目录

第 10 章 复合数据类型的定义和使用

在 C 语言程序设计中，除了基本的数据类型，如 char、short、int、float、double 等外，还可以自定义“结构”这种复合数据类型，它将各种不同类型的数据组织到一个数据结构中，简化了数据管理工作。在汇编语言程序设计中同样也可以使用结构。在学习汇编语言时，应掌握复合数据类型变量的存储方式、访问这类变量方式、及与机器指令的对应关系。

10．1 结构体

10.1.1 结构体定义

在汇编语言程序设计中，除了 byte、word、dword、qword、real4、real8 等类型外，也可以自己定义结构。在定义结构这种复合数据类型后，就可以定义该结构类型的变量。

结构体定义的一般格式如下：

结构名 struct
　　数据定义语句序列
结构名 ends

结构名应是一个合法的标识符，伪指令 struct（另一种等同写法是 struc）和 ends 需要配对使用。数据定义语句序列是一组变量的定义语句，每条变量定义语句的格式与 3.6 节中介绍的相同。

【例 10.1】 课程结构 course 的定义。

```
course   struct
   cid        dd  0          ; 课程编号
   ctitle     db  20 dup(0)  ; 课程名
   chour      db  0          ; 学时数
   cteacher   db  10 dup(0)  ; 主讲教师
   cterm      db  1          ; 开课学期
course  ends
```

也可以等价的写成如下形式：

```
course   struct
   cid       dword  0         ; 课程编号
   ctitle    byte  20 dup(0)  ; 课程名
```

```
    chour      byte   0           ; 学时数
    cteacher   byte   10 dup(0)   ; 主讲教师
    cterm      byte    1          ; 开课学期
course  ends
```

结构说明中的数据定义语句给定了结构类型中所含的变量，称为结构字段；相应的变量名称为字段名。一个结构中可以含有任意数目的字段，可以独立地存取。字段可以有名或无名，可以有初值或无初值。字段名代表了该字段的第一个字节相对于该结构开始位置的偏移地址。

与 C 语言一样，结构体中的一个字段除了可以是基本数据类型外，还可以是其他已定义的类型，例如，下面给出的了一个结构体中含有另一个结构体的示例。

```
department   struct
    dname        db   10   dup(0)    ; 系名
    daddress     db   10   dup(0)    ; 系的办公地址
    coursetable  course  <>          ; 课表
department   ends
```

在 department2 结构中含有 5 门课，结构体定义如下。

```
department2   struct
    dname        byte   10   dup(0)       ; 系名
    daddress     byte   10   dup(0)       ; 系的办公地址
    coursetable  course     5 dup (<>)    ; 课表, 5 门课
department2   ends
```

由此可见，结构类型中各个字段的定义方法与基本数据类型（byte、sbyte、word、sword、dword、sdword 等）变量的定义方法是类似的。在结构中包含另一个结构字段的定义方法与基本变量的定义方法也是类似的，只是用结构名代替了基本类型变量的数据定义伪指令，用“<>”来表示结构字段的初始值。

10.1.2 结构变量的定义

结构说明只是一种数据类型的描述，并未分配内存空间。只有在程序中定义一个结构变量时才进行存储分配。结构变量可以在数据段中定义，此时该变量是一个全局变量；结构变量也可以定义在子程序中，成为一个局部变量。结构说明一般放在程序开头，或者用头文件来存放自定义的结构类型，当然也可以放在其他位置，原则上只要结构变量定义之前即可。结构说明不属于任何段。

在数据段中，结构变量定义的一般格式为：

[变量名]　结构名 <字段赋值表 >

其中，变量名是当前定义的结构变量的名称，可以省略。结构名是在前面

说明结构类型时自定义的名字。字段赋值表用来给结构变量的各字段重新赋初值，其中各字段值的排列顺序及类型应与结构说明时的各字段相一致，中间用逗号分隔。如果某个字段采用在结构说明时指定的初值，那么可简单地用逗号表示；如果不打算对结构变量重新赋值，则可省去字段赋值表，但仍必须保留一对尖括号。下面是几种正确的定义形式。

```
.data
ke1   course  < >          ; 5 个字段均用结构说明时给的初值
ke2   course  <2102, 'math', 40, 'liming', 2>
      course  <2103, 'chinese', 80, ' zhangsan' , >
            ; cterm 字段未重新赋值，默认为 1
ke3   course  5 dup(<2104, ,60, ,>)
            ; 定义了 5 个相同的结构变量，对 cid、chour 赋了值
      course  10 dup(<>)
            ;  定义了 10 个结构变量，即预留了相应的存储空间
```

由此可见，对于结构变量的定义与基本类型（byte、word 等）变量的定义语法是相似的，差别是在给变量赋初值的形式上，结构变量要使用一对尖括号 <> 来“封装”各个字段。这种“封装”的做法也是比较容易理解的，否则单纯的以逗号来分隔，会产生歧义，不清楚该逗号都字段之间的分隔，还是多个结构数据之间的分隔。

在子程序中，定义局部的结构变量时，使用语句：

local　变量名[数量]：结构名

```
course   struct
   cid        dd   0           ; 课程编号
   ctitle     db   20 dup(0)    ; 课程名
   chour      db   0           ; 学时数
   cteacher   db   10 dup(0)    ; 主讲教师
   cterm      db   1           ; 开课学期
course   ends
```

10.1.3 结构变量的访问

（1）直接用变量名加字段名的方式访问

对有变量名的结构变量，不论是全局变量还是局部变量，都可通过结构变量名直接存取结构变量。若要存取其中某个字段，则可采用“结构变量名.结构字段名”的形式，这与 C 语言的用法是很相似的。例如：

```
mov   eax, ke2.cid        ; 将 2102 送到 eax 寄存器中
mov   al,  ke2.ctitle     ; ctitle 中的字符‘m’送到 al 中
mov   ah,  ke2.ctitle+2   ; ctitle 中的字符‘t’送到 ah 中
```

在上面的三条语句中，都直接使用了结构变量名 ke2。结构变量在它所定义的数据段中都有一个偏移地址，称结构首址，内部的每一个结构字段相对于结构首址都有一个位移量。当采用“结构变量名.结构字段名”存取结构字段时，该字段的偏移地址即为结构首址与该字段的位移量之和。对于数据段定义的全局变量，对应的机器指令中采用的是直接寻址方式 ds:[n]。上面三条指令的反汇编代码如下，其中 ke2 的地址为 0C850C3h。

```
mov    eax, dword ptr [ke2 (0C850C3h)]
mov    al,    byte ptr ds:[00C850C7h]
mov    ah,    byte ptr ds:[0C850C9h]
```

对于子程序中定义的局部变量，所对应的寻址方式是变址寻址[ebp+n]。

（2）寄存器间接加字段名的方式访问

另一种存取结构字段的方法是把结构首址先存入某个寄存器，然后用“[寄存器名].结构名.字段名”来访问一个字段。此时对应的寻址方式是变址寻址，移位量就是字段名所指定项在结构中的偏移地址。显然，在不同的结构中可以有相同的字段名，单纯一个字段名是不能确定是哪个结构中的字段，采用“结构名.字段名”的形式才具有唯一性。下面给出了一个用法示例。

```
mov    ebx，offset    ke2          ；获得结构变量 ke2 的结构首址→ebx
mov    al，[ebx].course.chour       ；将 ke2 中 chour 字段的值→al
```

该语句对应的反汇编指令是：mov al, [ebx+18H]。

另外一种写法是类似于强制类型转换方法，表示方法如下：

```
mov    al，(course ptr [ebx]).chour
```

在上面的指令中，是以(ebx)为起始地址，将其当成一个 course 结构的起始地址，访问其中的字段 chour，对应的机器指令同样是：mov al, [ebx+18H]。

从逻辑上看，ebx 可以是任意一个单元的地址，可以用各种符合语法规则的语句给 ebx 赋值，下面给出了几种强制类型转化的写法，有关结构的定义、函数说明等从略。

【例 10.2】 在以 x 为首地址的字节缓冲区中存储结构 course 中各字段的信息。

```
.data
  lpfmt   db '%s',0
  x       db  100 dup(0)
.code
 start:
    mov   (course ptr x).cid, 100
    invoke scanf, offset lpfmt, offset (course ptr x).ctitle
    lea     ebx, x
    mov     [ebx].course.chour, 40
    lea     esi, [ebx + offset course.cteacher]
    invoke scanf, offset lpfmt, esi
    mov    (course ptr [ebx]).cterm, 2
    invoke ExitProcess, 0
end start
```

上面的例子只给出了程序的片段，程序处理器选择伪指令、存储模型说明伪指令、函数说明、引用库说明等和以前的例子是一样的。结构定义一般放在

程序的开头。在调试程序时，也可以使用强制类型转换的方法，观察变量单元中的数据。例如，在监视窗口，输入 *(course *)&x, 就可以看到将 x 视为一个 course 结构的存储结果。当然，也可以在内存窗口观察 x 中存放的结果。

（3）寄存器间接寻址访问

可以将要访问的结构变量的首地址送给一个寄存器，然后将寄存器中的值再加上字段在结构中的相对地址，得到的寄存器的值即为要访问单元的地址。

```
mov   ebx，offset  ke2   ；获得结构变量 ke2 的结构首址→ebx
add   ebx，18H           ；  chour 在 course 中的偏移地址是 18H
mov   al，[ebx]          ；将 ke2 中 chour 字段的值→al
```

10.1.4 结构信息的自动计算

在编写程序时，有是需要知道一个字段在结构中的偏移地址以便更灵活的访问各个字段;有时需要知道一个结构所占长度,这样对结构数组进行访问时，可以从上一个元素的地址增加结构长度得到下一个元素的地址；有时还需要知道构数组的长度。这些信息除了用人工方法计算得到外，还可以由编译器自动的计算。汇编语言中提供了 offset、sizeof、size、type、length 等运算符来获取相关信息。

① 取偏移地址运算符 offset

offset 后面可以跟单个变量名、结构名.字段名、结构变量名.字段名，它们的含义有所不同。因此在使用时要注意符号的含义。

语法格式： offset 变量名

功　　能：对一个全局变量，获得其在段中的偏移地址。

语法格式： offset 结构名.字段名

功　　能：获得一个字段在一个结构中的偏移地址。

语法格式： offset 结构变量名.字段名

功　　能：获得一个结构变量中指定的字段在段中的偏移地址。

例如： 设在数据段中有如下变量定义：

ke2 course <2102，'math'，40，'liming'，2>

其中 course 是例 10.1 中定义的一个结构。

假设 ke2 在数据段中的偏移地址为 904003H 。

```
mov  ebx,  offset  ke2          ; (ebx)=904003H
mov  ebx,  offset  course.chour ; (ebx)=18H
mov  ebx,  offset  ke2.chour    ; (ebx)=90401BH
```

② 取结构的长度 type

语法格式： type 类型名 或 type 变量名

功　　能： 获得一个数据类型的长度，对于变量获得的是该变量对应的类

型的长度。

例如：
```
mov  eax,  type  byte     ;(eax)=1
mov  eax,  type  sbyte    ;(eax)=1
mov  ecx,  type  course   ; (ecx)=24H
mov  ecx,  type  ke1      ; (ecx)=24H
mov  ecx,  type  ke3      ; (ecx)=24H
```
其中 ke1,ke3 的定义如下：
```
ke1  course  < >
ke3  course  5 dup(<2104, ,60, ,>)
```

③ 取变量所含的元素个数 length

语法格式： length 变量名

功　　能： 获得定义该变量时第一个表达式对应的元素个数，当定义语句为 “n dup （表达式）”时，返回结果为 n，其他情况为 1。

④ 取结构或变量所含的数据存储区大小 size/sizeof

语法格式： size 结构名　或者　sizeof 结构名

功　　能： 获得结构的大小。

等价于 type 结构名。

语法格式： size 变量名　或者　sizeof 变量名

功　　能： 获得变量定义时的第 1 个表达式所占的单元字节数。

size 变量 = （length 变量）*（type 变量）

例：
```
mov  ecx,  sizeof  course  ; (ecx)=24H = 36
mov  ecx,  sizeof  ke1     ; (ecx)=24H = 36
mov  ecx,  sizeof  ke3     ; (ecx)=0B4H = 5*36 = 180
```

在 C 语言程序中，可以用 offsetof 函数来求一个字段的偏移地址，用 sizeof 求一个结构的大小。下面给出了用 C 语言定义的与本节中用汇编语言定义等价的结构。

```
typedef  struct {
    int  cid;           // 课程编号
    char ctitle[20];    // 课程名
    char chour;         // 学时数
    char cteacher[10]; // 主讲教师
    char cterm;        // 开课学期
}course;
#include <stddef.h>
……
printf("%d %d\n", sizeof(course), offsetof(course, ctitle));
```

上面的语句可显示结构的大小，以及给定字段在结构中的偏移地址。

10．2 结构变量的数据存储

10.2.1 汇编语言中结构变量的存储

下面给出了一个完整的例子以及其反汇编的代码，读者可以从中分析结构变量的各字段间地址的关系，以及数据存放的规律。

```
.686P
.model flat, c
  ExitProcess proto stdcall :dword
  scanf       proto :ptr sbyte, :VARARG
  includelib  libcmt.lib
  includelib  legacy_stdio_definitions.lib
course   struct
    cid      dd  ?          ;课程编号
    ctitle   db  20 dup(0)  ;课程名
    chour    db  0          ;学时数
    cteacher db 10 dup(0)   ;主讲教师
    cterm    db  1          ;开课学期
course  ends
.data
    lpfmt db '%s',0
    Ke1   course <>
    Ke2   course <2102,'math',40,'liming',2>
          course <2103,'chinese',80,'zhangsan',>
    ke3   course 5 dup(<2104, , 60, ,>)
          course 10 dup(<>)
.stack 200
.code
 main  proc
    mov    ke1.cid, 100
    invoke scanf, offset lpfmt, offset ke1.ctitle
    lea    ebx, ke1
    mov    [ebx].course.chour, 40
    mov    [ebx].course.cterm, 2
    invoke  ExitProcess, 0
 main endp
```

end

在程序运行过程中，输入了课程名（ctitle）为 english。在执行“call _ExitProcess@4”之间，可以在监视窗口观察结构变量 ke1 的地址（&ke1），在内存窗口观察以该地址为起始地址单元中的内容，进而看到各个字段中存放的数据所占的长度、字段中数据的存放形式，以及字段之间数据存放的顺序。结果如下：

64 00 00 00 65 6e 67 6c 69 73 68 00 00 00 00 00 00 00
00 00 00 00 00 00 28 00 00 00 00 00 00 00 00 00 00 02

其中，ke1 第 1 个字段 cid 定义为 dword，占 4 个字节，其值为 100，即 00000064H，一个双字数据在存储时采用“低字节内容存放在低地址单元”的模式，这与前面介绍的数据存放模式是相同的。ke1 的第 2 个字段 ctitle 出现在第一个字段后，从其起始地址开始，从低地址单元到高地址单元，依次存放字符串“english”中从左到右的各个字符的 ASCII 码，最后以 0 结束。第 3 个字段 chour 的占 1 个字节，其值为 28H，即 40。之后的 10 个字节为第 4 个字段 cteacher，它的值全为初始值 0,。最后一个字段为 cterm，其值为 2。

从该例可以看出：结构变量中各字段按定义时的先后顺序依次存放，每个字段所占长度由定义时的长度所确定。因此，对于一个结构变量，可以直接通过“变量名.字段名”的方式来访问一个字段，编译器会自动计算该字段在结构中的偏移地址。从机器语言的角度来看，并没有结构和结构字段的说法，只有访问单元的地址。但一个结构变量中各字段之间的地址关系是明确的，因而可由编译器自动完成翻译工作。

对于结构体中含有另一个结构体的情况，访问时与 C 语言类型，可以直接用“变量名.结构字段名.字段名”的形式，也可以先取到内层结构字段的首地址，然后再以该地址为起始地址，加上要访问内层结构字段在内层结构中的偏移地址。下面给出了一个具体的例子。结构体 course 和 department 的定义见 10.1.1 节。

```
.data
lpfmt   db '%s',0
mydepartment    department  <>
.code
 main  proc
    invoke scanf, offset lpfmt, offset mydepartment.dname
    invoke scanf, offset lpfmt, offset mydepartment.daddress
    lea    ebx,  mydepartment.coursetable
    mov    [ebx].course.cid, 1010
    add    ebx,  offset course.ctitle
    invoke scanf, offset lpfmt, ebx
    invoke ExitProcess, 0
```

```
course   struct
   cid        dd  0          ; 课程编号
   ctitle     db  20 dup(0)  ; 课程名
   chour      db  0          ; 学时数
   cteacher   db  10 dup(0)  ; 主讲教师
   cterm      db  1          ; 开课学期
course  ends
```

```
department   struct
   dname       db  10  dup(0)   ; 系名
   daddress    db  10  dup(0)   ; 系的办公地址
   coursetable  course  <>      ; 课表
department   ends
```

```
main endp
end
```

在程序中也可以用“mov mydepartment.coursetable.cid,1010”代替“mov [ebx].course.cid, 1010”。

在结构中嵌套有结构时，结构变量的存储形式与一般的结构变量的存储方式是相同的。

10.2.3 与 C 语言结构变量存储的差异

编写过 C 语言程序的人，会发现 C 语言结构的大小、以及各个字段在结构中的偏移地址似乎与汇编语言有所差距。下面给出了一个示例。

```
#include <stddef.h>
typedef  struct t{
   char f1;
   int  f2;
}temps;
int i = sizeof(temps);         // i=8
int j = offsetof(temps, f1);   // j=0
int k = offsetof(temps, f2);   // k=4
```

而同样的用汇编语言定义的结构，得到的结构大小和字段的偏移地址有所不同。

```
temps struct
  f1  byte 0
  f2  dword 0
temps ends
mov eax, sizeof temps      ; (eax)=5
mov ebx, offset temps.f1   ; (ebx)=0
mov ecx, offset temps.f2   ; (ecx)=1
```

产生这一差异的原因，是因为一个结构的大小是与其编译时采用的“结构体对齐”参数有关的。在 Visual Studio 中，可以在“C/C++/代码生成/结构成员对齐”中选择不同的对齐方式，也可以在程序中使用语句“#pragma pack(n)”来设置对齐方式。汇编语言中采用的是紧凑对齐模式，等价于在 C 语言程序中使用#pragma pack(1)的结果。

在 C 语言和汇编程序混合编程时，使用相同的结构定义时要注意编译时的对齐方式，否则同一个字段在 C 语言程序和汇编语言程序中有不同的地址，就会导致数据访问的逻辑错误。

10．3 union 联合体

除了 struct 结构外，Microsoft Visual Studio 开发平台也支持 union 结构。多数据类型联合（union）定义的语法如下：

```
联合体名称   union
      字段定义
[联合体名称]   ends
```

例如：

```
myunion     union
            num      dword   0
            chars    byte 4 dup(0)
        myunion      ends
```

与struct定义结构一样，定义联合体是不分配空间的。在数据段中定义变量才分配空间。例如： u1 myunion <34353637H>。

访问联合体变量的方法与访问结构变量的用法是类似的。下面给出了具体的例子。

```
mov   eax, u1.num
mov   al, u1.chars[2]
lea   ebx, u1
mov   al, [ebx].myunion.chars[2]
```

它们对应的反汇编代码如下,其中 u1 的起始地址为 00C4428Bh：

```
mov   eax,dword ptr ds:[00C4428Bh]
mov   al,byte ptr ds:[00C4428Dh]
lea   ebx,ds:[0C4428Bh]
mov   al,byte ptr [ebx+2]
```

我们也可以用下面的语句来获取有关 union 联合体的信息。

```
mov eax, offset myunion.num    ; (eax)=0
mov ebx, offset myunion.chars  ; (ebx)=0;
mov ecx, sizeof myunion        ; (ecx)=4
```

由此可见，myunion 联合体中的两个字段 num 和 chars 具有相同的偏移地址，它们指向了相同位置的单元，只是两个字段的类型不同。定义 union 联合体，可以根据需要用不同方式解读相同单元中的内容。

习题 10

10.1 定义一个名为 student 的结构，包括学生姓名、学号、年龄、电话号码等字段，字段长度自定。定义一个 student 结构的结构变量，给其各个字段

赋值，最后显示个字段。

10.2 在子程序调用时，若参数较多，可定义一个结构来存储这些参数。试编写一个子程序，实现将一个缓冲区拷贝的功能（类似 C 语言中的 memcpy）。要求定义一个结构体存储所有的参数，包括源缓冲区的首地址、目的缓冲区的首地址、拷贝的字节数。子程序中不得全局变量。

上机实践 10

10.1 设有如下 C 语言程序段,试分析函数调用时,结构变量参数是如何传递的。

```
typedef  struct t{
   int  cid;           // 课程编号
   char ctitle[20];    // 课程名
   char chour;         // 学时数
   char cteacher[10]; // 主讲教师
   char cterm;        // 开课学期
}course;
void f(course x)
{   ……
}
int main(int argc, char* argv[])
{
    course  temp;
    ……
    f(temp);
    ……
}
```