UNIT-6

Subtitling (CD)

machine independent optimization

machine independent optimization depends on only the arithmetic properties of the operations in the language and can be achieved using foll. criteria

→ The code should be analyzed completely and use alternative equivalent sequence of source code that will produce a minimum amount of target code.

→ use appropriate program structure in order to improve the efficiency of target code.

→ From the source program eliminate the unreachable code.

→ move two or more identical computations at one place and make use of the result instead of each time computing the expressions.

Semantic preserving transformations

→ These are a numbers of ways in which a compilers can improve a program without changing the function it computes.

→ Common subexpression elimination, copy propagation, dead code elimination, and constant folding are common examples of such function-preserving (or semantics-preserving) transformations.

→ Consider an example sorting program called quicksort to illustrate several important code-improving transformations.

→ In this program a[0] must contain the smallest of the sorted elements and a[max] contains the largest element.

C Code for quicksort

```
void quicksort(int m, int n)
  /* recursively sorts a[m] through a[n] */
{
  int i, j;
  int v, x;
  if (n <= m) return;
  /* fragment begins here */
    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[j];      /* swap a[i], a[j] */
    a[j] = x;
  }
    x = a[i];
  a[i] = a[n];        /* swap a[i], a[n] */
  a[n] = x;
  /* fragment ends here */
  quicksort(m, j);
  quicksort(i+1, n);
}
```

→ Intermediate representation consists of three-address statements, where temporary variables are used to hold all the results of intermediate expressions.

→ In this example we assume that integers occupy 4 bytes.

→ The 3-address statements for quicksort program is as follows :-

Three-address code के उदाहरण

(1) i = n-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*i
(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) a[t13] = x

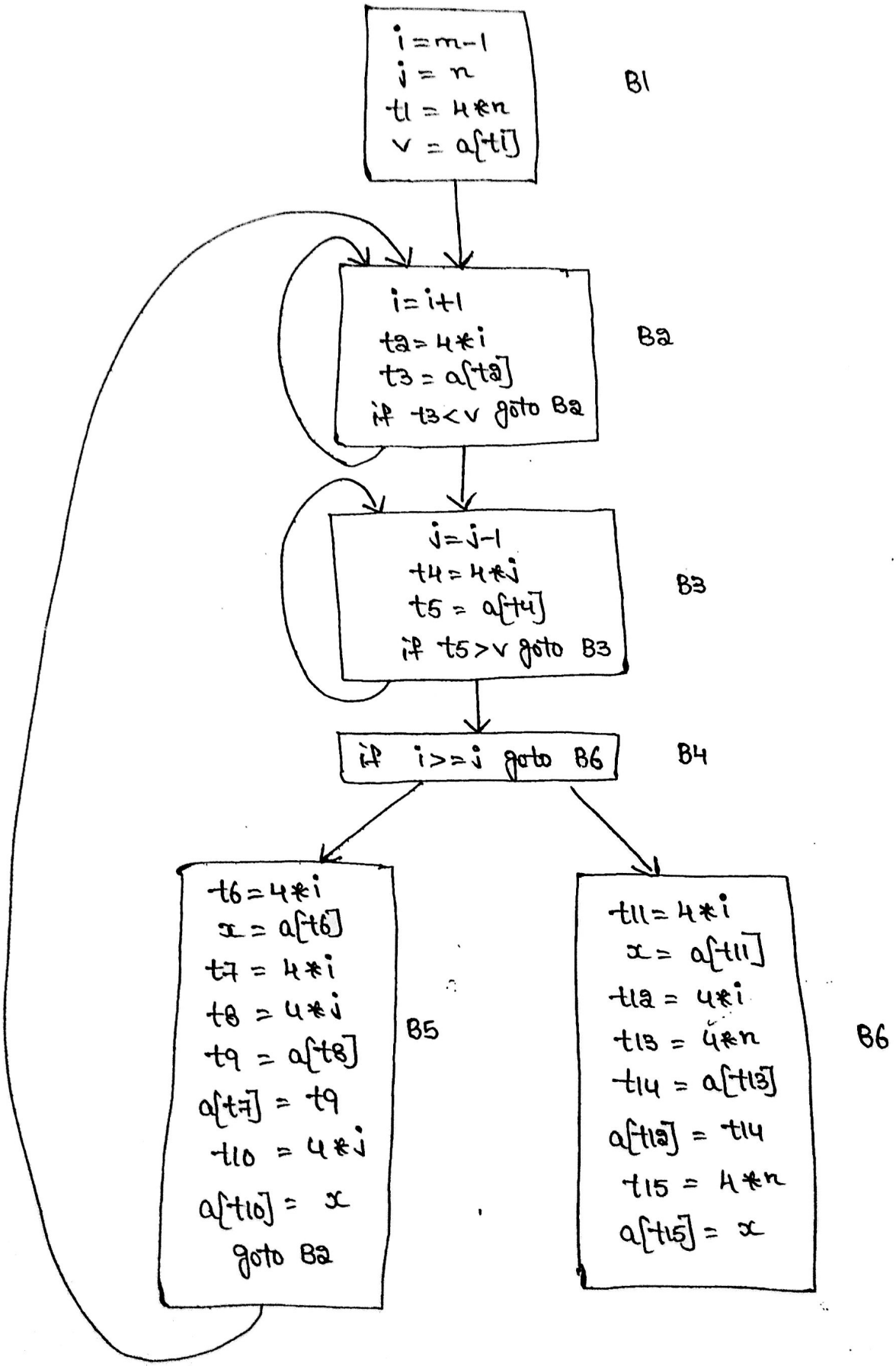
→ The following is the flowgraph for the quicksort program.

→ Block B1 is the entry node.

→ There are 3 loops. Blocks B2 and B3 are loops by themselves.

→ Blocks B2, B3, B4 and B5 together form a loop, with B2 the only entry point.

Flow graph for the quicksort



Common subexpression Elimination

An occurrence of an expression E is called a common subexpressio if E was previously computed and the values of variables in E have not changed since the previous computation. We can avoid recomputing the expression, if we can use the previously computed val

Example

① The assignments to t7 and t10 have the common subexpressions 4*i and 4*j respectively on RHS. They have been eliminated by using t6 instead of t7 and t8 instead of t10 in the Block B5.

After eliminating t7 and t10, the Block B5 is

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

B5

t7 - t6
t10 - t8

② The assignments to t12 and t15 have the common subexpressions 4*i and 4*n respectively on RHS. They have been eliminated by using t11 instead of t12 and t13 instead of t15 in the Block B6.

After eliminating t12 and t15, the Block B6 is

```
t11 = 4*i
x = a[t11]
t13 = 4*n
t14 = a[t13]
a[t11] = t14
a[t13] = x
```

B6

After local common subexpressions are eliminated, B5 still evaluates 4*i and 4*j. They have eliminated, by using t2 instead of t6 and t4 instead of t8.

After eliminating t6 and t8, Block B5 is

```
x = a[t2]
t9 = a[t4]
a[t2] = t9
a[t4] = x
goto B2
```
B5

The common subexpressions t11 and t13 have been eliminated, by using t2 instead of t11 and t1 instead of t13.

After eliminating t11 and t13, Block B6 is

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

B5 and B6 still having common subexpressions i.e, t9 = a[t4] in B5 and t5 = a[t4] in B3, and t3 = a[t2] in B2 and x = a[t2] in B5

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

Now in B6 the common subexpression is x = a[t2] in B6 and t3 = a[t2] in B2 then Block B6 becomes

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

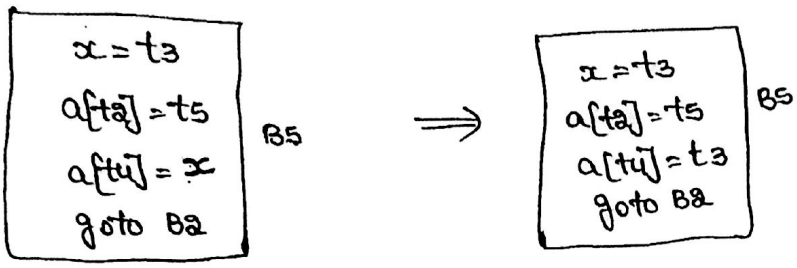
Copy Propagation

→ The form f=g called copy statements or copies.

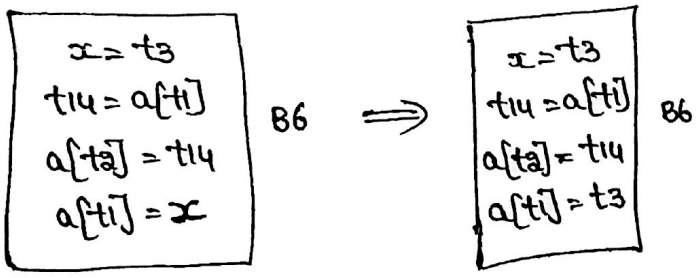
→ The idea behind the copy propagation transformation is to use 'g' for 'f' wherever possible after the copy statement f=g.

Example

① The assignment x=t3 in Block B5 is a copy statement, then copy-propagation applied to B5 yields,



② The assignment x=t3 in Block B6 is a copy statement, then copy-propagation applied to B6 yields,

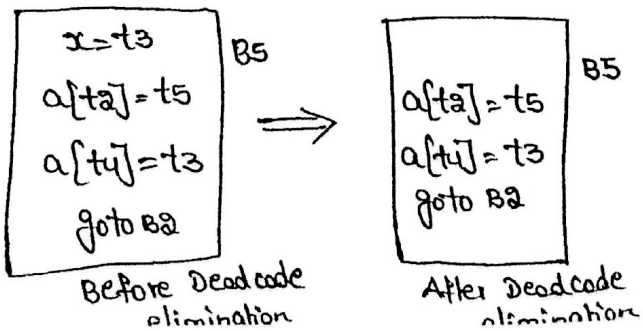


Dead Code Elimination

A variable is live at a point in a program if its value can be used subsequently, otherwise it is dead at that point, i.e, statements that compute values that never get used.

Example

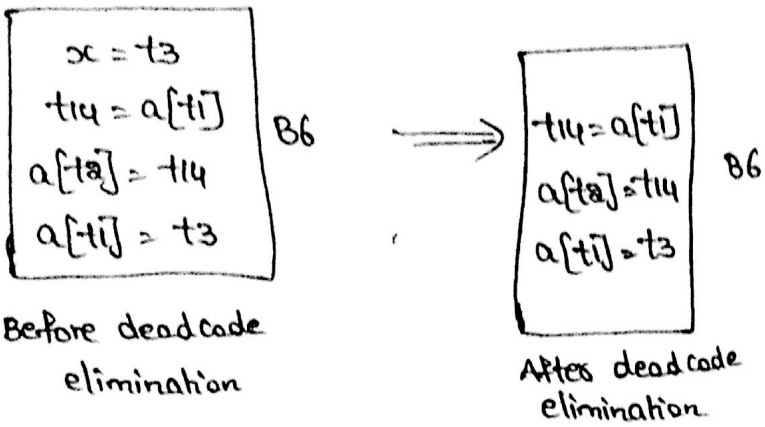
① The assignments x=t3 in Block B5 is a dead code. Dead code elimination applied to B5 yields



Before Dead code elimination

After Dead code elimination

The assignment x=t3 in Block B6 is a dead code. After eliminating dead code B6 is



Before dead code elimination

After dead code elimination

Constant Folding

→ The substitution of values for names whose values are constant is known as constant folding.

→ In the folding technique the computation of constant is done at compile time instead of execution time.

Example

① for the statement

$$\text{Length} = \left(\frac{22}{7}\right) * d$$

Here folding is implied by performing the computation of $\left(\frac{22}{7}\right)$ at compile time instead of execution time.

Loop optimization

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

These are ~~three~~ 6 techniques in loop optimizations.

1) Code motion
2) Induction variable elimination
3) Reduction in strength
4) Loop unrolling
5) Loop fusion
6) Loop invariant method.

1) Code motion

Code motion moves code outside a loop. This transformation takes an expression that yields the same result independent of the numbers of times a loop is executed (a loop-invariant computation) and places the expression before the loop.

Example

Consider the statement

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

Evaluation of limit-2 is a loop-invariant computation.

Code motion will result in the equivalent of,

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

2) Induction variable Elimination

Induction variable is a variable of the form,

$$V_i = \pm C_1$$
$$V_i = V_i \pm C_2$$

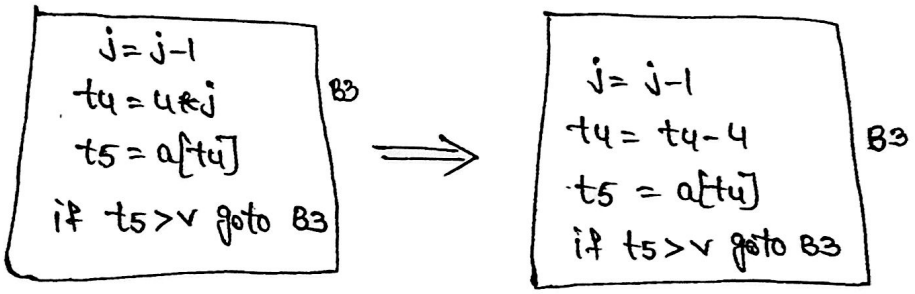
where C_1, C_2 are constants.

Example

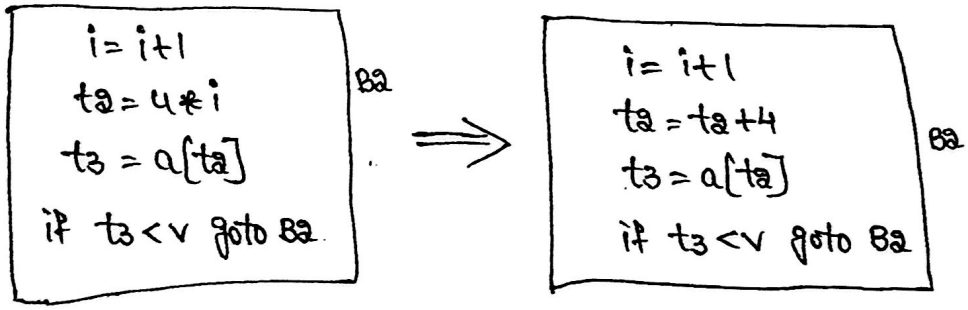
The values of j and t_4 remain in lock-step in B3. Every time the value of j decreases by 1, that of t_4 decreases by 4 because 4*j is assigned to t_4.

Therefore replace the assignment $t_4 = 4*j$ by $t_4 = t_4 - 4$. The only problem is that t_4 does not have a value when we enter block B3 for the first time. Since we must maintain the relationship $t_4 = 4*j$ on entry to the block B3, we place an initialization of t_4 at the end of the block where 'j' itself is initialized.

Block B3 is,



Block B2 is,



3) Reduction in strength

The replacement of an expensive operation by a cheaper one is termed reduction in strength.

Example The multiplication step $t_2 = 4*i$ in B2 was replaced by an addition step $t_2 = t_2 + 4$.

This replacement will speed up the object code, if addition takes less time than multiplication, as in the case of many machines.

④ Loop unrolling

In this method the number of jumps and tests can be reduced by writing the code two times.

for example

```
int i=1;
while(i<=100)
{
    a[i]=b[i];
    i++;
}
```

can be written as

```
int i=1;
while(i<=100)
{
    a[i]=b[i];
    i++;
    a[i]=b[i];
    i++;
}
```

⑤ Loop fusion

In Loop fusion method several loops are merged to one loop.

for example

```
for i=1 to n do
for j=1 to m do
a[i,j]=10
```

can be written as

```
for i=1 to n*m do
a[i]=10
```

⑥ Loop invariant method

In this optimization technique the computation inside the loop is avoided. for example

```
for i=0 to 10 do begin
K= i+a/b;
...
...
...
end;
```

can be written as

```
t=a/b
for i=0 to 10 do begin
K= i+t;
...
...
...
end;
```

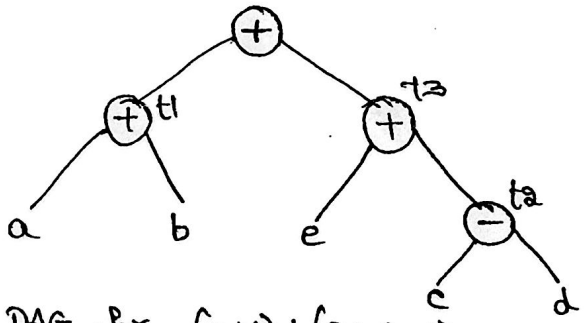
Instruction Scheduling

→ The order of three address code affects the cost of the object code being generated. By changing the order in which computations are done we can obtain the object code with minimum cost. Thus code optimization can be achieved by scheduling the instructions in proper order. This technique of code optimization is called instruction scheduling.

for example

```
t1 = a+b
t2 = c-d
t3 = e+t2
t4 = t1+t3
```

for the expression (a+b)+(e+(c-d))



DAG for (a+b)+(e+(c-d))

The code can be generated by translating the three address code line by line.

```
MOV  a, R0
ADD  b, R0      /* R0 = a+b */
MOV  c, R1
SUB  d, R1      /* R1 = c-d */
MOV  R0, t1     /* t1 = a+b */
MOV  e, R0
ADD  R0, R1     /* R1 = e+(c-d) */
MOV  t1, R0     /* R0 = a+b */
ADD  R1, R0     /* R0 = (a+b)+(e+(c-d)) */
MOV  R0, t4
```

Now if we change the ordering sequence of the above three address code

```
t2 = c-d
t3 = e+t2
t1 = a+b
t4 = t1+t3
```

then we get an improved code as

```
MOV  c, R0
SUB  d, R0
MOV  e, R1
ADD  R0,R1
MOV  a, R0
ADD  b, R0
ADD  R1, R0
MOV  R0, t4
```

Thus optimized code can be obtained by instruction scheduling.

Inter Procedural Optimization (IPO)

Inter-procedural optimization technique is a kind of code optimization in which collection of optimization techniques are used to improve the performance of the program that contain many frequently used functions of blocks. IPO reduces or eliminates duplicate calculations, inefficient use of memory and to simplify loops.

IPO may re-order the routines for better memory utilization The compiler tests for branches that are never taken and removes the code in that branch.

IPO also checks if constants are used properly or not.

Consider the following flow graph to perform interprocedural Optimizations (IPO) on the code.