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 toán hạng
- Tồn tại nhiều nhất một toán tử ở vế phải cộng thêm một toán tử gán
- Dạng tổng quát: x := y *op* z
- 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 như con trỏ tới lối vào của nó trong bảng ký hiệu

Mã trung gian của x + y * z

- t_1 := y*z
- t_2 := x+t_1

Các dạng mã ba địa chỉ phổ biến

- Mã 3 địa chỉ tương tự mã Assembly: lệnh có thể có nhãn, có những lệnh chuyển điều khiểnolcho 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.

Các dạng mã ba địa chỉ

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

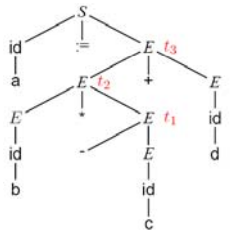
- Thuộc tính tổng hợp S.code biểu diễn mã ba địa chỉ của lệnh
- Các tên trung gian được sinh ra cho các tính toán trung gian
- Các biểu thức được liên hệ với hai thuộc tính tổng hợp
- E.place chứa địa chỉ chứa giá trị của E
- E.code mã ba địa chỉ để đánh giá E
- Hàm newtemp sinh ra các tên trung giant1, 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
$S \rightarrow id := E$	{ S.*code* = E.*code*||*gen*(id.*place* ':=' E.*place*) }
$E \rightarrow E_1 + E_2$	{E.*place*= *newtemp* ; E.*code* = E_1.*code* || E_2.*code* || || *gen*(E.*place*':='E_1.*place*'+'E_2.*place*) }
$E \rightarrow E_1 * E_2$	{E.*place*= *newtemp* ; E.*code* = E_1.*code* || E_2.*code* || || *gen*(E.*place*':='E_1.*place*'*'E_2.*place*) }
$E \rightarrow - E_1$	{E.*place*= *newtemp* ; E.*code* = E_1.*code* || || *gen*(E.*place* ':=' 'uminus' E_1.*place*) }
$E \rightarrow (E_1)$	{E.*place*= E_1.*place* ; E.*code* = E_1.*code*}
$E \rightarrow id$	{E.*place* = id.*place* ; E.*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



$t_1 := \text{uminus } c$
$t_2 := b * t_1$
$t_3 := t_2 + d$
$a := t_3$

Cài đặt câu lệnh 3 địa chỉ

Bộ bốn (Quadruples)

$t_1 := - c$
$t_2 := b * t_1$
$t_3 := - c$
$t_4 := b * t_3$
$t_5 := t_2 + t_4$
$a := t_5$

	op	*arg1*	*arg2*	*result*
(0)	uminus	c		t_1
(1)	*	b	t_1	t_2
(2)	uminus	c		t_3
(3)	*	b	t_3	t_4
(4)	+	t_2	t_4	t_5
(5)	:=	t_5		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)

$t_1 := - c$
$t_2 := b * t_1$
$t_3 := - c$
$t_4 := b * t_3$
$t_5 := t_2 + t_4$
$a := t_5$

	op	*arg1*	*arg2*
(0)	uminus	c	
(1)	*	b	(0)
(2)	uminus	c	
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	assign	a	(4)

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)

Sinh mã cho khai báo

Sử dụng biến toàn cục *offset*.

Các tên cục bộ trong chương trình con được truy xuất thông qua địa chỉ tương đối offset.

Sản xuất	**Quy tắc ngữ nghĩa**
P → M D	**{ }**
M → ε	**{*offset*:=0 }**
D → D; D	
D → id : T	**{ *enter*(id.*name*, T.*type*, *offset*)** **offset*:=offset* + T.*width* }**
T → integer	**{T.*type* = *integer*; T.*width* = 4 }**
T → real	**{T.*type* =*real*; T.*width* = *8* }**
T → array [num] of T_1	**{T.*type*=*array*(1..num.*val*,T_1.*type*)** **T.*width* = num.val * T_1.*width*}**

Lưu trữ thông tin về phạm vi

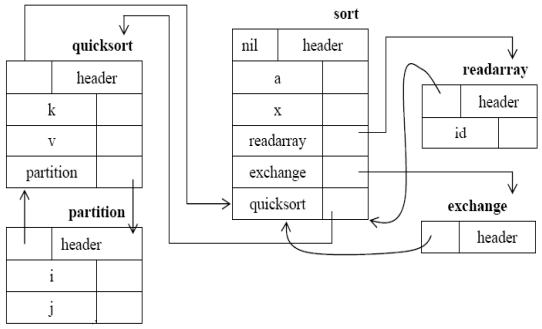
- Trong một ngôn ngữ mà chương trình con được phép khai báo lồng nhau, mỗi khi tìm thấy một CTC thì quá trình khai báo của chương trình bao nó bị tạm dừng.
- Văn phạm của khai báo này:
  P → D
  D → D; D | id : T | proc id ; D ; S
- Khi một khai báo chương trình con D → proc id D1; S được tạo ra thì các khai báo trong D1 được lưu trong bảng kí hiệu mới.

Khai báo chương trình con lồng nhau

- Ví dụ chương trình:

```
1)  Program sort;
2)  Var  a: array[0..10] of integer;
3)       x: integer;
4)    Procedure readarray;
5)       Var i: integer;
6)       Begin …a… end {readarray};
7)    Procedure exchange(i, j: integer);
8)       Begin {exchange} end;
9)    Procedure quicksort(m, n: integer);
10)      Var k, v: integer;
11)   Function partition(y,z: integer): integer;
12)      Begin ..a..v..exchange(i,j) end; {partition}
13)   Begin … end; {quicksort}
14) Begin … end; {sort}
```

Năm bảng kí hiệu của Sort



Các thủ tục trong tập quy tắc ngữ nghĩa

mktable(previous) – tạo một bảng kí hiệu mới, bảng này có previous chỉ đến bảng cha của bảng kí hiệu mới này.

enter(table,name,type,offset) – tạo ra một ô mới có tên *name* trong bảng kí hiệu được chỉ ra bởi *table* và đặt kiểu *type*, địa chỉ tương đối *offset* vào các trường bên trong ô đó.

enterproc(table,name,newbtable) – tạo ra một ô mới cho tên chương trình con vào table, newtable trỏ tới bảng kí hiệu của chương trình con này.

addwidth(table,width) – ghi tổng kích thước của tất cả các ô trong bảng kí hiệu vào header của bảng đó.

Xử lý các khai báo trong những chương trình con lồng nhau

P → M D { ***addwidth***(***top***(tblptr), ***top***(offset)); ***pop***(tblptr); ***pop***(offset) }

M → ε { t:=***mktable***(null); ***push***(t, tblptr); ***push***(0, offset)}

D → D_1 ; D_2

D → proc id ; N D_1 ; S { t:=***top***(tblpr); ***addwidth***(t,***top***(offset)); ***pop***(tblptr); ***pop***(offset); ***enterproc***(***top***(tblptr), id.name, t)}

N → ε {t:=***mktable***(***top***(tblptr)); ***push***(t,tblptr); ***push***(0,offset);}

D → id : T {***enter***(***top***(tblptr), id.name, T.***type***, ***top***(offset); ***top***(offset):=***top***(offset) + T.***width***

- **tblptr** – để giữ con trỏ bảng kí hiệu.
- **offset** – lưu trữ địa chỉ offset hiện tại của bảng kí hiệu trong **tblptr**.

Xử lý các khai báo trong những chương trình con lồng nhau

- Với sản xuất A → BC {action$_A$} thì các hoạt động trong cây con B, C được thực hiện trước A.
- Sản xuất **M → ε** khởi tạo stack ***tblptr*** với một bảng kí hiệu cho phạm vi ngoài cùng (chương trình sort) bằng lệnh ***mktable(nil)*** đồng thời đặt offset = 0.
- N đóng vai trò tương tự M khi một khai báo chương trình con xuất hiện, nó dùng lệnh ***mktable(top(tblptr))*** để tạo ra một bảng mới, tham số ***top*(tblptr)** cho giá trị con trỏ tới bảng lại được đẩy vào đỉnh stack tblptr và 0 được đẩy vào stack offset.
- Với mỗi khai báo **id: T** một ô mới được tạo ra cho id trong bảng kí hiệu hiện hành, stack tblptr không đổi, giá trị top(offset) được tăng lên bởi T.width.
- Khi **D → proc id ; N D_1 ; S** diễn ra thì kích thước của tất cả các đối tượng dữ liệu khai báo trong D_1 sẽ nằm trên đỉnh stack offset. Nó được lưu trữ bằng cách dùng Addwidth, các stack tblptr và offset bị đẩy và chúng ta thao tác trên các khai báo của chương trình con.

Tên trong bảng kí hiệu

- Xét ĐNTCP để sinh ra mã lệnh 3 địa chỉ cho lệnh gán

S→ id := E	*{p:=lookup(id.name);*
	if p <> nil then emit(p ':=' E.place) else error }
E→ E_1 + E_2	*{ E.place := newtemp;*
	emit(E.place ':=' E_1.place '+' E_2.place) }
E→ E_1 * E_2	*{ E.place := newtemp;*
	emit(E.place ':=' E_1.place '' E_2.place) }*
E→ - E_1	*{ E.place := newtemp;*
	emit(E.place ':=' 'unimus' E_1.place) }
E→ (E_1)	*{ E.place:=E_1.place) }*
E→ 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

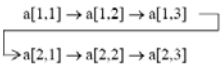
- Xét sản xuất D → proc id; ND_1; S
- Các tên trong lệnh gán sinh ra bởi kí hiệu không kết thúc S sẽ được khai báo trong chương trình con này hoặc trong chương trình chứa nó.
- Khi khai báo tới một tên thì trước hết hàm lookup sẽ tìm xem tên đó có trong bảng kí hiệu hiện hành hay không, nếu không thì 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
- ⇒ A[i] = i*w + c

Địa chỉ hóa các phần tử của mảng 2 chiều

- Theo dòng

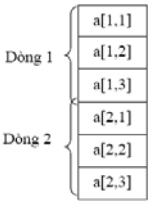


Địa chỉ tương đối của $A[i_1,i_2]$ =

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

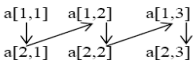
low_1, low_2: cận dưới cho i1 và i2

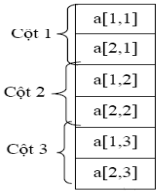
n2: số lượng các giá trị mà i2 có thể nhận. Nếu high2 là cận trên của i2 thì n2 = high2 – low2 + 1



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:
  E→ E or E | E and E | not E | (E) | 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

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:
```

ĐNTCPdùng số để biểu diễn các giá trị logic

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

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

$E \rightarrow$ not E_1 { *E.place:= newtemp; emit(E.place ':=' 'not'*

$E \rightarrow id_1$ relop id_2 { *E.place:= newtemp;*

('if' id_1.place relop.op id_2.place 'goto' nextstat +3);

(E.place ':=' '0'); emit('goto' nextstat +2);

(E.place ':=' '1') }

$E \rightarrow$ true { *E.place:= newtemp; emit(E.place ':=' '1')* }

$E \rightarrow$ false { *E.place:= newtemp; emit(E.place ':=' '0')* }

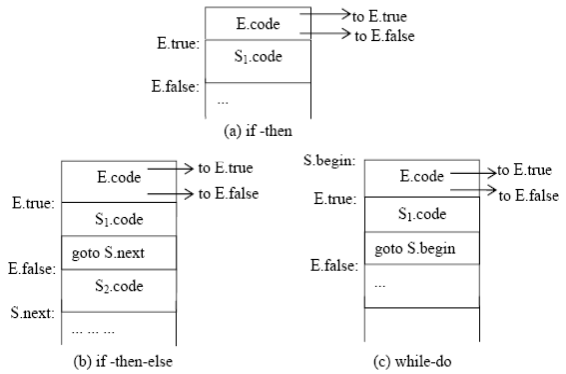
Nextstat cho biết chỉ số của câu lệnh 3 địa chỉ tiếp theo

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 E then S_1 | iF E then S_1 else S_2 | while E do S_1
- Với mỗi biểu thức E chúng ta kết hợp với 2 nhãn:
  - E.true: nhãn của dòng điều khiển nếu E là true
  - E.false: nhãn của dòng điều khiển nếu E 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

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



ĐNTCP cho các cấu trúc lập trình

SẢN XUẤT	QUY TẮC NGỮ NGHĨA
S→ if E then S_1	*E.true := newlabel;* *E.false := S.next;* *S_1.next := S.next;* *S.code := E.code || gen(E.true ':') || S_1.code*
S→ 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('goto' S.next) || gen(E.false ':') || S_2.code*
S→ while E do S_1	*S.begin := newlabel;* *E.true := newlabel;* *E.fasle := S.next;* *S_1.next := S.begin;* *S.code:= gen(S.begin':') || E.code || gen(E.true ':') || S_1.code || gen('goto' S.begin)*

Dịch biểu thức logic

Dịch biểu thức logic trong các cấu trúc lập trình

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

  If a<b then goto E.true else goto E.false
- Nếu E có dạng: E1 or E2 thì
  - Nếu E1 là true thì E cũng là true
  - Nếu E1là false thì phải đánh giá E2; E sẽ làtrue hay false phụ thuộc E2
- Tương tự với E1 and E2

Dịch biểu thức logic trong các cấu trúc lập trình

SẢN XUẤT	QUY TẮC NGỮ NGHĨA
E→ 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.false ':') || E_2.code*
E→ 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.true ':') || E_2.code*
E→ not E_1	*E_1.true := E.false;* *E_1.false := E.true;* *E.code := E_1.code*
E → (E_1)	*E_1.true := E.true;* *E_1.false := E.false;* *E.code := E_1.code*
E → id_1 relop id_2	*E.code := gen('if' id_1.place relop.op id_2.place 'goto' E.true) || gen('goto' E.false)*
E → true	*E.code:= gen('goto' E.true)*
E → 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

Xét văn phạm: $E \rightarrow E + E \mid E \text{ and } E \mid E \text{ relop } E \mid id$

Trong đó, E and E đòi hỏi hai đối số phải là logic. Trong khi + và relop có các đối số là biểu thức logic hoặc/và số học.

Để sinh mã lệnh trong trường hợp này, chúng ta dùng thuộc tính tổng hợp E.type có thể là arith hoặc bool. E sẽ có các thuộc tính kế thừa E.true và E.false đối với biểu thức số học.

Biểu thức logic ở dạng hỗn hợp

QT ngữ nghĩa kết hợp với E → E1 + E2

E.type := arith;
if *E_1.type = arith* **and** *E_2.type = arith* **then begin**
/ phép cộng số học bình thường */*
E.place := newtemp;
E.code := E_1.code || E_2.code || gen(E.place ':=' E_1.place '+' E_2.place)
end
else if *E_1.type = arith* **and** *E_2.type = bool* **then begin**
E.place := newtemp;
E_2.true := newlabel;
E_2.false := newlabel;
E.code := E_1.code || E_2.code || gen(E_2.true ':' E.place ':= ' E_1.place +1) ||
gen('goto' nextstat +1) || gen(E_2.false ':' E.place ':= ' E_1.place)
else if ...

Biểu thức logic ở dạng hỗn hợp

nếu có biểu thức logic nào có biểu thức số học, sinh mã lệnh cho E1, E2 bởi

E_2.true : E.place := E_1.place +1
goto nextstat +1
E_2.false : E.place := E_1.place