



Lecture 21

Intermediate Code Generation

Awanish Pandey

Department of Computer Science and Engineering
Indian Institute of Technology
Roorkee

March 26, 2025

Flow of Control

- $S \rightarrow \text{while } E \text{ do } S_1$

Flow of Control

- $S \rightarrow \text{while } E \text{ do } S_1$
- $S.\text{begin} :$

Flow of Control

- $S \rightarrow \text{while } E \text{ do } S_1$
- $S.\text{begin} :$
 $E.\text{code}$

Flow of Control

- $S \rightarrow \text{while } E \text{ do } S_1$
- $S.\text{begin} :$
 $E.\text{code}$
 if $E.\text{place} == 0$ goto $S.\text{after}$

Flow of Control

- $S \rightarrow \text{while } E \text{ do } S_1$
- $S.begin :$
 $E.code$
 if $E.place == 0$ goto $S.after$
 $S_1.code$

Flow of Control

- $S \rightarrow \text{while } E \text{ do } S_1$
- $S.\text{begin} :$
 $E.\text{code}$
 if $E.\text{place} == 0$ goto $S.\text{after}$
 $S_1.\text{code}$
 goto $S.\text{begin}$

Flow of Control

- $S \rightarrow \text{while } E \text{ do } S_1$
- $S.\text{begin} :$
 $E.\text{code}$
 if $E.\text{place} == 0$ goto $S.\text{after}$
 $S_1.\text{code}$
 goto $S.\text{begin}$
 $S.\text{after} :$

Flow of Control

- $S \rightarrow \text{while } E \text{ do } S_1$
- $S.begin :$
 $E.code$
 if $E.place == 0$ goto $S.after$
 $S_1.code$
 goto $S.begin$
 $S.after :$
- $S.begin = newlabel()$
 $S.after = newlabel()$
 $S.code = gen(S.begin :)||E.code||gen(\text{if } E.place == 0 \quad \text{goto}$
 $S.after)||S_1.code||gen(\text{goto } S.begin)||gen(S.after :)$

Flow of Control

- $S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$

Flow of Control

- $S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$
- $E.\text{code}$

Flow of Control

- $S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$
- $E.\text{code}$
if $E.\text{place} == 0$ goto $S.\text{else}$

Flow of Control

- $S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$
- $E.\text{code}$
if $E.\text{place} == 0$ goto $S.\text{else}$
 $S_1.\text{code}$

Flow of Control

- $S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$
- $E.\text{code}$
 $\text{if } E.\text{place} == 0 \text{ goto } S.\text{else}$
 $S_1.\text{code}$
 $\text{goto } S.\text{after}$

Flow of Control

- $S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$
- $E.\text{code}$
 $\text{if } E.\text{place} == 0 \text{ goto } S.\text{else}$
 $S_1.\text{code}$
 $\text{goto } S.\text{after}$
 $S.\text{else} :$

Flow of Control

- $S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$
- $E.\text{code}$
 if $E.\text{place} == 0$ goto $S.\text{else}$
 $S_1.\text{code}$
 goto $S.\text{after}$
 $S.\text{else} : S_2.\text{code}$

Flow of Control

- $S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$
- $E.\text{code}$
 $\text{if } E.\text{place} == 0 \text{ goto } S.\text{else}$
 $S_1.\text{code}$
 $\text{goto } S.\text{after}$
 $S.\text{else} : S_2.\text{code}$
 $S.\text{after} :$

Flow of Control

- $S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$
- $E.\text{code}$
 $\text{if } E.\text{place} == 0 \text{ goto } S.\text{else}$
 $S_1.\text{code}$
 $\text{goto } S.\text{after}$
 $S.\text{else} : S_2.\text{code}$
 $S.\text{after} :$
- $S.\text{else} = \text{newlabel}() \quad S.\text{after} = \text{newlabel}()$
 $S.\text{code} = E.\text{code} ||$
 $\text{gen}(\text{if } E.\text{place} == 0 \text{ goto } S.\text{else}) ||$
 $S_1.\text{code} || \text{gen}(\text{goto } S.\text{after}) ||$
 $\text{gen}(S.\text{else} :) ||$
 $S_2.\text{code} ||$
 $\text{gen}(S.\text{after} :)$

Names in Symbol Table

- $S \rightarrow id = E$

Names in Symbol Table

- $S \rightarrow id = E$
 $p = \text{lookup}(id.place);$
 if $p \neq \text{null}$ then $\text{emit}(p = E.place)$
 else error

Names in Symbol Table

- $S \rightarrow id = E$
 $p = \text{lookup}(id.place);$
 if $p \neq \text{null}$ then $\text{emit}(p = E.place)$
 else error
- $E \rightarrow id$

Names in Symbol Table

- $S \rightarrow id = E$
 $p = \text{lookup}(id.place);$
 if $p \neq \text{null}$ then $\text{emit}(p = E.place)$
 else error
- $E \rightarrow id$
 $p = \text{lookup}(id.name);$
 if $p \neq \text{null}$ then $E.place = p$
 else error

Addressing Array Elements

- Arrays are stored in a block of consecutive locations

Addressing Array Elements

- Arrays are stored in a block of consecutive locations
- Assume width of each element is w

Addressing Array Elements

- Arrays are stored in a block of consecutive locations
- Assume width of each element is w
- i^{th} element of array A begins in location

Addressing Array Elements

- Arrays are stored in a block of consecutive locations
- Assume width of each element is w
- i^{th} element of array A begins in location

$$base + (i - low) \times w \leftrightarrow i \times w + const$$

where $base$ is relative address of $A[low]$

- For $n - d$ arrays storage can be either row major or column major.

Addressing Array Elements

- Arrays are stored in a block of consecutive locations
- Assume width of each element is w
- i^{th} element of array A begins in location

$$base + (i - low) \times w \leftrightarrow i \times w + const$$

where base is relative address of $A[low]$

- For $n - d$ arrays storage can be either row major or column major.
- in case of 2-D array stored in row major form address of $A[i_1, i_2]$ can be calculated as

$$base + ((i_1 - low_1) \times n_2 + i_2 - low_2) \times w \leftrightarrow ((i_1 \times n_2) + i_2) \times w + constant$$

$$n_2 = high_2 - low_2 + 1$$

Type conversion within assignments

- $E \rightarrow E_1 + E_2$

Type conversion within assignments

- $E \rightarrow E_1 + E_2$
 $E.place = newtmp();$
 if $E_1.type = integer$ and $E_2.type = integer$
 then emit($E.place = "E_1.place \quad int + " E_2.place$);
 $E.type = integer$;

Type conversion within assignments

- $E \rightarrow E_1 + E_2$
 $E.place = newtmp();$
 if $E_1.type = integer$ and $E_2.type = integer$
 then emit($E.place = "E_1.place \quad int + " \quad E_2.place$);
 $E.type = integer$;
- Similar code if both $E_1.type$ and $E_2.type$ are *real*

Type conversion within assignments

- $E \rightarrow E_1 + E_2$
 $E.place = newtmp();$
 if $E_1.type = integer$ and $E_2.type = integer$
 then emit($E.place = "E_1.place \quad "int + " \quad E_2.place$);
 $E.type = integer$;
- Similar code if both $E_1.type$ and $E_2.type$ are *real*
- else if $E_1.type = int$ and $E_2.type = real$

Type conversion within assignments

- $E \rightarrow E_1 + E_2$
 $E.place = newtmp();$
if $E_1.type = integer$ and $E_2.type = integer$
then $emit(E.place = " E_1.place \quad "int + " \quad E_2.place);$
 $E.type = integer;$
- Similar code if both $E_1.type$ and $E_2.type$ are *real*
- else if $E_1.type = int$ and $E_2.type = real$ then $u = newtmp();$
 $emit(u = "int to real E_1.place);$
 $emit(E.place = " u "real + " E_2.place);$
 $E.type = real;$
- similar code if $E_1.type$ is *real* and $E_2.type$ is *integer*

Boolean Expression

- Compute logical values

Boolean Expression

- Compute logical values
- Change the flow of control

Boolean Expression

- Compute logical values
- Change the flow of control
- Boolean operators are: `and` `or` `not`

Boolean Expression

- Compute logical values
- Change the flow of control
- Boolean operators are: `and` `or` `not`
- $E \rightarrow E$ or E
 - | E and E
 - | `not` E
 - | (E)
 - | `id` `relop` id
 - | `true`
 - | `false`

Methods of translation

- Evaluate similar to arithmetic expressions

Methods of translation

- Evaluate similar to arithmetic expressions
 - ▶ Normally use 1 for true and 0 for false

Methods of translation

- Evaluate similar to arithmetic expressions
 - ▶ Normally use 1 for true and 0 for false
 - ▶ a or b and not c

Methods of translation

- Evaluate similar to arithmetic expressions
 - ▶ Normally use 1 for true and 0 for false
 - ▶ `a or b and not c`
 - ▶ `t1 = not c`

Methods of translation

- Evaluate similar to arithmetic expressions
 - ▶ Normally use 1 for true and 0 for false
 - ▶ `a or b and not c`
 - ▶ $t_1 = \text{not } c$
 - ▶ $t_2 = b \text{ and } t_1$

Methods of translation

- Evaluate similar to arithmetic expressions
 - ▶ Normally use 1 for true and 0 for false
 - ▶ `a or b and not c`
 - ▶ $t_1 = \text{not } c$
 - ▶ $t_2 = b \text{ and } t_1$
 - ▶ $t_3 = a \text{ or } t_2$
- Implement by flow of control

Methods of translation

- Evaluate similar to arithmetic expressions
 - ▶ Normally use 1 for true and 0 for false
 - ▶ a or b and not c
 - ▶ $t_1 = \text{not } c$
 - ▶ $t_2 = b$ and t_1
 - ▶ $t_3 = a$ or t_2
- Implement by flow of control
 - ▶ given expression E_1 or E_2
if E_1 evaluates to *true*
then E_1 or E_2 evaluates to *true*
without evaluating E_2

Syntax directed translation of boolean expressions

- $E \rightarrow E_1 \text{ or } E_2$

Syntax directed translation of boolean expressions

- $E \rightarrow E_1 \text{ or } E_2$
 $E.place = newtmp()$
 $emit(E.place = "E_1.place \text{ or } E_2.place")$

Syntax directed translation of boolean expressions

- $E \rightarrow E_1 \text{ or } E_2$
 $E.place = newtmp()$
 $emit(E.place = " E_1.place \text{or} E_2.place)$
- $E \rightarrow E_1 \text{ and } E_2$

Syntax directed translation of boolean expressions

- $E \rightarrow E_1 \text{ or } E_2$
 $E.place = newtmp()$
 $emit(E.place = "E_1.place \text{ or } E_2.place")$
- $E \rightarrow E_1 \text{ and } E_2$
 $E.place = newtmp()$
 $emit(E.place = "E_1.place \text{ and } E_2.place")$

Syntax directed translation of boolean expressions

- $E \rightarrow E_1 \text{ or } E_2$
 $E.place = newtmp()$
 $emit(E.place = "E_1.place \text{ or } E_2.place")$
- $E \rightarrow E_1 \text{ and } E_2$
 $E.place = newtmp()$
 $emit(E.place = "E_1.place \text{ and } E_2.place")$
- $E \rightarrow \text{not } E_1$

Syntax directed translation of boolean expressions

- $E \rightarrow E_1 \text{ or } E_2$
 $E.place = newtmp()$
 $emit(E.place = "E_1.place \text{ or } E_2.place")$
- $E \rightarrow E_1 \text{ and } E_2$
 $E.place = newtmp()$
 $emit(E.place = "E_1.place \text{ and } E_2.place")$
- $E \rightarrow \text{not } E_1$
 $E.place = newtmp()$
 $emit(E.place = "not E_1.place")$

Syntax directed translation of boolean expressions

- $E \rightarrow E_1 \text{ or } E_2$
 $E.place = newtmp()$
 $emit(E.place = "E_1.place \text{ or } E_2.place")$
- $E \rightarrow E_1 \text{ and } E_2$
 $E.place = newtmp()$
 $emit(E.place = "E_1.place \text{ and } E_2.place")$
- $E \rightarrow \text{not } E_1$
 $E.place = newtmp()$
 $emit(E.place = "not E_1.place")$
- $E \rightarrow (E_1)$ $E.place = E_1.place$

Syntax directed translation of boolean expressions

- $E \rightarrow id_1 \text{ relop } id_2$

Syntax directed translation of boolean expressions

- $E \rightarrow id_1 \text{ relop } id_2$
 $E.place = newtmp()$
 $emit(\text{if } id_1.place \text{ relop } id_2.place \text{ goto state}+3)$
 $emit(E.place = 0)$
 $emit(\text{goto state} + 2)$
 $emit(E.place = 1)$

Syntax directed translation of boolean expressions

- $E \rightarrow id_1 \text{ relop } id_2$
 $E.place = newtmp()$
 $emit(\text{if } id_1.place \text{ relop } id_2.place \text{ goto state}+3)$
 $emit(E.place = 0)$
 $emit(\text{goto state} + 2)$
 $emit(E.place = 1)$
- $E \rightarrow true$

Syntax directed translation of boolean expressions

- $E \rightarrow id_1 \text{ relop } id_2$
 $E.place = newtmp()$
 $\text{emit}(\text{if } id_1.place \text{ relop } id_2.place \text{ goto state}+3)$
 $\text{emit}(E.place = 0)$
 $\text{emit}(\text{goto state} + 2)$
 $\text{emit}(E.place = 1)$
- $E \rightarrow true$
 $E.place = newtmp()$
 $\text{emit}(E.place = '1')$

Syntax directed translation of boolean expressions

- $E \rightarrow id_1 \text{ relop } id_2$
 $E.place = newtmp()$
 $\text{emit}(\text{if } id_1.place \text{ relop } id_2.place \text{ goto state}+3)$
 $\text{emit}(E.place = 0)$
 $\text{emit}(\text{goto state} + 2)$
 $\text{emit}(E.place = 1)$
- $E \rightarrow true$
 $E.place = newtmp()$
 $\text{emit}(E.place = '1')$
- $E \rightarrow false$

Syntax directed translation of boolean expressions

- $E \rightarrow id_1 \text{ relop } id_2$
 $E.place = newtmp()$
 $emit(\text{if } id_1.place \text{ relop } id_2.place \text{ goto state}+3)$
 $emit(E.place = 0)$
 $emit(\text{goto state} + 2)$
 $emit(E.place = 1)$
- $E \rightarrow true$
 $E.place = newtmp()$
 $emit(E.place = '1')$
- $E \rightarrow false$
 $E.place = newtmp()$
 $emit(E.place = '0')$

Syntax directed translation of boolean expressions

- $E \rightarrow id_1 \text{ relop } id_2$
 $E.place = newtmp()$
 $emit(\text{if } id_1.place \text{ relop } id_2.place \text{ goto state}+3)$
 $emit(E.place = 0)$
 $emit(\text{goto state} + 2)$
 $emit(E.place = 1)$
- $E \rightarrow true$
 $E.place = newtmp()$
 $emit(E.place = '1')$
- $E \rightarrow false$
 $E.place = newtmp()$
 $emit(E.place = '0')$

Write 3AC Code for $a < b$ or $c < d$ and $e < f$

Short Circuit Evaluation of boolean expressions

- Translate boolean expressions without

Short Circuit Evaluation of boolean expressions

- Translate boolean expressions without
 - ▶ generating code for boolean operators

Short Circuit Evaluation of boolean expressions

- Translate boolean expressions without
 - ▶ generating code for boolean operators
 - ▶ evaluating the entire expression
- Flow if control statements

Short Circuit Evaluation of boolean expressions

- Translate boolean expressions without
 - ▶ generating code for boolean operators
 - ▶ evaluating the entire expression

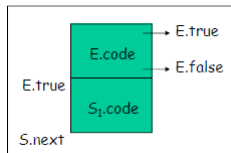
- Flow if control statements

$S \rightarrow \text{if } E \text{ then } S_1$

| $\text{if } E \text{ then } S_1 \text{ else } S_2$

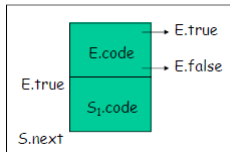
| $\text{while } E \text{ do } S_1$

If-then



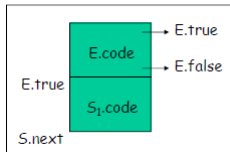
$S \rightarrow \text{if } E \text{ then } S_1$
 $E.true = \text{newlabel}()$

If-then



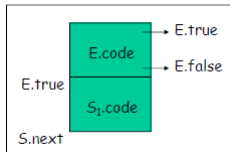
$S \rightarrow \text{if } E \text{ then } S_1$
 $E.true = \text{newlabel}()$
 $E.false = S.next$

If-then



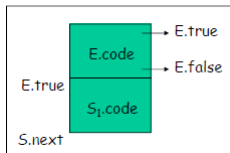
$S \rightarrow \text{if } E \text{ then } S_1$
 $E.true = \text{newlabel}()$
 $E.false = S.next$
 $S_1.next = S.next$

If-then



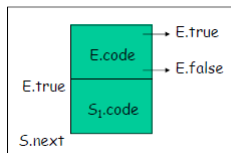
$S \rightarrow \text{if } E \text{ then } S_1$
 $E.true = \text{newlabel}()$
 $E.false = S.next$
 $S_1.next = S.next$
 $S.code = E.code||$

If-then



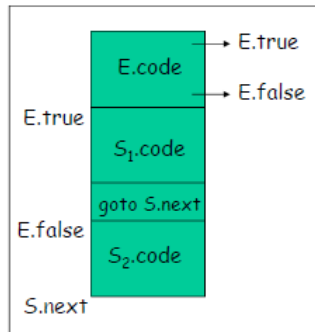
$S \rightarrow \text{if } E \text{ then } S_1$
 $E.true = \text{newlabel}()$
 $E.false = S.next$
 $S_1.next = S.next$
 $S.code = E.code ||$
 $\text{gen}(E.true " : ") ||$

If-then

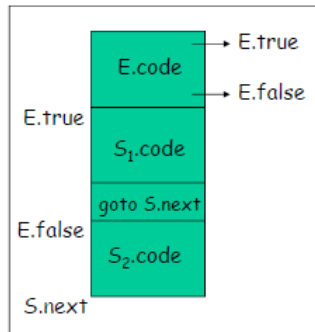


$S \rightarrow \text{if } E \text{ then } S_1$
 $E.true = \text{newlabel}()$
 $E.false = S.next$
 $S_1.next = S.next$
 $S.code = E.code ||$
 $\text{gen}(E.true " : ") ||$
 $S_1.code$

If-else

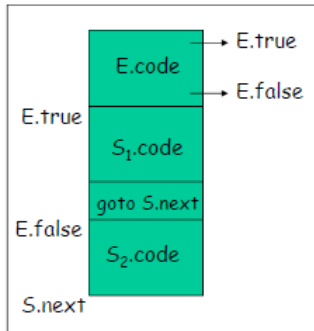


If-else



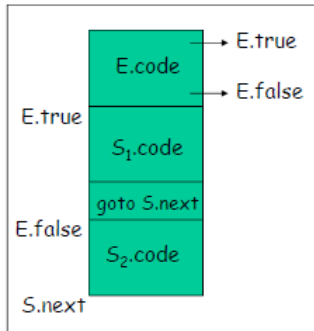
$S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$

If-else



$S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$
 $E.\text{true} = \text{newlabel}()$

If-else

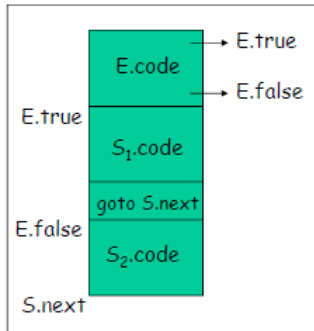


$S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$

$E.true = \text{newlabel}()$

$E.false = \text{newlabel}()$

If-else



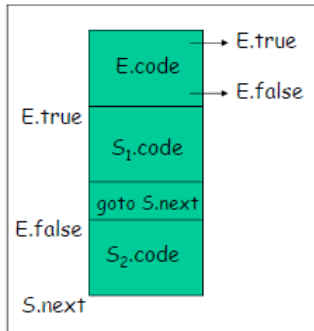
$S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$

$E.true = \text{newlabel}()$

$E.false = \text{newlabel}()$

$S_1.next = S.next$

If-else



$S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$

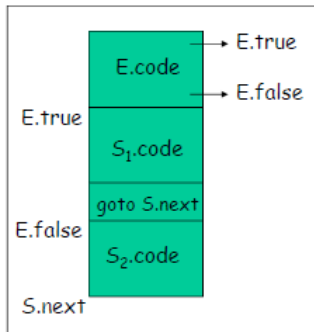
$E.true = \text{newlabel}()$

$E.false = \text{newlabel}()$

$S_1.next = S.next$

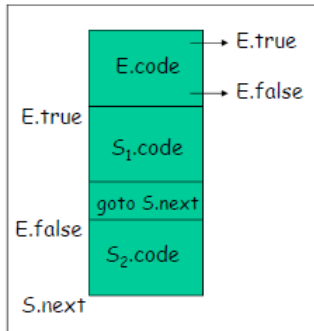
$S_2.next = S.next$

If-else



```
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||
```

If-else



$S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$

$E.true = \text{newlabel}()$

$E.false = \text{newlabel}()$

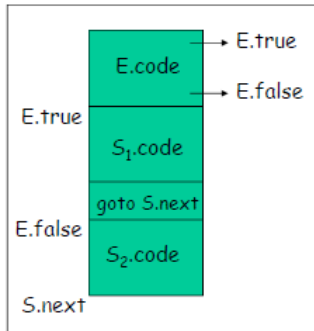
$S_1.next = S.next$

$S_2.next = S.next$

$S.code = E.code ||$

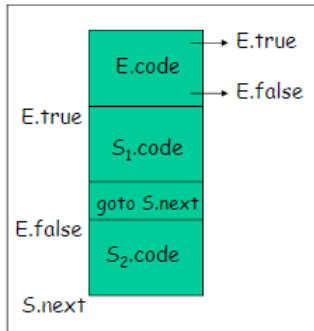
$\text{gen}(E.true " : ") ||$

If-else



```
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||
```

If-else



$S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$

$E.true = \text{newlabel}()$

$E.false = \text{newlabel}()$

$S_1.next = S.next$

$S_2.next = S.next$

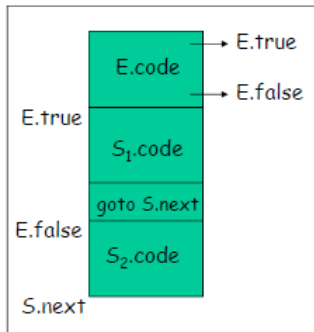
$S.code = E.code ||$

$\text{gen}(E.true : " : ") ||$

$S_1.code ||$

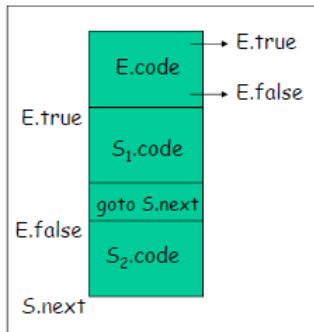
$\text{gen}(\text{goto } S.next) ||$

If-else



```
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(gotoS.next)||  
gen(E.false" : ")||
```

If-else



$S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$

$E.true = \text{newlabel}()$

$E.false = \text{newlabel}()$

$S_1.next = S.next$

$S_2.next = S.next$

$S.code = E.code ||$

$\text{gen}(E.true " : ") ||$

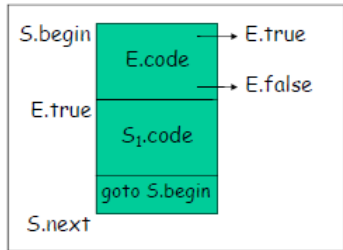
$S_1.code ||$

$\text{gen}(\text{goto } S.next) ||$

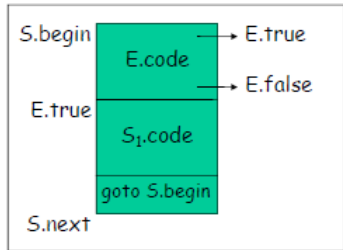
$\text{gen}(E.false " : ") ||$

$S_2.code$

While

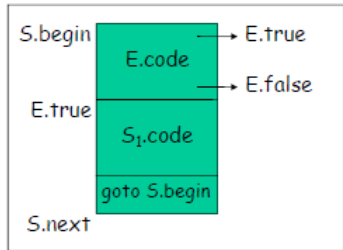


While



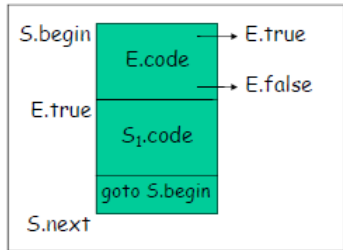
$S \rightarrow \text{while } E \text{ do } S_1$

While



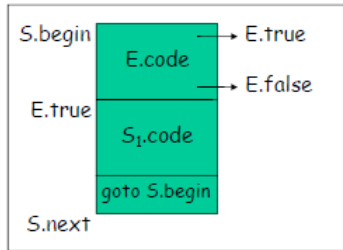
$S \rightarrow \text{while } E \text{ do } S_1$
 $S.begin = \text{newlabel}()$

While



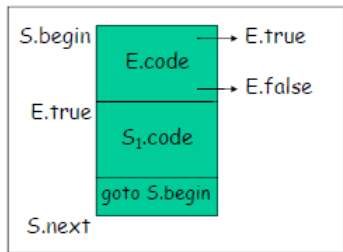
$S \rightarrow \text{while } E \text{ do } S_1$
 $S.begin = \text{newlabel}()$
 $E.true = \text{newlabel}()$

While



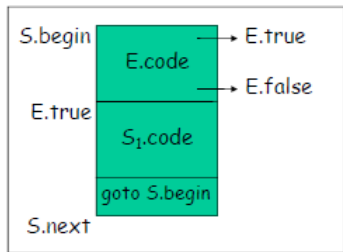
```
S → while E do S1  
S.begin = newlabel()  
E.true = newlabel()  
E.false = S.next
```

While



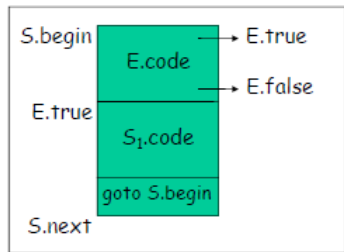
```
S → while E do S1  
S.begin = newlabel()  
E.true = newlabel()  
E.false = S.next  
S1.next = S.begin
```

While



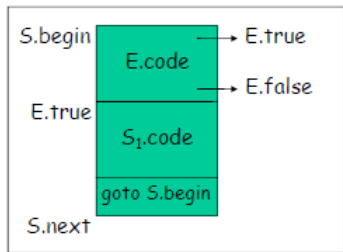
```
S → while E do S1  
S.begin = newlabel()  
E.true = newlabel()  
E.false = S.next  
S1.next = S.begin  
S.ocde = gen(S.begin' :')||
```

While



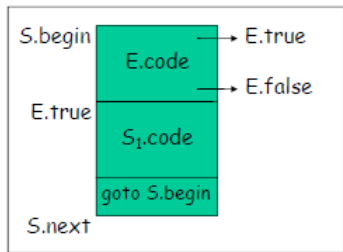
```
S → while E do S1  
S.begin = newlabel()  
E.true = newlabel()  
E.false = S.next  
S1.next = S.begin  
S.ocde = gen(S.begin' :')||  
E.code||
```

While



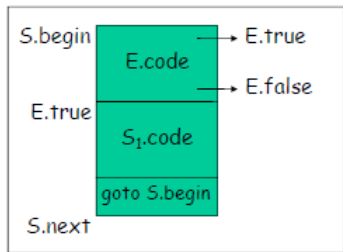
```
S → while E do S1  
S.begin = newlabel()  
E.true = newlabel()  
E.false = S.next  
S1.next = S.begin  
S.ocde = gen(S.begin' :')||  
E.code||  
gen(E.true' :')||
```


While



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

While



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

Control flow translation of boolean expression

- $E \rightarrow E_1 \text{ or } E_2$

Control flow translation of boolean expression

- $E \rightarrow E_1 \text{ or } E_2$
 $E_1.true = E.true$
 $E_1.false = \text{newlabel}()$
 $E_2.true = E.true$
 $E_2.false = E.false$
 $E.code = E_1.code \parallel \text{gen}(E_1.false)$
 $\parallel E_2.code$

Control flow translation of boolean expression

- $E \rightarrow E_1 \text{ or } E_2$
 $E_1.true = E.true$
 $E_1.false = \text{newlabel}()$
 $E_2.true = E.true$
 $E_2.false = E.false$
 $E.code = E_1.code \parallel \text{gen}(E_1.false)$
 $\parallel E_2.code$
- $E \rightarrow E_1 \text{ and } E_2$

Control flow translation of boolean expression

- $E \rightarrow E_1 \text{ or } E_2$
 $E_1.true = E.true$
 $E_1.false = \text{newlabel}()$
 $E_2.true = E.true$
 $E_2.false = E.false$
 $E.code = E_1.code \parallel \text{gen}(E_1.false)$
 $\parallel E_2.code$
- $E \rightarrow E_1 \text{ and } E_2$
 $E_1.true = \text{newlabel}()$
 $E_1.false = E.false$
 $E_2.true = E.true$
 $E_2.false = E.false$
 $E.code = E_1.code \parallel \text{gen}(E_1.true)$
 $\parallel E_2.code$