اصول طراحی کامپایلر

درس ۲۱

تولید کد میانی

Intermediate Code Generation

کاظم فولادی قلعه
دانشکده مهندسی، پردیس فارابی
دانشگاه تهران

http://courses.fouladi.ir/compiler

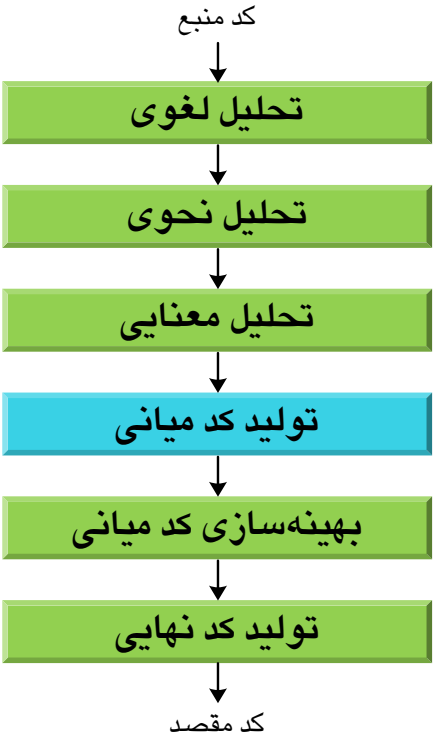
تولید کد میانی

۱

مقدمه

مراحل فرآیند کامپایل در یک نگاه

تولید کد میانی



تولید کد میانی

تولید کد میانی

مثال

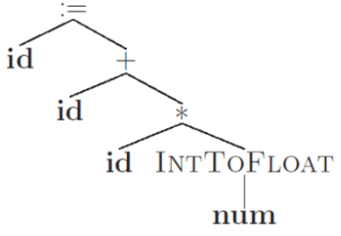
حاشیه‌نویسی‌شده
درخت ساختارها
annotated tree of structures

→

مولد کد میانی
Intermediate Code Generator

→

دنباله‌ی اجزای کد میانی
sequence of interm. code parts




```
temp1 := inttofloat(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

کد سه‌آدرسی

تولید کد میانی

۲

کد میانی

کد سه‌آدرسی

THREE-ADDRESS CODE

کد سه‌آدرسی: هر دستورالعمل حداکثر سه عملوند دارد.

op: + - * / uminus and or xor not	(*op*,x,y,z)	x := y *op* z	دودویی *Binary*	انتساب *Assignment*
	(*op*,x,y,)	x := *op* y	تکی *Unary*	
	(:=,x,y,)	x := y	ساده *Simple*	کپی *Copy*
	(copyi,x,y,i) (icopy,x,i,y)	x := y[i] x[i] := y	اندیس‌دار *Indexed*	
	(copya,x,y,) (copyc,x,y,) (acopy,x,y,)	x := &y x := *y *x := y	آدرس و اشاره‌گر *Address and Pointer*	
relop: < = > ≤ ≠ ≥	(jump,L, ,)	goto L	غیرشرطی *Binary*	پرش *Jump*
	(jumpif*relop*,x,y,L1) (jump,L2, ,)	if x *relop* y goto L1 [else goto L2]	شرطی *Unary*	
	(param,x1, ,) (param,x2, ,) ... (param,x*n*, ,) (call, p, *n*,)	param X1 param X2 ... param X*n* call p, *n*	$P(X1,X2,\ldots,Xn)$	فراخوانی روال *Procedure Call*

کد سه‌آدرسی

پیاده‌سازی با چهارتایی‌ها

Quadruples

	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

کد سه‌آدرسی

پیاده‌سازی با سه‌تایی‌ها

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

کد سه‌آدرسی

پیاده‌سازی با سه‌تایی‌های غیرمستقیم

Indirect triples

	statement
(0)	(14)
(1)	(15)
(2)	(16)
(3)	(17)
(4)	(18)
(5)	(19)

	op	arg1	arg2
(14)	uminus	c	
(15)	*	b	(14)
(16)	uminus	c	
(17)	*	b	(16)
(18)	+	(15)	(17)
(19)	assign	a	(18)

تولید کد میانی

۳

تولید کد میانی برای ساختارهای یک زبان برنامه‌سازی

کد برای دستور انتساب

ASSIGNMENT

تعریف S–خصیصه‌ای برای دستور انتساب

PRODUCTION	SEMANTIC RULES
$S \rightarrow \text{id} := E$	$S.code := E.code \parallel gen(\text{id}.place\ ':='\ E.place)$
$E \rightarrow E_1 + E_2$	$E.place := newtemp;$ $E.code := E_1.code \parallel E_2.code \parallel$ $gen(E.place\ ':='\ E_1.place\ '+'\ E_2.place)$
$E \rightarrow E_1 * E_2$	$E.place := newtemp;$ $E.code := E_1.code \parallel E_2.code \parallel$ $gen(E.place\ ':='\ E_1.place\ '*'\ E_2.place)$
$E \rightarrow -E_1$	$E.place := newtemp;$ $E.code := E_1.code \parallel gen(E.place\ ':='\ \text{'uminus'}\ E_1.place)$
$E \rightarrow (E_1)$	$E.place := E_1.place; E.code := E_1.code$
$E \rightarrow \text{id}$	$E.place := \text{id}.place; E.code := ''$

خصیصه‌های ترکیبی

$S.code$	حاوی کد سه‌آدرسی برای دستور S
$E.place$	حاوی شناسه‌ای برای متغیر موقتی با محتوای مقدار E
$E.code$	حاوی کد سه‌آدرسی برای ارزیابی مقدار E

تابع‌ها

$newtemp$	تولیدکننده‌ی متغیرهای موقتی با نام‌های متمایز $t_1, t_2, t_3, \ldots$ (متغیرهای موقتی هم وارد جدول نمادها می‌شوند)
gen	کد سه‌آدرسی را تولید می‌کند: ▪ هر چیزی که داخل " " قرار دارد، به عنوان لیترال رشته‌ای در نظر گرفته می‌شود. ▪ سایر اجزای آرگومان ارزیابی می‌شود.
$emit$	* در عمل می‌توان کد حاصل را به‌جای قرار دادن در خصیصه‌ی $code$، به خروجی فرستاد.
$\parallel$	عملگر الحاق رشته‌ها

کد برای دستور انتساب

ترجمه

ASSIGNMENT

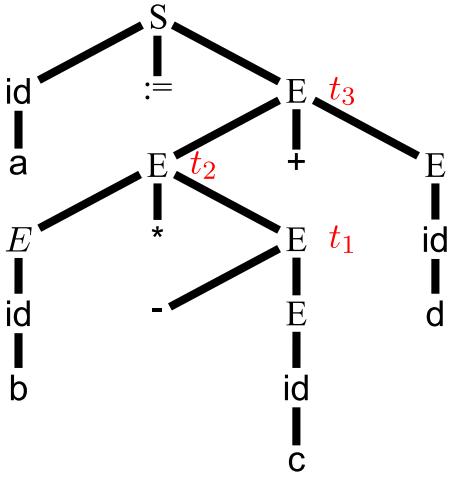
تعریف S–خصیصه‌ای برای دستور انتساب

PRODUCTION	SEMANTIC RULES
$S \rightarrow \mathsf{id} := E$	$S.code := E.code \parallel gen(\mathsf{id}.place\ ':='\ E.place)$
$E \rightarrow E_1 + E_2$	$E.place := newtemp;$ $E.code := E_1.code \parallel E_2.code \parallel$ $gen(E.place\ ':='\ E_1.place\ '+'\ E_2.place)$
$E \rightarrow E_1 * E_2$	$E.place := newtemp;$ $E.code := E_1.code \parallel E_2.code \parallel$ $gen(E.place\ ':='\ E_1.place\ '*'\ E_2.place)$
$E \rightarrow -E_1$	$E.place := newtemp;$ $E.code := E_1.code \parallel gen(E.place\ ':='\ '\mathsf{uminus}'\ E_1.place)$
$E \rightarrow (E_1)$	$E.place := E_1.place; E.code := E_1.code$
$E \rightarrow \mathsf{id}$	$E.place := \mathsf{id}.place; E.code := ''$

کد برای دستور انتساب

مثال

Given the assignment a := b ∗ −c + d the code generated by the above grammar is:



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

کد برای دستور انتساب

انتساب و جدول نمادها

هر شناسه، درواقع اشاره‌گری است به مدخل آن نام در جدول نمادها
(سایر اطلاعات آن نام مانند آدرس متغیر و ... نیز برای تولید کد نهایی لازم است)

تابع *lookup* برای دسترسی به جدول نمادها

lookup(id.*name*)

مقدار *nil* : اگر مدخل مربوط به **id** در جدول نمادها پیدا نشود.

اشاره‌گر به مدخل id : اگر مدخل مربوط به **id** در جدول نمادها پیدا شود.

این تابع به‌گونه‌ای پیاده‌سازی می‌شود که حوزه‌ی دید متغیرها را در نظر بگیرد:
اگر شناسه در جدول نماد فعلی ظاهر نشده بود، جدول نماد احاطه‌کننده‌ی آن بررسی می‌شود.

emit

کد حاصل را به‌جای قرار دادن در خصیصه‌ی ***code***، به خروجی می‌فرستیم.

شرط لازم: کد ناپایانه‌ی سمت چپ باید از الحاق کد ناپایانه‌های سمت راست با همان ترتیب به دست آید (با وجود تعدادی رشته‌ی اضافی بین آنها).

کد برای دستور انتساب

انتساب و جدول نمادها: ترجمه

PRODUCTION	SEMANTIC RULES
$S \rightarrow \mathsf{id} := E$	$p := lookup(\mathsf{id}.name)$; if $p \neq nil$ then $emit(p\,' :='\ E.place)$ else $error$
$E \rightarrow E_1 + E_2$	$E.place := newtemp$; $emit(E.place\,' :='\ E_1.place\,' +'\ E_2.place)$
$E \rightarrow E_1 * E_2$	$E.place := newtemp$; $emit(E.place\,' :='\ E_1.place\,' *'\ E_2.place)$
$E \rightarrow -E_1$	$E.place := newtemp$; $emit(E.place\,' :='\ '\mathsf{uminus}'\ E_1.place)$
$E \rightarrow (E_1)$	$E.place := E_1.place$
$E \rightarrow \mathsf{id}$	$p := lookup(\mathsf{id}.name)$; if $p \neq nil$ then $E.place := p$ else $error$

کد برای دستور انتساب

عبارت‌های بولی

کاربرد عبارت‌های بولی

- مقادیر منطقی
- عبارت‌های شرطی

گرامر عبارت‌های بولی

$$E \rightarrow E \text{ or } E \mid E \text{ and } E \mid \text{not } E \mid (E) \mid \text{id relop id} \mid \text{true} \mid \text{false}$$

روش‌های ارزیابی عبارت‌های بولی

بازنمایی عددی *Numerical Representation*	کدپرشی ـ مدار کوتاه *Jump Code / Short Circuit*

بازنمایی
`true` با عدد `1`
`false` با عدد `0`
و برخورد با آنها مشابه عبارات حسابی
	محاسبه‌ی مقدار عبارت بولی
	با حرکت در طول برنامه

ارزیابی عبارت‌های بولی از سمت چپ به راست

`and` و `or` شرکت‌پذیر از چپ

ترتیب اولویت‌ها:
`or` (پایین‌ترین)
`and`
`not` (بالاترین)

کد برای دستور انتساب

عبارت‌های بولی: ارزیابی: مثال ۱

ترجمه برای عبارت بولی

```
a or (b and (not c))
```

$$t_1 := \text{not c}$$

$$t_2 := \text{b and } t_1$$

$$t_3 := \text{a or } t_2$$

کد برای دستور انتساب

عبارت‌های بولی: ارزیابی: مثال ۲

ترجمه برای عبارت رابطه‌ای

```
a < b
```

که معادل است با دستور شرطی

```
if  a < b  then  1  else  0
```

(ترجمه‌ی آن شامل پرش به دستورات برچسب‌دار می‌شود):

$$
\begin{aligned}
100: &\quad \text{if a} < \text{b goto } 103\\
101: &\quad t := 0\\
102: &\quad \text{goto } 104\\
103: &\quad t := 1\\
104: &
\end{aligned}
$$

کد برای دستور انتساب

عبارت‌های بولی: ترجمه (روش بازنمایی عددی)

PRODUCTION	SEMANTIC RULES
$E \rightarrow E_1$ or E_2	$E.place := newtemp;$ $emit(E.place\ ':='\ E_1.place\ 'or'\ E_2.place)$
$E \rightarrow E_1$ and E_2	$E.place := newtemp;$ $emit(E.place\ ':='\ E_1.place\ 'and'\ E_2.place)$
$E \rightarrow$ not E_1	$E.place := newtemp;$ $emit(E.place\ ':='\ 'not'\ E_1.place)$
$E \rightarrow (E_1)$	$E.place := E_1.place$
$E \rightarrow \mathrm{id}_1$ relop id_2	$E.place := newtemp;$ $emit('if'\ \mathrm{id}_1.place\ \text{relop}.op\ \mathrm{id}_2.place\ 'goto'\ nextstat + 3);$ $emit(E.place\ ':='\ '0');$ $emit('goto'\ nextstat + 2);$ $emit(E.place\ ':='\ '1')$
$E \rightarrow$ true	$E.place := newtemp;\ emit(E.place\ ':='\ '1')$
$E \rightarrow$ false	$E.place := newtemp;\ emit(E.place\ ':='\ '0')$

nextstat	حاوی اندیس دستورالعمل بعدی در کد سه‌آدرسی (مقدار آن توسط ***emit*** افزایش می‌یابد)
relop.*op*	خصیصه‌ی ترکیبی برای تعیین عملگر رابطه‌ای مورد استفاده در **relop**

کد برای دستور انتساب

عبارت‌های بولی: ترجمه (روش بازنمایی عددی): مثال

$E \rightarrow \mathsf{id}_1$ relop id_2	$E.place := newtemp;$ $emit(\text{'if'}\ \mathsf{id}_1.place\ \text{relop}.op\ \mathsf{id}_2.place\ \text{'goto'}\ nextstat + 3);$ $emit(E.place\ ' :=' \ '0');$ $emit(\text{'goto'}\ nextstat + 2);$ $emit(E.place\ ' :=' \ '1')$

ترجمه برای عبارت

`a < b or c < d and e < f`

```
100: if a < b goto 103
101: t1 := 0
102: goto 104
103: t1 := 1 /* true */
104: if c < d goto 107
105: t2 := 0 /* false */
106: goto 108
```

```
107: t2 := 1
108: if e < f goto 111
109: t3 := 0
110: goto 112
111: t3 := 1
112: t4 := t2 and t3
113: t3 := t1 or t4
```

کد برای دستور انتساب

عبارت‌های بولی: روش کد پرشی

تعریف L–خصیصه‌ای برای ارزیابی عبارت بولی

عبارت بولی به دنباله‌ای از پرش‌های شرطی و غیرشرطی
به برچسب‌های $E.true$ و $E.false$ ترجمه می‌شوند.

خصیصه‌های موروثی	
$E.true$	حاوی برچسب مقصد پرش در صورت درستی E
$E.false$	حاوی برچسب مقصد پرش در صورت نادرستی E

`a < b`	`if a < b goto E.true` `else goto E.flase`
E_1 or E_2	اگر E_1 درست باشد، آن‌گاه قرار می‌دهیم E_1.true = E.true وگرنه E_2 باید ارزیابی شود، در این صورت E_1.false = برچسب اولین دستور در کد E_2
E_1 and E_2	اگر E_1 نادرست باشد، آن‌گاه قرار می‌دهیم E_1.false = E.false وگرنه E_2 باید ارزیابی شود، در این صورت E_1.true = برچسب اولین دستور در کد E_2
not E_1	فقط باید برچسب‌های true و false برای E را تعویض کنیم.

کد برای دستور انتساب

عبارت‌های بولی: ترجمه (روش کد پرشی)

PRODUCTION	SEMANTIC RULES
$E \rightarrow E_1$ or E_2	$E_1.true := E.true;\ \ E_1.false := newlabel;$ $E_2.true := E.true;\ \ E_2.false := E.false;$ $E.code := E_1.code \parallel gen(E_1.false\ ':') \parallel E_2.code$
$E \rightarrow E_1$ and E_2	$E_1.true := newlabel;\ \ E_1.false := E.false;$ $E_2.true := E.true;\ \ E_2.false := E.false;$ $E.code := E_1.code \parallel gen(E_1.true\ ':') \parallel E_2.code$
$E \rightarrow$ not E_1	$E_1.true := E.false;\ \ E_1.false := E.true;$ $E.code := E_1.code$
$E \rightarrow (E_1)$	$E_1.true := E.true;\ \ E_1.false := E.false;$ $E.code := E_1.code$
$E \rightarrow$ id$_1$ relop id$_2$	$gen('\text{if}'\ \text{id}_1.place\ \text{relop}.op\ \text{id}_2.place\ '\text{goto}'\ E.true) \parallel$ $gen('\text{goto}'\ E.false)$
$E \rightarrow$ true	$E.code := gen('\text{goto}'\ E.true)$
$E \rightarrow$ false	$E.code := gen('\text{goto}'\ E.false)$

کد برای دستور انتساب

عبارت‌های بولی: ترجمه (روش کد پرشی) [گرامر کامل‌تر]

Production	Semantic Rules
$E \rightarrow E_1$ **or** E_2	$E_1.true := E.true; \quad E_1.false := newlabel;$ $E_2.true := E.true; \quad E_2.false := E.false;$ $E.code := E_1.code \| \| gen(E_1.false' :') \| \| E_2.code;$
$E \rightarrow E_1$ **and** E_2	$E_1.true := newlabel; \quad E_1.false := E.false;$ $E_2.true := E.true; \quad E_2.false := E.false;$ $E.code := E_1.code \| \| gen(E_1.true' :') \| \| E_2.code;$
$E \rightarrow$ **not** E_1	$E_1.true := E.false; \quad E_1.false := E.true; \quad E.code := E_1.code;$
$E \rightarrow E_1$ **relop** E_2	$E.code := E_1.code \| \| E_2.code$ $\| \| gen('\texttt{if}'\ E_1.place\ \textbf{relop}.op\ E_2.place\ '\texttt{goto}'\ E.true) \| \| gen('\texttt{goto}'\ E.false)$
$E \rightarrow$ **true**	$E.code := gen('\texttt{goto}'\ E.true)$
$E \rightarrow$ **false**	$E.code := gen('\texttt{goto}'\ E.false)$

کد برای دستورهای کنترل جریان برنامه

هر دستورالعمل کد سه‌آدرسی می‌تواند دارای برچسب نمادین باشد

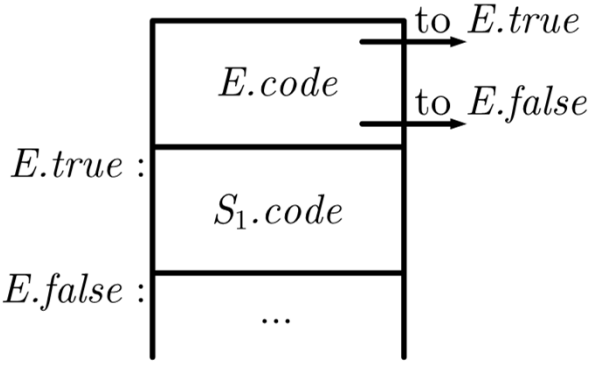
تابع‌ها	
newlabel	تولیدکننده‌ی برچسب برای هر دستور کد سه‌آدرسی

خصیصه‌های موروثی		
دو برچسب برای E	***E.true***	حاوی برچسب مقصد پرش در صورت درستی E
	E.false	حاوی برچسب مقصد پرش در صورت نادرستی E
	S.next	حاوی برچسب متصل شده به نخستین دستور پس از کد S

* این روش تعداد بسیار زیادی برچسب تولید می‌کند. روش backpatching فقط در صورت نیاز برچسب تولید می‌کند.

کد برای دستورهای کنترل جریان برنامه

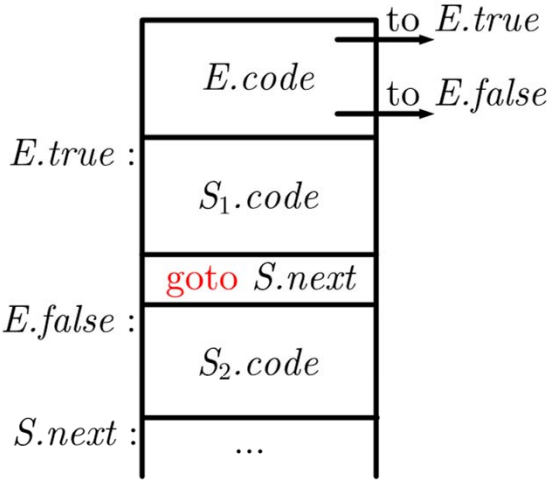
ترجمه‌ی دستور شرطی if-then



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

کد برای دستورهای کنترل جریان برنامه

ترجمه‌ی دستور شرطی if-then-else

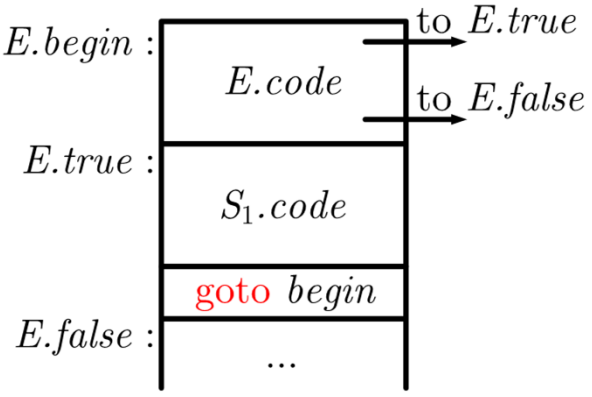


$S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$

$$
\begin{aligned}
&E.true := newlabel; \quad E.false := newlabel; \\
&S_1.next := S.next; \quad S_2.next := S.next; \\
&S.code := E.code \parallel gen(E.true\,' :') \parallel S_1.code \parallel \\
&\qquad\qquad gen('\text{goto } S.next) \parallel \\
&\qquad\qquad gen(E.false\,' :') \parallel S_2.code
\end{aligned}
$$

کد برای دستورهای کنترل جریان برنامه

ترجمه‌ی دستور تکرار while-do



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

$E.begin := newlabel; \|$
$E.true := newlabel; \quad E.false := S.next; \|$
$S_1.next := E.begin; \|$
$S.code := gen(E.begin\ ':') \| E.code \|$
$gen(E.true\ ':') \| S_1.code \|$
$gen('\text{goto}'\ E.begin)$

کد برای دستورهای کنترل جریان برنامه

ترجمه‌ی دستورهای کنترل جریان

PRODUCTION	SEMANTIC RULES
$S \rightarrow$ if E then S_1	$E.true := newlabel;\ \ E.false := S.next;$ $S_1.next := S.next;$ $S.code := E.code \parallel gen(E.true\ ':') \parallel S_1.code$
$S \rightarrow$ if E then S_1 else S_2	$E.true := newlabel;\ \ E.false := newlabel;$ $S_1.next := S.next;\ \ S_2.next := S.next;$ $S.code := E.code \parallel gen(E.true\ ':') \parallel S_1.code \parallel$ $gen('\text{goto}'\ S.next) \parallel$ $gen(E.false\ ':') \parallel S_2.code$
$S \rightarrow$ while E doS_1	$E.begin := newlabel; \parallel$ $E.true := newlabel;\ \ E.false := S.next; \parallel$ $S_1.next := E.begin; \parallel$ $S.code := gen(E.begin\ ':') \parallel E.code \parallel$ $gen(E.true\ ':') \parallel S_1.code \parallel$ $gen('\text{goto}'\ E.begin)$

کد برای دستورهای کنترل جریان برنامه

دستور switch-case

```
switch expr
{
    case V1 : S1
    case V2 : S2
    ...
    case Vk : Sk
    default : Sd
}
```

دنباله‌ی ترجمه

- ارزیابی عبارت $expr$
- یافتن مقدار مطابقت‌یافته با مقدار $expr$ (V_i)
  - اگر هیچ مطابقتی وجود نداشت، مطابقت با *default* انجام می‌شود.
- اجرای دستور متناظر با مقدار مطابقت‌یافته (S_i)

کد برای دستورهای کنترل جریان برنامه

دستور switch-case: پیاده‌سازی

```
switch expr
{
    case V1 : S1
    case V2 : S2
    ...
    case Vk : Sk
    default : Sd
}
```

پیاده‌سازی ۱

```
    code to evaluate E into t
    goto test
L1: code for S1
    goto next
    ...
Lk: code for Sk
    goto next
Ld: code for Sd
    goto next
test:
    if t = V1 goto L1
    ...
    if t = Vk goto Lk
    goto Ld
next:
    ...
```

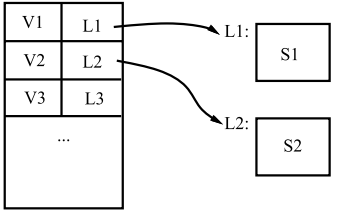
پیاده‌سازی ۲

```
    code to evaluate E into t
    if t <> V1 goto L1
    code for S1
    goto next
L1: if t <> V2 goto L2
    code for S2
    goto next
    ...
L(k-1): if t <> Vk goto Lk
    code for Sk
    goto next
Lk: code for Sd
next:
```

کد برای دستورهای کنترل جریان برنامه

دستور switch-case: پیاده‌سازی: روش‌های مطابقت و یافتن آدرس پرش

روش‌های مطابقت و یافتن آدرس پرش			
آزمون ترتیبی گزینه‌ها *Sequential Test*	**جدول مراجعه** *Lookup Table*	**جدول درهم‌سازی** *Hash Table*	**بک‌پچینگ** *Bachpatching*
بررسی نوبتی تک‌تک گزینه‌ها در کد	استفاده از یک جدول مراجعه برای یافتن آدرس پرش به کمک یک حلقه	وقتی تعداد مدخل‌ها بیش از ۱۰ مورد است، باید از یک جدول درهم‌سازی برای یافتن مدخل مناسب استفاده کرد.	یک سری از دستورات پرش که مقصد پرش آنها موقتاً نامشخص باقی گذاشته می‌شود، ایجاد می‌شود. در جدول برچسب‌ها: هر مدخل حاوی لیستی از مکان‌ها است که باید بعداً پر شوند (backpatch شوند). * همین روش قابل استفاده برای پیاده‌سازی دستور GOTO



تولید کد برای عبارت‌های بولی در حالت مخلوط

MIXED-MODE

عبارت‌های بولی در حالت مخلوط	

قرار دادن
`true` مساوی با عدد `1`
`false` مساوی با عدد `0`
و برخورد با حاصل به‌صورت حسابی
	عبارت‌های بولی
	حاوی زیرعبارت‌های محاسباتی

`(a < b) + (b < a)`	`(a + b) < c`

گرامر عبارت‌های بولی مخلوط

$$E \rightarrow E + E \mid E \text{ and } E \mid E \text{ relop } E \mid \text{id}$$

حالت‌ها			
$E + E$	E and E	E relop E	id
حاصل محاسباتی آرگومان‌ها مخلوط	حاصل بولی آرگومان‌ها بولی	حاصل بولی آرگومان‌ها مخلوط	محاسباتی

تولید کد برای عبارت‌های بولی در حالت مخلوط

خصیصه‌ها و متغیرهای لازم

MIXED-MODE

خصیصه‌های موروثی	
$E.true$	حاوی برچسب مقصد پرش در صورت درستی E
$E.false$	حاوی برچسب مقصد پرش در صورت نادرستی E
$S.next$	حاوی برچسب متصل شده به نخستین دستور پس از کد S

خصیصه‌های ترکیبی	
$E.place$	حاوی شناسه‌ای برای متغیر موقتی با محتوای مقدار E
$E.type$	حاوی نوع متغیر موقتی با محتوای مقدار E (با مقادیر $arith$ یا $bool$)

$nextstat$	حاوی اندیس دستورالعمل بعدی در کد سه‌آدرسی (مقدار آن توسط $emit$ افزایش می‌یابد)

تولید کد برای عبارت‌های بولی در حالت مخلوط

ترجمه

MIXED-MODE

$E \rightarrow E_1 + E_2$ | $E.type := arith;$
if $E_1.type := arith$ and $E_2.type := arith$ then begin
$E.place := newtemp;$
$E.code := E_1.code \parallel E_2.code \parallel$
$gen(E.place\,' :=' \; E_1.place\,' +' \; E_2.place)$
end
esle if $E_1.type := arith$ and $E_2.type := bool$ then begin
$E.place := newtemp;$
$E_2.true := newlabel; E_2.false := newlabel;$
$E.code := E_1.code \parallel E_2.code \parallel$
$gen(E_2.true' :' \; E.place\,' :=' \; E_1.place + 1) \parallel$
$gen('\text{goto}' nextstat + 1) \parallel$
$gen(E_2.false' :' \; E.place\,' :=' \; E_1.place)$
end
else if ...

تولید کد برای آرایه‌ها

آرایه‌ی یک‌بعدی

1-D Array

آرایه‌ی یک‌بعدی $A[i]$	
low	کران پایین در اندیس
w	عرض عنصر داده‌ای
$base$	آدرس شروع

آدرس برای عنصر $A[i]$

$$= base + (i - low) * w$$

$$= i * w + \underbrace{(base - low * w)}$$

قابل محاسبه در زمان کامپایل

تولید کد برای آرایه‌ها

آرایه‌ی دوبعدی

گسترش سطری *Row Major*	گسترش ستونی *Column Major*
قرار دادن آرایه به صورت سطربه‌سطر در حافظه	قرار دادن آرایه به صورت ستون‌به‌ستون در حافظه
▷ $A[1,1], A[1,2], A[1,3], A[2,1], A[2,2], \ldots$ ▷ ***Advantage: A[i,j] = A[i][j].*** ▷ ***A[1] means the first row.***	▷ $A[1,1], A[2,1], A[1,2], A[2,2], A[1,3], \ldots$

تولید کد برای آرایه‌ها

آرایه‌ی دوبعدی

2-D Array

آرایه‌ی دوبعدی $A[i_1,i_2]$	
low_i	کران پایین در اندیس بعد i
n_i	تعداد عناصر در بعد i
w	عرض عنصر داده‌ای
$base$	آدرس شروع

آدرس برای عنصر $A[i_1,i_2]$

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

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

قابل محاسبه در زمان کامپایل

تولید کد برای آرایه‌ها

آرایه‌ی چندبعدی

MULTI-DIMENTIONAL ARRAY

آرایه‌ی k–بعدی $A[i_1,i_2,\dots,i_k]$	
low_i	کران پایین در اندیس بعد i
n_i	تعداد عناصر در بعد i
w	عرض عنصر داده‌ای
$base$	آدرس شروع

آدرس برای عنصر $A[i_1,i_2,\dots,i_k]$

$$= (i_1 * \Pi_{i=2}^{k} n_i i_2 * \underbrace{\Pi_{i=3}^{k} n_i} + \cdots + i_k) * w + \underbrace{(base - low_1 * \Pi_{i=2}^{k} n_i * w - \cdots - low_k * w)}$$

قابل محاسبه در زمان کامپایل — قابل محاسبه در زمان کامپایل

تولید کد برای آرایه‌ها

ترجمه

$L \rightarrow Elist\,]$	$L.place := newtemp;$ $L.offset := newtemp;$ **emit(**$L.offset$, “:=”, $Elist.w$, “*”, $Elist.place$**);**
$L \rightarrow \mathbf{id}$	$L.place := \mathbf{id}.place;$ $L.offset := nil;$
$Elist \rightarrow Elist_1, E$	$t := newtemp;$ $m := Elist_1.ndim + 1;$ **emit(**t, “:=”, $Elist_1.place$, “*”, ***limit***$(Elist_1.array,m)$**);** **emit(**t, “:=”, t, “+”, $E.place$**);** $Elist.array := Elist_1.array;$ $Elist.place := t;$ $Elist.ndim := m;$
$Elist \rightarrow \mathbf{id}\,[\,E$	$Elist.place := E.place;$ $Elist.ndim := 1;$ $p := \textbf{\textit{lookup}}(\mathbf{id}.name)$ $Elist.w := p.size;$ $Elist.array := p.place$

تولید کد برای روال‌ها

```
call P(X1, X2, …, Xn)
```

دنباله‌ی ترجمه	
فراخوانی	○ اختصاص فضا برای رکورد فعالیت (تخصیص **ایستا** یا **پویا** (در پشته))
	○ ارزیابی آرگومان‌ها (قرار دادن در فضای پارامترهای رکورد فعالیت روال فراخوانی‌شده)
	○ ذخیره‌سازی وضعیت فعلی ماشین (قرار دادن در محل مشخص در رکورد فعالیت روال)
برگشت	○ قراردادن مقدار بازگشتی (قرار دادن در محل مشخص در رکورد فعالیت روال)
	○ بازیابی رکورد فعالیت

تولید کد برای روال‌ها

ترجمه

$S \rightarrow \textbf{call}\ \ \textbf{id}(Elist)$	**for each** item p on the queue $Elist.queue$ **do**
	emit("param", p);
	emit("call", **id**.$place$, $Elist.queue.size$);

$Elist \rightarrow Elist_1, E$	append $E.place$ to the end of $Elist.queue$
$Elist \rightarrow E$	initialize $Elist.queue$ to contain only $E.place$

می‌توان از یک صف برای نگهداری پارامترها استفاده کرد،
آن‌گاه کدها را برای پارامترها تولید نمود و سپس کد فراخوانی روال:

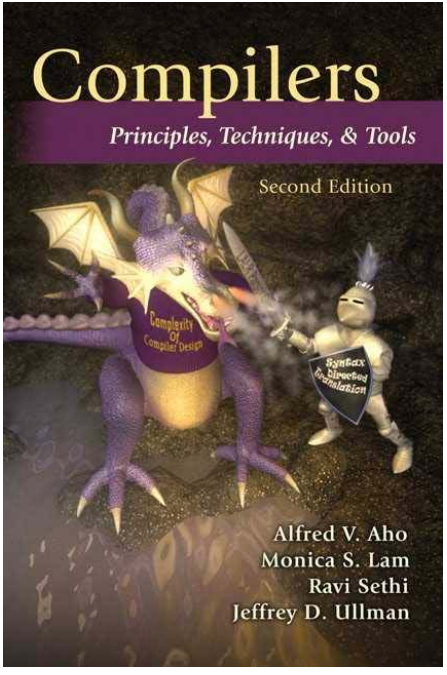
```
code for E_1, store in t_1
code for E_2, store in t_2
...
code for E_k, store in t_k
param t_1
param t_2
...
param t_k
call p, k
```

تولید کد میانی

۴

منابع

منبع اصلی



A. V. Aho, M. S. Lam, R. Sethi, J. D. Ullman,
Compilers: Principles, Techniques and Tools,
Second Edition, Addison-Wesley, 2007.

Chapter 6 (6.4, 6.6, 6.7, 6.8, 6.9)