Лекция 15

Раздел 4. Структуры и записи

4.2. Определение структур и объединений

Рассмотренные нами ранее массивы представляют собой совокупность однотипных элементов. Но часто в приложениях возникает необходимость рассматривать некоторую совокупность данных разного типа как некоторый единый тип. Это очень актуально, например, для программ баз данных, где с одним объектом может ассоциироваться совокупность данных разного типа.

Структуры и объединеия - это специальные конструкции ассемблера, позволяющие смешивать и объединять данные различных типов. Структура представляет тип данных, в котором содержится один или более элементов данных, называемых членом структуры.

Структура отличается от записи тем, что члены структуры всегда представлены определенным количеством байтов. Размер структуры определяется суммарным размером всех элементов данных, входящих в нее.

Объединения подобны структурам, за исключением того, что все члены объединения занимают один и тот же участок памяти. Размер объединеия таким образом определяется размером самого большого его члена. Объединеия применяются в тех случаях, когда блок памяти должен представлять один из нескольких возможных типов данных,

Ассемблер позволяет применять вложенные структуры и объединения, хотя злоупотребление этим значительно усложняет восприятие программы.

С целью повысить удобство использования языка ассемблера в него также был введен такой тип данных.

По определению **структура** — это тип данных, состоящий из фиксированного числа элементов разного типа.

Для использования структур в программе необходимо выполнить три действия.

1. Задать шаблон структуры. По смыслу это означает определение нового типа данных, который впоследствии можно использовать для определения переменных этого типа.
2. Определить экземпляр структуры. Этот этап подразумевает инициализацию конкретной переменной с заранее определенной (с помощью шаблона) структурой.
3. Организовать обращение к элементам структуры.

Очень важно хорошо понимать разницу между описанием структуры в программе и ее определением. Описание структуры в программе означает лишь указание компилятору ее схемы, или шаблона; память при этом не выделяется. Компилятор извлекает из этого шаблона информацию о расположении полей структуры и их значениях по умолчанию. Определение структуры означает указание транслятору на выделение памяти и присвоение этой области памяти символического имени. Описать структуру в программе можно только один раз, а определить — любое количество раз. После того как структура определена, то есть ее имя связано с именем переменной, возможно обращение к полям структуры по их именам.

Описание шаблона структуры

Описание шаблона структуры имеет следующий синтаксис:

имя_структуры STRUC
<описание полей>
имя_структуры ENDS

Здесь <описание полей> представляет собой последовательность директив описания данных **DB, DW, DD, DQ** и **DT**. Их операнды определяют размер полей и при

необходимости — начальные значения. Этими значениями будут, возможно, инициализироваться соответствующие поля при определении структуры.

Как мы уже отметили, при описании шаблона память не выделяется, так как это всего лишь информация для транслятора. Местоположение шаблона в программе может быть произвольным, но, следуя логике работы однопроходного транслятора, шаблон должен быть описан раньше, чем определяется переменная с типом данных структуры. То есть при описании в сегменте данных переменной с типом некоторой структуры ее шаблон необходимо описать в начале сегмента данных либо перед ним.

Рассмотрим работу со структурами на примере моделирования базы данных о сотрудниках некоторого отдела. Для простоты, чтобы уйти от проблем преобразования информации при вводе, условимся, что все поля символьные.

Определим структуру записи этой базы данных следующим шаблоном:

```
worker struc ;информация о сотруднике
 nam db 30 dup (" ")      ;фамилия, имя, отчество
 sex db " "               ;пол
 position db 30 dup (" ") ;должность
 age db 2 dup (" ")       ;возраст
 standing db 2 dup (" ")  ;стаж
 salary db 4 dup (" ")    ;оклад в рублях
 birthdate db 8 dup (" ") ;дата рождения
worker ends
```

Создание экземпляра структуры

Для использования описанной с помощью шаблона структуры в программе необходимо определить переменную с типом данных структуры. Для этого используется следующая синтаксическая конструкция:

[имя переменной] имя_структуры <[список значений]>

Здесь:

имя переменной — идентификатор переменной данного структурного типа. Задание имени переменной необязательно. Если его не указать, будет просто выделена область памяти размером в сумму длин всех элементов структуры;

список значений — заключенный в угловые скобки список начальных значений элементов структуры, разделенных запятыми, но его задание необязательно..

Если список указан не полностью, то все поля структуры для данной переменной инициализируются значениями из шаблона. Допускается инициализация отдельных полей, но в этом случае пропущенные поля, которые будут инициализированы значениями из шаблона структуры, должны отделяться запятыми. Если при определении новой переменной с типом данных структуры мы ***согласны*** со всеми ***значениями*** полей в ее шаблоне (то есть заданными по умолчанию), то нужно просто написать ***угловые скобки***.

Пример:

```
victor worker <>
```

Для примера определим несколько переменных с типом описанной ранее структуры:

```
data segment
sotrl worker <"Гурко Андрей Вячеславович",,'художник', '33' , '15' , '1800' , '26.01.64 >
sotr2 worker <"Михайлова Наталья Геннадьевна",'ж','программист', '30','10'.'1680'.'27.10.58>
sotr3 worker <"Степанов Юрий Лонгинович",,'художник', '38' , '20','1750','01.01.58'>
sotr4 worker <"Юрова Елена Александровна",'ж','связист', '32' , '2' , , '09.01.66>
sotrS worker <> ;здесь все значения по умолчанию
data ends
```

Методы работы со структурами

Смысл введения структурного типа данных в любой язык программирования состоит в объединении разнотипных переменных в один объект. В языке должны быть средства доступа к этим переменным внутри конкретного экземпляра структуры. Для того чтобы сослаться в команде на поле некоторой структуры, используется специальный оператор. (точка):

адресное_выражение . имя_поля_структуры

Здесь:

- **адресное_выражение** — идентификатор переменной некоторого структурного типа или выражение в скобках в соответствии с указанными ранее синтаксическими правилами (рис.);
- **имя_поля_структуры** — имя поля из шаблона структуры (это на самом деле тоже адрес, а точнее, смещение поля от начала структуры).

Оператор «.» (точка) вычисляет выражение (**адресное_выражение**) + (**имя_поля_структуры**).

Продемонстрируем с помощью определенной нами структуры worker некоторые приемы работы со структурами. К примеру, требуется извлечь в регистр **AX** значения поля с возрастом. Так как вряд ли возраст трудоспособного человека может быть больше 99 лет, то после помещения содержимого этого символьного поля в регистр **AX** его будет удобно преобразовать в двоичное представление командой **AAD**. Будьте внимательны, так как из-за принципа хранения данных «младший байт по младшему адресу» старшая цифра возраста будет помещена в **AL**, а младшая — в **AH**. Для корректировки достаточно использовать команду ***xchg ah,al***:

```
mov ax,word ptr sotrl.age ;в al возраст sotrl
xchg ah, al
;а можно и так:
lea bx,sotrl
mov ax,word ptr [bx].age
xchg ah, al
aad ; преобразование BCD в двоичное представление
```

Давайте представим, что сотрудников не четверо, а намного больше, и к тому же их число и информация о них постоянно меняются. В этом случае теряется смысл явного определения переменных с типом worker для конкретных личностей.

Ассемблер разрешает определять не только отдельную переменную с типом структуры, но и массив структур.

К примеру, определим массив из 10 структур типа worker:

```
mas_sotr worker 10 dup (<>)
```

Дальнейшая работа с массивом структур производится так же, как и с одномерным массивом. Здесь возникает несколько вопросов. Как быть с размером и как организовать индексацию элементов массива? Аналогично другим идентификаторам, определенным в программе, транслятор назначает имени типа структуры и имени переменной с типом структуры атрибут типа. Значением этого атрибута является размер в байтах, занимаемый полями структуры. Извлечь это значение можно с помощью оператора:

TYPE имя_типа_структуры

После того как становится известным размер экземпляра структуры, организация индексации в массиве структур не представляет особой сложности.

Пример:

```
worker struc
……..
worker ends
……
mas_sotr worker 10 dup (<>)
……..
mov bx, type worker ;bx=77=4Dh
lea di,mas_sotr
;извлечь и вывести на экран пол всех сотрудников:
mov cx,10
cycl:
mov dl,[di].sex
... ;вывод на экран содержимого
;поля sex структуры worker
add di.bx ;к следующей структуре в массиве mas_sort
loop cycl
```

Как выполнить копирование поля из одной структуры в соответствующее поле другой структуры? Или как выполнить копирование всей структуры?

Пример: выполним копирование поля ***nam*** третьего сотрудника в поле ***nam*** пятого сотрудника:

```
worker struc
 ……..
worker ends
 …….
mas_sotr worker 10 dup (<>)
……..
mov bx,offset mas_sotr
mov si,(type worker)*2 ;si=77*2
add si,bx
mov di,(type worker)*4 ;si=77*4
add di,bx
mov cx,30
rep movsb ; пересылка элементов последовательности (ds:esi/si) в (es:edi/di) памяти
```

Объединения

Представим ситуацию, когда мы используем некоторую область памяти для размещения того или иного объекта программы (переменной, массива или структуры). Вдруг после некоторого этапа работы у нас отпадает надобность в этих данных. В обычном случае память остается занятой до конца работы программы.

Конечно, ее можно было бы задействовать для хранения других переменных, но без принятия специальных мер нельзя изменить тип и имя данных/Неплохо было бы иметь возможность переопределить эту область памяти для объекта с другими типом и именем. Ассемблер предоставляет такую возможность в виде специального типа данных, называемого объединением.

Объединение — тип данных, позволяющий трактовать одну и ту же область памяти как данные, имеющие разные типы и имена.

Описание объединений в программе напоминает описание структур, то есть сначала указывается шаблон, в котором с помощью директив описания данных перечисляются имена и типы полей:

имя_объединения UNION
<описание полей>
имя_объединения ENDS

Отличие объединений от структур состоит, в частности, в том, что при определении переменной типа объединения память выделяется в соответствии с размером максимального элемента. Обращение к элементам объединения происходит по их именам, но при этом нужно, конечно, помнить, что все поля в объединении накладываются друг на друга. Одновременная работа с элементами объединения исключена. В качестве элементов объединения можно использовать и структуры.

Пример, который мы сейчас рассмотрим, примечателен тем, что кроме демонстрации собственно типа данных объединение, в нем показывается возможность взаимного вложения структур и объединений. Постарайтесь внимательно отнестись к анализу этой программы. Основная идея здесь в том, что указатель на память, формируемый программой, может быть представлен в виде:

- 16-разрядного смещения;
- 32-разрядного смещения;
- пары из 16-разрядного смещения и 16-разрядной сегментной составляющей адреса;
- пары из 32-разрядного смещения и 16-разрядного селектора.

Какие из этих указателей можно применять в конкретной ситуации, зависит от режима адресации (use16 или use32) и режима работы процессора. Шаблон объединения, описанный в примере, позволяет упростить формирование и использование указателей различных типов.

Вложенные объединения

Ассемблер позволяет определять вложенные директивы **UNION** и **ENDS** внутри открытого описания данных типа объединения.

В структуре каждый элемент данных начинается сразу за окончанием предыдущего. В объединении каждый элемент начинается по тому же смещению, что и предыдущий. Позволив одному элементу данных содержать целое объединение, можно получить большие возможности гибкого управления данными.

Пример:

```
CUNION   STRUC
 CTYPE    DB ?
  UNION     ;Начало объединения
   STRUC
      CT0PAR1      DW 1
      CT0PAR2      DB 2
    ENDS
   STRUC
    CT1PAR1      DB 3
    CT1PAR2      DD 4
   ENDS
  ENDS
ENDS
```

Члены объединений этого примера имеют следующие параметры:

Имя	Тип	Смещение	Значение по умолчанию
CTYPE	BYTE	0	?
CT0PAR1	WORD	1	1
CT0PAR2	BYTE	3	2
CT1PAR1	BYTE	1	3
CT1PAR2	DWORD	2	4

1	2	3	4	5	6
CTYPE	CT0PAR1 CT1PAR1	CT0PAR1 CT1PAR2	CT0PAR2 CT1PAR2	CT1PAR2	CT1PAR2

Длина этой структуры составляет **6** байт.

Объединения отличаются от структур тем, что их члены перекрывают друг друга. При инициализации данных типа объединения необходимо проявлять осторожность, так как Ассемблер позволяет только одному члену экземпляра объединения иметь инициализирующее значение.

Например:

```
BUNION {}
```

является допустимым, так как все три члена объединения в объявлении данных типа объединения были неинициализированы. Этот оператор эквивалентен

```
DB 4 DUP(?)
```

В данном примере резервируется 4 байта, так как размер объединения определяется размером максимального его члена (в данном случае - двойное слово).

Например:

```
BUNION {z=1}
```

эквивалентно

```
DB 1
DB 3 DUP(?)
```

Конструкция

```
BUNION {x=1,z=2}
```

Недопустима

Альтернативным методом инициализации экземпляра объединения является использование инициализатора с угловыми скобками (<>). Значения в инициализаторе такого вида не имеют имен, поэтому они должны обязательно следовать в том же порядке, в каком описаны соответствующие члены объединения в определении структур или объединений. Ключевое слово **?** указывает, что данный член структуры будет неинициализированным. Например:

```
ASTRUC <'abc',,?>
```

является эквивалентом

```
DB 'abc'
DW 1
DD ?
```

При указании в инициализаторе экземпляра объединения меньшего количества значений инициализации Ассемблер использует для всех оставшихся членов экземпляра объединения те значения, которые были указаны при определении типа этой объединения. Таким образом, вместо ASTRUC <'abc',,> можно указать ASTRUC <'abc'>.

Если применяется инициализатор с угловыми скобками, то при инициализации вложенных объединений используются специальные соглашения. Для каждого вложенного уровня необходимо указать дополнительную пару угловых скобок. Например, для типа данных CUNION:

```
CUNION <0,<<2,>,?>>
```

эквивалентно

```
DB 0
DW 2
DB 2
DB 2 DUP(?) ;чтобы сохранить правильный размер  объединения
```

Пример: использование объединения

```
masm
model small
stack 256
.586P
pnt struc       ;структура pnt, содержащая вложенное объединение
        union           ;описание вложенного в структуру объединения
                offs_16         dw      ?
                offs_32         dd      ?
        ends            ;конец описания объединения
        segm    dw      ?
ends            ;конец описания структуры
.data
point   union   ;определение объединения, содержащего вложенную структуру
        off_16 dw       ?
        off_32 dd       ?
        point_16        pnt     <>
        point_32        pnt     <>
point   ends
tst     db      "Строка для тестирования"
adr_data        point   <>      ;определение экземпляра объединения
.code
main:
        mov     ax,@data
        mov     ds,ax
        mov     ax,seg tst
;записать адрес сегмента строки tst в поле структуры adr_data
        mov     adr_data.point_16.segm,ax
;когда понадобится можно извлечь значение из этого поля обратно, в регистр bx:
        mov     bx,adr_data.point_16.segm
;формируем смещение в поле структуры adr_data
        mov     ax,offset tst     ;смещение строки в ax
        mov     adr_data.point_16.offs_16,ax
;аналогично, когда понадобится, можно извлечь значение из этого поля:
        mov     bx,adr_data.point_16.offs_16
        exit:
        mov     ax,4c00h
        int     21h
end     main
```

Какой будет выглядеть строка, чтоб **segm=18** ?

Когда вы будете работать в защищенном режиме процессора и использовать 32-разрядные адреса, то аналогичным способом можете заполнить и задействовать описанное ранее объединение.