CONTROL FLOW.

Boolean Expressions

- This section concentrates on the use of boolean expressions to alter the flow of control.
- For clarity, we introduce a new nonterminal B for this purpose.
- Here we consider boolean expressions generated by the following grammar:

  $B \rightarrow B \,||\, B \mid B \,\&\&\, B \mid !B \mid (B) \mid E \text{ rel } E \mid \text{true} \mid \text{false}$

- Given the expression $B_1 || B_2$, if we determine that B_1 is true, then we can conclude that the entire expression is true without having to evaluate B_2.

Short-Circuit Code

- In short-circuit code, the boolean operators &&, ||, and ! translate into jumps. The operators themselves do not appear in the code; instead, the value of a boolean expression is represented by a position in the code sequence.
- Example: The statement

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

  might be translated into the code given below.

  ```
      if x < 100 goto L2
      ifFalse x > 200 goto L1
      ifFalse x != y goto L1
  L2: x = 0
  L1:
  ```

  - In this translation, the boolean expression is true if control reaches label L_2. If the expression is false, control goes immediately to L_1, skipping the assignment $x = 0$.

Flow-of-Control Statements

- We now consider the translation of boolean expressions into three-address code in the context of statements such as those generated by the following grammar:

  $S \rightarrow$ if (B) S_1

  $S \rightarrow$ if (B) S_1 else S_2

  $S \rightarrow$ while (B) S_1

- Here both B and S have synthesized attribute 'code', which gives the translation into three-address instructions.
- $S \rightarrow$ if (B) S_1 :
  - The translation of 'if (B) S_1' consists of B.code followed by S_1.code, as given in figure below.

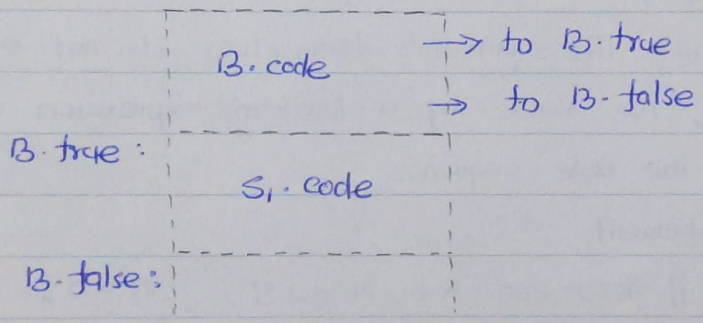

Fig: if (B) S_1

  - Within B.code are jumps based on the value of B. If B is true, control flows to the first instruction of S_1.code, and if B is false, control flows to the instruction immediately following S_1.code.
  - The labels for the jumps in B.code and S.code are managed using inherited attributes. With a boolean expression B, we associate two labels: 'B.true', the label to which the control flows if B is true, and 'B.false', the label to which the control flows if B is false. With a statement S, we associate an inherited attribute 'S.next' denoting a label for the instruction immediately after the code for S.

- The syntax-directed definition given below produces three-address code for boolean expressions in the context of control-flow statements

PRODUCTION	SEMANTIC RULES
$P \rightarrow S$	S.next = newlabel() P.code = S.code || label(S.next)
$S \rightarrow$ assign	S.code = assign.code
$S \rightarrow$ if (B) S_1	B.true = newlabel() B.false = S_1.next = S.next S.code = B.code || label(B.true) || S_1.code
$S \rightarrow$ if (B) S_1 else S_2	B.true = newlabel() B.false = newlabel() S_1.next = S_2.next = S.next S.code = B.code || label(B.true) || S_1.code || gen('goto' S.next) || label(B.false) || S_2.code
$S \rightarrow$ while (B) S_1	begin = newlabel() B.true = newlabel() B.false = S.next S_1.next = begin S.code = label(begin) || B.code || label(B.true) || S_1.code || gen('goto' begin)
$S \rightarrow S_1 S_2$	S_1.next = newlabel() S_2.next = S.next S.code = S_1.code || label(S_1.next) || S_2.code

Fig: Syntax-directed definition for flow-of-control statements

- We assume that 'newlabel()' creates a new label each time it is called, and that 'label(L)' attaches label L to the next three-address instruction to be generated.
- A program consists of a statement generated by P → S. The semantic rules associated with this production initialize S.next to a new label. P.code consists of S.code followed by the new label 'S.next'.
- The code for 'if-else' and 'while' statements are given below:

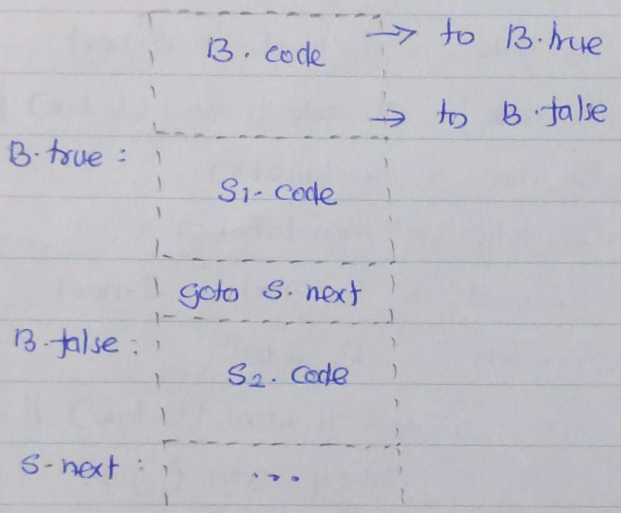

Fig: Code for if-else statement

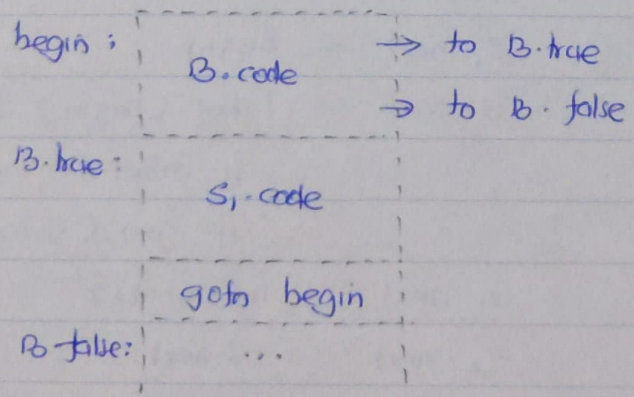

Fig: Code for while statement.