


伸缩因子 1、2、4 和 8 覆盖了所有基本简单数据类型的大小。

 **练习题 3.36** 考虑下面的声明：

```
short    S[7];
short   *T[3];
short  **U[6];
int     V[8];
double *W[4];
```

填写下表，描述每个数组的元素大小、整个数组的大小以及元素  $i$  的地址：

数组	元素大小	整个数组的大小	起始地址	元素 $i$
S			$x_S$	
T			$x_T$	
U			$x_U$	
V			$x_V$	
W			$x_W$	

### 3.8.2 指针运算


C 语言允许对指针进行运算，而计算出来的值会根据该指针引用的数据类型的大小进行伸缩。也就是说，如果  $p$  是一个指向类型为  $T$  的数据的指针， $p$  的值为  $x_p$ ，那么表达式  $p+i$  的值为  $x_p + L \cdot i$ ，这里  $L$  是数据类型  $T$  的大小。

单操作数操作符 ‘&’ 和 ‘\*’ 可以产生指针和间接引用指针。也就是，对于一个表示某个对象的表达式  $Expr$ ， $\&Expr$  是给出该对象地址的一个指针。对于一个表示地址的表达式  $AExpr$ ， $*AExpr$  给出该地址处的值。因此，表达式  $Expr$  与  $*\&Expr$  是等价的。可以对数组和指针应用数组下标操作。数组引用  $A[i]$  等同于表达式  $*(A+i)$ 。它计算第  $i$  个数组元素的地址，然后访问这个内存位置。

扩展一下前面的例子，假设整型数组  $E$  的起始地址和整数索引  $i$  分别存放在寄存器  $\%rdx$  和  $\%rcx$  中。下面是一些与  $E$  有关的表达式。我们还给出了每个表达式的汇编代码实现，结果存放在寄存器  $\%eax$  (如果是数据) 或寄存器  $\%rax$  (如果是指针) 中。

表达式	类型	值	汇编代码
$E$	$\text{int}^*$	$x_E$	<code>movq %rdx,%rax</code>
$E[0]$	$\text{int}$	$M[x_E]$	<code>movl (%rdx),%rax</code>
$E[i]$	$\text{int}$	$M[x_E + 4i]$	<code>movl (%rdx,%rcx,4),%eax</code>
$\&E[2]$	$\text{int}^*$	$x_E + 8$	<code>leaq 8(%rdx),%rax</code>
$E+i-1$	$\text{int}^*$	$x_E + 4i - 4$	<code>leaq -4(%rdx,%rcx,4),%rax</code>
$*(E+i-3)$	$\text{int}$	$M[x_E + 4i - 12]$	<code>movl -12(%rdx,%rcx,4),%eax</code>
$\&E[i]-E$	$\text{long}$	$i$	<code>movq %rcx,%rax</code>

在这些例子中，可以看到返回数组值的操作类型为  $\text{int}$ ，因此涉及 4 字节操作 (例如 `movl`) 和寄存器 (例如  $\%eax$ )。那些返回指针的操作类型为  $\text{int}^*$ ，因此涉及 8 字节操作 (例如 `leaq`) 和寄存器 (例如  $\%rax$ )。最后一个例子表明可以计算同一个数据结构中的两个指针之差，结果的数据类型为  $\text{long}$ ，值等于两个地址之差除以该数据类型的大小。

 **练习题 3.37** 假设短整型数组  $S$  的地址  $x_S$  和整数索引  $i$  分别存放在寄存器  $\%rdx$  和  $\%rcx$  中。对下面每个表达式，给出它的类型、值的表达式和汇编代码实现。如果结果