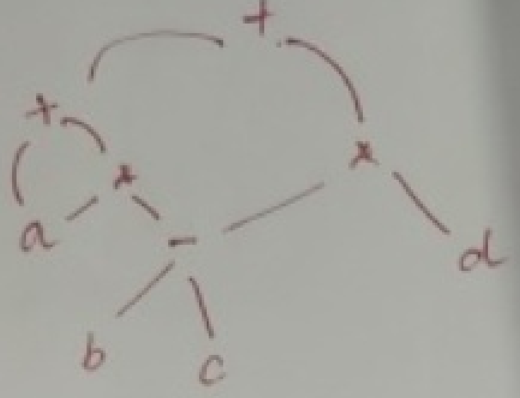
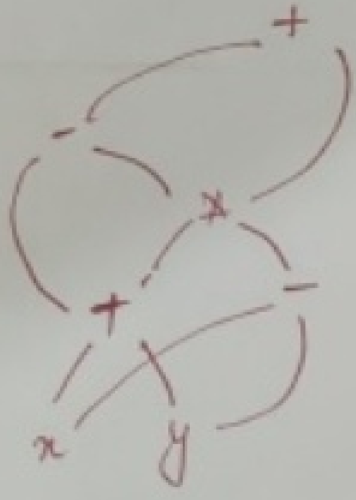
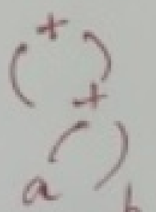
1) $a + a * (b - c) + (b - c) * d$



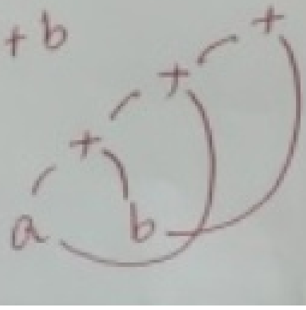
2) $((x + y) - ((x + y) * (x - y))) + ((x + y) * (x - y))$

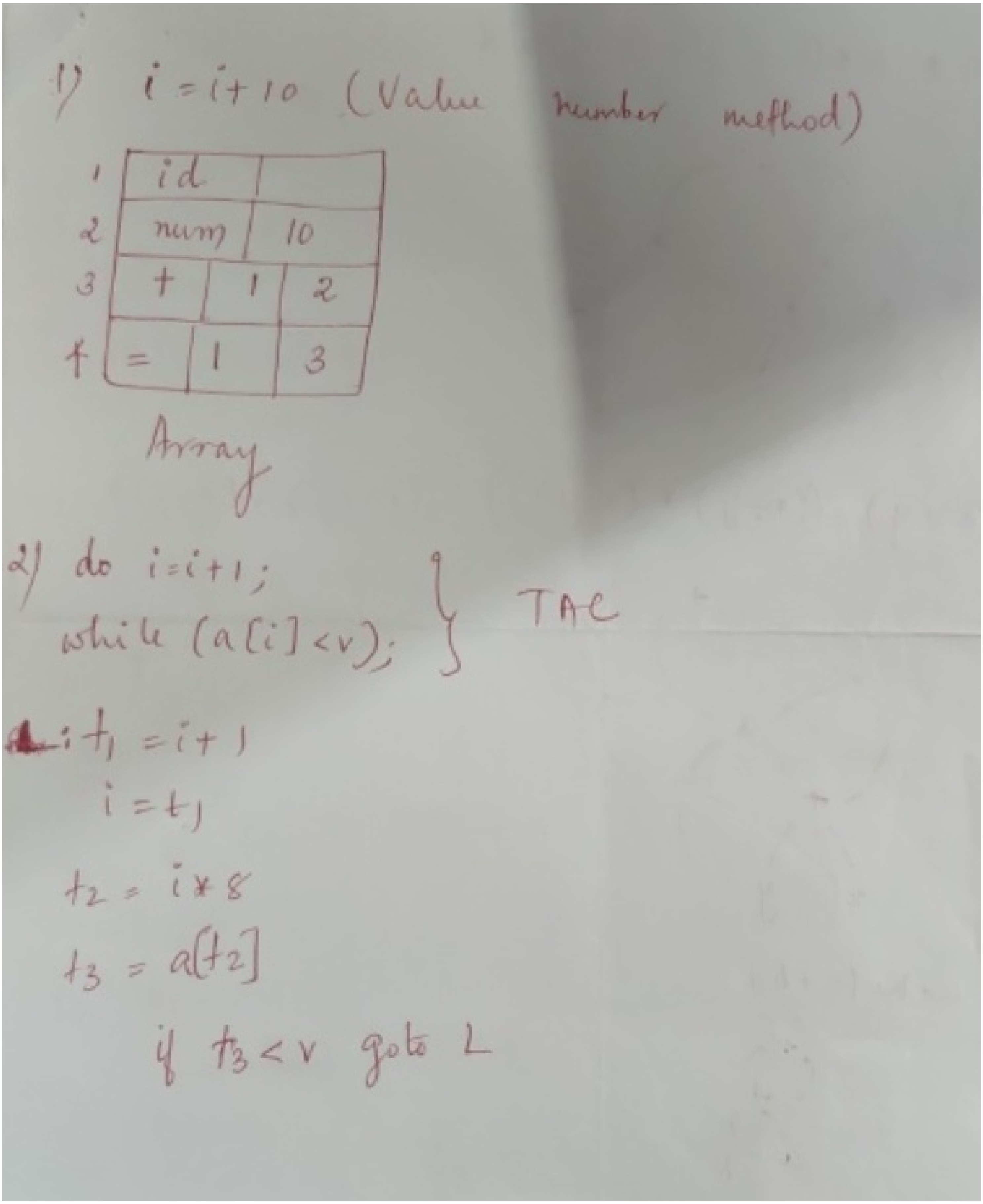


3) $a + b + (a + b)$



4) $a + b + a + b$





1) $i = i + 10$ (Value number method)

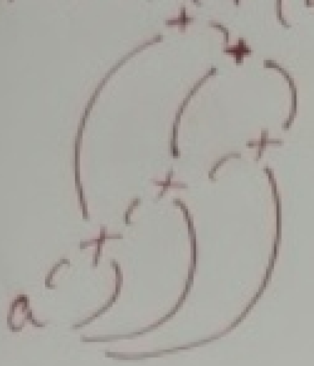
1	id		
2	num	10	
3	+	1	2
4	=	1	3

Array

2) do $i = i + 1$;
while $(a[i] < v)$; } TAC

L: $t_1 = i + 1$
$i = t_1$
$t_2 = i * 8$
$t_3 = a[t_2]$
if $t_3 < v$ goto L

(5) a+a+((a+a+a+(a+a+a+a)))



(6)

main () {
    int i
  int a[10];
  While (i <=10)
      a[i] = 0;
}

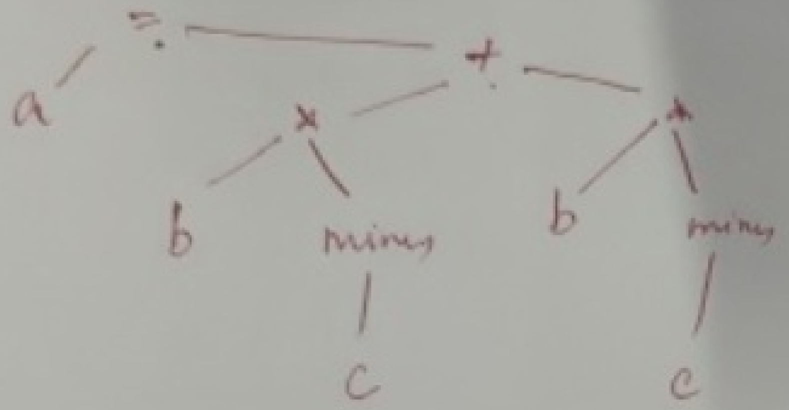
TAC

L1: if i<=10 goto L2
  goto L3

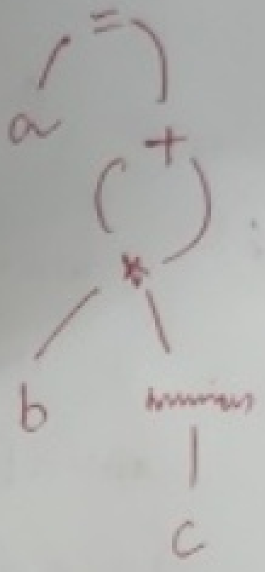
L2 t1 = i * 4;
  ~~= a[t1]~~
  ~~t2 = a~~
  a[t1] = 0;
  goto L1
L3 : end

(7) $a = b * -c + b * -c$;

Syntax tree



DAG



(8) $x[i] = y$

Ans => $t_1 = i*4$

$x[t_1] = y$

(9) $x = y[i]$

$t_1 = i*4$

$x = y[t_1]$

(10) $n = f(a[i])$;

$t_1 = i*4$

$t_2 = a[t_1]$

param t_2

$t_3 =$ call $f, 1$

$n = t_3$.

$a := (-a+b) * ((-a+b) * c)$

t_1 = minus a

$t_2 = t_1 + b$

$t_3 = t_2 * c$

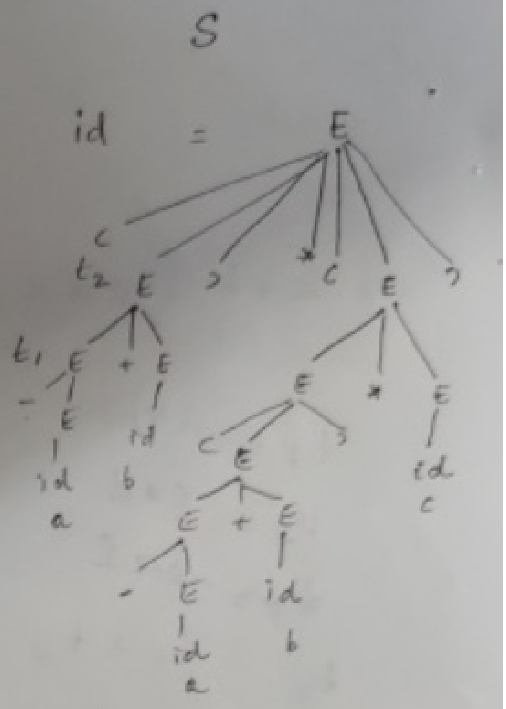
$t_4 = t_2 * t_3$

$a := t_4$

t_3 = minus a

$t_4 = t_3 + b$

$t_5 = t_4 * c$



$y = a+b$

$x[i] = y$

$z = z+2$

$y[i] = z$

$y = a+b$

$t_1 = a+b$

$y = t_1$

$t_2 = i * 4$

$x[t_2] = y$

$t_3 = z+2$

$z = t_3$ $\rightarrow t_4 = i*4$

$y[t_4] = z$

$x = f(0, y+1) - 1$

$t_1 = y+1$

param 0

param t_1

call f, 2
return t_2 } or t_2 = call f, 2
return t_2 //optional

↳ //optional

$t_3 = t_2 - 1$

$x = t_3$

$x = a*b + c*d - e*f$

$t_1 = a*b$

$t_2 = c*d$

~~$t_3 = e*f$~~

$t_3 = t_1 + t_2$

$t_4 = e*f$

$t_5 = t_3 - t_4$

$x = t_5$

$-(a+b)*(c+d)+(a+b+c)$

$t_1 = a+b$

$t_2 =$ minus t_1

$t_3 = c+d$

$t_4 = t_2 * t_3$

$t_5 = t_1 + c$

$t_6 = t_4 + t_5$

$a = -b * (c+d)/e$

$t_1 =$ minus b

$t_2 = c+d$

$t_3 = t_1 * t_2$

4) TAC

1) i:=1

2) j:=1

3) t_1 = i*8

4) t_2 = a[t_1]

5) t_3 = j*c

6) t_4 = t_2 + t_3

7) j = t_4

8) t_5 = i+1

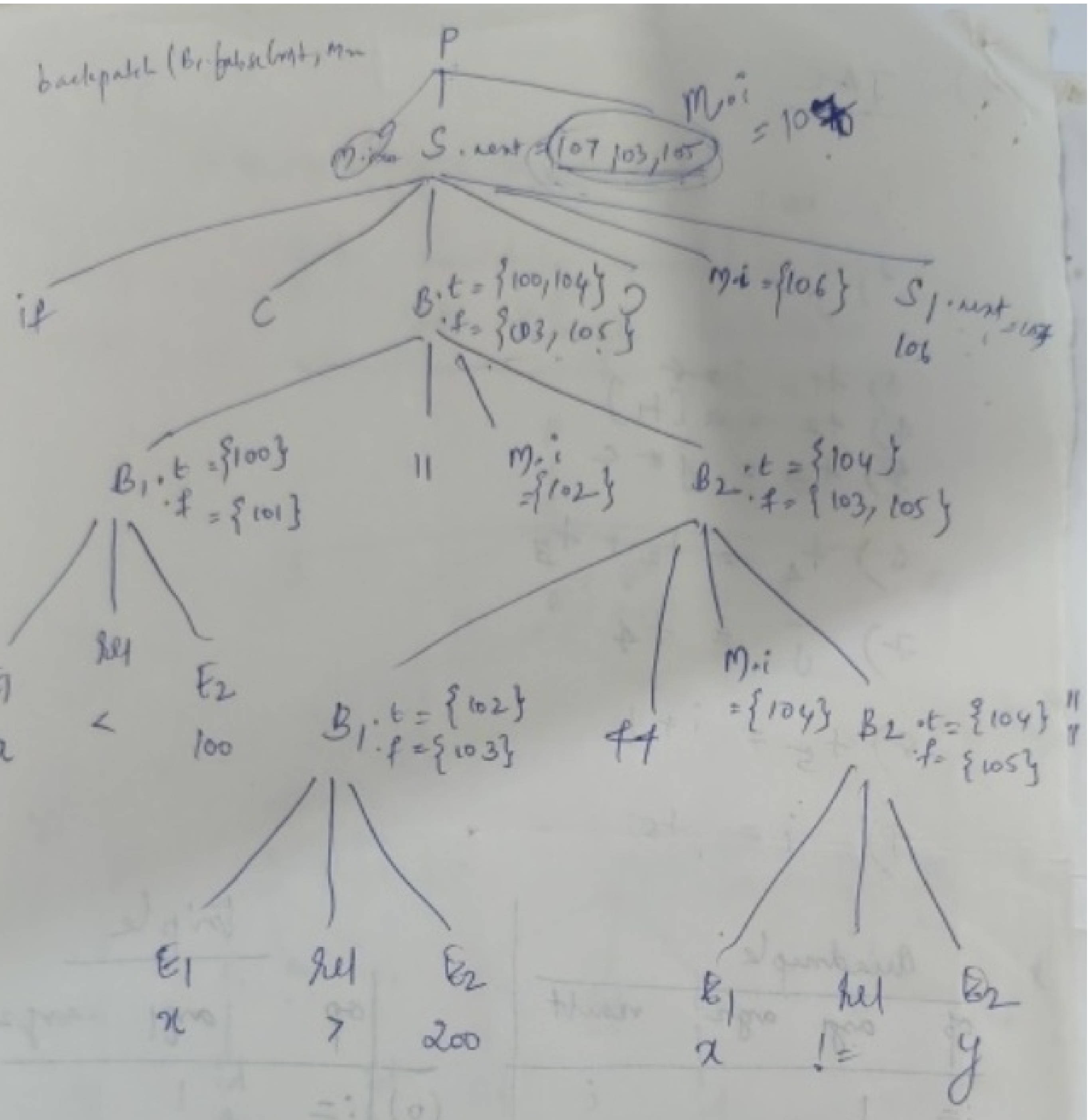
9) i = t_5

Quadruple

op	arg1	arg2	result
:=	1		i
:=	1		j
*	i	8	t_1
=[]	a	t_1	t_2
*	j	c	t_3
+	t_2	t_3	t_4
:=	t_4		j
+	i	1	t_5
:=			

triple

	op	arg1	arg2
(0)	:=	1	
(1)	:=	1	
(2)	*	~~(0)~~i	8
(3)	=[]	a	(2)
(4)	*	j	c
(5)	+	(3)	(4)
(6)	:=	(5)	
(7)	+	~~[illegible]~~i	1
(8)	:=	(7)	

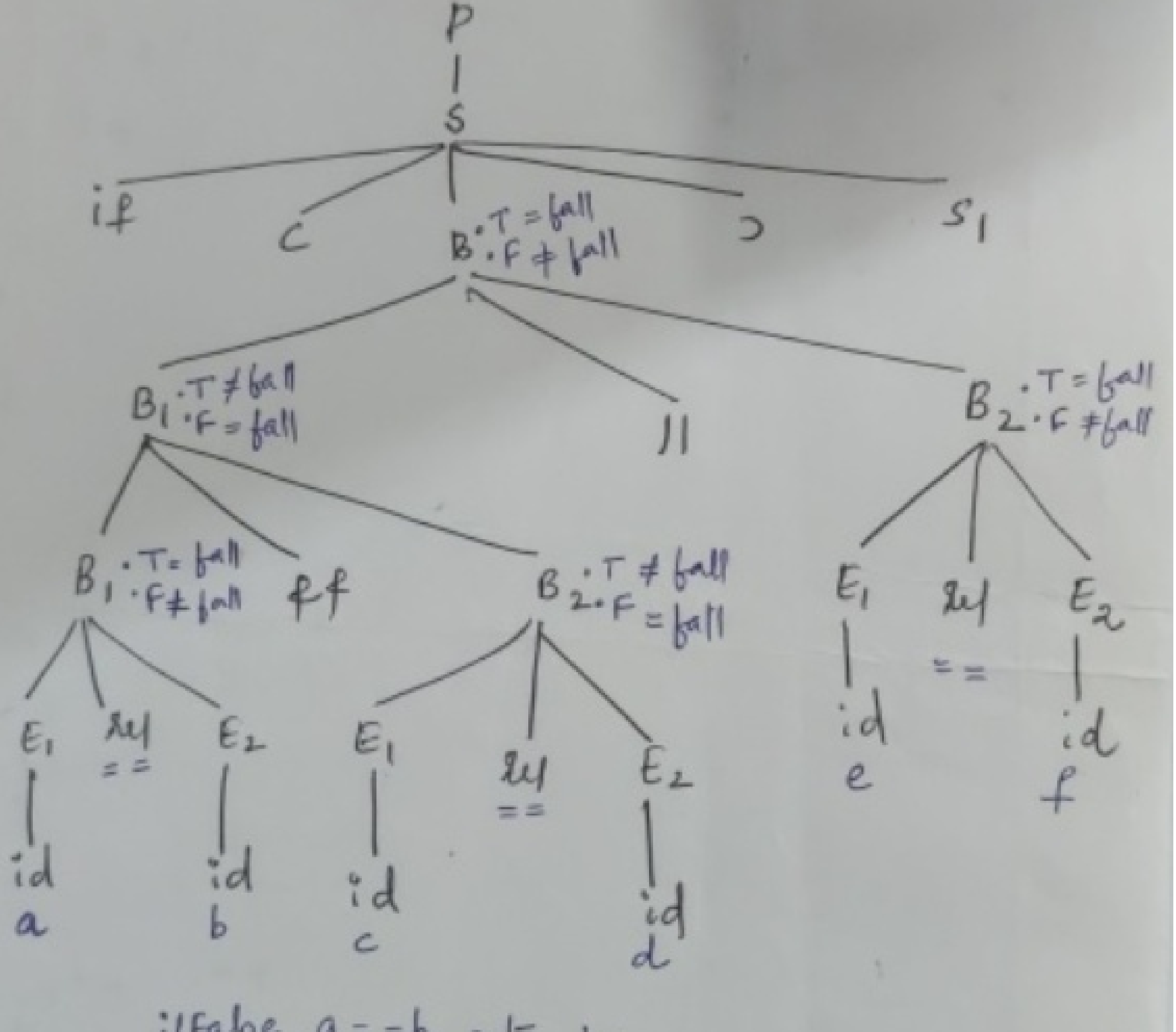


```
100 if x < 100 goto 106
101      goto 102
102 if x > 200 goto 104
103    goto 105
104 if x != y got 106
105      goto 107
```

106 :
107 :

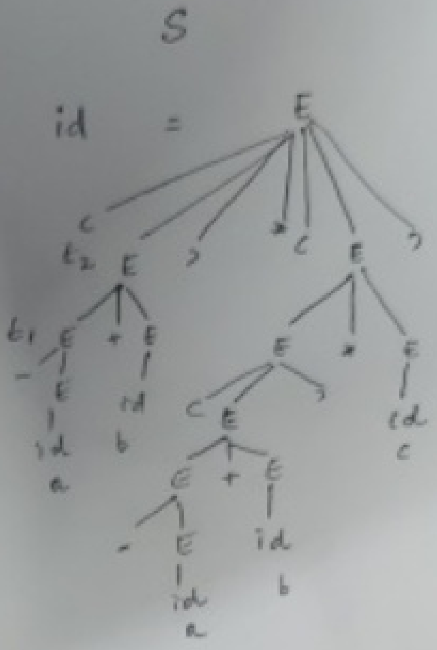
~~b) x = a[i][j] + b[i][j]~~

goto - avoiding translation scheme
fall through technique



```
   ifFalse a==b goto L4
   if c==d goto L2
L4: ifFalse e==f goto L1
L2: x==1
L1:
```

$a := (-a+b) * ((-a+b) * c)$



$t_1 =$ minus a

$t_2 = t_1 + b$

$t_3 = t_2 * c$

$t_4 = t_2 * t_3$

$a := t_4$

$t_3 =$ minus a

$t_4 = t_3 + b$

$t_5 = t_4 * c$

$y = a+b$

$x[i] = y$

$z = z+2$

$y[i] = z$

$y = a+b$

$t_1 = a+b$

$y = t_1$

$t_2 = i*4$

$x[t_2] = y$

$t_3 = z+2$

$z = t_3$ → $t_4 = i*4$

$y[t_4] = z$

$x = f(0, y+1) - 1$

$t_1 = y+1$

param 0

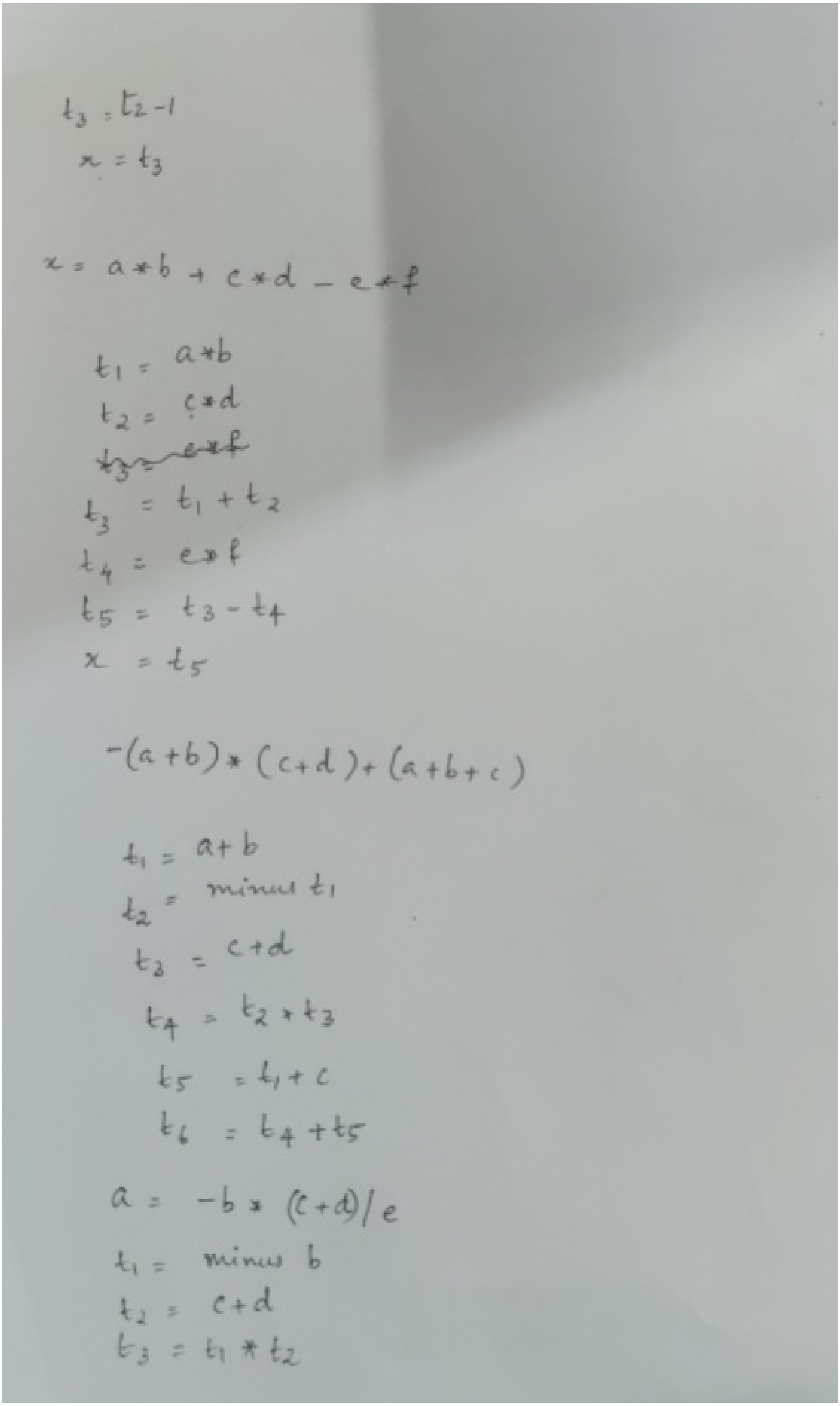
param t_1

call f, 2 }

return t_2 } or $t_2 =$ call f, 2

↳ //optional

return t_2 //optional



$t_3 = t_2 - 1$

$x = t_3$

$x = a*b + c*d - e*f$

$t_1 = a*b$

$t_2 = c*d$

~~$t_3 = e*f$~~

$t_3 = t_1 + t_2$

$t_4 = e*f$

$t_5 = t_3 - t_4$

$x = t_5$

$-(a+b)*(c+d)+(a+b+c)$

$t_1 = a+b$

$t_2 =$ minus t_1

$t_3 = c+d$

$t_4 = t_2 * t_3$

$t_5 = t_1 + c$

$t_6 = t_4 + t_5$

$a = -b * (c+d)/e$

$t_1 =$ minus b

$t_2 = c+d$

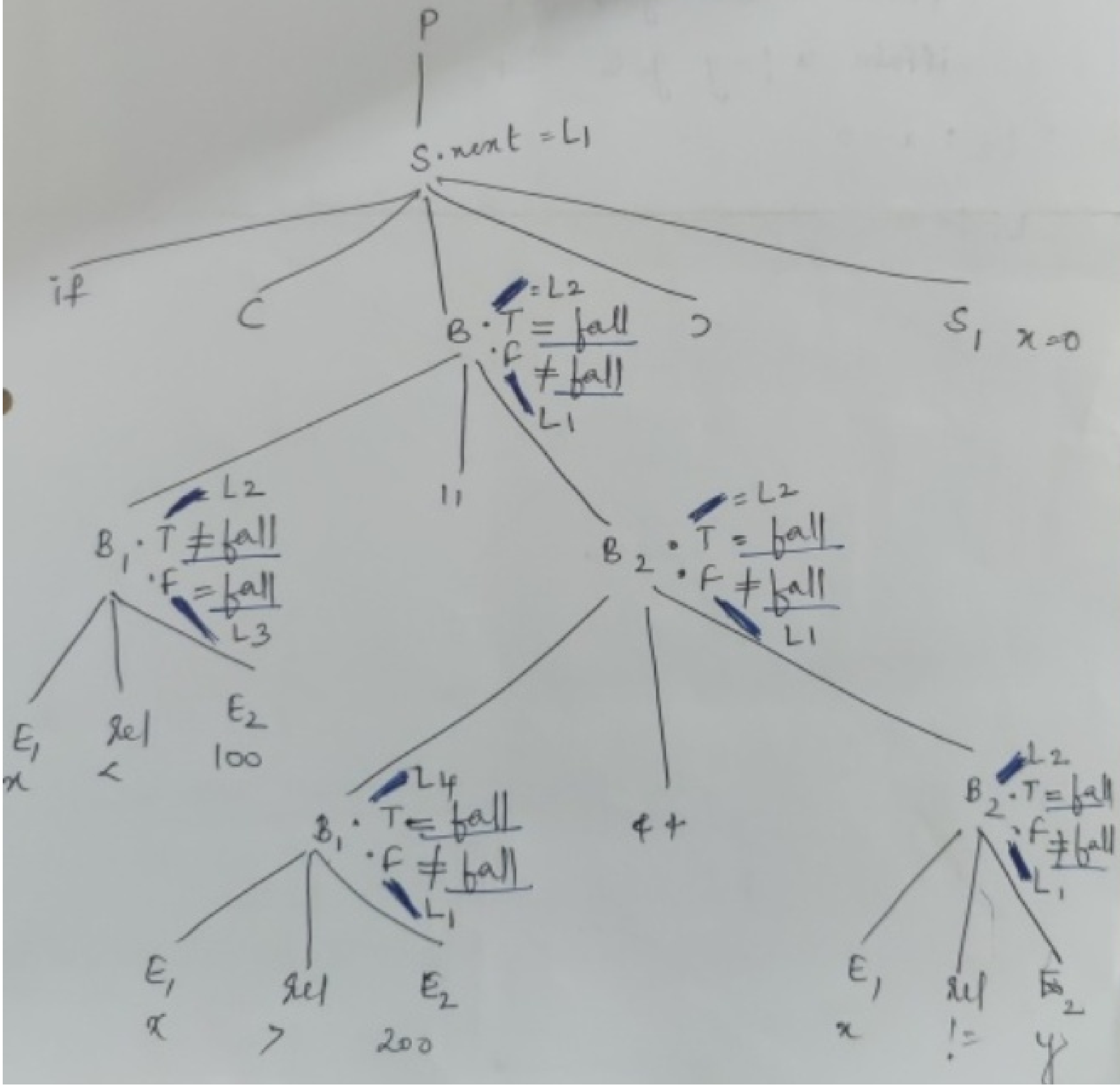
$t_3 = t_1 * t_2$

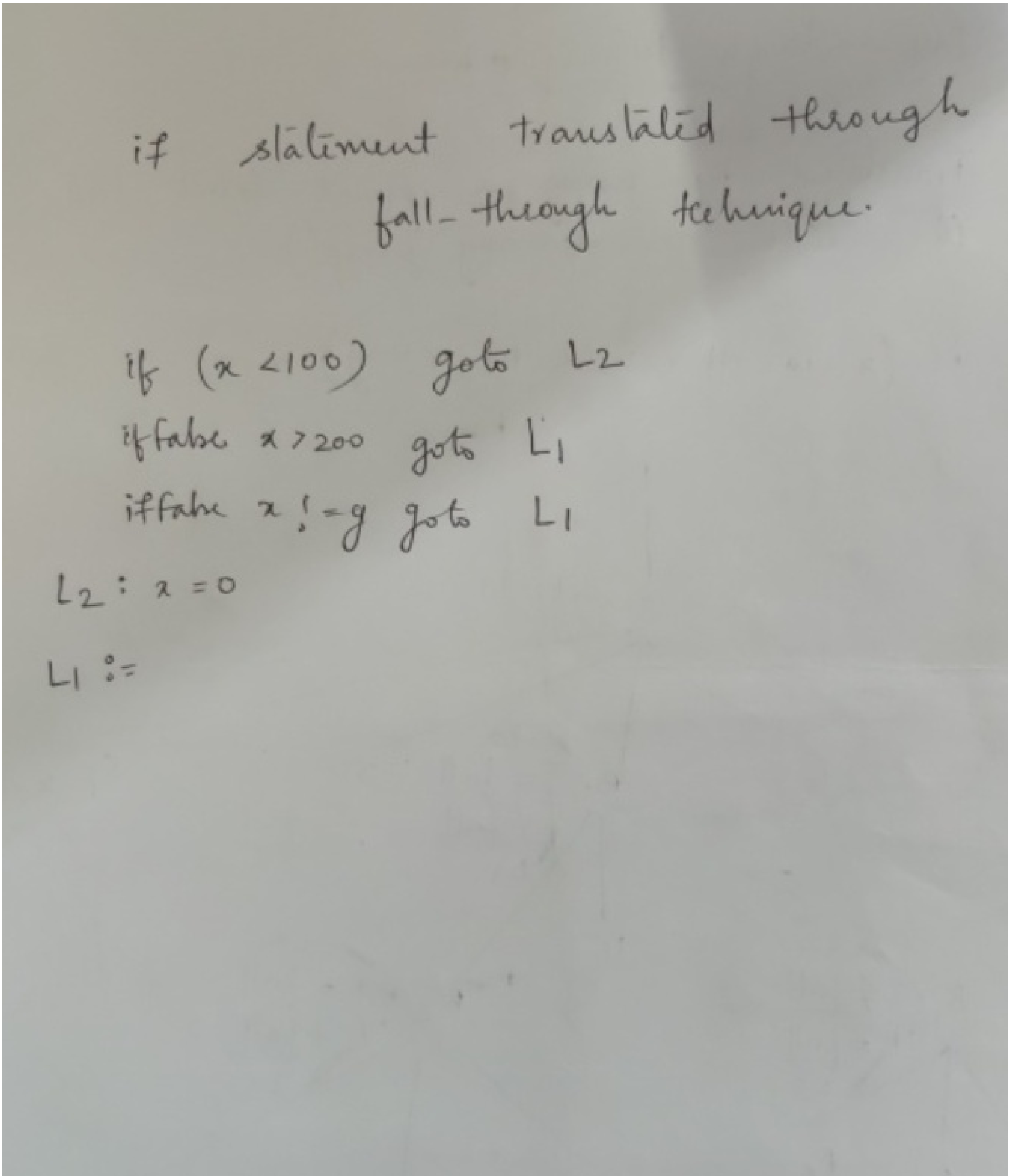
Avoiding Redundant Gotos

Fall-through technique

Translated by using a special label fall (i.e don't generate any jump')

if (x<100 || x >200 && x!=y) x=0;

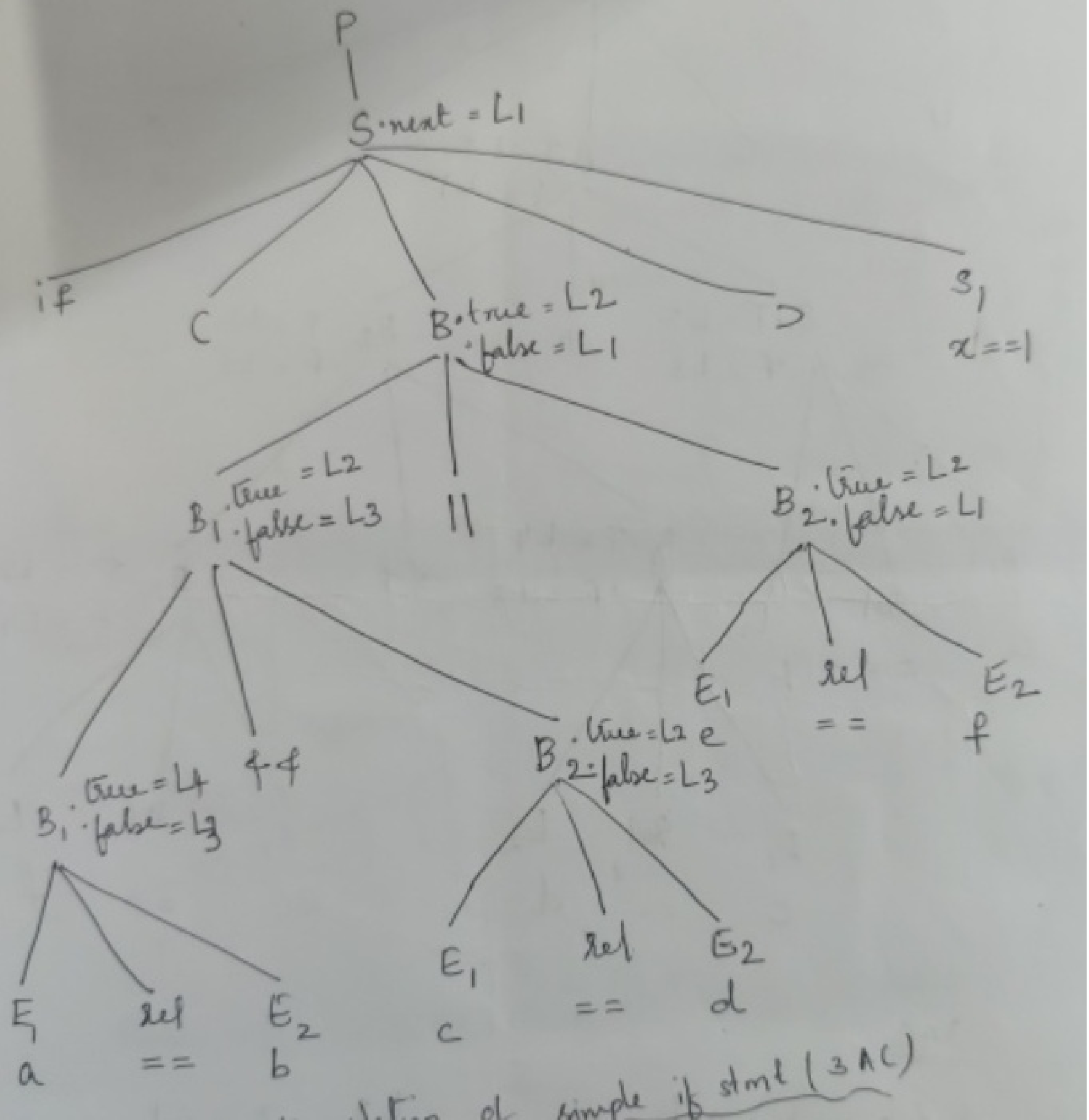




if statement translated through fall-through technique.

```
    if (x <100) goto L2
    iffalse x >200 goto L1
    iffalse x != y goto L1
L2 : x = 0
L1 :=
```

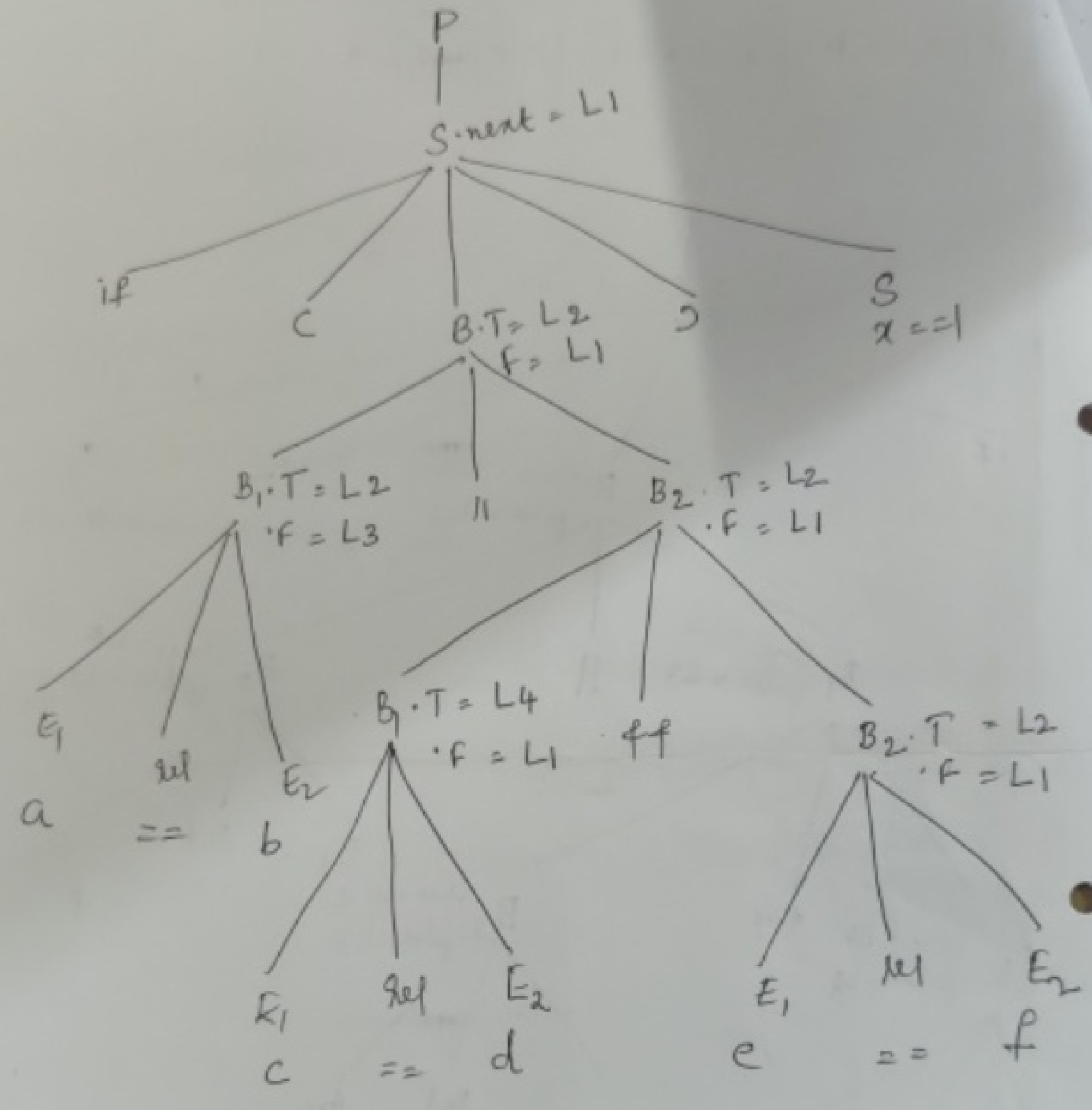
if(a==b && c==d || e==f) x==1;



Control flow translation of simple if stmt (3AC)

```
    if a==b goto L4
       goto L3
L4: if c==d goto L2
       goto L3
L3: if e==f goto L2
       goto L1

L2: x==1
L1:
```

if (a==b || c==d ++ e==f) x=1



3AC

if (a==b) goto L2
goto L3
L3: if c==d goto L4
goto L1
L4: if e==f goto L2
goto L1

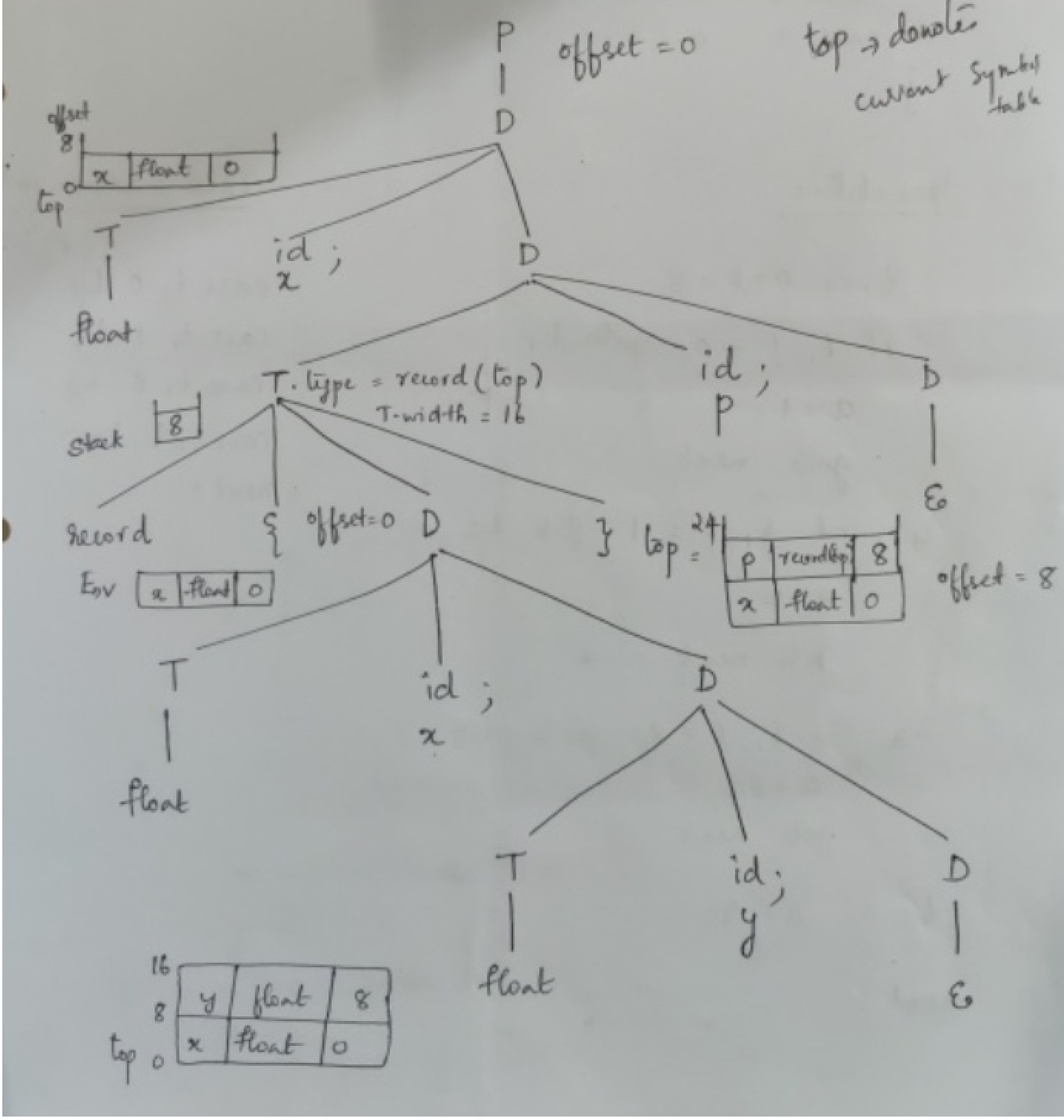
L2: x ==1

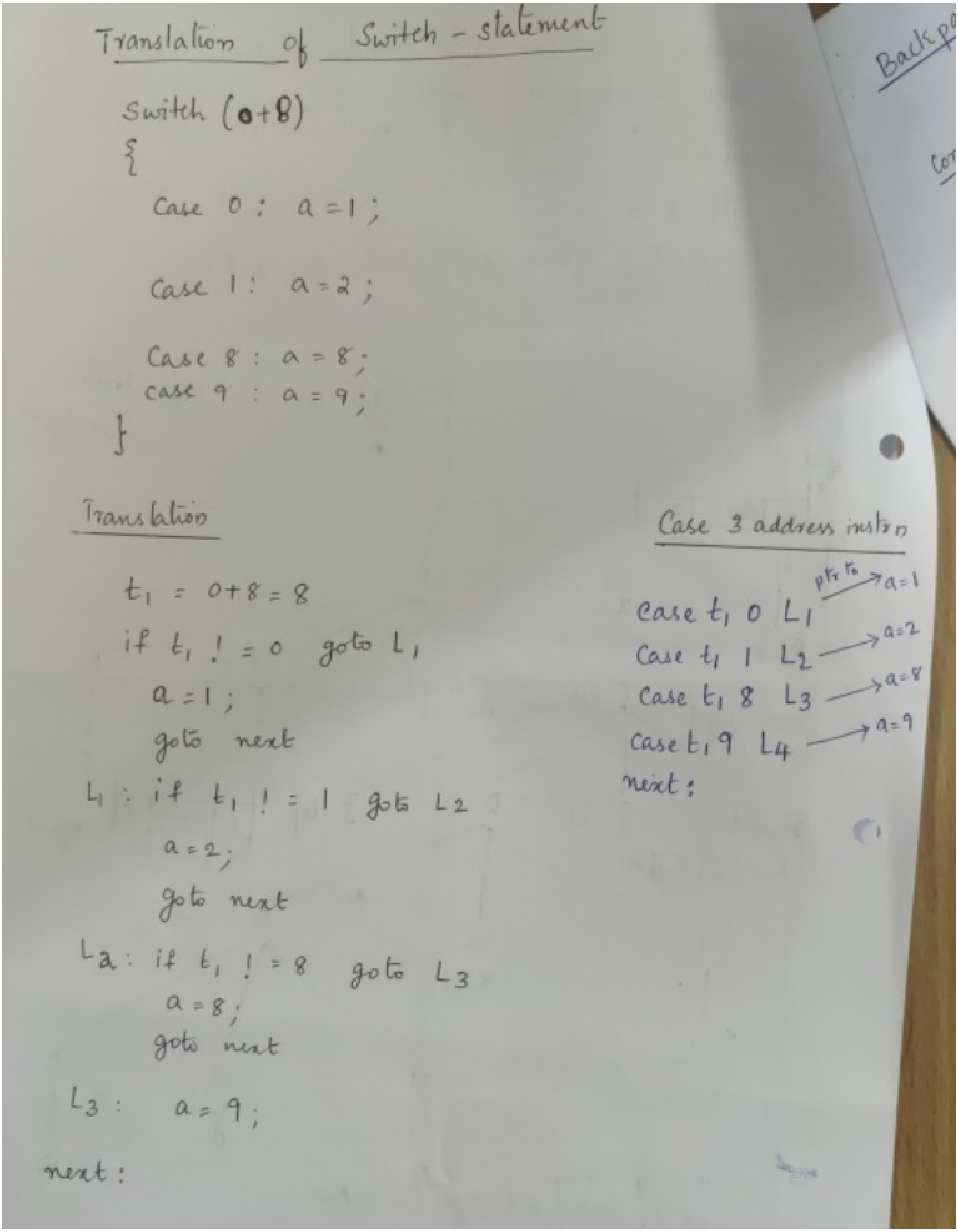
L1:

Determine the types and relative addresses for the identifiers in the following sequence of declarations:

float x ;

record { float x ; float y ; } p ;





Translation of Switch - statement

```
switch (0+8)
{
  Case 0 :  a = 1 ;

  Case 1 :  a = 2 ;

  Case 8 :  a = 8 ;
  case 9 :  a = 9 ;
}
```

Translation

```
    t1 = 0+8 = 8
    if t1 ! = 0  goto L1
      a = 1 ;
      goto next
L1 : if t1 ! = 1  goto L2
      a = 2 ;
      goto next
L2 : if t1 ! = 8  goto L3
      a = 8 ;
      goto next
L3 :  a = 9 ;

next :
```

Case 3 address instrn

case t_1 0 L_1 $\xrightarrow{\text{ptr to}}$ a=1

Case t_1 1 L_2 $\longrightarrow$ a=2

Case t_1 8 L_3 $\longrightarrow$ a=8

case t_1 9 L_4 $\longrightarrow$ a=9

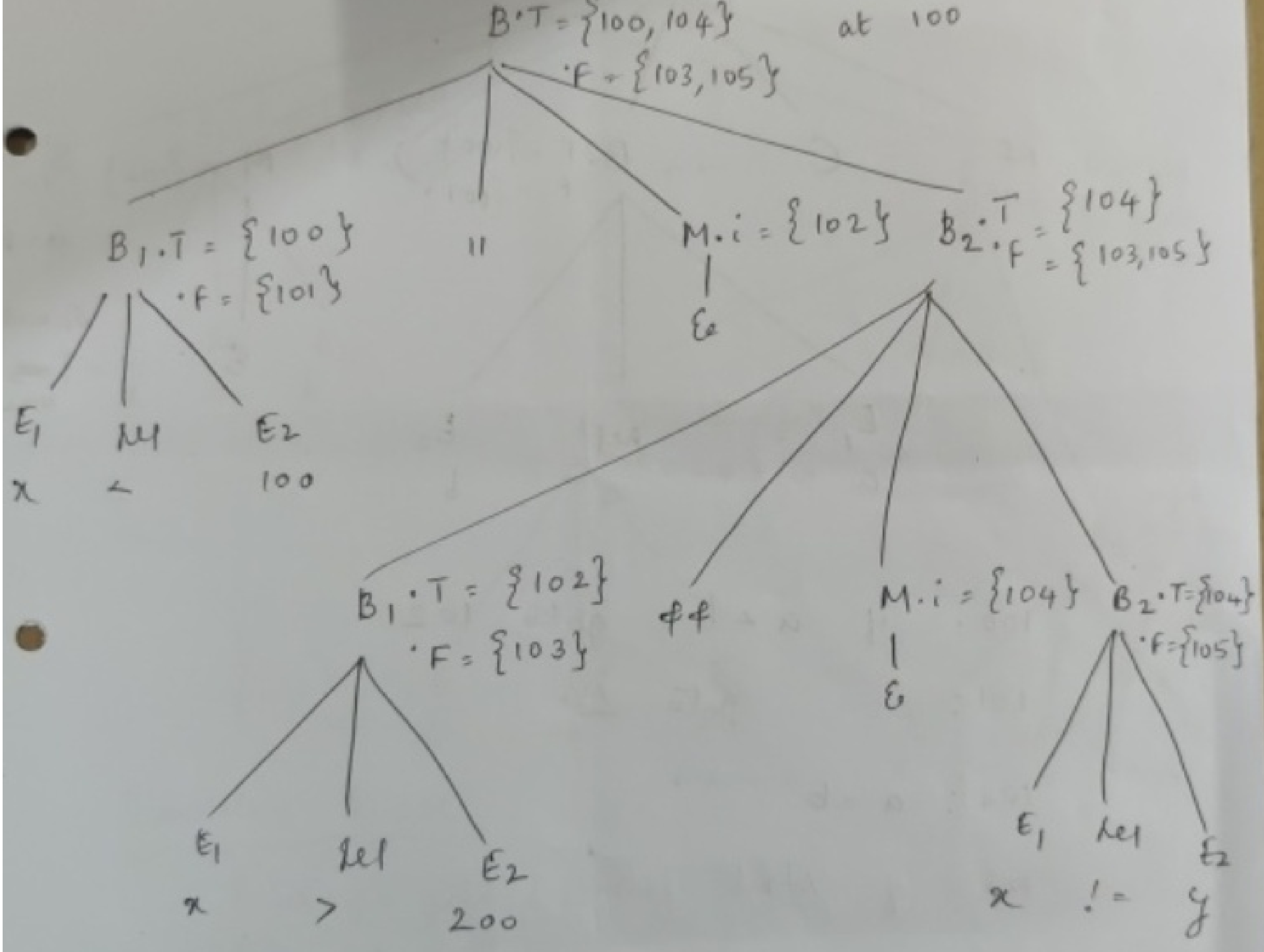
next :

Backpatching

lists of jumps are passed as synthesized attributes

Consider expression
$x < 100 \; || \; x > 200 \; \&\& \; x \; != y$

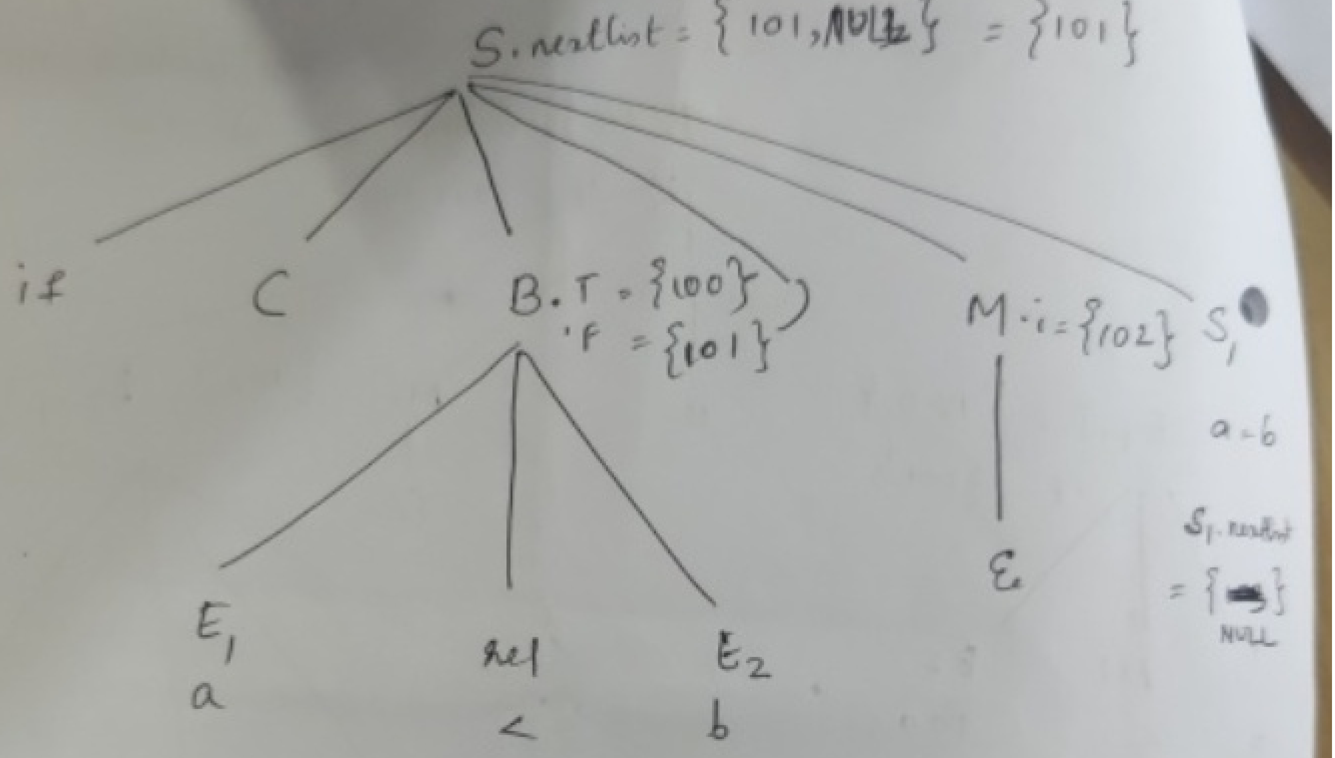
* Arbitrarily start instruction numbers at 100



```
100: if x < 100 goto —
101:     goto 102
102: if x > 200 goto 104
103:     goto —
104: if x != y goto
105:     goto —
```

```
106: if Stmt True
107: if Stmt False
```

if (a < b)   a = b;



100 : if a < b goto 102

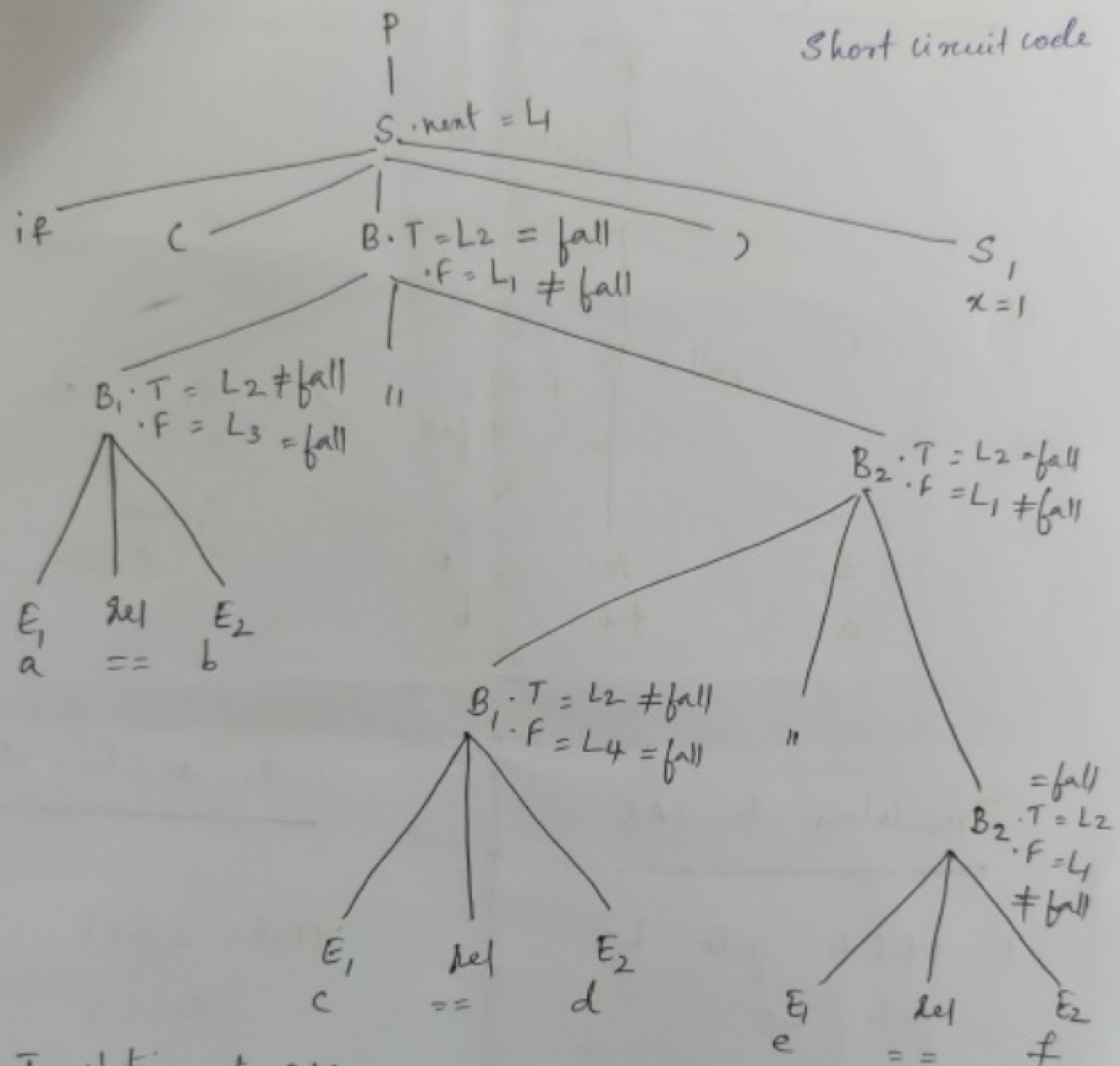
101 : goto 103

102 : a = b

103 : if stmt false

Goto Avoiding Scheme (Fall-through Technique)

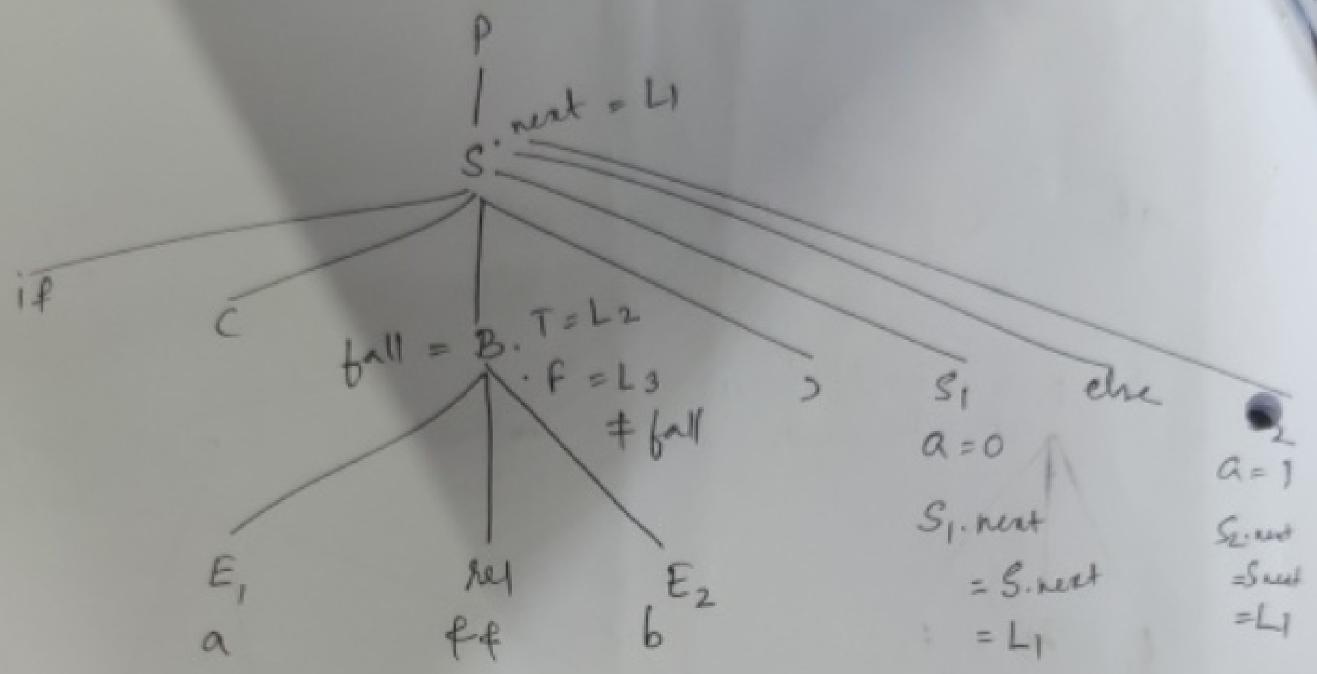
Short circuit code



Translation to 3AC

```
    if a==b goto L2
    if c==d goto L2
    ifFalse e==f goto L1

L2:  x = 1

L1:
```

if (a &&b) a=0 else a=1



Translation to 3AC

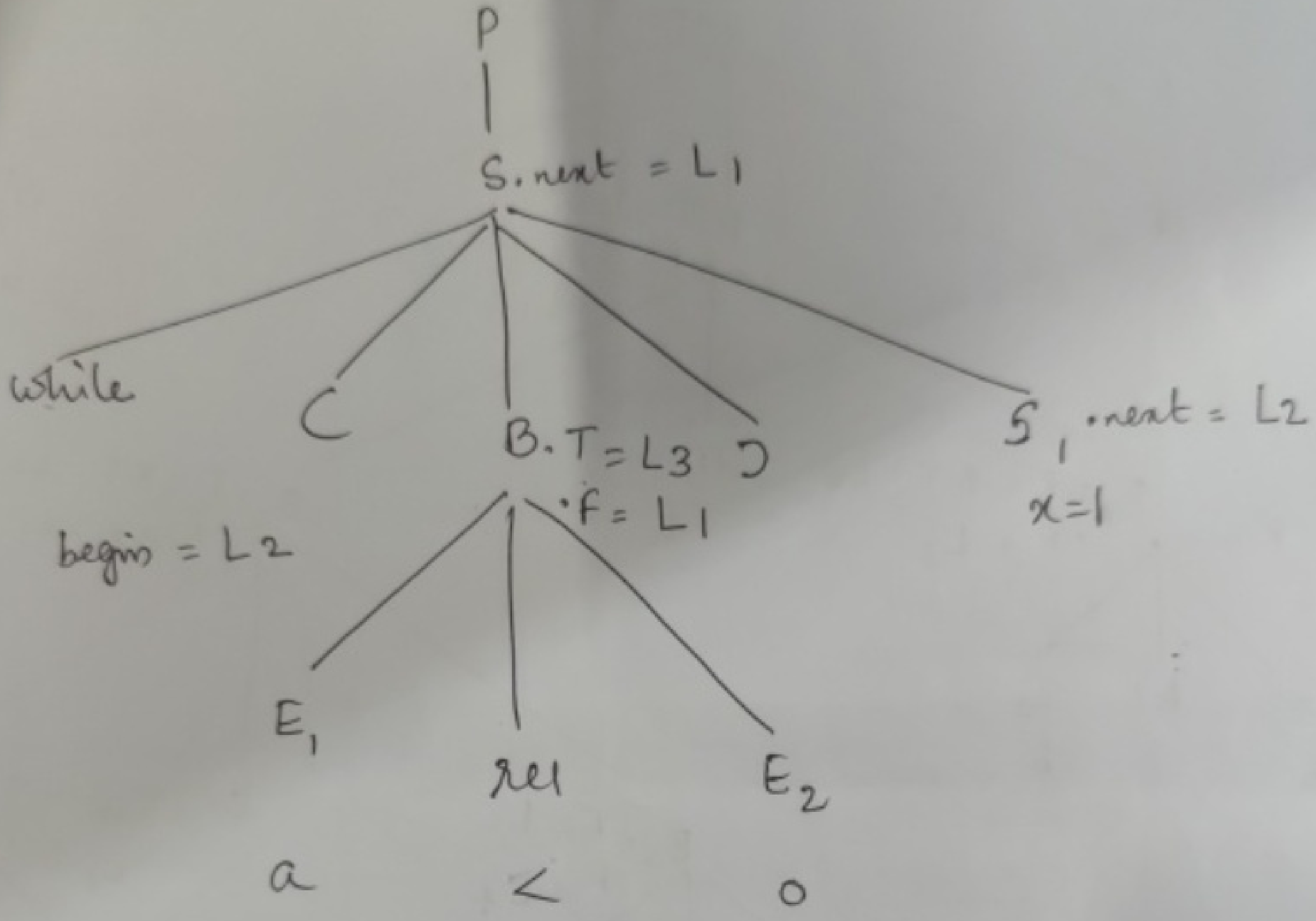
```
if a && b goto L2
    goto L3

L2 : a = 0

    goto L1

L3 : a = 1

L1 :
```

Goto Avoiding scheme

```
ifFalse a && b

    goto L3

L2 : a = 0
     goto L1
L3 : a = 1
L1 :
```

Control Flow

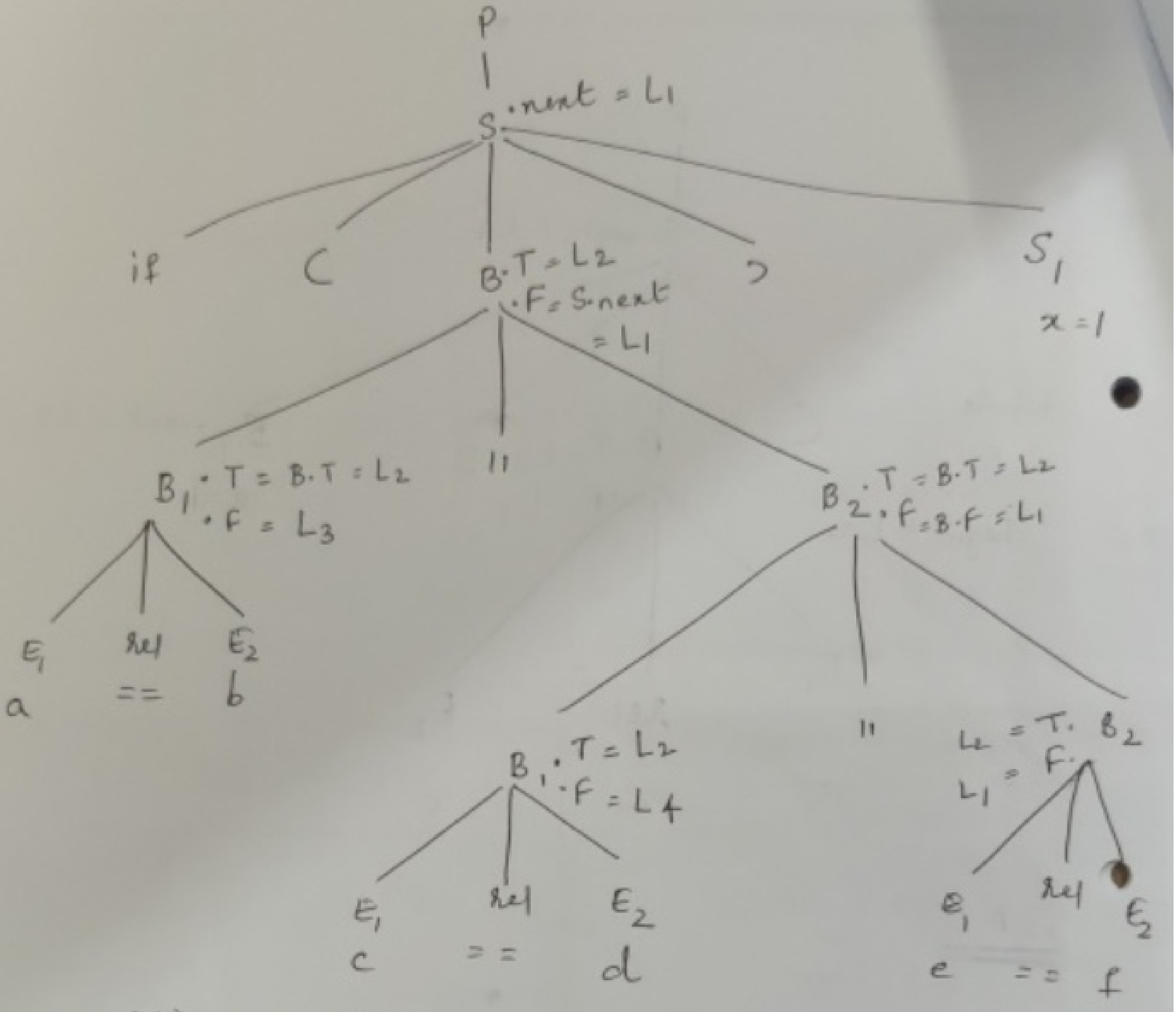
while (a<0) x=1;



3 A C

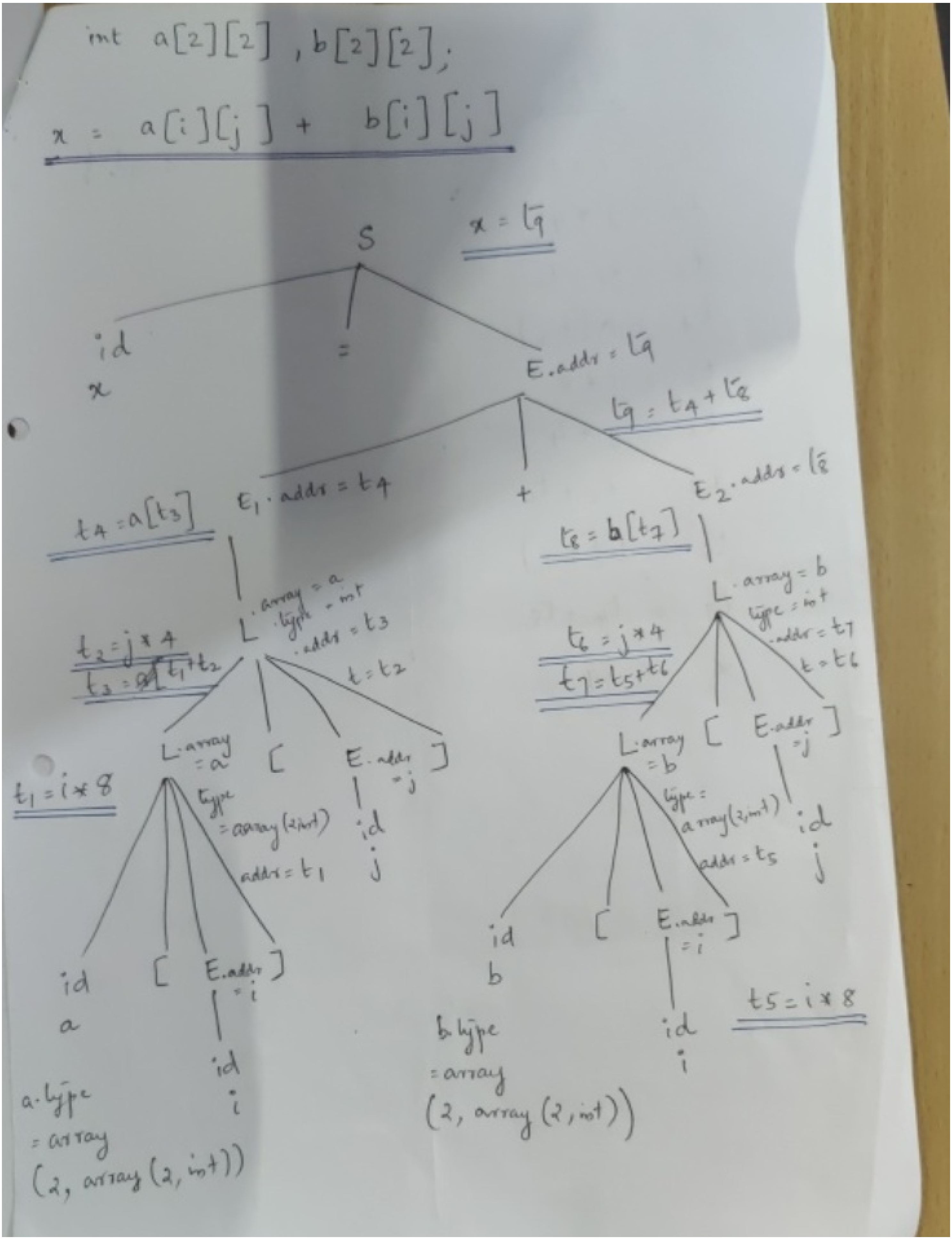
```
L2: if a < 0 goto L3
        goto L1

  L3 : x = 1
   goto L2

L1 :
```

if (a==b !! c==d !! e==f) x=1;



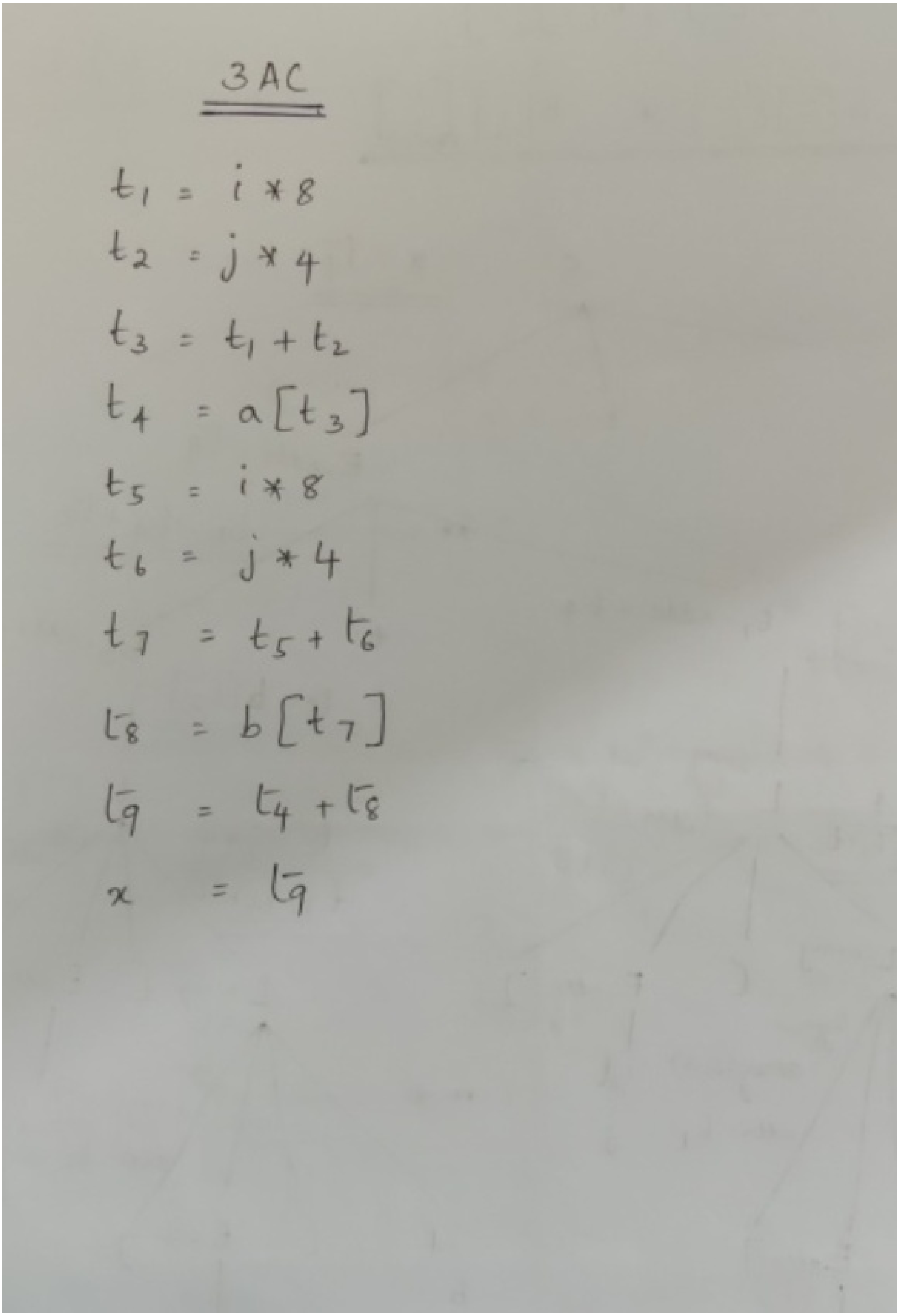
Translation to 3AC

```
     if  a==b  goto L2
           goto L3
L3:  if  c==d  goto L2
           goto L4
L4:  if  e==f goto L2
           goto L1
  L2:  x=1
  L1:
```



int a[2][2] , b[2][2];

x = a[i][j] + b[i][j]



3AC

$t_1 = i * 8$

$t_2 = j * 4$

$t_3 = t_1 + t_2$

$t_4 = a[t_3]$

$t_5 = i * 8$

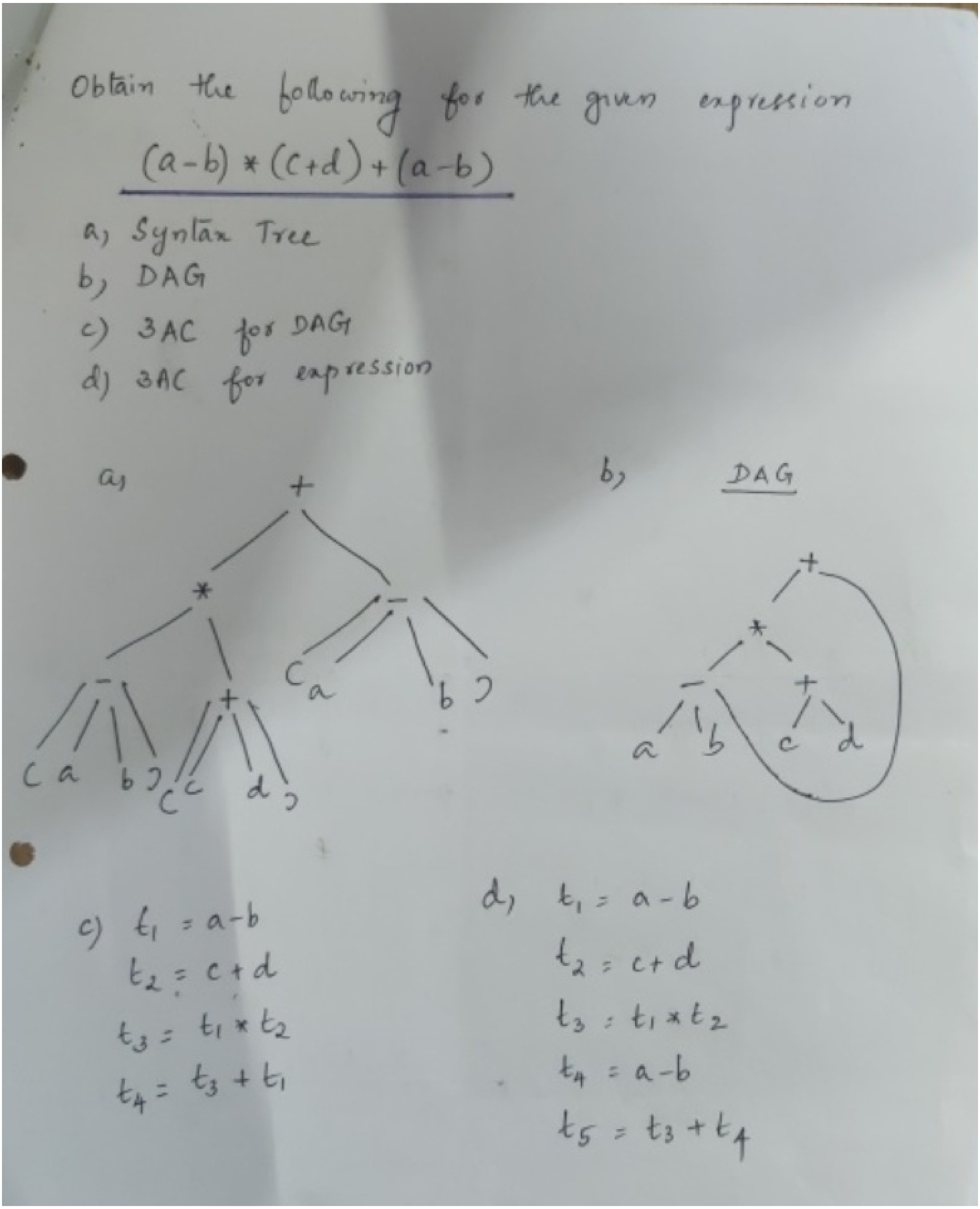
$t_6 = j * 4$

$t_7 = t_5 + t_6$

$t_8 = b[t_7]$

$t_9 = t_4 + t_8$

$x = t_9$



Obtain the following for the given expression

$(a-b)*(c+d)+(a-b)$

a) Syntax Tree

b) DAG

c) 3AC for DAG

d) 3AC for expression

a)

+
*
-
(a b)
+
(c d)
-
(a b)

b) DAG

+
*
-
+
a b c d

c) $t_1 = a-b$

$t_2 = c+d$

$t_3 = t_1 * t_2$

$t_4 = t_3 + t_1$

d) $t_1 = a-b$

$t_2 = c+d$

$t_3 = t_1 * t_2$

$t_4 = a-b$

$t_5 = t_3 + t_4$

Intermediate code for procedures

① $n = f(a[i])$ a = array of integers

$t_1 = i * 4$

$t_2 = a[t_1]$

param t_2

$t_3 = \text{call } f, 1$

$n = t_3$

Quadruple

	op	arg1	arg2	Result
(0)	*	i	4	t_1
(1)	=[]	a	t_1	t_2
(2)	param			x
(3)	call	f	1	t_3
(4)	=	t_3		n

② $x = f(y+1) + 2$

$t_1 = y+1$

param t_1

$t_2 = \text{call } f, 1$

$t_3 = t_2 + 2$

$x = t_3$

Quadruple

op	arg1	arg 2	Result
+	y	1	t_1
param			t_1
call	f	1	t_2
+	t_2	2	t_3
=	t_3		x

③ $g = gcd(x-y, x)$

$t_1 = x-y$

param t_1

param x

$t_2 = \text{call } gcd, 2$

$g = t_2$

Quadruple

op	arg1	arg2	Result
-	x	y	t_1
param			t_1
param			x
call	gcd	2	t_2
=	t_2		g