是指针的话，要保存在寄存器%rax 中，如果数据类型为 short，就保存在寄存器元素%ax 中。

表达式	类型	值	汇编代码
S+ 1			
S[3]			
&S[i]			
S[4*i+ 1]			
S+ i-5			

3.8.3　嵌套的数组

当我们创建数组的数组时，数组分配和引用的一般原则也是成立的。例如，声明

```
int A[5][3];
```

等价于下面的声明

```
typedef int row3_t[3];
row3_t A[5];
```

数据类型 row3_t 被定义为一个 3 个整数的数组。数组 A 包含 5 个这样的元素，每个元素需要 12 个字节来存储 3 个整数。整个数组的大小就是 4×5×3=60 字节。

数组 A 还可以被看成一个 5 行 3 列的二维数组，用 A[0][0]到 A[4][2]来引用。数组元素在内存中按照“行优先”的顺序排列，意味着第 0 行的所有元素，可以写作 A[0]，后面跟着第 1 行的所有元素(A[1])，以此类推，如图 3-36 所示。

这种排列顺序是嵌套声明的结果。将 A 看作一个有 5 个元素的数组，每个元素都是 3 个 int 的数组，首先是 A[0]，然后是 A[1]，以此类推。

要访问多维数组的元素，编译器会以数组起始为基地址，(可能需要经过伸缩的)偏移量为索引，产生计算期望的元素的偏移量，然后使用某种 MOV 指令。通常来说，对于一个声明如下的数组：

T D[*R*][*C*];

它的数组元素 D[i][j]的内存地址为

$$\&D[i][j] = x_D + L(C \cdot i + j) \qquad (3.1)$$

这里，L 是数据类型 T 以字节为单位的大小。作为一个示例，考虑前面定义的 5×3 的整型数组 A。假设 x_A、i 和 j 分别在寄存器%rdi、%rsi 和%rdx 中。然后，可以用下面的代码将数组元素 A[i][j]复制到寄存器%eax 中：

行	元素	地址
A[0]	A[0][0]	x_A
	A[0][1]	x_A+4
	A[0][2]	x_A+8
A[1]	A[1][0]	x_A+12
	A[1][1]	x_A+16
	A[1][2]	x_A+20
A[2]	A[2][0]	x_A+24
	A[2][1]	x_A+28
	A[2][2]	x_A+32
A[3]	A[3][0]	x_A+36
	A[3][1]	x_A+40
	A[3][2]	x_A+44
A[4]	A[4][0]	x_A+48
	A[4][1]	x_A+52
	A[4][2]	x_A+56

图 3-36　按照行优先顺序存储的数组元素

```
   A in %rdi, i in %rsi, and j in %rdx
1     leaq    (%rsi,%rsi,2), %rax     Compute 3i
2     leaq    (%rdi,%rax,4), %rax     Compute x_A + 12i
3     movl    (%rax,%rdx,4), %eax     Read from M[x_A + 12i + 4j]
```

正如可以看到的那样，这段代码计算元素的地址为 $x_A+12i+4j=x_A+4(3i+j)$，使用了 x86-64 地址运算的伸缩和加法特性。