CODE OPTIMIZATION

The Code optimization is required to produce an efficient code. The improvement over Intermediate code by transformation is called optimization.

Code optimization
- machine Independent
- Machine Dependent

there are three Criteria that we applied optimization.

(I) Must preserve meaning of program
(2) speedup program by measurable amount of time.
(II) must worth the effort.

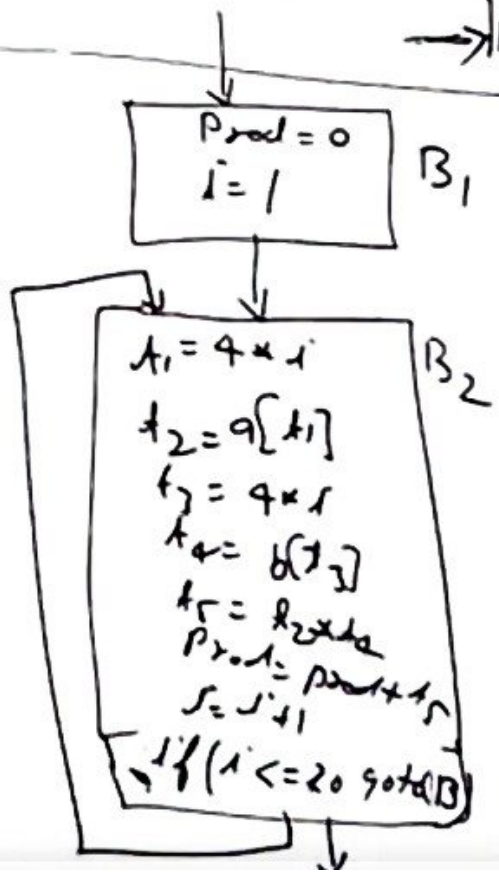
Term Used in Code optimization

1. Basic Blocks :— Break Intermediate Code into Blocks
2. flow Graph ⟶ Graphical Representational Basic Block

Ex: Convert following program into three Address code and Identify the Basic Blocks.


```
begin
  Prod = 0;
  i = 1;
  do
    begin
      Prod = Prod + a[i] * b[i]
      i = i+1;
    end
  while i <= 20
end
```

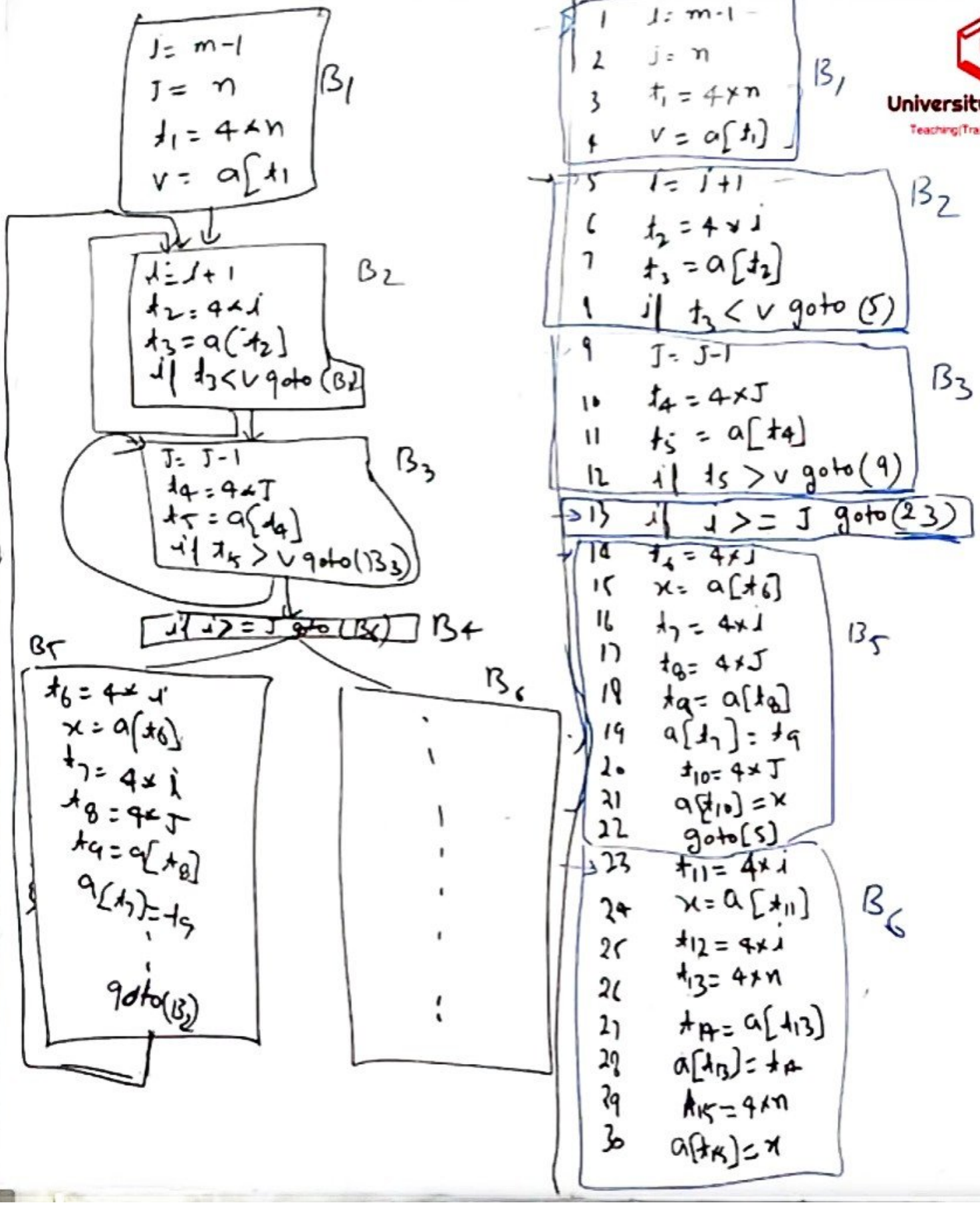
```
1.  Prod = 0
2.  i = 1
3.  t1 = 4 * i
4.  t2 = a[t1]
5.  t3 = 4 * i
6.  t4 = b[t3]
7.  t5 = t2 * t4
8.  Prod = Prod + t5
9.  i = i + 1
10. if i <= 20 goto (3)
```



CODE OPTIMIZATION

Local — Global

1. Function preserving transformation
   (a) Common Sub expression elimination (b) Copy propagation
   (c) dead code elimination (d) Constant folding
2. Loop Optimization
   (a) Code motion (b) Induction variable.
   (c) Reduction in strength.

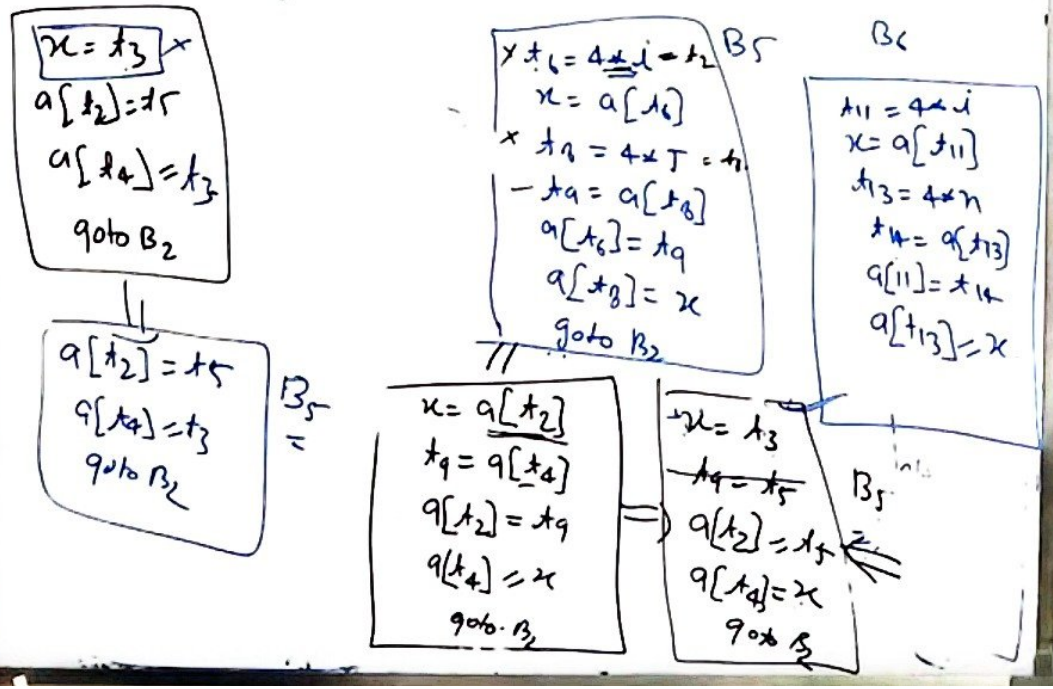

J= m-1
I= n
t1 = 4*n
V = a[t1
B1
I = I+1
t2 = 4*I
t3 = a[t2]
if t3<V goto (B2)
B2
J= J-1
t4 = 4*J
t5 = a[t4]
if t5 > V goto(B3)
B3
if I >= J goto (B5)
B4
t6 = 4*I
X = a[t6]
t7 = 4*I
t8 = 4*J
t9 = a[t8]
a[t7] = t9
goto(B2)
B5
B6
1 J= m-1
2 I= n
3 t1 = 4*n
4 V = a[t1]
5 I = I+1
6 t2 = 4*I
7 t3 = a[t2]
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[t13] = t14
29 t15 = 4*n
30 a[t15] = X

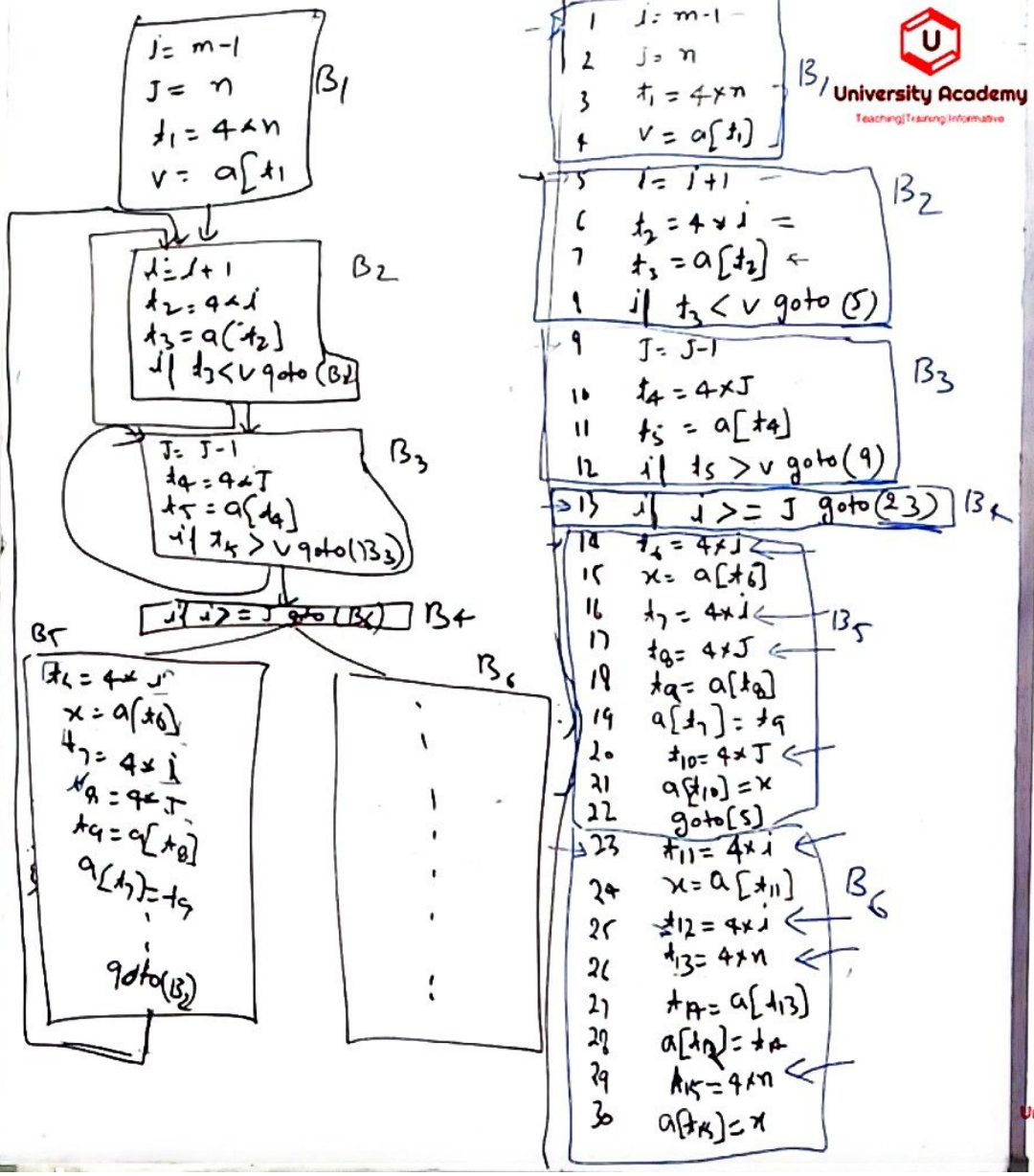
CODE OPTIMIZATION

- Local
- Global

1. Function Preserving transformation.
   (a) Common Sub expression elimination. (b) Copy propagation.
   (c) dead code elimination. (d) Constant folding
2. Loop Optimization.
   (a) Code motion. (b) Induction variable.
   (c) Reduction in strength.

Common Sub expression Elimination:





University Academy
Teaching | Learning | Informative

Loop Unrolling.

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

→

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

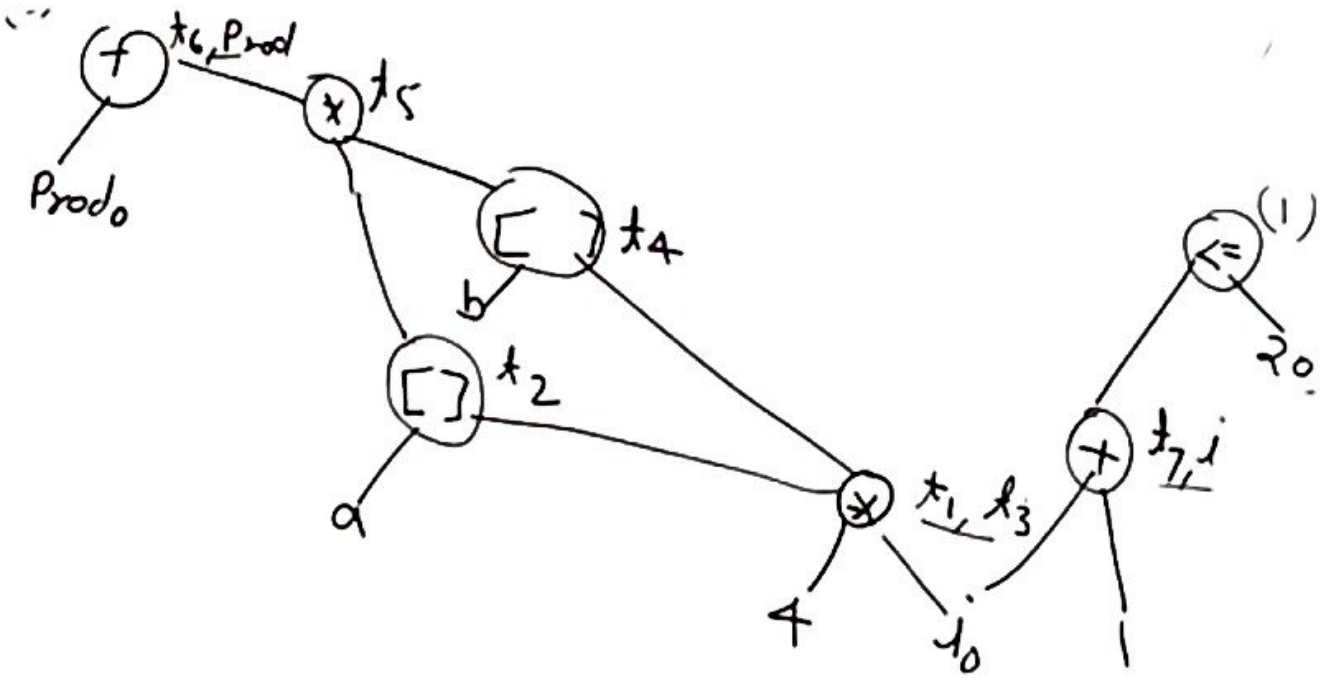
Loop fusion or Loop Jumping.

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

CODE OPTIMIZATION

Application of DAG

1. Determining Common subexpression and eliminate it.
2. Determining which Name are used inside but valued outside the Block.
3. Determining which statement of Block could have their computed value but used outside the Block.



CODE OPTIMIZATION

Application of DAG

1. Determining Common subexpression and eliminate it.
2. Determining which Name are used inside but valuated outside the Block.
3. Determining which statement of Block could have their computed value but used outside the Block.

1. $a = b * c$
2. $d = b$
3. $e = d * c$
4. $b = e$
5. $f = b + c$
6. $g = f + d$

Ex Block of three Address Code is given below.

t_1 t_2 t_3 t_5 t_6 t_7

1. $t_1 = 4 * i$
2. $t_2 = a[t_1]$
3. $t_3 = 4 * i$ X
4. $t_4 = b[t_3]$
5. $t_5 = t_2 * t_4$
6. $t_6 = prod + t_5$
7. $prod = t_6$
8. $t_7 = i + 1$
9. $i = t_7$
10. if $i <= 20$ goto (1)

B1

1. $t_1 = 4 * i$
2. $t_2 = a[t_1]$
3. $t_4 = b[t_1]$
4. $t_5 = t_2 * t_4$
5. $prod = prod + t_5$
6. $i = i + 1$
7. if $i < 20$ goto (1)

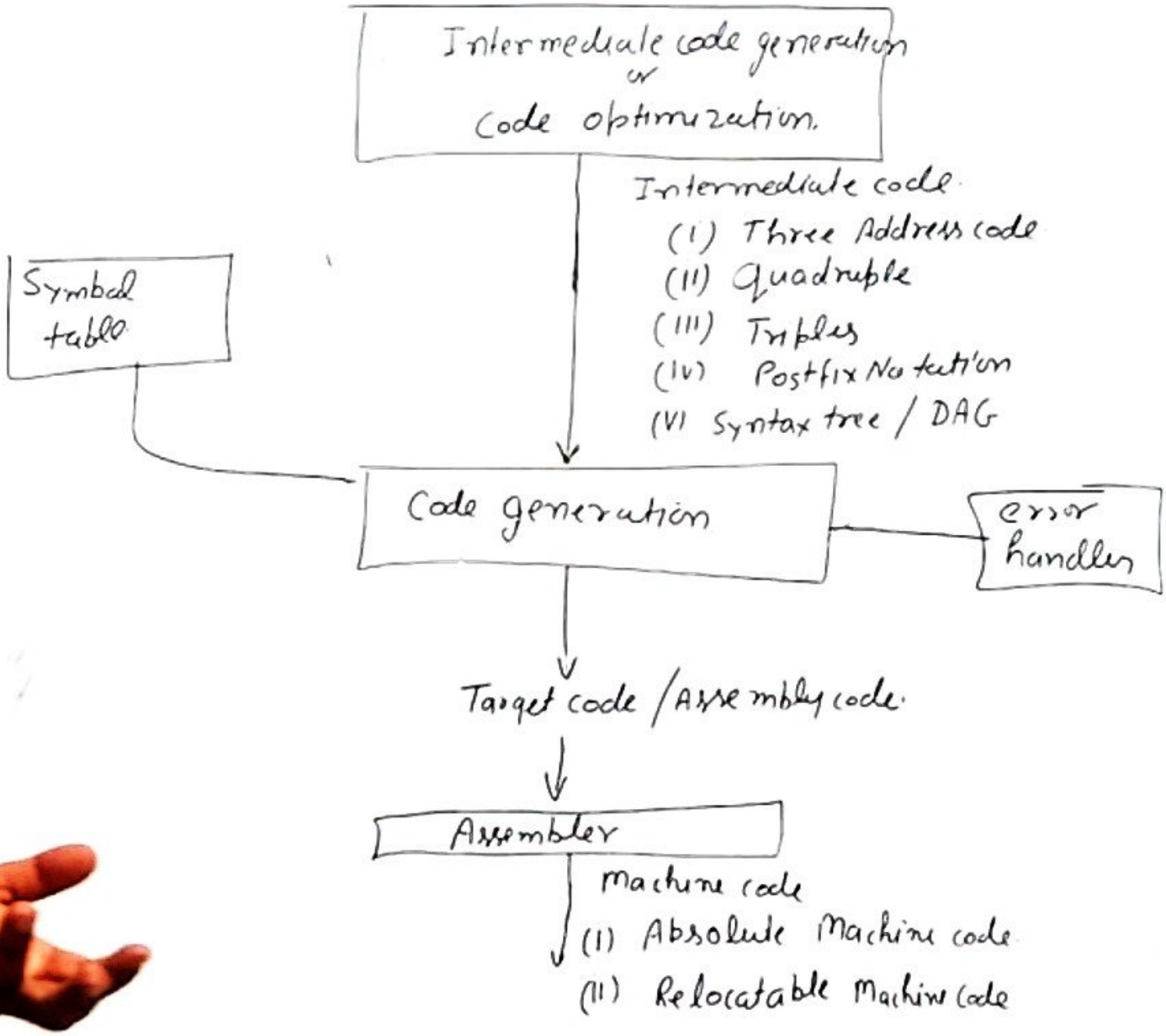
CODE GENERATION

Code generation is the final activity of compiler code generation is the process of creating Assembly / machine language. There are some properties of code generation

(I) Correctness
(II) high Quality
(III) Efficient use of resource of target machine.
(IV) Quick code generation

Absolute Machine Code: Fixed location in memory and Imidiatly executed. Useful for small program

Relocatable Machine Code: Not a fixed locatin, code can be placed wherever linker find the room in RAM.
Usefull for commercial combiles

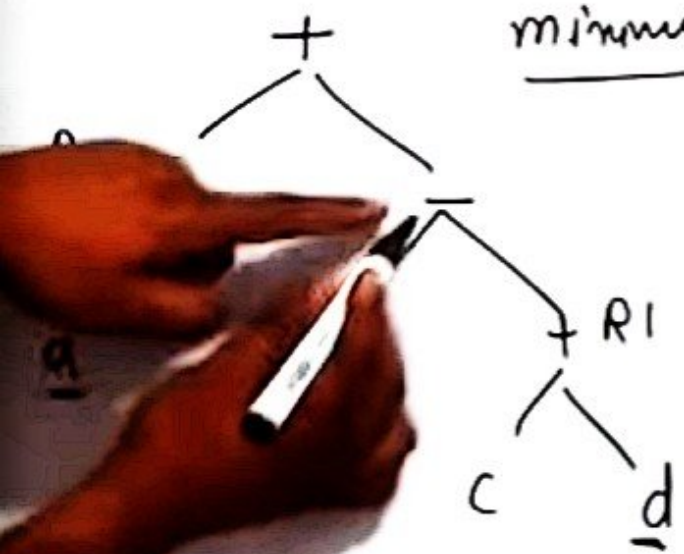

Intermediate code generation
or
Code optimization.
Intermediate code
(I) Three Address code
(II) Quadruple
(III) Triples
(IV) Postfix Notation
(VI Syntax tree / DAG
Symbol table
Code generation
error handler
Target code / Assembly code.
Assembler
Machine code
(I) Absolute Machine code
(II) Relocatable Machine code

CODE GENERATION

Issue in code generation

1. Input to Code generation.
2. Target program
3. Memory Management
4. Instruction selection
5. Register Allocation

minimum Registers



R0 ← a
R1 ← b
R0 ← R0 − R1
R1 ← c
R2 ← d
R1 ← R1 + R2
R2 ← e
R2 − R1

CODE GENERATION

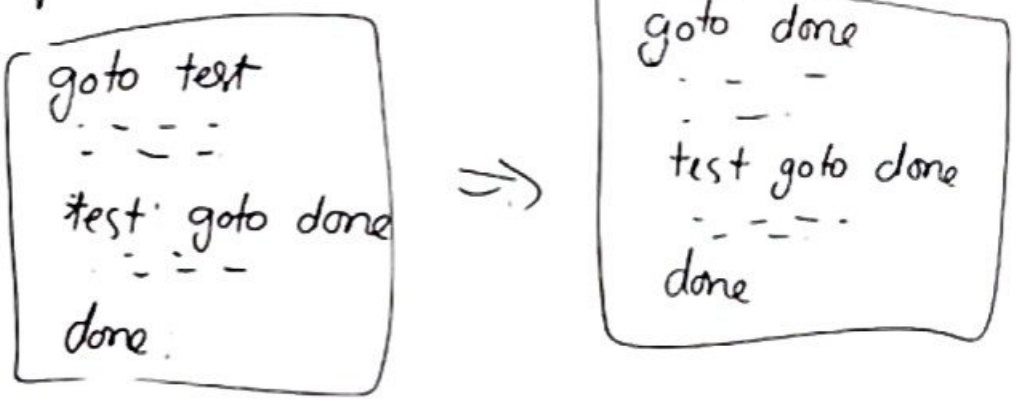
PEEPHOLE OPTIMIZATION

A statement by statement code generation strategy generate target code that contain redundant Instruction. to optimize such target code we use Peephole optimization technique.

1. Redudent Instruction

(i) mov R0, a
(ii) mov a R0
} eliminate (ii) instruction

(2) Flow of control optimization



③ Algebraic simplification

$x = x + 0$
or
$x = x * 1$
} eliminate such Instruction

④ Reduction strength.

Addition is cheaper than multiplication

Substraction is cheaper than division.

⑤ Machine idioms

$x = x + 1$ → Remove and use Auto Increment.

$x = x - 1$ → Remove and use Auto decrement.