

## 四、中间代码生成 (11. 表达式的翻译)

魏恒峰

hfwei@nju.edu.cn

2024 年 05 月 10 日



产生式	语义规则
$S \rightarrow id = E ;$	$S.code = E.code \parallel$ $gen(top.get(id.lexeme) '=' E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = new Temp()$ $E.code = E_1.code \parallel E_2.code \parallel$ $gen(E.addr '=' E_1.addr '+' E_2.addr)$
$  - E_1$	$E.addr = new Temp()$ $E.code = E_1.code \parallel$ $gen(E.addr '=' 'minus' E_1.addr)$
$  ( E_1 )$	$E.addr = E_1.addr$ $E.code = E_1.code$
$  id$	$E.addr = top.get(id.lexeme)$ $E.code = ''$

产生式	语义规则
$S \rightarrow id = E ;$	$S.code = E.code \parallel$ $gen(top.get(id.lexeme) \neq E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = new Temp()$ $E.code = E_1.code \parallel E_2.code \parallel$ $gen(E.addr \neq E_1.addr + E_2.addr)$
$  - E_1$	$E.addr = new Temp()$ $E.code = E_1.code \parallel$ $gen(E.addr \neq 'minus' E_1.addr)$
$  ( E_1 )$	$E.addr = E_1.addr$ $E.code = E_1.code$
$  id$	$E.addr = top.get(id.lexeme)$ $E.code = ''$

$S \rightarrow id = E ;$	$\{ gen(top.get(id.lexeme) \neq E.addr); \}$
$  L = E ;$	$\{ gen(L.array.base \neq L.addr \neq E.addr); \}$
$E \rightarrow E_1 + E_2$	$\{ E.addr = new Temp();$ $gen(E.addr \neq E_1.addr + E_2.addr); \}$
$  id$	$\{ E.addr = top.get(id.lexeme); \}$
$  L$	$\{ E.addr = new Temp();$ $gen(E.addr \neq L.array.base \neq L.addr); \}$
$L \rightarrow id [ E ]$	$\{ L.array = top.get(id.lexeme);$ $L.type = L.array.type.elem;$ $L.addr = new Temp();$ $gen(L.addr \neq E.addr * L.type.width); \}$
$  L_1 [ E ]$	$\{ L.array = L_1.array;$ $L.type = L_1.type.elem;$ $t = new Temp();$ $L.addr = new Temp();$ $gen(t \neq E.addr * L.type.width);$ $gen(L.addr \neq L_1.addr + t); \}$

## 表达式的中间代码翻译

综合属性  $E.code$ : 中间代码

产生式	语义规则
$S \rightarrow id = E ;$	$S.code = E.code \parallel$ $gen(top.get(id.lexeme) '=' E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = new Temp()$ $E.code = E_1.code \parallel E_2.code \parallel$ $gen(E.addr '=' E_1.addr '+' E_2.addr)$
$  - E_1$	$E.addr = new Temp()$ $E.code = E_1.code \parallel$ $gen(E.addr '=' 'minus' E_1.addr)$
$  ( E_1 )$	$E.addr = E_1.addr$ $E.code = E_1.code$
$  id$	$E.addr = top.get(id.lexeme)$ $E.code = ''$

综合属性  $E.addr$ : 变量名 (包括临时变量)、常量

# 表达式的中间代码翻译

综合属性  $E.code$ : 中间代码

产生式	语义规则
$S \rightarrow id = E ;$	$S.code = E.code \parallel$ $gen(top.get(id.lexeme) '=' E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = new Temp()$ $E.code = E_1.code \parallel E_2.code \parallel$ $gen(E.addr '=' E_1.addr '+' E_2.addr)$
$  - E_1$	$E.addr = new Temp()$ $E.code = E_1.code \parallel$ $gen(E.addr '=' 'minus' E_1.addr)$
$  ( E_1 )$	$E.addr = E_1.addr$ $E.code = E_1.code$
$  id$	$E.addr = top.get(id.lexeme)$ $E.code = ''$

$a = b + -c$

$t_1 = \text{minus } c$

$t_2 = b + t_1$

$a = t_2$

综合属性  $E.addr$ : 变量名 (包括临时变量)、常量

```
int main() {  
    int a = 0, b = 1, c = 2;  
  
    a = b + -c;  
  
    return 0;  
}
```

```
int main() {  
    int a = 0, b = 1, c = 2;  
  
    a = b + -c;  
  
    return 0;  
}
```

```
%7 = sub nsw i32 0, %6  
%8 = add nsw i32 %5, %7  
store i32 %8, i32* %2, align 4
```

## 数组引用的中间代码翻译

声明 :  $\text{int } a[2][3]$

数组引用 :  $x = a[1][2]; a[1][2] = x$



## 数组引用的中间代码翻译

声明 : `int a[2][3]`

数组引用 :  $x = a[1][2]; a[1][2] = x$

需要计算  $a[1][2]$  相对于**数组基地址**  $a$  的**偏移地址**

$$\text{addr}(a[1][2]) = \text{base} + 1 \times 12 + 2 \times 4$$

	类型	宽度
$a$	$\text{array}(2, \text{array}(3, \text{integer}))$	24
$a[i]$	$\text{array}(3, \text{integer})$	12
$a[i][j]$	integer	4

int a[2][3]

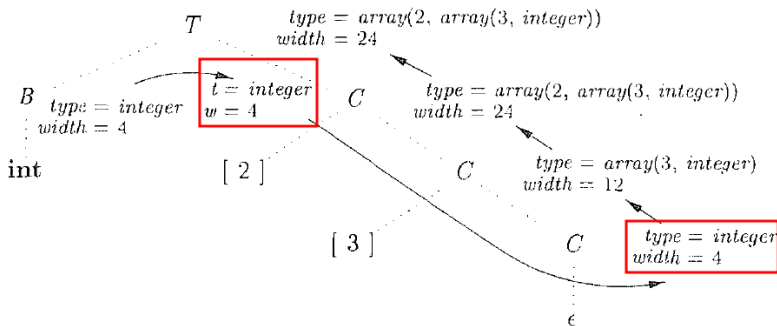


图 6-16 数组类型的语法制导翻译

## 综合属性 $L.array(.base)$ : 数组基地址 (即, 数组名)

```

S → id = E ; { gen( top.get(id.lexeme) != E.addr); }

    | L = E ; { gen( L.array.base '[' L.addr ']' != E.addr); }

E → E1 + E2 { E.addr = new Temp();
                gen(E.addr != E1.addr '+' E2.addr); }

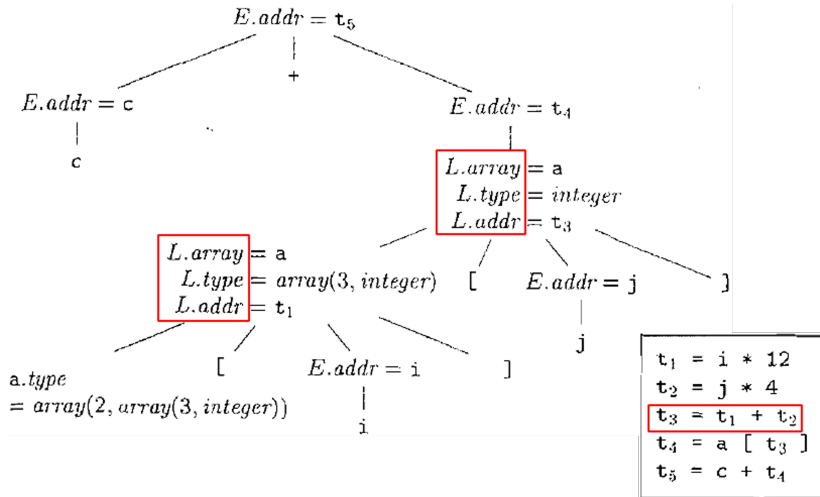
    | id      { E.addr = top.get(id.lexeme); }

    | L      { E.addr = new Temp();
                gen(E.addr != L.array.base '[' L.addr ']'); }

L → id [ E ] { L.array = top.get(id.lexeme);
                L.type = L.array.type.elem;
                L.addr = new Temp();
                gen(L.addr != E.addr '*' L.type.width); }

    | L1 [ E ] { L.array = L1.array;
                L.type = L1.type.elem;
                t = new Temp();
                L.addr = new Temp();
                gen(t != E.addr '*' L.type.width);
                gen(L.addr != L1.addr '+' t); }
    
```

## 综合属性 $L.addr$ : 偏移地址



$c + a[i][j]$

```
int main() {  
    int a[2][3] = { 0 };  
  
    int i = 1, j = 2;  
    int c = 10, d = 20;  
  
    d = c + a[i][j];  
  
    return 0;  
}
```

%2 = alloca [2 x [3 x i32]], align 16

```
int main() {  
    int a[2][3] = { 0 };  
  
    int i = 1, j = 2;  
    int c = 10, d = 20;  
  
    d = c + a[i][j];  
  
    return 0;  
}
```

%2 = alloca [2 x [3 x i32]], align 16

```
int main() {  
    int a[2][3] = { 0 };  
  
    int i = 1, j = 2;  
    int c = 10, d = 20;  
  
    d = c + a[i][j];  
  
    return 0;  
}
```

```
%8 = load i32, i32* %5, align 4 %8: c  
%9 = load i32, i32* %3, align 4 %9: i  
%10 = sext i32 %9 to i64  
%11 = getelementptr inbounds [2 x [3 x i32]], [2 x [3 x i32]]* %2, i64 0, i64 %10  
%12 = load i32, i32* %4, align 4 %12: j  
%13 = sext i32 %12 to i64  
%14 = getelementptr inbounds [3 x i32], [3 x i32]* %11, i64 0, i64 %13  
%15 = load i32, i32* %14, align 4 %15: a[i][j]  
%16 = add nsw i32 %8, %15  
store i32 %16, i32* %6, align 4
```

GEP provides a way to **access arrays and manipulate pointers**.

#### Overview: ¶

The 'getelementptr' instruction is used to get the address of a subelement of an aggregate data structure. It performs address calculation only and does not access memory. The instruction can also be used to calculate a vector of such addresses.

<https://llvm.org/docs/LangRef.html#getelementptr-instruction>

GEP abstract away details like **size of types**.



getelementptr

<base-type>, <base-type>\* <base-addr>, [i32/i64 <index>]+

getelementptr

<base-type>, <base-type>\* <base-addr>, [i32/i64 <index>]+

<base-type>: base type used for **the first** index

- ▶ It does *not* change the pointer type.
- ▶ It offsets by the <base-type>.

getelementptr

<base-type>, <base-type>\* <base-addr>, [i32/i64 <index>]+

<base-type>: base type used for **the first** index

- ▶ It does *not* change the pointer type.
- ▶ It offsets by the <base-type>.

Further indices:

- ▶ Offset **inside** arrays (aggregate types)
- ▶ *Change* the pointer type by removing one layer of “aggregation”

## getelementptr

<base-type>, <base-type>\* <base-addr>, [i32/i64 <index>]+

```
%8 = load i32, i32* %5, align 4 %8:c
%9 = load i32, i32* %3, align 4 %9:i
%10 = sext i32 %9 to i64
%11 = getelementptr inbounds [2 x [3 x i32]], [2 x [3 x i32]]* %2, i64 0, i64 %10
%12 = load i32, i32* %4, align 4 %12:j
%13 = sext i32 %12 to i64
%14 = getelementptr inbounds [3 x i32], [3 x i32]* %11, i64 0, i64 %13
%15 = load i32, i32* %14, align 4 %15:a[i][j]
%16 = add nsw i32 %8, %15
store i32 %16, i32* %6, align 4
```

getelementptr [2 x [3 x i32]], [2 x [3 x i32]]\* %2, i64 0,  
i64 %10, i64 %13

# The Often Misunderstood GEP Instruction @ LLVM Docs

## The Often Misunderstood GEP Instruction @ LLVM Docs



LLVM IR Tutorial: Phis, GEPs and Other Things, Oh My! @ bilibili

Thank  
You!



Office 926

hfwei@nju.edu.cn