ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

# Bài 11
# Sinh mã trung gian

# Nội dung

- Mã ba địa chỉ
- Sinh mã cho lệnh gán
- Sinh mã cho các biểu thức logic
- Sinh mã cho các cấu trúc lập trình

# Mã trung gian

- Một chương trình với mã nguồn được chuyển sang chương trình tương đương trong ngôn ngữ trung gian bằng bộ sinh mã trung gian.
- Ngôn ngữ trung gian được người thiết kế trình biên dịch quyết định, có thể là:
  - Cây cú pháp
  - Ký pháp Ba Lan sau (hậu tố)
  - Mã 3 địa chỉ …

# Mã trung gian

- Được sản sinh dưới dạng một chương trình cho một máy trừu tượng
- Mã trung gian thường dùng : mã ba địa chỉ, tương tự mã assembly
- Chương trình là một dãy các lệnh. Mỗi lệnh gồm tối đa 3 định danh.
- Tồn tại nhiều nhất một toán tử ở vế phải cộng thêm một toán tử gán
- x,y,z là các địa chỉ , tức là tên, hằng hay các tên trung gian do trình biên dịch sinh ra
  - Tên trung gian phải được sinh để thực hiện các phép toán trung gian
  - Các địa chỉ được thực hiện thường là con trỏ tới lối vào của nó trong bảng ký hiệu

# Mã trung gian của t2=x + y * z

- t1 := y*z
- t2 := x+t1

# Tập mã lệnh ba địa chỉ điển hình

- Mã 3 địa chỉ tương tự mã Assembly: lệnh có thể có nhãn, có những lệnh chuyển điều khiển cho các cấu trúc lập trình.
  1. *Lệnh gán* x := y *op* z.
  2. *Lệnh gán với phép toán 1 ngôi* : x := *op* y.
  3. *Lệnh sao chép*: x := y.
  4. *Lệnh nhảy không điều kiện*: *goto L, L là nhãn của một lệnh*
  5. *Lệnh nhảy có điều kiện x relop y goto L.*

# Tập mã lệnh ba địa chỉ điển hình

6. Lời gọi thủ tục param x và call p,n để gọi thủ tục p với n tham số . Return y là giá trị thủ tục trả về

   param $x_1$

   param $x_2$

   . . .

   param $x_n$

   Call p,n

7. Lệnh gán có chỉ số x:=y[i] hay x[i]:=y

# Sinh mã trực tiếp từ ĐNTCP

- Thuộc tính tổng hợp S.code biểu diễn mã ba địa chỉ của lệnh
- Các tên trung gian được sinh ra cho các tính toán trung gian
- Các biểu thức được liên hệ với hai thuộc tính tổng hợp
  - E.place chứa địa chỉ chứa giá trị của E
  - E.code mã ba địa chỉ để đánh giá E
- Hàm newtemp sinh ra các tên trung giant1, t2,. .
- Hàm gen sinh mã ba địa chỉ
- Trong thực tế, code được gửi vào file thay cho thuộc tính code

# Dịch trực tiếp cú pháp thành mã 3 địa chỉ

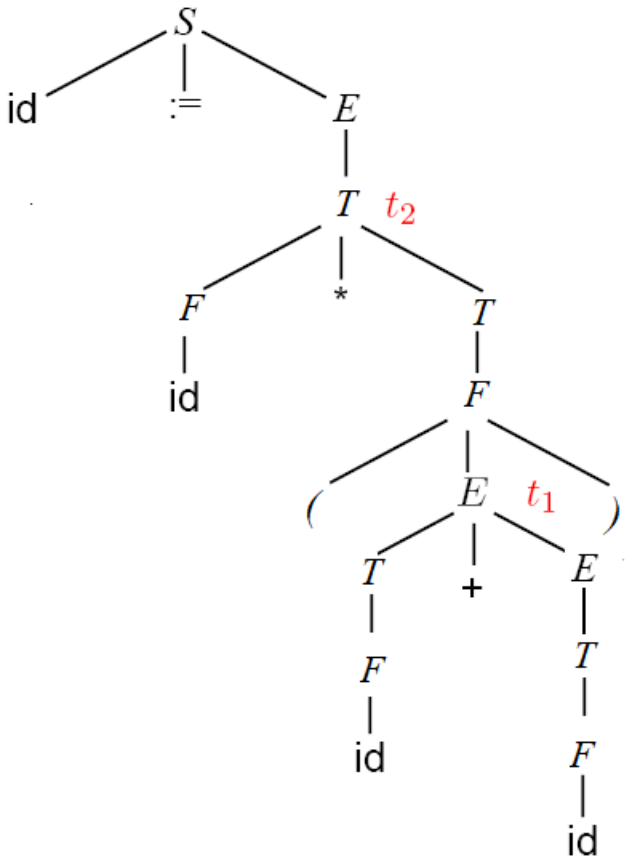
| Sản xuất | Quy tắc ngữ nghĩa |
|---|---|
| S → id := E | {S.*code* = E.*code* \|\| *gen*(id.*place* ':=' E.*place*)} |
| E → T+$E_1$ | {E.*place* = *newtemp* ; E.*code* = T.*code* \|\| $E_1$.*code* \|\| *gen*(E.*place*':='T.*place*'+'$E_1$.*place*) } |
| E → T | {E.*place* = T.*place* ; E.*code* = T.*code*} |
| T → F* $T_1$ | {T.*place* = *newtemp* ; T.*code* = F.*code* \|\| $T_1$.*code* \|\| *gen*(T.*place*':='F.*place*'*'$T_1$.*place*) } |
| T → F | {T.*place* = F.*place* ; T.*code* = *F.code*} |
| F → (E) | {F.*place*= E.*place* ; F.*code* = E.*code*} |
| F → id | {F.*place* = id.*place* ; F.*code* = '' } |

Hàm ***newtemp*** trả về một dãy các tên khác nhau $t_1$, $t_2$… cho lời gọi kế tiếp.

E.*place:* là tên sẽ giữ giá trị của E

E.*code:* là dãy các câu lệnh 3 địa chỉ dùng để ước lượng E

# Mã cho lệnh gán a := b * (c + d)




$$t_1 := c + d$$
$$t_2 := b * t_1$$
$$a := t_2$$

# Cài đặt câu lệnh 3 địa chỉ

**Bộ bốn (Quadruples)**

$t_1$:= c + d

$t_2$:=b * $t_1$

a:=$t_2$

| | *op* | *arg1* | *arg2* | *result* |
|---|---|---|---|---|
| (0) | + | c | d | $t_1$ |
| (1) | * | b | $t_1$ | $t_2$ |
| (2) | := | $t_2$ | | a |

**Tên tạm phải được thêm vào bảng kí hiệu khi chúng được tạo ra.**

# Cài đặt câu lệnh 3 địa chỉ

- Bộ ba (Triples)

$t_1$:= c+d

$t_2$:=b * $t_1$

a:=$t_2$

| | *op* | *arg1* | *arg2* |
|---|---|---|---|
| (0) | + | c | d |
| (1) | * | b | (0) |
| (2) | assign | a | (1) |

**Tên tạm không được thêm vào trong bảng kí hiệu.**

# Các dạng khác của câu lệnh 3 địa chỉ

- Ví dụ:
  x[i]:=y  x:=y[i]
- Sử dụng 2 cấu trúc bộ ba

| | *op* | *arg1* | *arg2* |
|---|---|---|---|
| (0) | [ ] | x | i |
| (1) | := | (0) | y |

| | *op* | *arg1* | *arg2* |
|---|---|---|---|
| (0) | [ ] | y | i |
| (1) | := | x | (0) |

# Cài đặt câu lệnh 3 địa chỉ

- Bộ 3 gián tiếp: sử dụng một danh sách các con trỏ các bộ 3

| | *op* | | *op* | *arg1* | *arg2* |
|---|---|---|---|---|---|
| (0) | (14) | (14) | uminus | c | |
| (1) | (15) | (15) | * | b | (14) |
| (2) | (16) | (16) | uminus | c | |
| (3) | (17) | (17) | * | b | (16) |
| (4) | (18) | (18) | + | (15) | (17) |
| (5) | (19) | (19) | assign | a | (18) |

# ĐNTCP để sinh ra mã lệnh 3 địa chỉ cho lệnh gán

$S \rightarrow$ id := E  {*p:=lookup( id.name);*
*if p <> nil then emit( p ':=' E.place) else error* }

$E \rightarrow E_1 + E_2$  { *E.place := newtemp;*
*emit(E.place ':=' $E_1$.place '+' $E_2$.place)* }

$E \rightarrow E_1 * E_2$  { *E.place := newtemp;*
*emit(E.place ':=' $E_1$.place '*' $E_2$.place)* }

$E \rightarrow - E_1$  { *E.place := newtemp;*
*emit(E.place ':=' 'unimus' $E_1$.place)* }

$E \rightarrow ( E_1 )$  { *E.place:=$E_1$.place)* }

$E \rightarrow$ id  { *p:=lookup( id.name);*
*if p <> nil then E.place := p else error* }

# Tên trong bảng kí hiệu

- Hàm lookup sẽ tìm trong bảng kí hiệu xem có hay không một tên được cho bởi *id.name*. Nếu có thì trả về con trỏ của ô, nếu không thì trả về nil.

- Thủ tục emit để đưa mã 3 địa chỉ vào một tập tin output chứ không xây dựng thuộc tính code cho các kí hiệu chưa kết thúc như gen. Quá trình dịch thực hiện bằng cách đưa ra một tập tin output nếu thuộc tính code của kí hiệu không kết thúc trong vế trái sản xuất được tạo ra bằng cách nối thuộc tính code của kí hiệu không kết thúc trong vế phải theo đúng thứ tự xuất hiện của các kí hiệu chưa kết thúc ở vế phải.

# Tên trong bảng kí hiệu

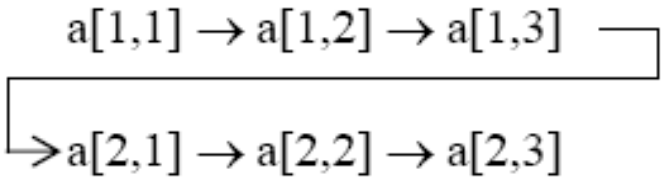
- Xét sản xuất D → proc id; $ND_1$; S
- Các tên trong lệnh gán sinh ra bởi kí hiệu không  kết thúc S sẽ được khai báo trong chương trình con này hoặc trong chương trình chứa nó.
- Khi khai báo tới một tên thì trước hết hàm lookup sẽ tìm xem tên đó có trong bảng kí hiệu hiện hành hay không, nếu không thì dùng con trỏ trong header của bảng để tìm bảng kí hiệu bao nó và tìm trong đó, nếu không tìm thấy trong tất cả các mức thì lookup trả về nil.

# Địa chỉ hóa các phần tử của mảng

- Các phần tử của mảng có thể truy xuất nhanh nếu chúng được lưu trữ trong một khối ô nhớ kế tiếp nhau. Trong mảng một chiều, nếu kích thước của một phần tử là w thì địa chỉ tương đối phần tử thứ i của mảng A được tính theo công thức:
- A[i] = base + (i-low)*w
- Trong đó:
    - Low: cận dưới tập chỉ số
    - Base: địa chỉ tương đối của ô nhớ cấp phát cho mảng(địa chỉ tương đối của A[low])
- Tương đương A[i] = i*w + (base – low*w)
- Trong đó:
    - c = base – low*w có thể được tính tại thời gian dịch và lưu trong bảng kí hiệu
- $\Rightarrow$ A[i] = i*w + c

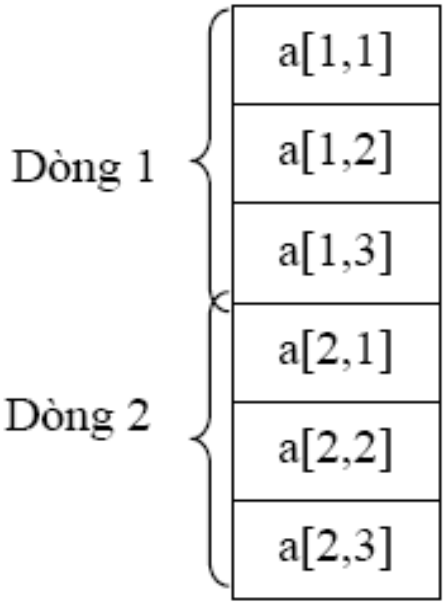
# Địa chỉ hóa các phần tử của mảng 2 chiều

- Theo dòng




Địa chỉ tương đối của $A[i_1,i_2]$ =

$base + ((i_1 - low_1)*n_2 + i_2 - low_2)*w$

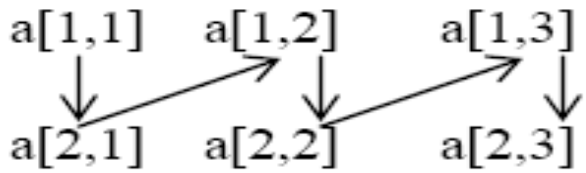


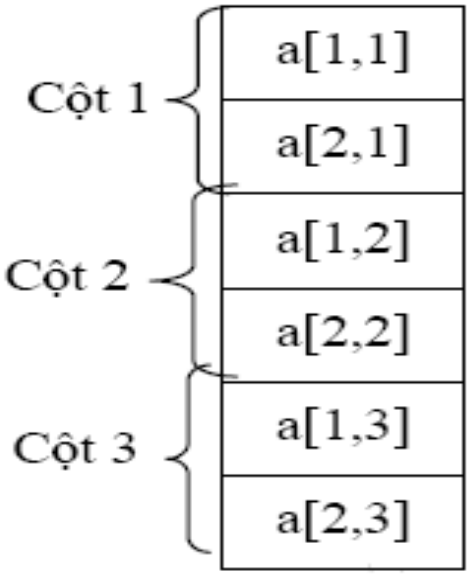

$low_1$, $low_2$: cận dưới cho i1 và i2

n2: số lượng các giá trị mà i2 có thể nhận. Nếu high2 là cận trên của i2 thì n2 = high2 – low2 + 1

# Địa chỉ hóa các phần tử của mảng 2 chiều

- Theo cột



# Sinh mã biểu thức Logic

- Biểu thức logic được sinh bởi văn phạm sau:

  B→ B or B | B and B | not B | (B) | id relop id | true |false

- Trong đó:
  - Or và And kết hợp trái
  - Or có độ ưu tiên thấp nhất tiếp theo là And, và Not
  - Những thông tin này có thể them vào bộ phân tích cú pháp dưới lên sử dụng quan hệ thứ bậc toán tử. Kết quả cho 1 cây phân tích cú pháp với các phép toán được thực hiện theo đúng thứ tự ưu tiên

# Biểu diễn bằng số

- Mã hóa true và false bằng các số và ước lượng một biểu thức boole tương tự như đối với biểu thức số học
- Có thể biểu diễn true là 1; false là 0
- Hoặc các số khác 0 là true, 0 là false

Ví dụ: biểu thức a or b and not c

- Mã 3 địa chỉ:

```
t1 = not c
t2 = b and t1
t3 = a or t2
```

- Biểu thức quan hệ a<b tương đương lệnh điều kiện if a<b then 1 else 0. Mã 3 địa chỉ tương ứng:

```
100: if a<b goto 103
101: t:=0
102: goto 104
103: t:= 1
104:
```

# ĐNTCP dùng số để biểu diễn các giá trị logic

| Sản xuất | Quy tắc ngữ nghĩa |
|---|---|
| $B \rightarrow B_1$ or $B_2$ | B.place:= newtemp();<br>emit( B.place ':=' B1.place 'or' B2.place) |
| $B \rightarrow B_1$ and $B_2$ | B.place:= newtemp();<br>emit( B.place ':=' B1.place 'and' B2.place) |
| $B \rightarrow$ not $B_1$ | B.place := newtemp();<br>emit(B.place ':=' 'not' B1.place) |
| $B \rightarrow id_1$ relop $id_2$ | B.place := newtemp();<br>emit('if' id1.place relop id2.place 'goto' nextstat + 3);<br>emit(B.place':=''0') ;emit('goto' nextstat + 2)<br>emit(B.place':=''1') |
| $B \rightarrow$ true | B.place = newtemp(); emit (B.place ':=''1') |
| $B \rightarrow$ false | B.place = newtemp(); emit (B.place ':=''0') |

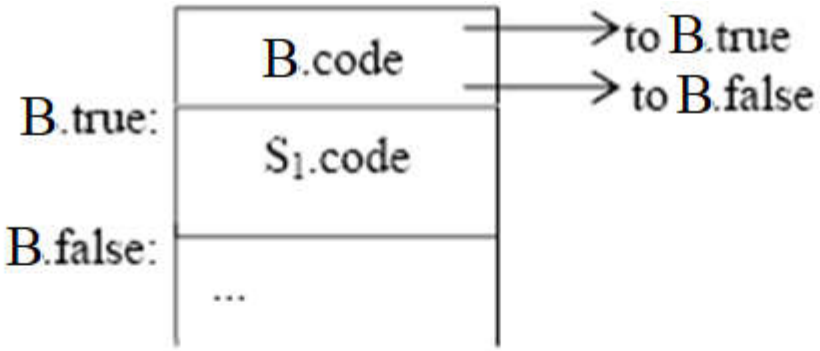
Nextstat cho biết nhãn của câu lệnh 3 địa chỉ tiếp theo.

Emit: đặt câu lệnh 3 địa chỉ vào tập tin, emit làm tăng nextstat sau khi thực hiện

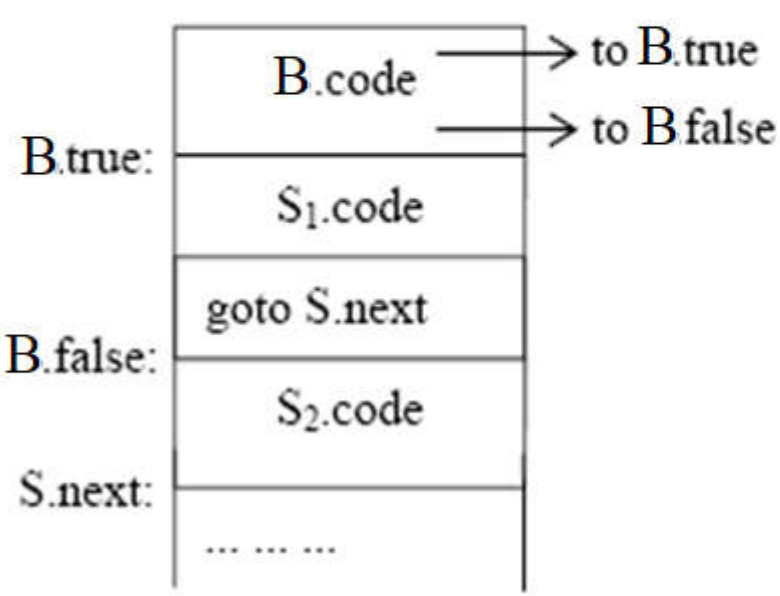
# Sinh mã cho các cấu trúc lập trình

- Biểu diễn các giá trị của biểu thức Boole bằng biểu thức đã đến được trong một chương trình.
- Ví dụ: cho câu lệnh sau
- S→ if B then $S_1$ | iF B then $S_1$ else $S_2$ | while B do $S_1$
- Với mỗi biểu thức B chúng ta kết hợp với 2 nhãn:
  - B.true: nhãn của dòng điều khiển nếu B là true
  - B.false: nhãn của dòng điều khiển nếu B là false
  - S.code: mã lệnh 3 địa chỉ được sinh ra bởi S
  - S.next: là nhãn mã lệnh 3 địa chỉ đầu tiên sẽ thực hiện sau mã lệnh của S
  - S.begin: nhãn địa chỉ lệnh đầu tiên được sinh ra cho S là lệnh while

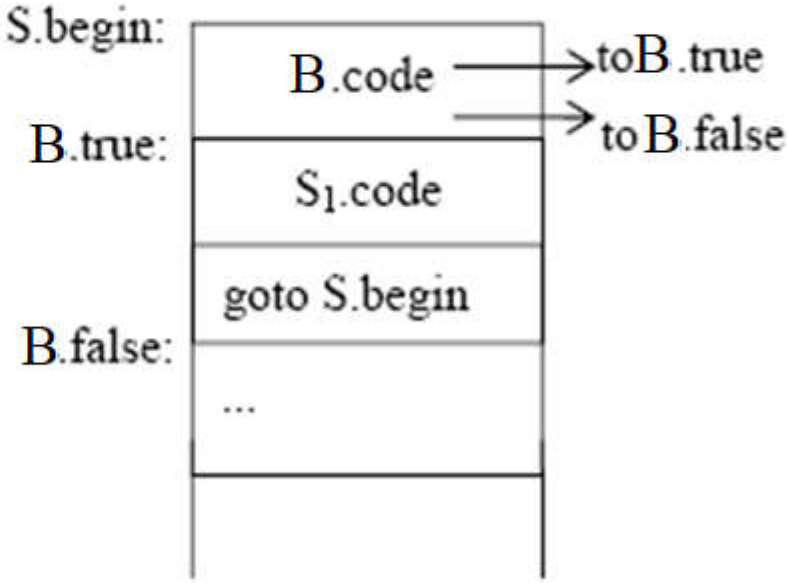
# Mã lệnh của các lệnh if-then, if-then-else, while-do




(a) if -then




(b) if -then-else




(c) while-do

# ĐNTCP cho các cấu trúc lập trình

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $P \rightarrow S$ | $S.next = newlabel()$<br>$P.code = S.code \mid\mid label(S.next)$ |
| $S \rightarrow$ **assign** | $S.code =$ **assign**$.code$ |
| $S \rightarrow$ **if** $B$ **then** $S_1$ | $B.true = newlabel()$<br>$B.false = S_1.next = S.next$<br>$S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$ |
| $S \rightarrow$ **if** $B$ **then** $S_1$ **else** $S_2$ | $B.true = newlabel()$<br>$B.false = newlabel()$<br>$S_1.next = S_2.next = S.next$<br>$S.code = B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ S.next)$<br>$\mid\mid label(B.false) \mid\mid S_2.code$ |
| $S \rightarrow$ **while** $B$ **do** $S_1$ | $begin = newlabel()$<br>$B.true = newlabel()$<br>$B.false = S.next$<br>$S_1.next = begin$<br>$S.code = label(begin) \mid\mid B.code$<br>$\mid\mid label(B.true) \mid\mid S_1.code$<br>$\mid\mid gen('\texttt{goto}'\ begin)$ |
| $S \rightarrow S_1\ S_2$ | $S_1.next = newlabel()$<br>$S_2.next = S.next$<br>$S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$ |

# Dịch biểu thức logic trong các cấu trúc lập trình

- Nếu B có dạng: a<b thì mã lệnh sinh ra có dạng

  If a<b then goto B.true else goto B.false

- Nếu B có dạng: B1 or B2 thì
  - Nếu B1 là true thì B cũng là true
  - Nếu B1là false thì phải đánh giá B2; B sẽ là true hay false phụ thuộc B2
- Tương tự với B1 and B2

# Dịch biểu thức logic trong các cấu trúc lập trình

| SẢN XUẤT | QUY TẮC NGỮ NGHĨA |
|---|---|
| $E \rightarrow E_1$ or $E_2$ | $E_1.true := E.true;$<br>$E_1.false := newlabel;$<br>$E_2.true := E.true;$<br>$E_2.false := E.false;$<br>$E.code := E_1.code \mid\mid gen(E.false ':') \mid\mid E_2.code$ |
| $E \rightarrow E_1$ and $E_2$ | $E_1.true := newlabel;$<br>$E_1.false := E.false;$<br>$E_2.true := E.true;$<br>$E_2.false := E.false;$<br>$E.code := E_1.code \mid\mid gen(E.true ':') \mid\mid E_2.code$ |
| $E \rightarrow$ not $E_1$ | $E_1.true := E.false;$<br>$E_1.false := E.true;$<br>$E.code := E_1.code$ |
| $E \rightarrow (E_1)$ | $E_1.true := E.true;$<br>$E_1.false := E.false;$<br>$E.code := E_1.code$ |
| $E \rightarrow id_1$ relop $id_2$ | $E.code := gen('if' id_1.place\ relop.op\ id_2.place$<br>$'goto' E.true) \mid\mid gen('goto' E.false)$ |
| $E \rightarrow$ true | $E.code := gen('goto' E.true)$ |
| $E \rightarrow$ false | $E.code := gen('goto' E.false)$ |

# Biểu thức logic ở dạng hỗn hợp

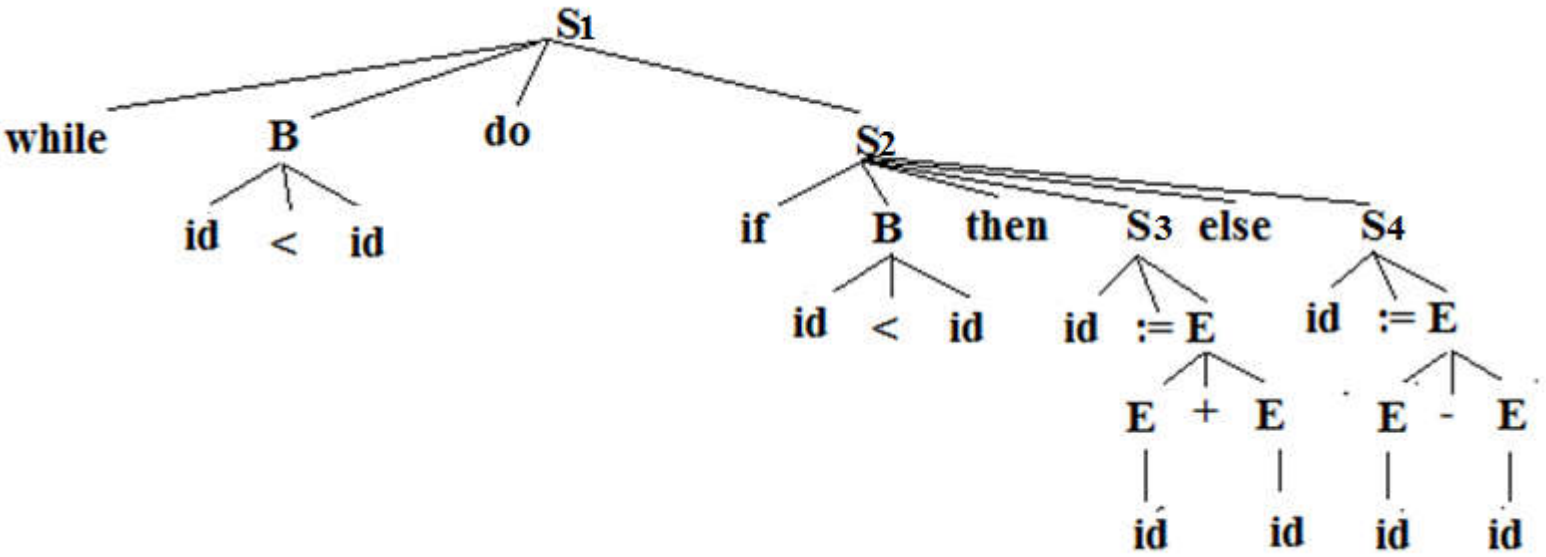
- Thực tế, các biểu thức logic thường chứa các biểu thức số học như trong (a+b)<c
- Để đơn giản, ta vẫn dùng 2 ký hiệu không kết thúc E cho biểu thức số học và B cho biểu thức logic
- Nếu dung chung một lý hiệu không kết thúc, cần lưu them thuộc tính kind để chỉ biểu thức là số học hay logic

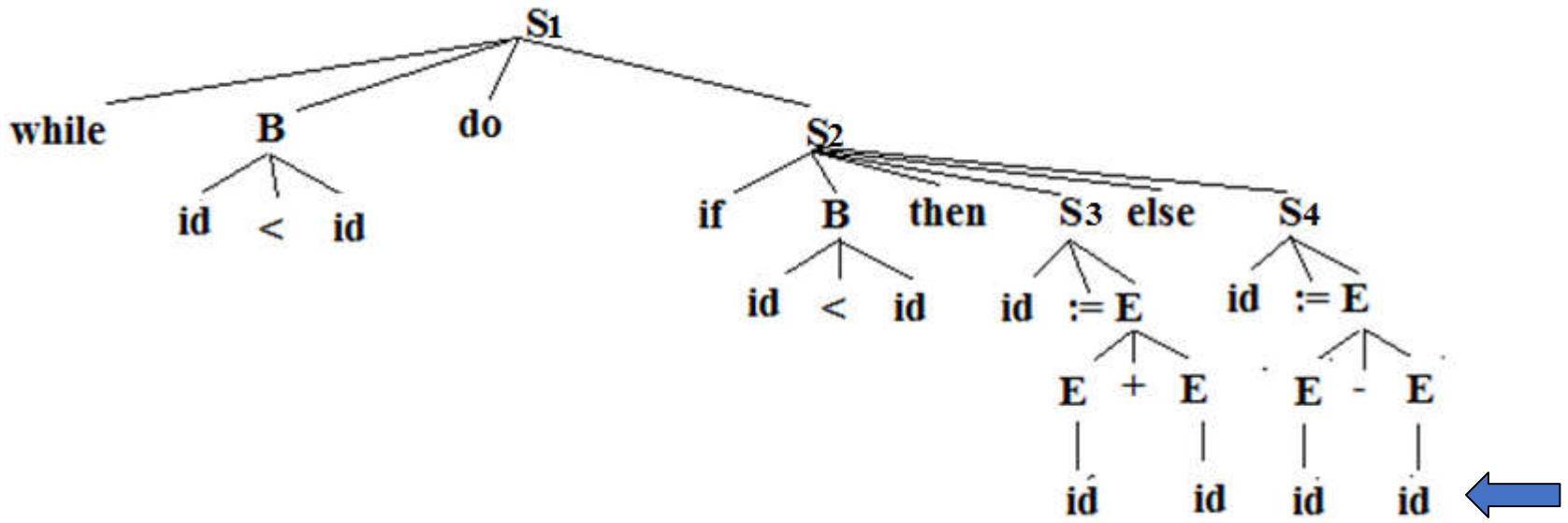
# Ví dụ

- Sinh mã trung gian cho lệnh sau:

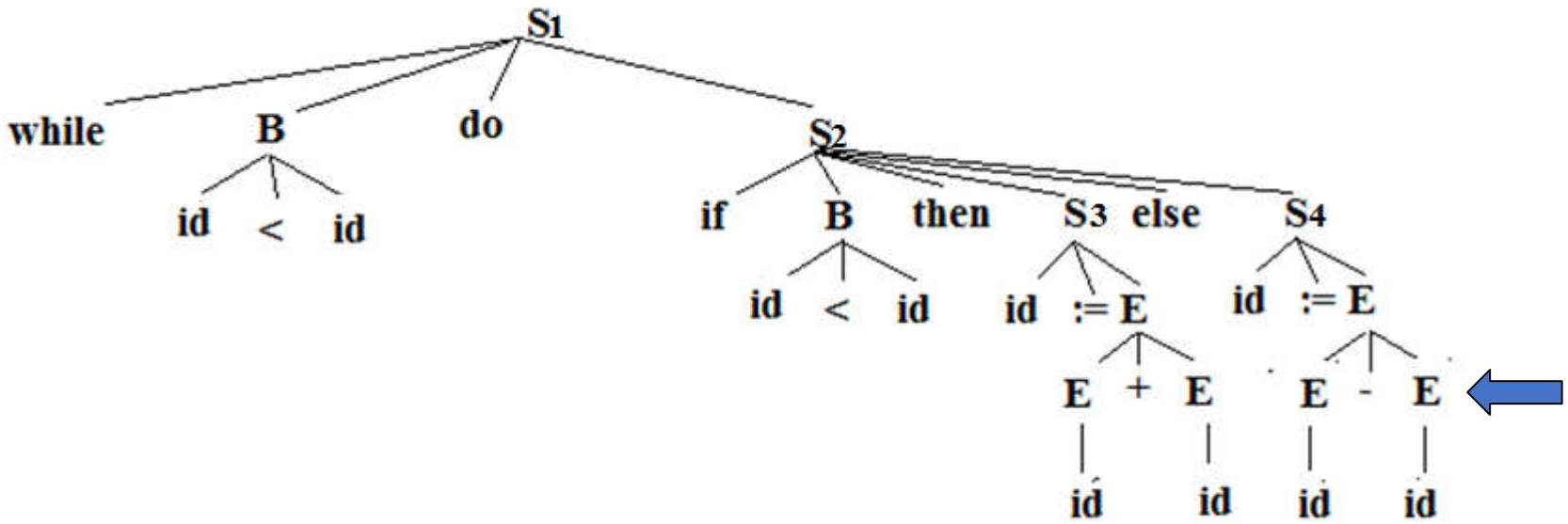
```
while a < b do
if c < d then
x := y + z
else
x := y - z
```

# Cây PT cú pháp của lệnh






| Thuộc tính |
| --- |
| id.place =z |




**Quy tắc ngữ nghĩa**

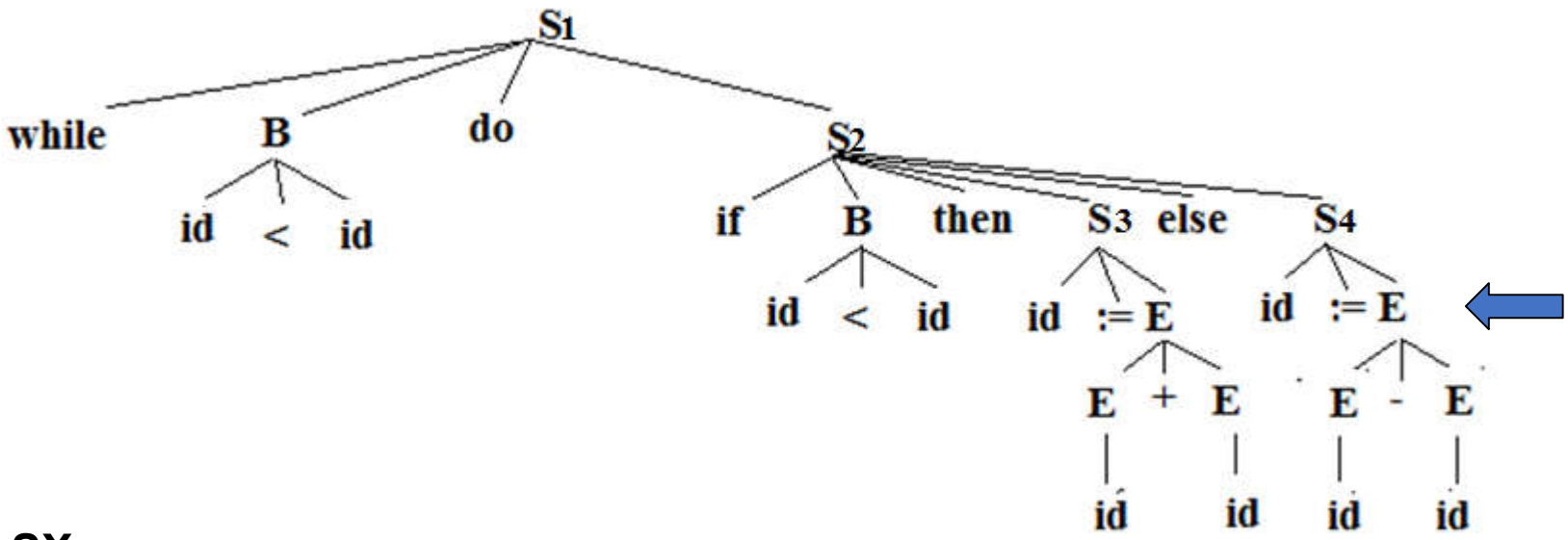
E.place = id.place

E.code=“”

**Thuộc tính**

E.place =z

E.code=“”




**SX:**

$E \rightarrow E_1 - E_2$

**Quy tắc ngữ nghĩa**

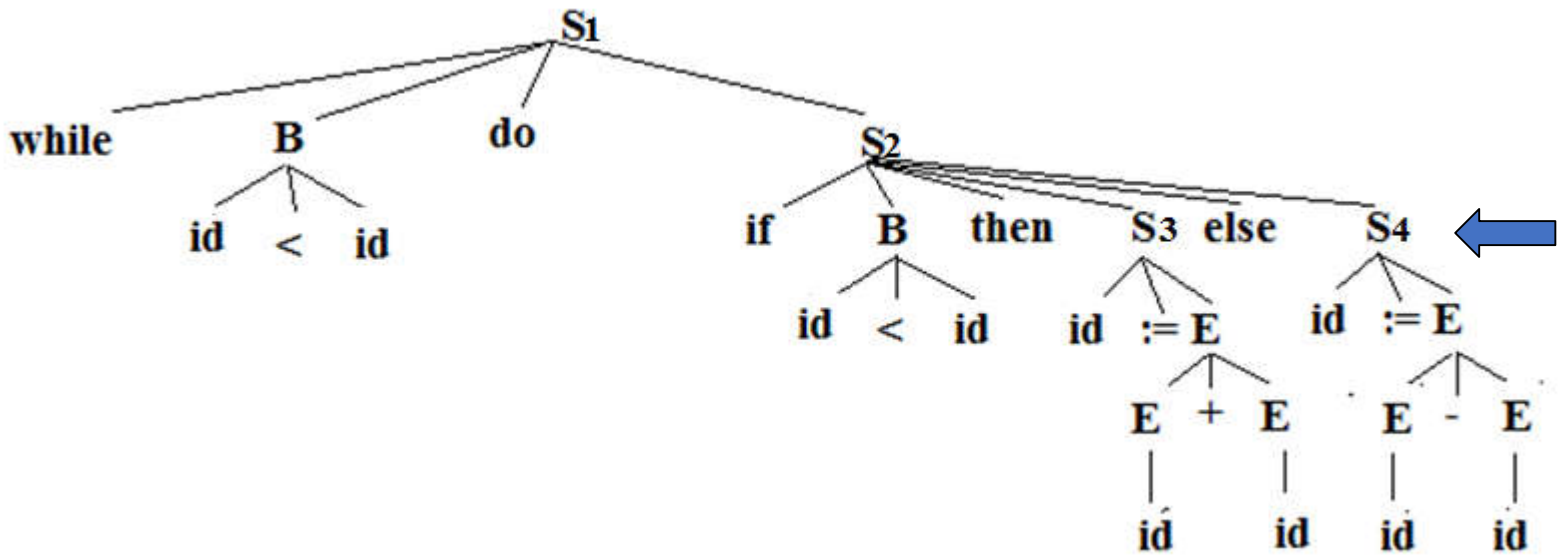
E.place = newtemp()  t1

E.Code =$E_1$.code || $E_2$.code ||gen (E.place := $E_1$.place "-" $E_2$.place)

**Thuộc tính**

E.place = newtemp()  t1

E.code= "t1:=y-z"



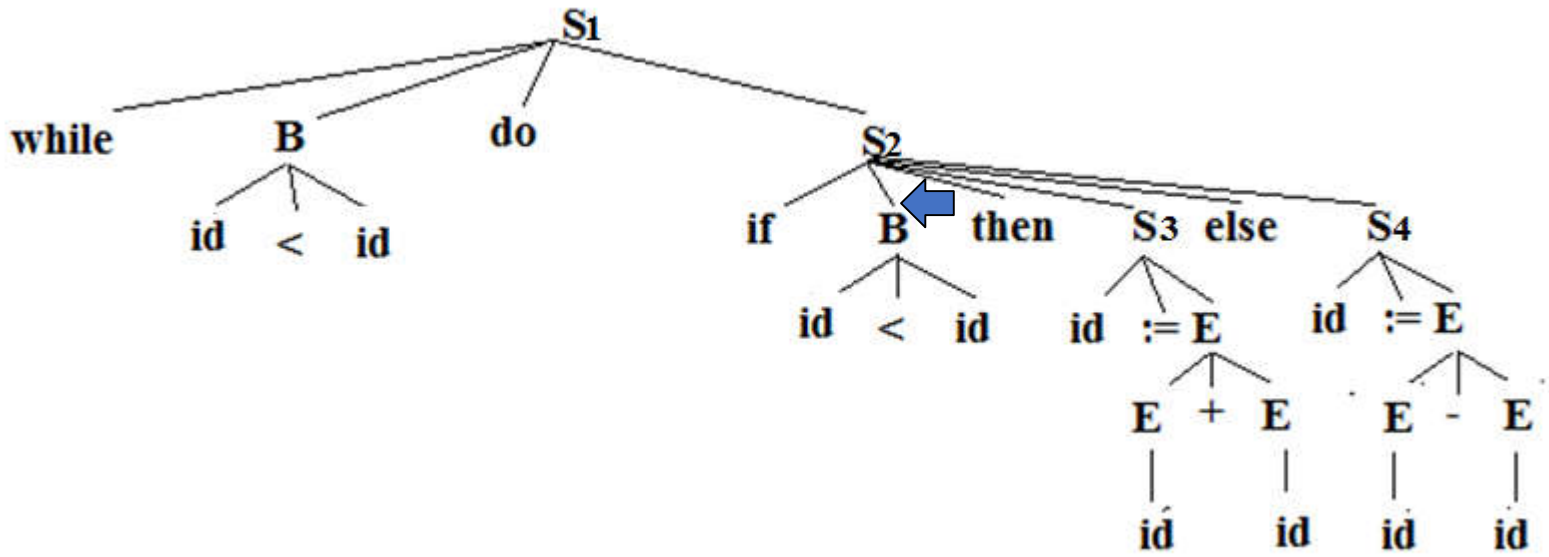

| SX | Quy tắc ngữ nghĩa |
|---|---|
| S → id := E | { S.*code* = E.*code*\|\|*gen*(id.*place* ':=' E.*place*) } |

**Thuộc tính**

id.place = x

S.code= "t1:=y-z

x:=t1"




**SX:**

$B \rightarrow E_1 < E_2$

**QTNN:**

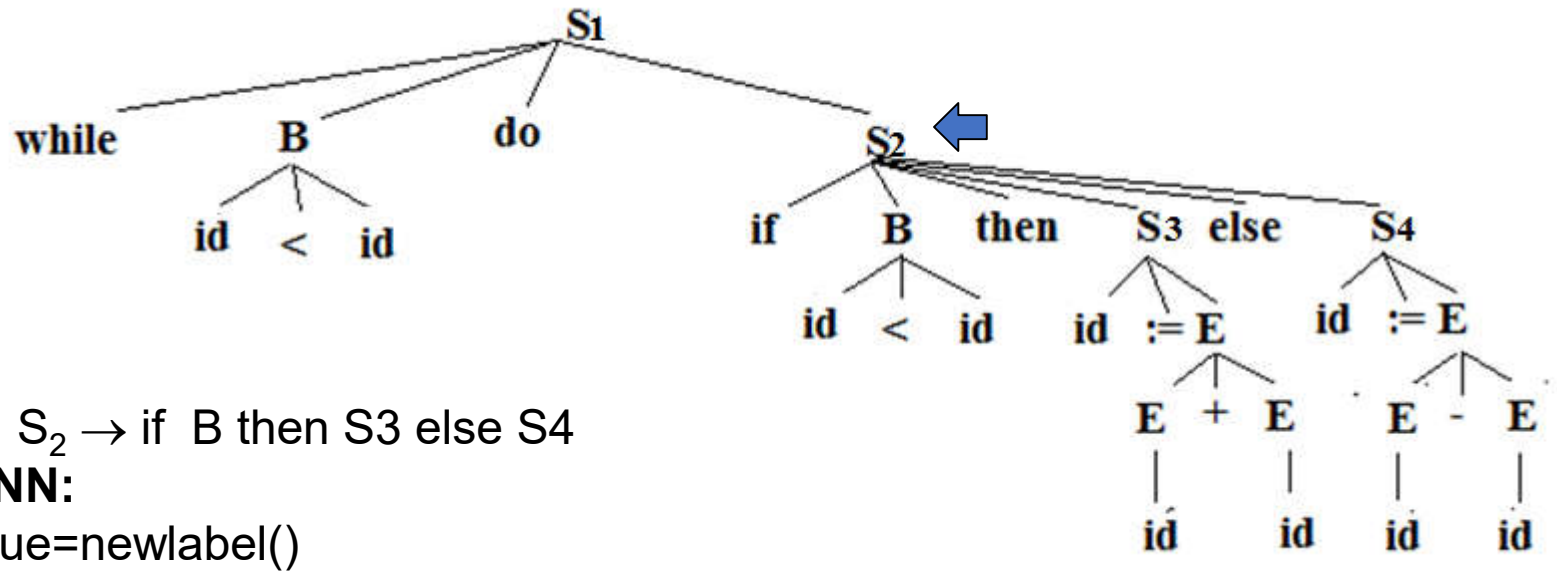
B.code = $E_1$.code || $E_2$.code || gen (‘if’ $E_1$.place < $E_2$.place) ‘goto’ B. true || gen (‘goto’ B.false )

**Thuộc tính**

B.code

if c < d goto $L_1$

goto $L_2$




**SX:** $S_2 \rightarrow$ if B then S3 else S4

**QTNN:**

B.true=newlabel()

B.false=newlabel()

$S_3$.next = $S_4$.next = $S_2$.next

$S_2$.code = B.code || label (B.true) || $S_3$.code || gen('goto' $S_2$.next || label(B.false) || $S_4$.code

**Thuộc tính (B của lệnh if)**

B.true = newlabel() -> $L_1$

B.false= newlable() -> $L_2$

$S_2$.next = $S_3$.next= $S_4$.next

$S_2$.code

    if c < d goto $L_1$

    goto $L_2$

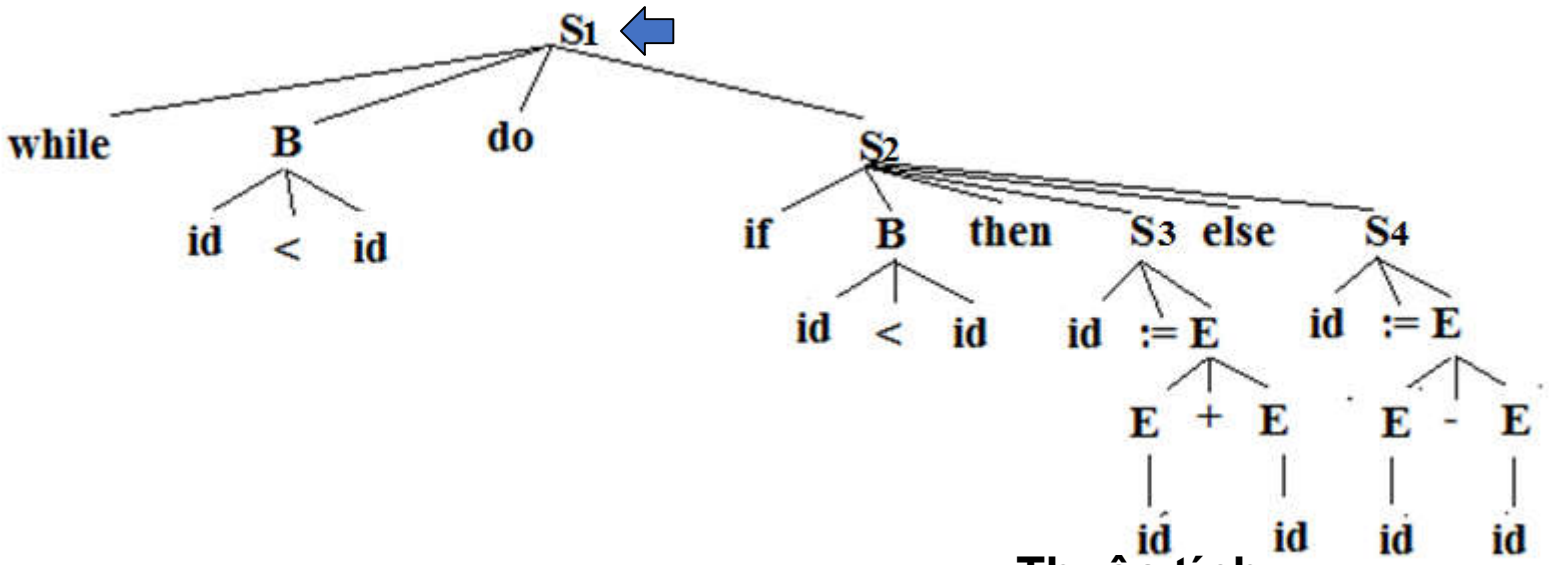
$L_1$: t2= y + z

    x = t2

    goto $S_2$.next

$L_2$: t1 = y – z

    x = t1




**SX:**

**$S_1 \rightarrow$ while B do $S_2$**

**QTNN:**

$S_1$.begin = newlabel()

B.True = newlabel()

B.False = $S_1$.next

$S_1$.code =

$S_1$.begin || B.code

|| B.true || $S_2$.code

||gen ('goto' $S_1$.begin)

**Thuộc tính**

$S_1$.begin = newlabel() -> $L_3$

B.true= newlabel() -> $L_4$ B của lệnh while

B.False = S.next

$S_2$.next = begin = $L_3$ => $S_2$.next = $S_3$.next = $L_3$

S.code

$L_3$: if a< b goto $L_4$

goto $L_0$

$L_4$: if c < d goto $L_1$

goto $L_2$

$L_1$: t1= y + z

x = t1

goto $L_3$

$L_2$:t2 = y – z

x = t1

goto L3

Nhãn $L_0$ sẽ xuất hiện trong chương trình khi sử dụng các quy tắc ngữ nghĩa của luật $S \rightarrow S_1S_2$