Lecture 21

Intermediate Code Generation

Awanish Pandey

Department of Computer Science and Engineering
Indian Institute of Technology
Roorkee

March 26, 2025

Flow of Control

- $S \rightarrow while\ E\ do\ S_1$

Flow of Control

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

Flow of Control

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

Flow of Control

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

Flow of Control

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

Flow of Control

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

Flow of Control

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

Flow of Control

- $S \rightarrow while\ E\ do\ S_1$
- $S.begin:$
  $E.code$
  $if\ \ E.place == 0\ goto\ S.after$
  $S_1.code$
  $goto\ S.begin$
  $S.after:$
- $S.begin = newlabel()$
  $S.after = newlabel()$
  $S.code = gen(S.begin:)||E.code||gen(if\ \ E.place == 0 \quad goto$
  $S.after)||S1.code||gen(goto\ \ S.begin)||gen(S.after:)$

Flow of Control

- $S \rightarrow if\ \ E\ \ then\ S_1\ \ else\ S_2$

Flow of Control

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

Flow of Control

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

Flow of Control

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

Flow of Control

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

Flow of Control

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

Flow of Control

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

Flow of Control

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

Flow of Control

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

Names in Symbol Table

- $S \rightarrow id = E$

Names in Symbol Table

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

Names in Symbol Table

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

Names in Symbol Table

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

$$base + ((i_1{-}low_1)xn_2 + i_2 - low_2)xw \leftrightarrow ((i_1xn_2) + i_2)xw + constant$$

$$n_2 = high_2 - low_2 + 1$$

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 $E_2.type = integer$
  $then \; emit(E.place ``=" E_1.place \quad ``int +" \quad E_2.place);$
  $E.type = integer;$

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 $u = newtmp();$
  $emit(u`` = " inttoreal \;\; E_1.place);$
  $emit(E.place`` = " u``real + " E_2.place);$
  $E.type = real;$
- similar code if $E_1.type$ is real and $E_2.type$ is integer

real+ will be the operator for real number addition here.

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`
- $$\begin{array}{ll} E \rightarrow & E \text{ or } E \\ & \mid \text{ E and } E \\ & \mid \text{ not } E \\ & \mid (E) \\ & \mid \text{ id relop } id \\ & \mid \ true \\ & \mid \ false \end{array}$$

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`
  - $t_1 =$ `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`
  - $t_1 =$ `not c`
  - $t_2 =$ `b 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`
  - $t_1 =$ `not c`
  - $t_2 =$ `b and` t_1
  - $t_3 =$ `a 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`
  - $t_1 =$ `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

called short circuit evaluation of booleans

Syntax directed translation of boolean expressions

- $E \rightarrow E_1$ or E_2

Syntax directed translation of boolean expressions

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

Syntax directed translation of boolean expressions

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

Syntax directed translation of boolean expressions

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

Syntax directed translation of boolean expressions

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

Syntax directed translation of boolean expressions

- $E \rightarrow E_1$ or E_2
  $E.place = newtmp()$
  $emit(E.place``=" E_1.place``or" E_2.place)$
- $E \rightarrow E_1$ and E_2
  $E.place = newtmp()$
  $emit(E.place``=" E_1.place``and" E_2.place)$
- $E \rightarrow not\ E_1$
  $E.place = newtmp()$
  $emit(E.place``=" \quad ``not" E_1.place)$

Syntax directed translation of boolean expressions

implementation similar to arithmetic expressions

- $E \rightarrow E_1$ or E_2
  $E.place = newtmp()$
  $emit(E.place``="\ E_1.place``or"\ E_2.place)$
- $E \rightarrow E_1$ and E_2
  $E.place = newtmp()$
  $emit(E.place``="\ E_1.place``and"\ E_2.place)$
- $E \rightarrow not\ E_1$
  $E.place = newtmp()$
  $emit(E.place``="\quad ``not"\ E_1.place)$
- $E \rightarrow (E1) \quad E.place = E_1.place$

Syntax directed translation of boolean expressions

- $E \rightarrow id_1$ relop id_2

Syntax directed translation of boolean expressions

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

Syntax directed translation of boolean expressions

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

Syntax directed translation of boolean expressions

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

Syntax directed translation of boolean expressions

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

Syntax directed translation of boolean expressions

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

Syntax directed translation of boolean expressions

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

state will keep track of line number

- $E \rightarrow true$
  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

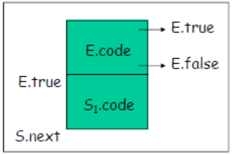
- Translate boolean expressions without
  - generating code for boolean operators
  - evaluating the entire expression
- Flow if control statements

  S $\rightarrow$ if E then S_1

  $|$ if E then S_1 else S_2

  $|$ while E do S_1

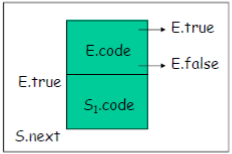
If-then



$S \rightarrow$ if E then S_1

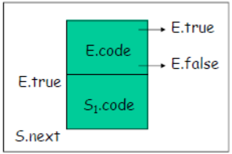
$E.true = newlabel()$

If-then



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

If-then



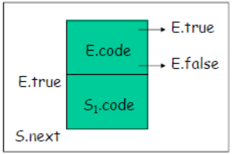
$S \rightarrow$ if E then S_1

$E.true = newlabel()$

$E.false = S.next$

$S_1.next = S.next$

If-then



$S \rightarrow$ if E then S_1

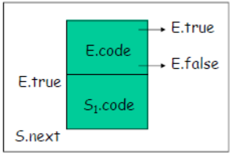
$E.true = newlabel()$

$E.false = S.next$

$S_1.next = S.next$

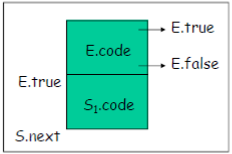
$S.code = E.code||$

If-then



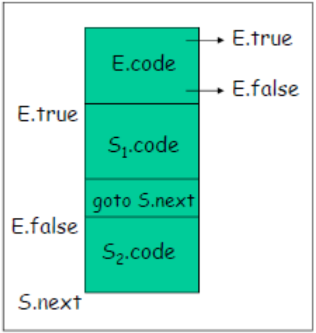
$S \rightarrow$ if E then S_1
$E.true = newlabel()$
$E.false = S.next$
$S_1.next = S.next$
$S.code = E.code||$
$gen(E.true``:")||$

If-then

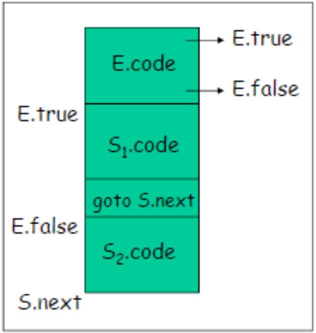


$S \rightarrow$ if E then S_1
$E.true = newlabel()$
$E.false = S.next$
$S_1.next = S.next$
$S.code = E.code||$
$gen(E.true``:")||$
$S1.code$

If-else

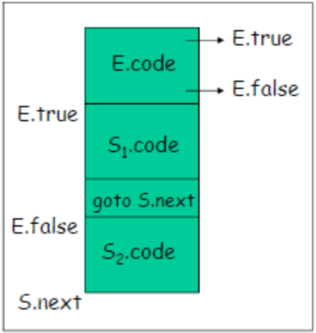

E.true
E.code
E.false
E.true
S_1.code
goto S.next
E.false
S_2.code
S.next

If-else



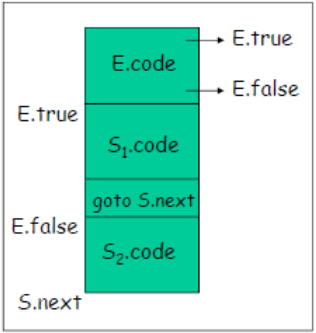
$S \rightarrow$ if E then S_1 else S_2

If-else



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

If-else

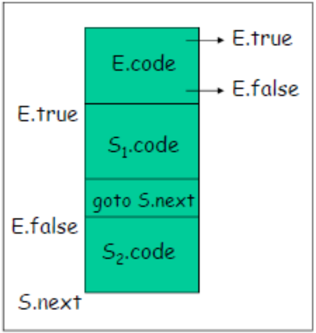


$S \rightarrow$ if E then S_1 else S_2

$E.true = newlabel()$

$E.false = newlabel()$

If-else



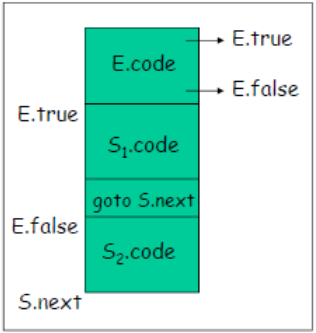
$S \rightarrow$ if E then S_1 else S_2

$E.true = newlabel()$

$E.false = newlabel()$

$S_1.next = S.next$

If-else



$S \rightarrow$ if E then S_1 else S_2

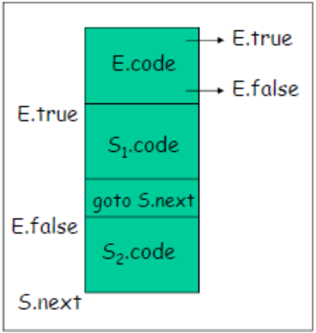
$E.true = newlabel()$

$E.false = newlabel()$

$S_1.next = S.next$

$S_2.next = S.next$

If-else



$S \rightarrow$ if E then S_1 else S_2

$E.true = newlabel()$

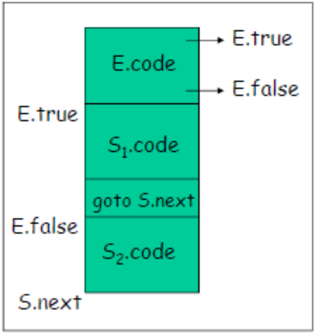
$E.false = newlabel()$

$S_1.next = S.next$

$S_2.next = S.next$

$S.code = E.code||$

If-else



$S \rightarrow$ if E then S_1 else S_2

$E.true = newlabel()$

$E.false = newlabel()$

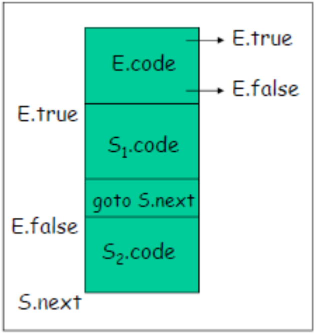
$S_1.next = S.next$

$S_2.next = S.next$

$S.code = E.code||$

$gen(E.true\text{“ : ”})||$

If-else



$S \rightarrow$ if E then S_1 else S_2

$E.true = newlabel()$

$E.false = newlabel()$

$S_1.next = S.next$

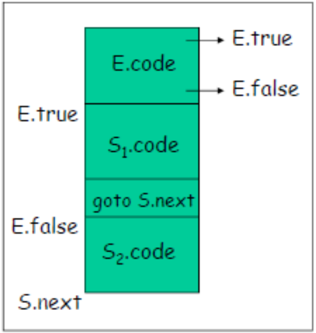
$S_2.next = S.next$

$S.code = E.code||$

$gen(E.true``:")||$

$S_1.code||$

If-else



$S \rightarrow$ if E then S_1 else S_2

$E.true = newlabel()$

$E.false = newlabel()$

$S_1.next = S.next$

$S_2.next = S.next$

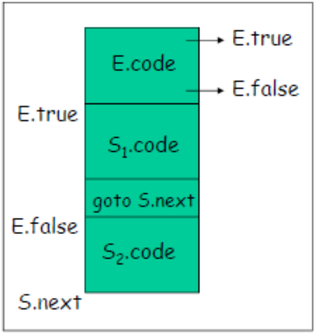
$S.code = E.code||$

$gen(E.true``:")||$

$S_1.code||$

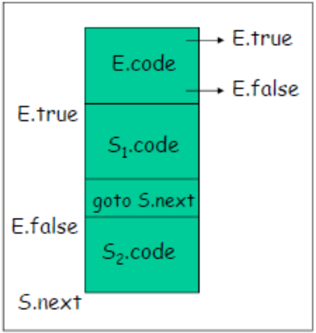
$gen(gotoS.next)||$

If-else



$S \rightarrow$ if E then S_1 else S_2
$E.true = newlabel()$
$E.false = newlabel()$
$S_1.next = S.next$
$S_2.next = S.next$
$S.code = E.code||$
$gen(E.true``:")||$
$S_1.code||$
$gen(gotoS.next)||$
$gen(E.false``:")||$

If-else



$S \rightarrow$ if E then S_1 else S_2

$E.true = newlabel()$

$E.false = newlabel()$

$S_1.next = S.next$

$S_2.next = S.next$

$S.code = E.code||$

$gen(E.true``:")||$

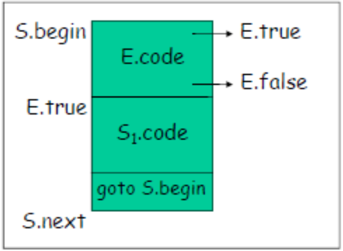
$S_1.code||$

$gen(gotoS.next)||$

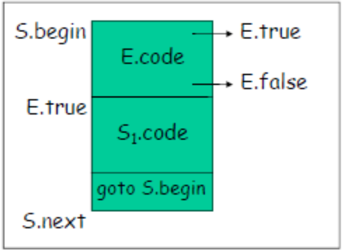
$gen(E.false``:")||$

$S_2.code$

While

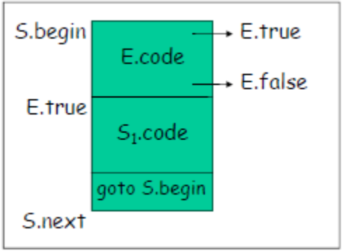


While



$S \rightarrow$ while E do S_1

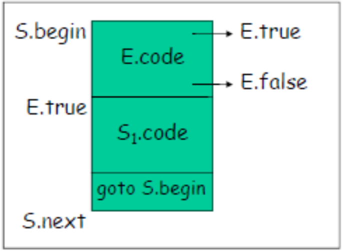
While



$S \rightarrow$ while E do S_1

$S.begin = newlabel()$

While

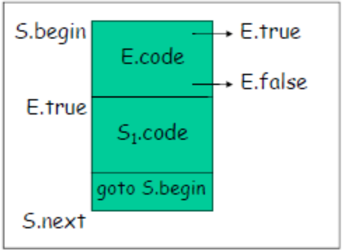


$S \rightarrow$ while E do S_1

$S.begin = newlabel()$

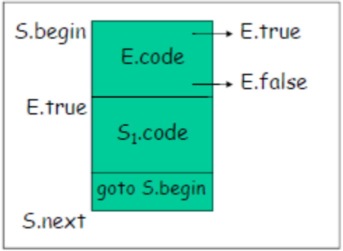
$E.true = newlabel()$

While



$S \rightarrow$ while E do S_1
$S.begin = newlabel()$
$E.true = newlabel()$
$E.false = S.next$

While



$S \rightarrow$ while E do S_1

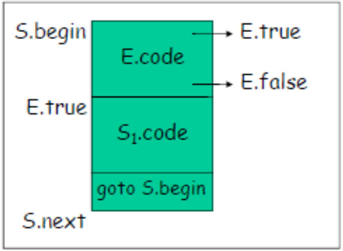
$S.begin = newlabel()$

$E.true = newlabel()$

$E.false = S.next$

$S_1.next = S.begin$

While



$S \rightarrow$ while E do S_1

$S.begin = newlabel()$

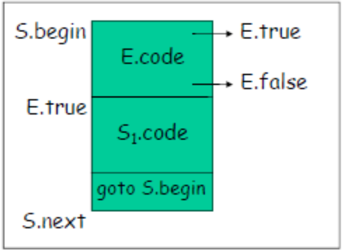
$E.true = newlabel()$

$E.false = S.next$

$S_1.next = S.begin$

$S.ocde = gen(S.begin' :')||$

While



$S \rightarrow$ while E do S_1

$S.begin = newlabel()$

$E.true = newlabel()$

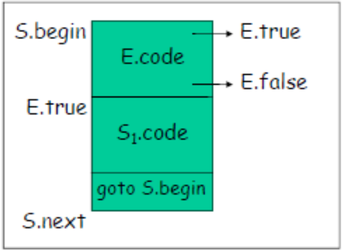
$E.false = S.next$

$S_1.next = S.begin$

$S.ocde = gen(S.begin' :')||$

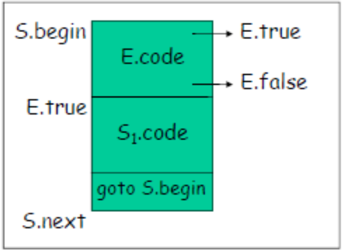
$E.code||$

While



$S \rightarrow$ while E do S_1
$S.begin = newlabel()$
$E.true = newlabel()$
$E.false = S.next$
$S_1.next = S.begin$
$S.ocde = gen(S.begin' :')||$
$E.code||$
$gen(E.true' :')||$

While



$S \rightarrow$ while E do S_1

$S.begin = newlabel()$

$E.true = newlabel()$

$E.false = S.next$

$S_1.next = S.begin$

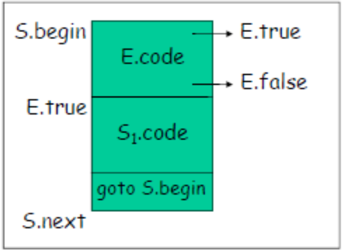
$S.ocde = gen(S.begin' :')||$

$E.code||$

$gen(E.true' :')||$

$S_1.code||$

While



$S \rightarrow$ while E do S_1

$S.begin = newlabel()$

$E.true = newlabel()$

$E.false = S.next$

$S_1.next = S.begin$

$S.ocde = gen(S.begin' :')||$

$E.code||$

$gen(E.true' :')||$

$S_1.code||$

$gen(gotoS.begin)$

Control flow translation of boolean expression

- $E \rightarrow E_1$ or E_2

Control flow translation of boolean expression

- $E \rightarrow E_1$ or E_2 $\quad E_1.true = E.true$

  $E_1.false = \text{newlabel()}$

  $E_2.true = \text{E.true}$

  $E_2.false = \text{E.false}$

  $\text{E.code} = E_1\text{.code} \mid\mid \text{gen}(E_1\text{.false})$

  $\qquad\qquad \mid\mid E_2\text{.code}$

Control flow translation of boolean expression

- $E \rightarrow E_1$ or E_2 $\quad E_1.true = E.true$
  $E_1.false =$ newlabel()
  $E_2.true =$ E.true
  $E_2.false =$ E.false
  E.code $= E_1$.code $||$ gen(E_1.false)
  $||$ E_2.code
- $E \rightarrow E_1$ and E_2

Control flow translation of boolean expression

- $E \rightarrow E_1$ or E_2 $\quad E_1.true = E.true$

  $E_1.false = \text{newlabel()}$

  $E_2.true = \text{E.true}$

  $E_2.false = \text{E.false}$

  $\text{E.code} = E_1.\text{code} \mid\mid \text{gen}(E_1.\text{false})$

  $\qquad\qquad \mid\mid E_2.\text{code}$

- $E \rightarrow E_1$ and E_2 $\quad E_1.\text{true} = \text{newlabel()}$

  $E_1.\text{false} = \text{E.false}$

  $E_2.\text{true} = \text{E.true}$

  $E_2.\text{false} = \text{E.false}$

  $\text{E.code} = E_1.\text{code} \mid\mid \text{gen}(E_1.\text{true})$

  $\qquad\qquad \mid\mid E_2.\text{code}$

short circuit evaluation.