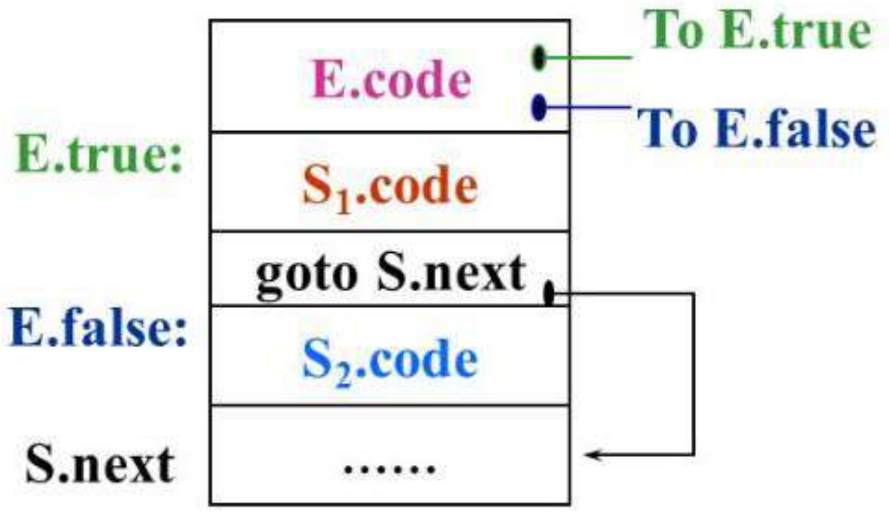
Intermediate Code generation for *Boolean Expressions*

- Boolean Expression
  - Logical values
  - Conditional Expression – change the flow of program (if-else, do-while)
- Boolean operator
  - And
  - Or (lowest precedence)
  - Not
- Example
  - $E \rightarrow E$ or E
  - $E \rightarrow E$ and E
  - $E \rightarrow$ not E
  - $E \rightarrow (E)$
  - $E \rightarrow$ id relop id
  - $E \rightarrow$ TRUE $E \rightarrow$ id
  - $E \rightarrow$ FALSE

Intermediate Code generation for ***Boolean Expressions***

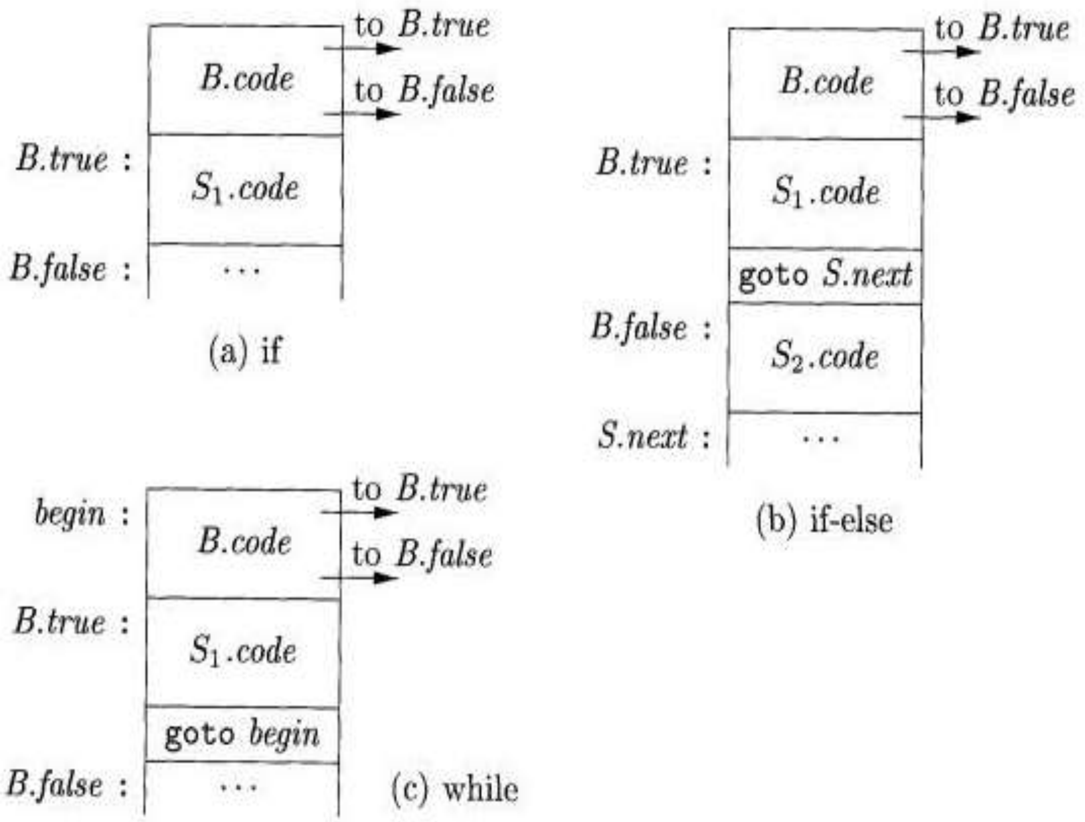
- Numerical representation of Boolean Expression
  - Example1: A or B and C
    - Three Address Sequence:
    - T1=B and C
    - T2=A or T1
  - Example2: A<B → if A<B then 1 else 0
    - Three Address Sequence:
    - 1. If A<B goto (4)
    - 2. T=0
    - 3. goto (5)
    - 4. t=1
    - 5. ---

Attributes used for “if E then S1 else S2”



Flow of Control Statements

If E then S1
IF E then S1 else S2
While E do S1

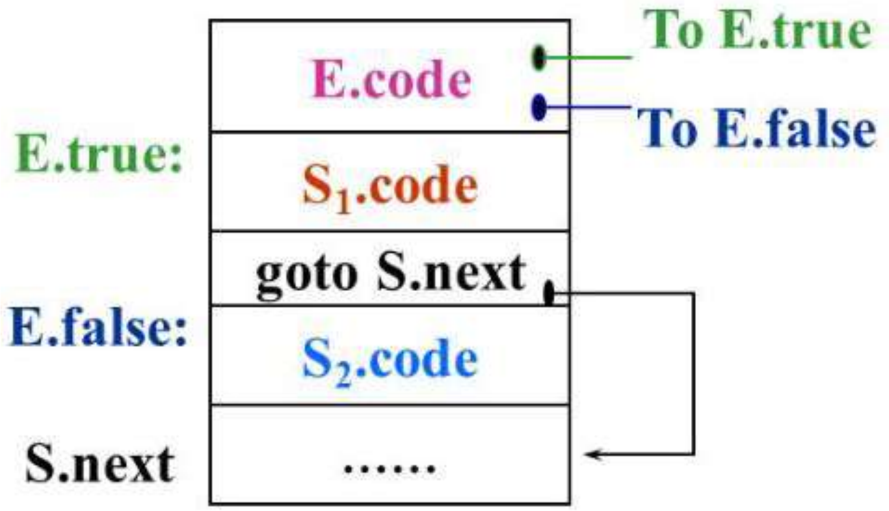


(a) if

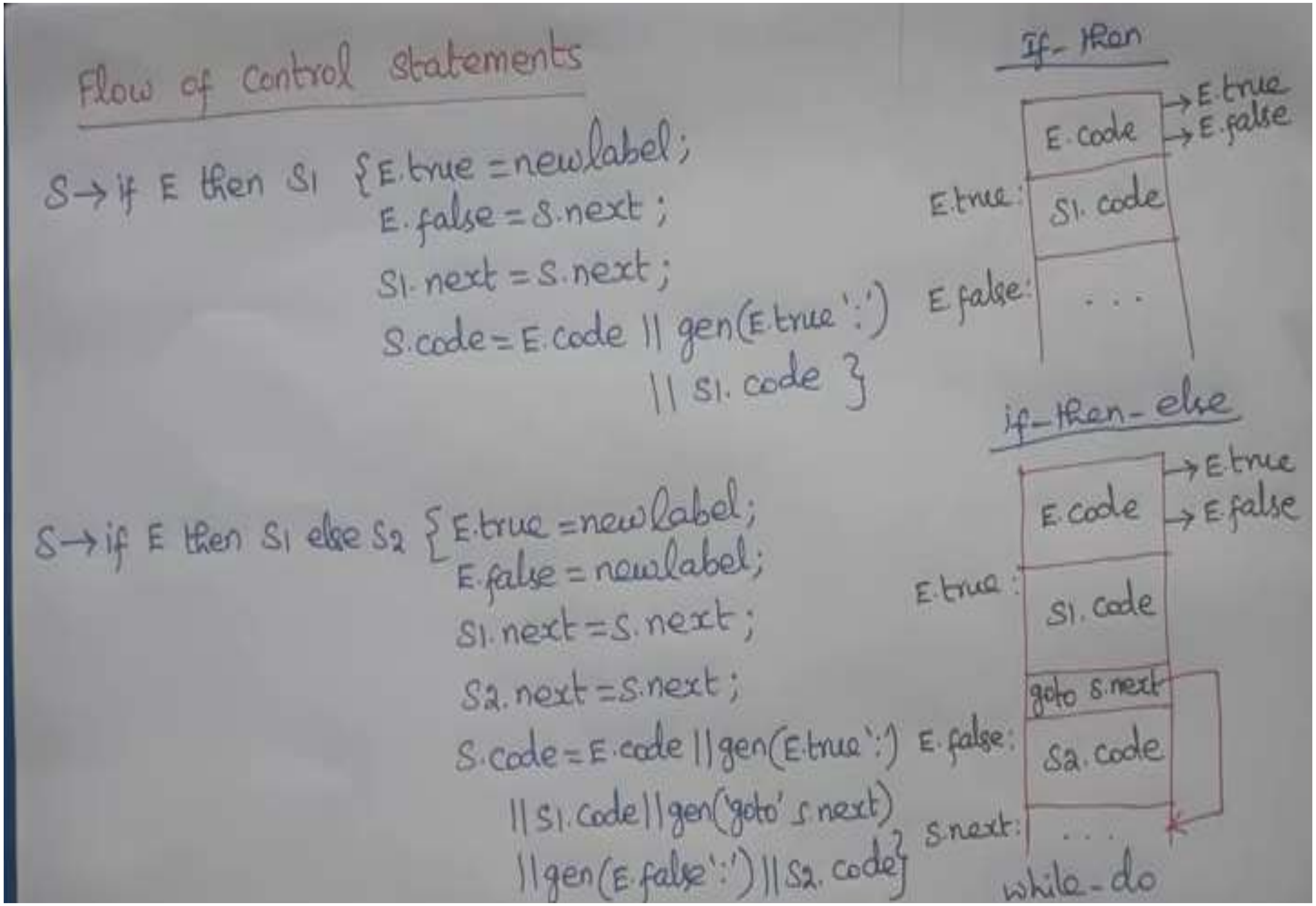
(b) if-else

(c) while

Attributes used for "if E then S1 else S2"



Flow of Control Statements



Flow of control statements

S → if E then S1 { E.true = newlabel;
E.false = S.next;
S1.next = S.next;
S.code = E.code || gen(E.true ':')
|| S1.code }

S → if E then S1 else S2 { E.true = newlabel;
E.false = newlabel;
S1.next = S.next;
S2.next = S.next;
S.code = E.code || gen(E.true ':')
|| S1.code || gen('goto' S.next)
|| gen(E.false ':') || S2.code }

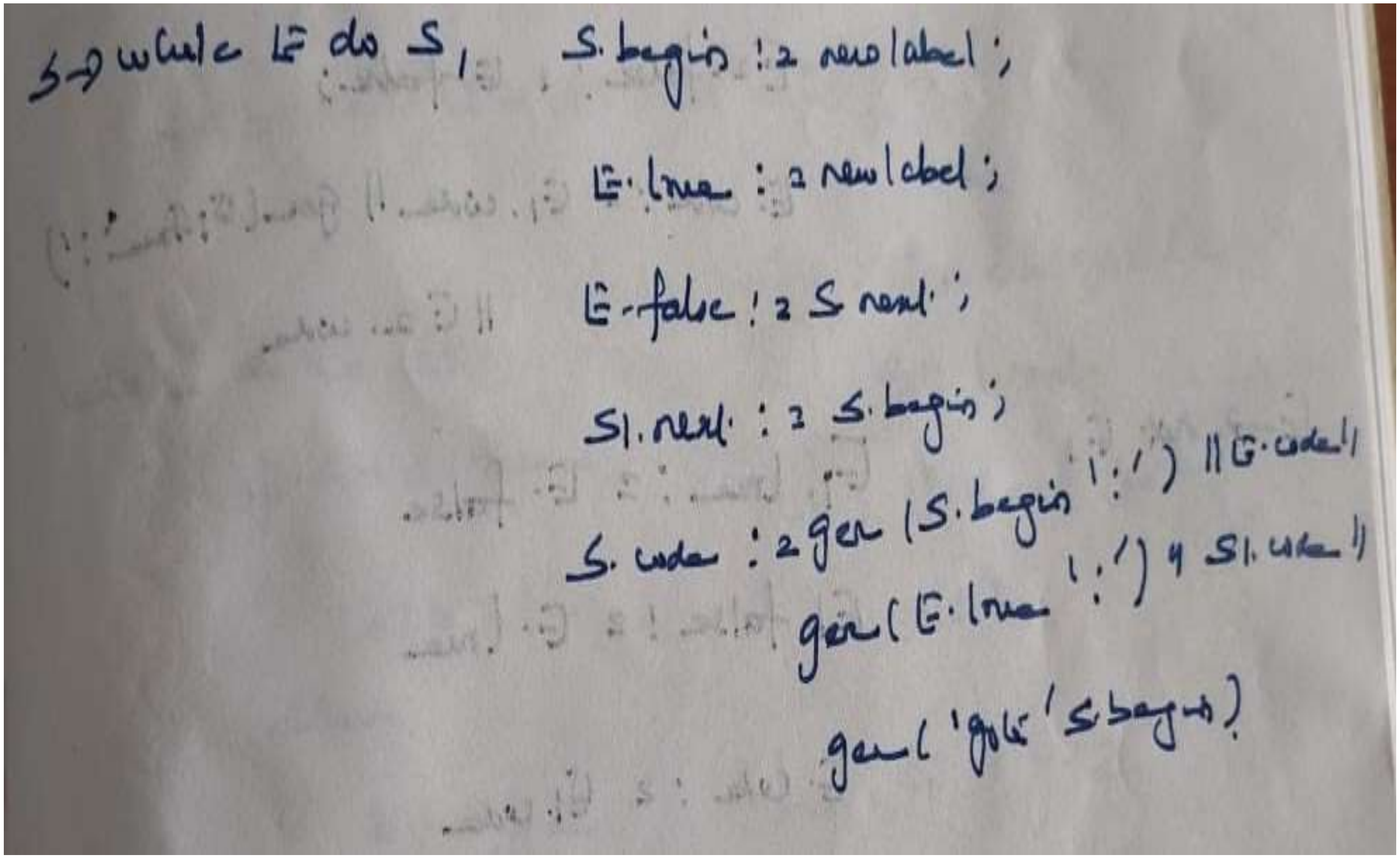
If - then

	E.code	→ E.true → E.false
E.true:	S1.code	
E.false:	. . .	

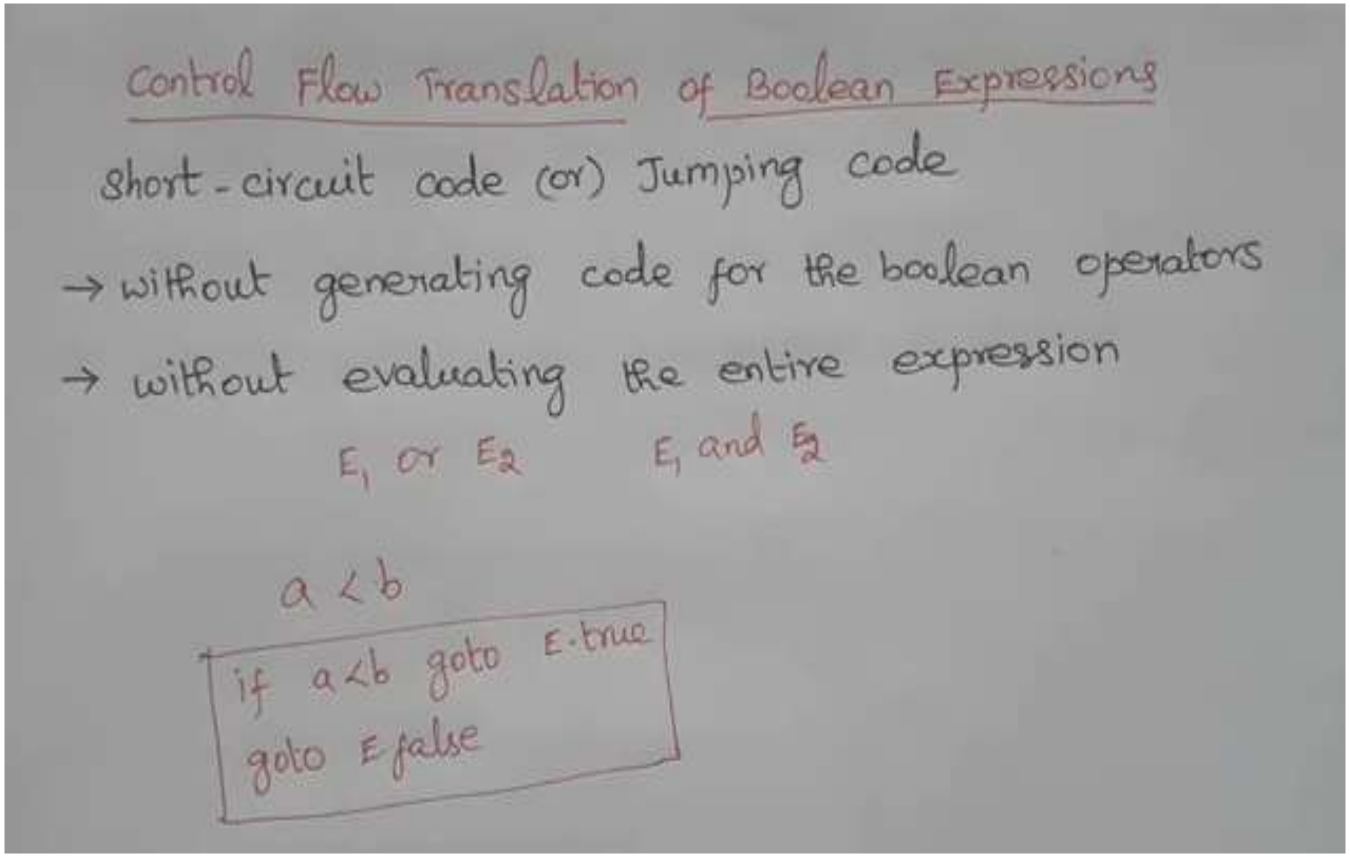
if - then - else

	E.code	→ E.true → E.false
E.true:	S1.code	
	goto S.next	
E.false:	S2.code	
S.next:	. . .	

while - do



```
S → while E do S1    S.begin := newlabel;
                     E.true := newlabel;
                     E.false := S.next;
                     S1.next := S.begin;
                     S.code := gen(S.begin ':') || E.code ||
                               gen(E.true ':') || S1.code ||
                               gen('goto' S.begin)
```



Control Flow Translation of Boolean Expressions

Short-circuit code (or) Jumping code

→ without generating code for the boolean operators

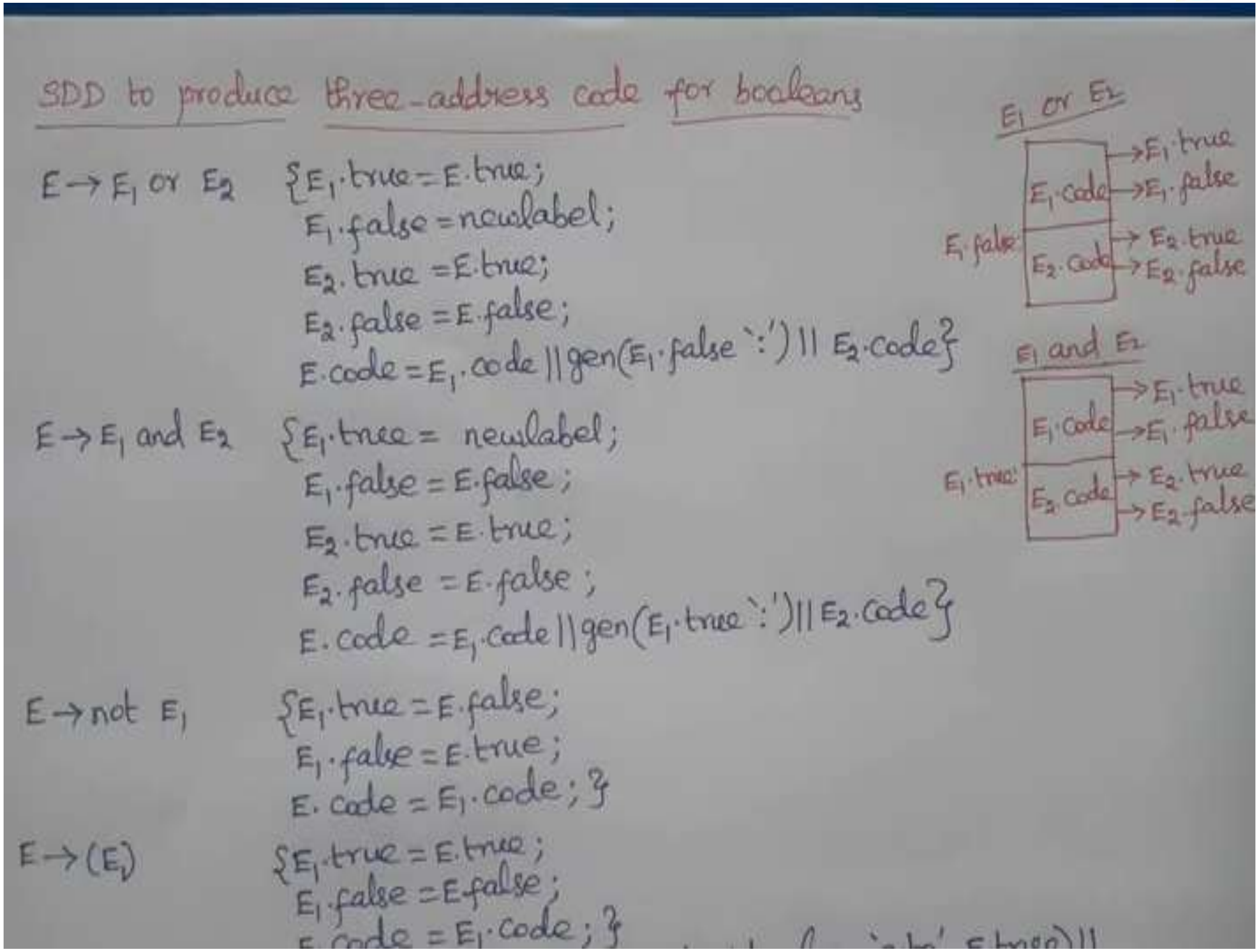
→ without evaluating the entire expression

E_1 or E_2 E_1 and E_2

$a < b$

```
if a<b goto E.true
goto E.false
```

Boolean Expressions



SDD to produce three-address code for booleans

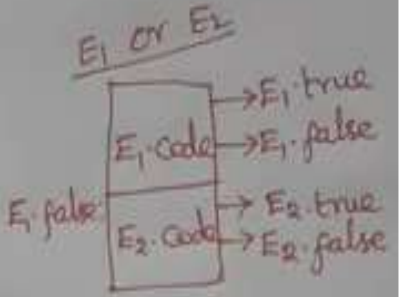
$E \to E_1$ or E_2 { E_1.true = E.true;
E_1.false = newlabel;
E_2.true = E.true;
E_2.false = E.false;
E.code = E_1.code || gen(E_1.false ':') || E_2.code }

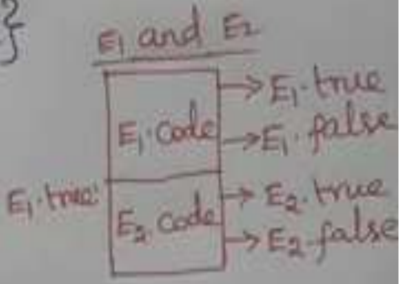
$E \to E_1$ and E_2 { E_1.true = newlabel;
E_1.false = E.false;
E_2.true = E.true;
E_2.false = E.false;
E.code = E_1.code || gen(E_1.true ':') || E_2.code }

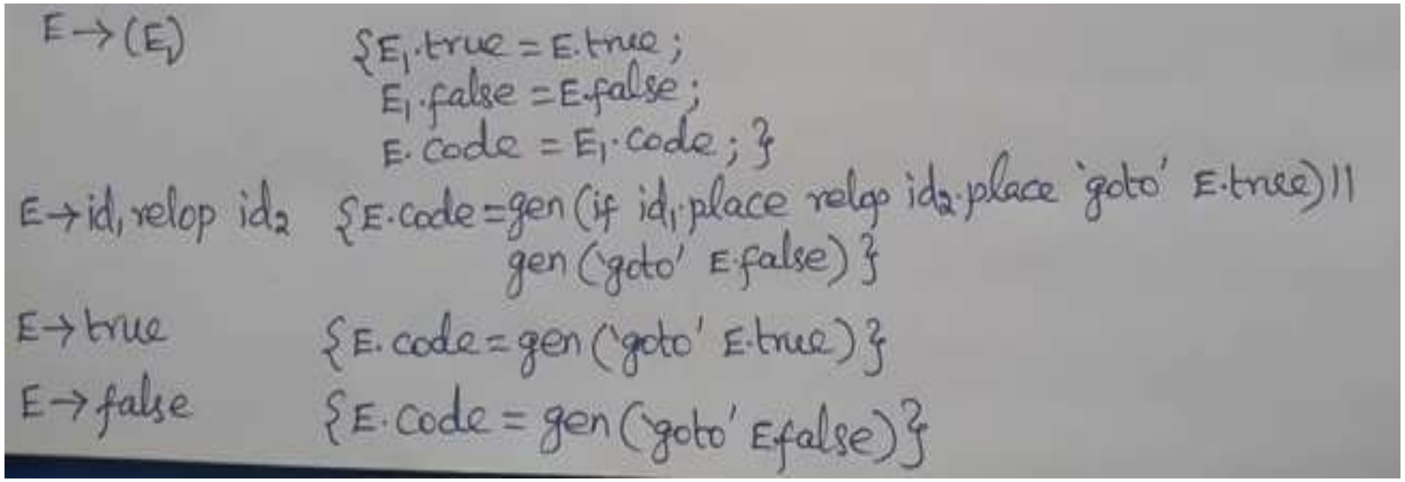
$E \to$ not E_1 { E_1.true = E.false;
E_1.false = E.true;
E.code = E_1.code; }

$E \to (E_1)$ { E_1.true = E.true;
E_1.false = E.false;
E.code = E_1.code; }

[illegible] 'goto' E.true) ||





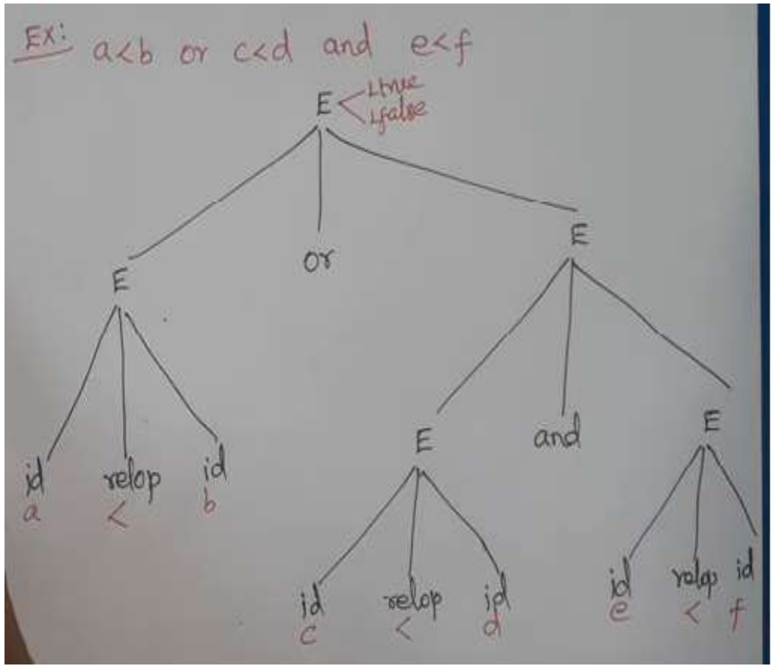


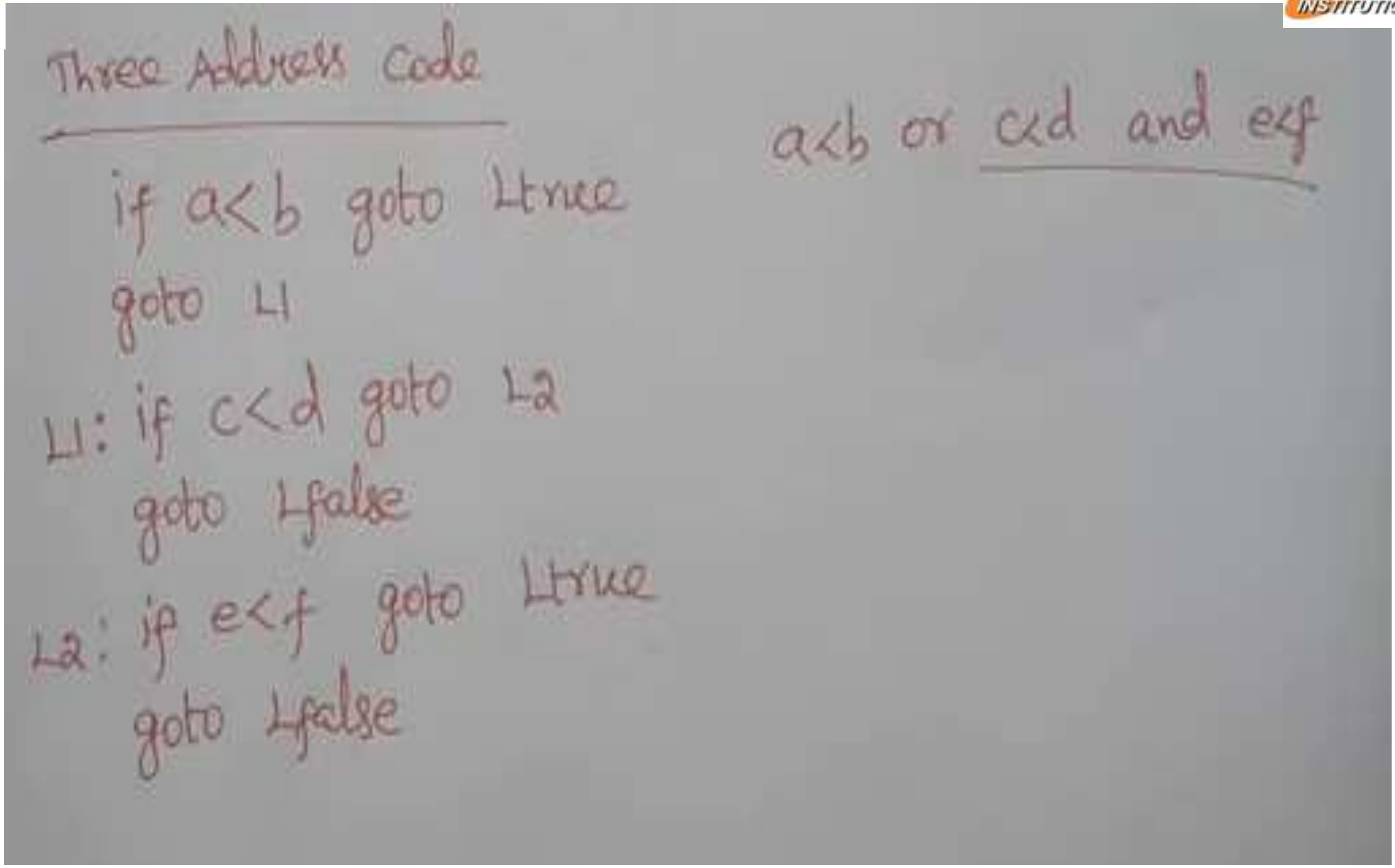
$E \rightarrow (E_1)$ { $E_1.true = E.true$;
$E_1.false = E.false$;
$E.code = E_1.code$; }

$E \rightarrow id_1$ relop id_2 { E.code = gen(if id_1.place relop id_2.place 'goto' E.true) ||
gen('goto' E.false) }

$E \rightarrow$ true { E.code = gen('goto' E.true) }

$E \rightarrow$ false { E.code = gen('goto' E.false) }





Three Address Code

$a<b$ or $\underline{c<d \text{ and } e<f}$

```
    if a<b goto Ltrue
    goto L1
L1: if c<d goto L2
    goto Lfalse
L2: if e<f goto Ltrue
    goto Lfalse
```

BACKPATCHING

- Process of backpatching
  - A marker Non-terminal M – next instruction to be executed
  - Example
    - E→E1 and M E2
    - Incomplete jumps with unfilled labels → E.truelist and E.falselist
    - E1 – false , E is also false → E1.falselist becomes a part of E.flaselist
    - E1 – true → E2 test → E1.truelist becomes the beginning code for E2 ← marker non-terminal M