链接器使用图 7-10 中算法的第 13 行修改了引用：

```
*refptr = (unsigned) (ADDR(r.symbol) + r.addend)
        = (unsigned) (0x601018       + 0)
        = (unsigned) (0x601018)
```

在得到的可执行目标文件中，该引用有下面的重定位形式：

```
4004d9:  bf 18 10 60 00          mov    $0x601018,%edi   %edi = &array
```

综合到一起，图 7-12 给出了最终可执行目标文件中已重定位的.text 节和.data 节。在加载的时候，加载器会把这些节中的字节直接复制到内存，不再进行任何修改地执行这些指令。

```
1  00000000004004d0 <main>:
2    4004d0:  48 83 ec 08             sub    $0x8,%rsp
3    4004d4:  be 02 00 00 00          mov    $0x2,%esi
4    4004d9:  bf 18 10 60 00          mov    $0x601018,%edi    %edi = &array
5    4004de:  e8 05 00 00 00          callq  4004e8 <sum>      sum()
6    4004e3:  48 83 c4 08             add    $0x8,%rsp
7    4004e7:  c3                      retq

8  00000000004004e8 <sum>:
9    4004e8:  b8 00 00 00 00          mov    $0x0,%eax
10   4004ed:  ba 00 00 00 00          mov    $0x0,%edx
11   4004f2:  eb 09                   jmp    4004fd <sum+0x15>
12   4004f4:  48 63 ca                movslq %edx,%rcx
13   4004f7:  03 04 8f                add    (%rdi,%rcx,4),%eax
14   4004fa:  83 c2 01                add    $0x1,%edx
15   4004fd:  39 f2                   cmp    %esi,%edx
16   4004ff:  7c f3                   jl     4004f4 <sum+0xc>
17   400501:  f3 c3                   repz retq
```

a) 已重定位的.text 节

```
1  0000000000601018 <array>:
2    601018:  01 00 00 00 02 00 00 00
```

b) 已重定位的.data 节

图 7-12　可执行文件 prog 的已重定位的.text 节和.data 节。原始的 C 代码在图 7-1 中

练习题 7.4　本题是关于图 7-12a 中的已重定位程序的。

A. 第 5 行中对 sum 的重定位引用的十六进制地址是多少？

B. 第 5 行中对 sum 的重定位引用的十六进制值是多少？

练习题 7.5　考虑目标文件 m.o 中对 swap 函数的调用(图 7-5)。

```
9:   e8 00 00 00 00          callq  e <main+0xe>      swap()
```

它的重定位条目如下：

```
r.offset = 0xa
r.symbol = swap
r.type   = R_X86_64_PC32
r.addend = -4
```

现在假设链接器将 m.o 中的.text 重定位到地址 0x4004d0，将 swap 重定位到地址