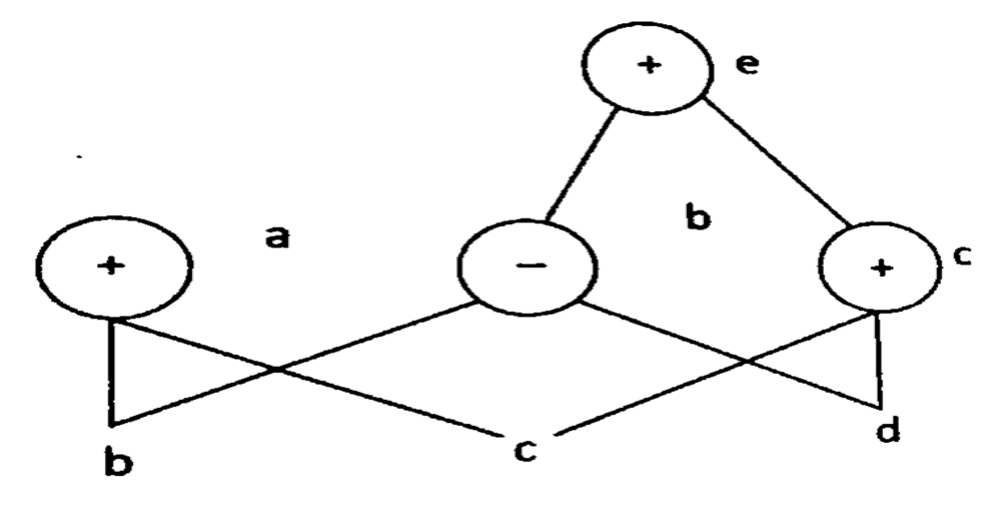
```
B4
if i>=j goto B6
```

```
B5
x := t3
a[t2] := t5
a[t4] := x
goto B2
```

```
B6
x := t3
t14 := a[t1]
a[t2] := t14
a[t1] := x
```

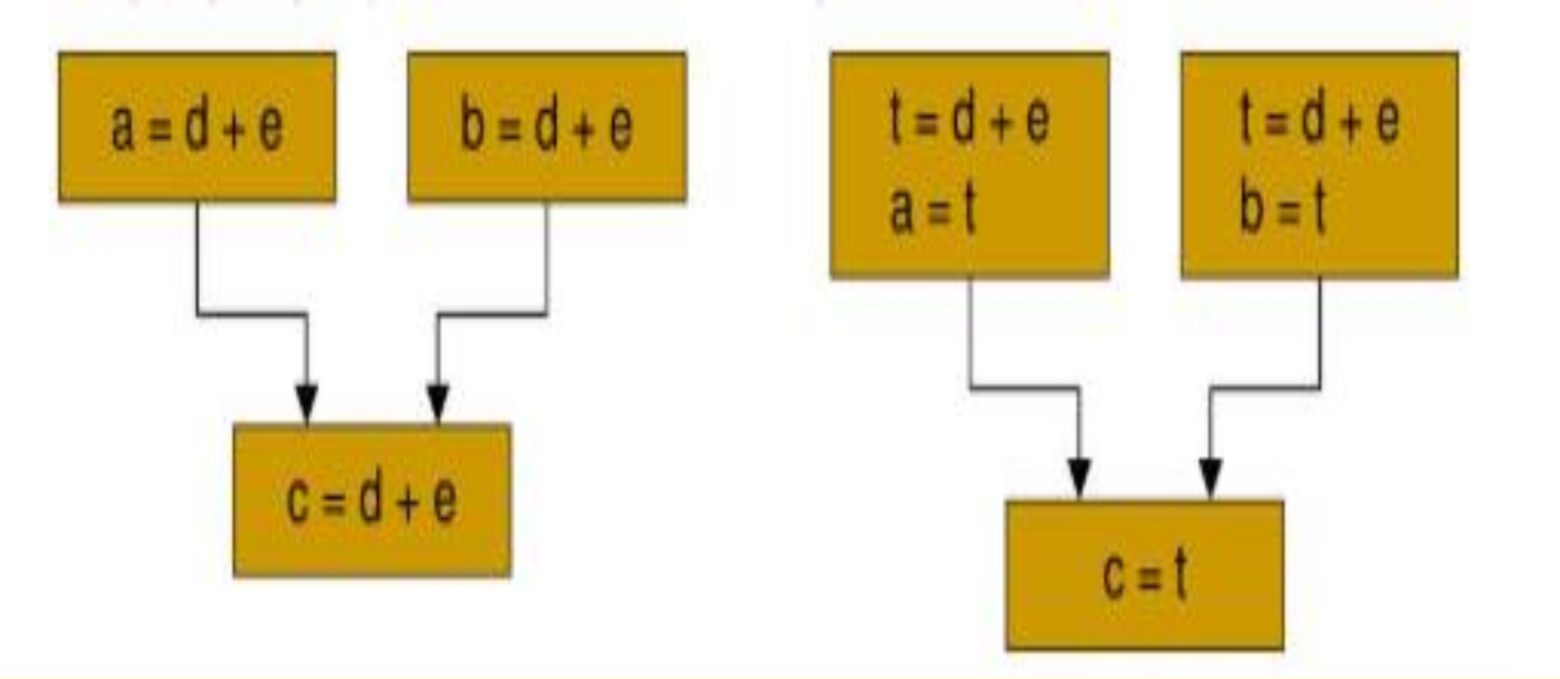
- Example : a :=b+c
  b :=a-d
  c :=b+c
  d :=a-d

  If ‘a’ is a sub expression code,then delete node a.



Copy Propagation

- Assignment of the form X :=Y are called copy statements or copies.
- The idea behind this is to use Y for X Wherever possible after the copy statement X :=Y.



a = d + e

b = d + e

c = d + e

t = d + e
a = t

t = d + e
b = t

c = t

Example:

- From the common sub-expression elimination
- Consider the block B5,

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

- After copy propagation,

```
x:=t3; =>this can be eliminated.
a[t2]:=t5;
a[t4]:=t3;
goto B2;
```

Dead code elimination

- The code which is never be executed with in the program is called dead code. Example unreachable code and un used variables

```
 int  add(int a,int b){
   int x,y,z;
return (a+b);
printf(“hello compiler”);
printf(“how are you”);
}
```

Constant Folding

- The process of recognizing and evaluating constant expressions at compile time rather than at run time is called constant folding.
- **Examples**

```
For (i=0;i<n;i++){
  y=y+8+7+4*12;
}
```

```
x=8+7+4*12;
For (i=0;i<n;i++){
  y=y+8+7+4*12
}
```

Loop Optimization

- Three types of loop optimizations:
  - ✓Code Motion
  - ✓Induction-Variable elimination
  - ✓Reduction in strength

Code Motion

- It moves code outside the loop
- Thus transformation takes an expression that yields the same result independent of the number of times a loop is executed and places the expression before the loop.

Example

Consider the stmt:

```
while(i<=limit-2)
```

Code motion :

```
t :=limit-2;
while(i<=t)
```

Induction-variable elimination

- Any two variables are said to be induction variables ,if there is a change in any one of the variable, then there is a corresponding change in the other variable.

Before

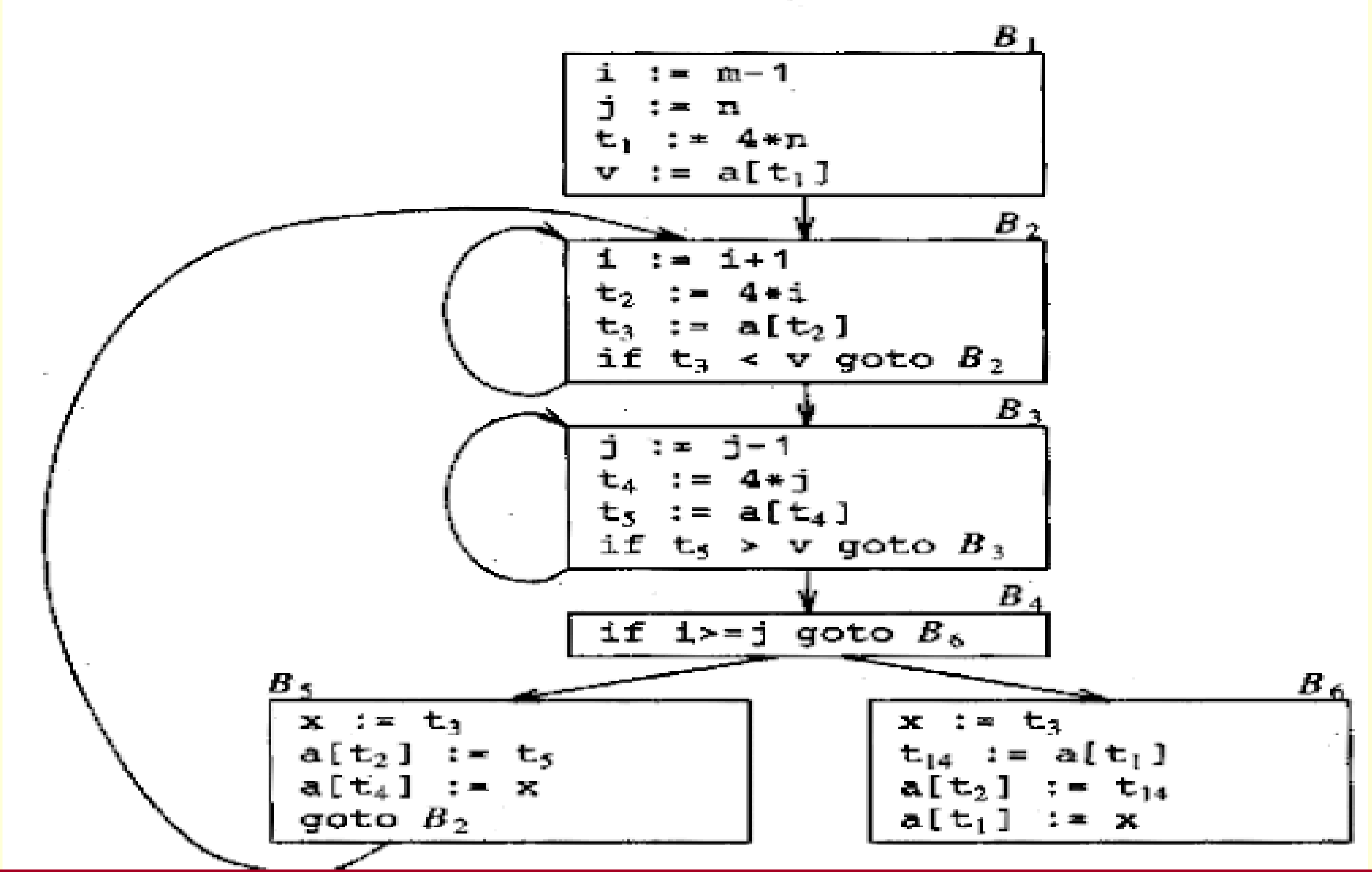
```
B3(in self loop)
j := j+1
t4 := 4*j;
t5 :=a[t4];
If t5 >v goto B3
```

After

```
B3 (in self -loop)
j :=j+1;
t4 :=t4+4
t5 :=a[t4];
If t5 >v goto B3
```

Induction-variable elimination

- Flow graph after induction-variable elimination and strength reduction:



B_1

$i := m-1$
$j := n$
$t_1 := 4*n$
$v := a[t_1]$

B_2

$i := i+1$
$t_2 := 4*i$
$t_3 := a[t_2]$
if $t_3 < v$ goto B_2

B_3

$j := j-1$
$t_4 := 4*j$
$t_5 := a[t_4]$
if $t_5 > v$ goto B_3

B_4

if $i>=j$ goto B_6

B_5

$x := t_3$
$a[t_2] := t_5$
$a[t_4] := x$
goto B_2

B_6

$x := t_3$
$t_{14} := a[t_1]$
$a[t_2] := t_{14}$
$a[t_1] := x$

Reduction in strength

- The replacement of an expensive operation by a cheaper one.
- Example :
- step t2 :=4*i; in B2
- Replaced with t2 :=t2+4;
- This replacement will speed up the object code ,if addition takes less time than multiplication

Introduction to Global Data Flow Analysis

- The compiler needs to collect information about the program as a whole and to distribute this information to each block in the flow graph
- The data-flow information that can be optimising compiler collects by a process is known as Data-flow analysis.
- The generation and killing of process depends on the desired information on the data-flow analysis to be solved.
- Data-flow analysis is affected by the control construct in a problem.

OPTIMISATION OF BASIC BLOCKS

Basic Blocks

- A basic block code sequence is entered at the beginning and exited only at the end.

Basic Block=Sequence of instructions with single entry and exit.

- Extend analysis of register use to program units larger than expression but still completely analysable at compile time .
- If the first instruction of B1 is executed ,so is a remainder of the block

Data-Flow Analysis of Structured Programs

- Flow graphs for control flow constructs such as do-while statements have a useful property.
- There is a single beginning point at which the control enters and a single end point at which the controls leaves, when execution of the statement is over.

GATE QUESTIONS

Question 1

Which of the following comment about peephole optimization is true?

A)It is applied to a small part of the code and applied repeatedly

B)It can be used to optimize intermediate code

C)It can be applied to a portion of the code that is not contiguous

D)It is applied in the symbol table to optimize the memory requirements.

It is applied to a small part of the code and applied repeatedly

Question 2

Which of the following class of statement usually produces no executable code when compiled?

A)Declaration

B)Assignment statements

C)Input and output statements

D)Structural statements

A)Declaration

Question 3

Substitution of values for names (whose values are constants) is done in

A)Local optimization

B)Loop optimization

C)Constant folding

D)Strength reduction

C)Constant folding

Question 4

In compiler terminology reduction in strength means

A)Replacing run time computation by compile time computation

B)Removing loop invariant computation

C)Removing common subexpressions

D)Replacing a costly operation by a relatively cheaper one

Option D

Question 5

Which of the following statements about peephole optimization is False?

A)It is applied to a small part of the code

B)It can be used to optimize intermediate code

C)To get the best out of this, it has to be applied repeatedly

D)It can be applied to the portion of the code that is not contiguous

It can be applied to the portion of the code that is not contiguous

Question 6

The graph that shows basic blocks and their successor relationship is called:

A)DAG

B)Control graph

C)Flow graph

D)Hamiltonian graph

Flow graph

Question 7

In compiler optimization, operator strength reduction uses mathematical identities to replace slow math operations with faster operations. Which of the following code replacements is an illustration of operator strength reduction ?

- A)Replace P + P by 2 * P or Replace 3 + 4 by 7.
- B)Replace P * 32 by P << 5
- C)Replace P * 0 by 0
- D)Replace (P << 4) – P by P * 15

- Replace P * 32 by P << 5

Question 8

In _______, the bodies of the two loops are merged together to form a single loop provided that they do not make any references to each other.

- A)Loop unrolling
- B)Strength reduction
- C)Loop concatenation
- D)Loop jamming

- Loop jamming

Question 9

Loop unrolling is a code optimization technique:

A)That avoids tests at every iteration of the loop.

B)That improves performance by decreasing the number of instructions in a basic block.

C)That exchanges inner loops with outer loops

D)That reorders operations to allow multiple computations to happen in parallel

- That avoids tests at every iteration of the loop

Question 10

Peer-hole optimization is a form of :

A)Loop optimization

B)Local optimization

C)Constant folding

D)Data flow analysis

B)Local optimization

Question 11

Dead-code elimination in machine code optimization refers to :

- A) Removal of all labels.
- B) Removal of values that never get used.
- C) Removal of function which are not involved.
- D) Removal of a module after its use.
- Removal of values that never get used.

Question 12

In the context of compiler design, “reduction in strength” refers to :

- A)Code optimization obtained by the use of cheaper machine instructions
- B)Reduction in accuracy of the output
- C)Reduction in the range of values of input variables
- D)Reduction in efficiency of the program

- Code optimization obtained by the use of cheaper machine instructions

Question 13

Some code optimizations are carried out on the intermediate code because

A)They enhance the portability of the compiler to other target processors

B)Program analysis is more accurate on intermediate code than on machine code

C)The information from dataflow analysis cannot otherwise be used for optimization

D)The information from the front end cannot otherwise be used for optimization

A and B are true

Question 14

Which one of the following is FALSE?

A)A basic block is a sequence of instructions where control enters the sequence at the beginning and exits at the end.

B)Available expression analysis can be used for common sub expression elimination.

C)Live variable analysis can be used for dead code elimination.

D)x = 4 ∗ 5 => x = 20 is an example of common subexpression elimination.

Answer is D

Question 15

One of the purposes of using intermediate code in compilers is to

A)make parsing and semantic analysis simpler.

B)improve error recovery and error reporting.

C)increase the chances of reusing the machine-independent code optimizer in other compilers.

D)improve the register allocation.

- Option is C

Question 16

Consider the following C code segment.

```
for (i = 0, i<n; i++)
• {
•     for (j=0; j<n; j++)
•     {
•         if (i%2)
•         {
•             x += (4*j + 5*i);
•             y += (7 + 4*j);
•         }
•     }
• }
```

Which one of the following is false?

A)The code contains loop invariant computation

B)There is scope of common sub-expression elimination in this code

C)There is scope of strength reduction in this code

D)There is scope of dead code elimination in this code

Option is D

Question 17

The identification of common sub-expression and replacement of runtime computations by compile-time computations is:

A.Local optimisation

B.Constant folding

C.Loop Optimisation

D.Data flow analysis

Question 18

Code optimisation is responsibility of:

A. Application programmer

B. Syatem programmer

C. Operating System

D. All of the above

Option D

Thank You