Тема 7. Адресация элементов массива

7.1. Общие сведения

Быстрый доступ к элементам массива легко осуществляется, когда элементы хранятся в блоке из последовательных ячеек памяти. Размер каждого элемента массива обозначим через *d*, тогда *i*-й элемент массива *A* начинается с адреса

$$base + (i - low) \times d, \qquad (1)$$

где *low* – нижняя граница индекса, а *base* – относительный адрес памяти, выделенной под массив, т. е. относительный адрес $A[low]$. Выражение (1) может быть частично вычислено во время компиляции, если переписать его как

$$i \times d + (base - low \times d).$$

Подвыражение $c = base - low \times d$ можно вычислить в момент обработки объявления массива. Полагая, что величина *c* хранится в записи таблицы символов для *A*, относительный адрес $A[i]$ мы получим просто прибавлением $i \times d$ к *c*.

Предвычисления в процессе компиляции могут быть применены и для вычисления адресов элементов многомерных массивов. Двумерный массив обычно хранится либо по строкам, либо по столбцам. В случае двумерного массива, хранимого по строкам, относительный адрес $A[i_1, i_2]$ может быть вычислен по формуле

$$base + ((i_1 - low_1) \times n_2 + i_2 - low_2) \times d,$$

где low_1 и low_2 – нижние границы значений i_1 и i_2, а n_2 – число значений, которые может принимать i_2. Таким образом, если $high_2$ – верхняя граница значения i_2, то $n_2 = high_2 - low_2 + 1$. Полагая, что в процессе компиляции неизвестны только i_1 и i_2, можно переписать приведенное выше выражение следующим образом.

$$((i_1 \times n_2) + i_2) \times d + (base - ((low_1 \times n_2) + low_2) \times d) \qquad (2)$$

Второе слагаемое этого выражения можно вычислить в процессе компиляции.

Можно обобщить хранение по строкам на случай многомерного массива. Обобщение состоит в хранении элементов массива таким образом, что при последовательном сканировании блока памяти самый правый индекс изменяется наиболее быстро. Выражение (2) обобщается в этом случае в следующее выражение для относительного адреса $A[i_1, i_2, \ldots, i_k]$.

$$((\ldots((i_1 n_2 + i_2)n_3 + i_3)\ldots)n_k + i_k)d + base - ((\ldots((low_1 n_2 + low_2)n_3 + low_3)\ldots)n_k + low_k)d \qquad (3)$$

Поскольку для всех j $n_j = high_j - low_j + 1$ считается фиксированным, второе слагаемое в (3) (выделено красным цветом) может быть вычислено компилятором и сохранено в записи таблицы символов для A.

7.2. Схема трансляции для адресации элементов массива

Основная проблема при генерации кода для работы с элементами массива состоит в том, чтобы связать данное вычисление с грамматикой. Пусть имеются следующие продукции для работы с элементами массива (*E* – нетерминал для выражения)

$L \rightarrow$ **id** [*Elist*] | **id**

Elist $\rightarrow$ *Elist* , *E* | *E*

Для того чтобы при группировании выражений индексов в *Elist* были допустимы различные пределы n_j у разных размерностей массива, следует переписать указанные продукции следующим образом.

$L \rightarrow$ *Elist*] | **id**

Elist $\rightarrow$ *Elist* , *E* | **id** [*E*

Таким образом, имя массива в процессе формирования *L* присоединяется к левому выражению индекса, а не к *Elist*. Эти продукции позволяют передать в качестве синтезируемого атрибута *Elist.array* указатель на запись в таблице символов для имени массива.

Нетерминал *Elist* имеет также атрибут *Elist.ndim* для записи количества размерностей в *Elist*. Функция *limit*(*array*, *j*) возвращает n_j, количество элементов *j*-й размерности массива, на запись которого в таблице символов указывает *array*. И наконец, *Elist.addr* обозначает временную переменную, хранящую значение, вычисленное по индексному выражению в *Elist*.

Elist, порождающий первые *m* индексов ссылки на элемент *k*-мерного массива $A[i_1, i_2, \ldots, i_k]$, будет генерировать трехадресный код для вычисления

$$(\ldots((i_1 n_2 + i_2)n_3 + i_3)\ldots)n_m + i_m, \qquad (4)$$

используя рекуррентные соотношения

$$e_1 = i_1$$

$$e_m = e_{m-1} \times n_m + i_m \qquad (5)$$

Таким образом, когда $m = k$, все, что нужно для вычисления члена в первой строке (3), – это умножение на размер *d*. Заметим, что здесь i_j представляет значение выражения, и код для его вычисления внедрен в код для вычисления (4).

Нетерминал *L* имеет два атрибута – *L.addr* и *L.offset*. Если *L* представляет собой простое имя, *L.addr* является указателем на запись в таблице символов для этого имени, а *L.offset* имеет значение **null**, указывающее, что *L* – простое имя, а не ссылка на элемент массива.

Семантические действия будут добавляться к следующей грамматике.

(1) $S \rightarrow L := E$

(2) $E \rightarrow E + E$

(3) $E \rightarrow (E)$

(4) $E \rightarrow L$

(5) $L \rightarrow Elist$]

(6) $L \rightarrow$ **id**

(7) $Elist \rightarrow Elist , E$

(8) $Elist \rightarrow$ **id** [E

Используется процедура *Gen* для формирования трехадресной команды.

Мы генерируем обычное присвоение, если L представляет собой простое имя, и индексированное присвоение по адресу, обозначенному L, в противном случае.

(1) $S \rightarrow L := E$ {**if** *L.offset* = **null then** /* *L* – простое имя */
Gen(*L.addr*’:=’*E.addr*)
else *Gen*(*L.addr*’[‘*L.offset*’]’’:=’*E.addr*)}

Код для арифметических выражений:

(2) $E \rightarrow E_1 + E_2$ {*E.addr* := *NewTemp*()
Gen(*E.addr*’:=’E_1.*addr*’+’ E_2.*addr*)}

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

Когда ссылка на массив *L* свертывается в *E*, требуется значение *L* как значение элемента массива. Таким образом, необходимо использовать индексацию для получения содержимого ячейки памяти *L.addr*[*L.offset*].

(4) $E \rightarrow L$ {**if** *L.offset* = **null then** /* *L* – простое имя */
E.addr := *L.addr*
else begin
E.addr := *NewTemp*()
Gen(*E.addr*’:=’*L.addr*’[‘*L.offset*’]’)
end}

Ниже *L.offset* является новой временной переменной, представляющей первое слагаемое в (3); функция *width*(*Elist.array*) возвращает значение *d* в (3). *L.addr* представляет второе слагаемое в (3), возвращаемое функцией *c*(*Elist.array*).

$$((\ldots((i_1 n_2 + i_2)n_3 + i_3)\ldots)n_k + i_k)d + base - ((\ldots((low_1 n_2 + low_2)n_3 + low_3)\ldots)n_k + low_k)d \qquad (3)$$

(5) $L \rightarrow Elist\ ]$ {*L.addr* := *NewTemp*()
L.offset := *NewTemp*()
Gen(*L.addr* ‘:=’ *c*(*Elist.array*))
Gen(*E.offset*’:=’*Elist.addr*’*‘*width*(*Elist.array*))}

Нулевое смещение указывает на простое имя.

(6) $L \rightarrow$ **id** {*L.addr* = **id**.*ns*
L.offset := **null**}

При рассмотрении следующего индексного выражения применяем рекуррентное соотношение (5).

$e_1 = i_1$
$e_m = e_{m-1} \times n_m + i_m$ (5)

В следующем действии $Elist_1.addr$ соответствует e_{m-1} из (5), а $Elist.addr$ – e_m. Заметим, что если $Elist_1$ имеет $m - 1$ компонент, то *Elist* в левой части продукции имеет m компонентов.

(7) $Elist \rightarrow Elist_1 , E$ {*t* := *NewTemp*()
m := $Elist_1.ndim + 1$
Gen(*t* ':=' $Elist_1.addr$ '*' *limit*($Elist_1.array$, *m*))
Gen(*t* ':=' *t* '+' *E.addr*)
Elist.array := $Elist_1.array$
Elist.addr := *t*
Elist.ndim := *m*}

$(\ldots((i_1 n_2 + i_2)n_3 + i_3)\ldots)n_m + i_m,$ (4)

E.addr хранит как значение выражения *E*, так и значение (4) для $m = 1$ (при $m = 1$ эти значения, по сути, одинаковы).

(8) *Elist* → **id** [*E* {*Elist.array* := **id**.*ns*
Elist.addr := *E.addr*
Elist.ndim := 1}

Пример.

Пусть A – массив размером 10×20 с $low_1 = low_2 = 1$. Таким образом, $n_1 = 10$, а $n_2 = 20$. Примем d равным 4. Для присвоения $x := A[y, z]$ рассмотренная схема трансляции формирует следующую последовательность трехадресных инструкций.

$t_1 := y * 20$
$t_1 := t_1 + z$
$t_2 := c$ /* Константа $c = base_A - 84$ */
$t_3 := 4 * t_1$
$t_4 := t_2[t_3]$
$x := t_4$

Для каждой переменной вместо **id**.*ns* мы использовали ее имя.

(1)	$S \to L := E$	{**if** *L.offset* = **null then** /* *L* – простое имя */ *Gen*(*L.addr*’:=’*E.addr*) **else** *Gen*(*L.addr*’[‘*L.offset*’]’’:=’*E.addr*)}
(4)	$E \to L$	{**if** *L.offset* = **null then** /* *L* – простое имя */ *E.addr* := *L.addr* **else begin** *E.addr* := *NewTemp*() *Gen*(*E.addr*’:=’*L.addr*’[‘*L.offset*’]’) **end**}
(5)	$L \to Elist$]	{*L.addr* := *NewTemp*() *L.offset* := *NewTemp*() *Gen*(*L.addr* ‘:=’ *c*(*Elist.array*)) *Gen*(*E.offset*’:=’*Elist.addr*’*‘*width*(*Elist.array*))}
(6)	$L \to$ **id**	{*L.addr* = **id**.*ns* *L.offset* := **null**}
(7)	$Elist \to Elist_1$, E	{*t* := *NewTemp*() *m* := $Elist_1$.*ndim* + 1 *Gen*(*t* ’:=’$Elist_1$.*addr*’*’*limit*($Elist_1$.*array*, *m*)) *Gen*(*t* ‘:=’ *t* ‘+’*E.addr*) *Elist.array* := $Elist_1$.*array* *Elist.addr* := *t* *Elist.ndim* := *m*}
(8)	$Elist \to$ **id** [E	{*Elist.array* := **id**.*ns* *Elist.addr* := *E.addr* *Elist.ndim* := 1}