**Константа в ассемблере.**

Мы уже знаем, что значение (конкретное число) можно присвоить переменной, предварительно определив размер этой переменной в байт, слово, двойное слово и т.д.:

* DB — Define Bite.
* DW — Define Word.
* DD — Define Double Word.
* DQ — Define Quatro Bites.
* DT — Define Tetro Bites.

Константа — символ, синоним конкретного числа (выражения, строки), которое, в отличие от переменной нельзя изменить.

Для задания констант применяются обозначения:

* .const — все данные будут восприниматься как константы, до момента изменения сегмента (.data, .code и т.п.).
* = (знак равно).
* equ (может использоваться для создания идентификатора, константного выражения, строки).

**Структура в ассемблере.**

Структура в ассемблере (structure) — это совокупность переменных, объединенных одним именем. Переменные называются полями и могут быть разными по размеру. Очень удобно обращаться к данным по именам полей. Структура — основа абстракции, «блочности» кода. Понятие КЛАСС в языках высокого уровня есть не что иное, как разновидность структуры. Только в качестве полей в классе кроме данных присутствуют ещё и функции. В качестве поля в структуру может входить структура (пример — в коде).

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 заранее определенной (с помощью шаблона) структурой.
3. *Организовать обращение к элементам структуры.*

Очень важно, чтобы вы с самого начала уяснили, в чем разница между описанием структуры в программе и ее определением.  
***Описать структуру*** в программе означает лишь указать ее схему или шаблон; память при этом не выделяется. Этот шаблон можно рассматривать лишь как информацию для транслятора о расположении полей и их значении по умолчанию.  
***Определить структуру*** — значит, дать указание транслятору выделить память и присвоить этой области памяти символическое имя.  
Описать структуру в программе можно только один раз, а определить — любое количество раз.  
  
**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 описания шаблона* в программе может быть произвольным, но, следуя логике работы однопроходного транслятора, он должен быть расположен до того места, где определяется переменная с типом данной структуры. То есть при описании в сегменте данных переменной с типом некоторой структуры ее шаблон необходимо поместить в начале сегмента данных либо перед ним.  
  
===

* MY\_STRUCT\_1 STRUC; структура в ассемблере объявляется словом **STRUC**  
  member\_1 dw ?;  
  member\_2 db ?;  
  MY\_STRUCT\_1 ENDS;

….

* my\_struct MY\_STRUCT\_1 <?>; структура в ассемблере, созданная на основе объявления.
* mov my\_struct.member\_1 ,33h;используем конкретный экземпляр в коде.

Константы, массив и структура в ассемблере — наиболее часто используемые организованные виды данных, однако есть и другие. Если посмотреть на структуру повнимательнее, со стороны нашего подхода к программированию, как к совокупности кода и данных, то остальные виды сгруппированных данных : массив, перечисление, объединение, битовые поля — фактически являются разновидностями структуры, реализованные в целях экономии процессорного времени и объёма памяти.

**Массив в ассемблере.**

Массив — структура данных, хранящих значения, которые идентифицируются по индексам, начиная с нулевого индекса. Рассмотрим работу с одномерным массивом — с учётом нашего начального уровня.

* my\_mass\_1 db 10 dup(8); создать байтовый массив, состоящий из 10 байт и заполнить его цифрами 8. Реально : дублировать (DUPlicate) 10 раз число 8.
* mov my\_mass\_1[0],1 ; поместить в первый байт число 1 (поля массива считаются с нуля, а не с единицы).

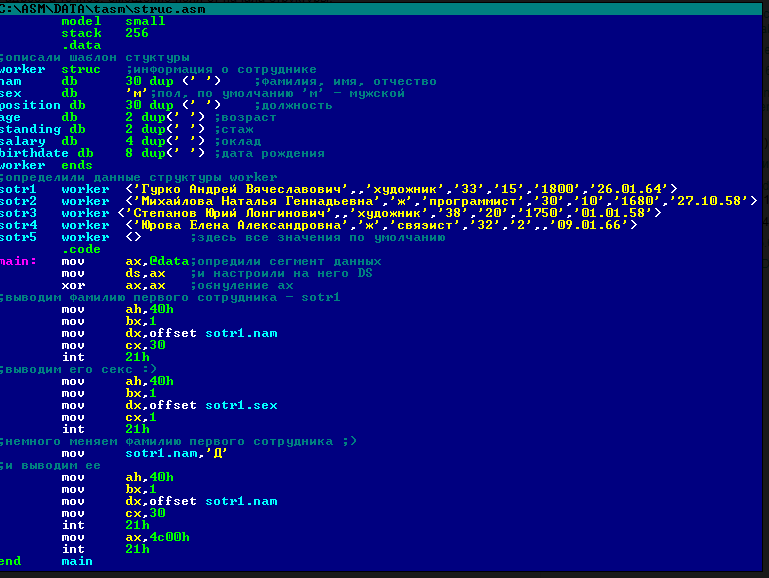
**===**

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:  
  
;описали шаблон стуктуры  
worker struc ;информация о сотруднике  
nam db 30 dup (' ') ;фамилия, имя, отчество  
sex db 'м';пол, по умолчанию 'м' - мужской  
position db 30 dup (' ') ;должность  
age db 2 dup(' ') ;возраст  
standing db 2 dup(' ') ;стаж  
salary db 4 dup(' ') ;оклад   
birthdate db 8 dup(' ') ;дата рождения  
worker ends  
  
**Определение данных с типом структуры**  
  
Для использования описанной с помощью шаблона структуры в программе необходимо определить переменную с типом данной 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.  
  Его задание также необязательно.  
  Если список указан не полностью, то все поля структуры для данной переменной инициализируются значениями из шаблона, если таковые заданы.  
  Допускается инициализация отдельных полей, но в этом случае пропущенные поля должны отделяться запятыми. Пропущенные поля будут инициализированы значениями из шаблона структуры. Если при определении новой переменной с типом данной структуры мы согласны со всеми значениями полей в ее шаблоне (то есть заданными по умолчанию), то нужно просто написать угловые скобки.  
  К примеру: victor worker <>.

Для примера определим несколько переменных с типом описанной выше структуры.  
  
sotr1 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','свяэист','32','2',,'09.01.66'>  
sotr5 worker <> ;здесь все значения по умолчанию  
  
**Методы работы со структурой**  
  
Идея введения структурного типа 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 — символ "." (точка). Он используется в следующей синтаксической конструкции:  
  
адресное\_выражение.имя\_поля\_структуры  
  
Здесь:

* ***адресное\_выражение*** — идентификатор переменной некоторого структурного типа или выражение в скобках указывающее на ее адрес
* ***имя\_поля\_структуры*** — имя поля из шаблона структуры. Это, на самом деле, тоже адрес, а точнее, смещение поля от начала структуры.

****

та программа уже выводит на экран информацию о первом сотруднике. Запускаем, анализируем, просветляемся...

#### Методы работы со структурой

Идея введения структурного типа 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 — символ "**.**" (точка). Он используется в следующей синтаксической конструкции:

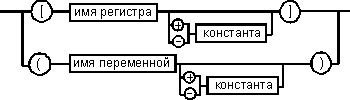
|  |
| --- |
| адресное\_выражение.имя\_поля\_структуры |

Здесь:

* адресное\_выражение — идентификатор переменной некоторого структурного типа или выражение в скобках в соответствии с указанными ниже синтаксическими правилами (рис. 1);
* имя\_поля\_структуры — имя поля из шаблона структуры.
* Это, на самом деле, тоже адрес, а точнее, смещение поля от начала структуры.

Таким образом оператор "**.**" (точка) вычисляет выражение

|  |
| --- |
| (адресное\_выражение) + (имя\_поля\_структуры) |



**Рис. 5.** Синтаксис адресного выражения в операторе обращения к полю структуры

Продемонстрируем на примере определенной нами структуры ***worker*** некоторые приемы работы со структурами.

К примеру, извлечь в **ax** значения поля с возрастом. Так как вряд ли возраст трудоспособного человека будет больше величины 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 al,ah**:

|  |
| --- |
| mov ax,word ptr sotr1.age ;в al возраст sotr1  xchg ah,al |

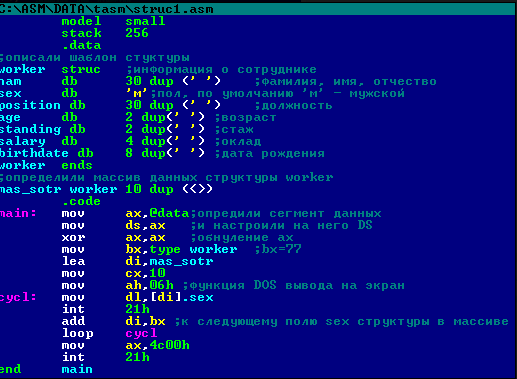
а можно и так:

|  |
| --- |
| lea bx,sotr1  mov ax,word ptr [bx].age  xchg ah,al |

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.

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

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.  
Язык ассемблера 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 этой структуры. Извлечь это значение можно с помощью оператор **type**.  
После того как стал известен размер экземпляра структуры, организовать индексацию в массиве структур не представляет особой сложности.  
К примеру, программа выводит на экран содержимое поля sex всех структур worker в массиве mas\_sotr:

****

**Перечисление в ассемблере.**

Перечисления (enum) представляет собой структуру, состоящую из именованных констант. Создана для удобства программирования в Си. При этом можно просто перечислять константы, компилятор будет присваивать им целые значения в порядке возрастания, начиная с нуля. Если присвоить полю конкретное значение, которое не совпадает с простой последовательностью, отсчёт следующих, не определённых полей будет происходить по алгоритму «+1».  
enum eDirection  
{  
RIGHT, // по умолчанию = 0  
LEFT, // = 1  
DOWN=5, // = 5 — присвоили, если бы не присвоить, то DOWN==2 (предыдущее поле +1)  
UP // = 6 (+1)  
};

В ассемблере перечисление (enum) как отдельный, самостоятельный тип данных не существует. При необходимости используют структуру с целочисленными полями либо просто отдельные целочисленные константы.

* \_ENUM STRUC;В ассемблере Нет ENUM, используем STRUC  
  RIGHT db ?;  
  LEFT db ?;  
  DOWN db ?;  
  UP db ? ;  
  \_ENUM ENDS
* my\_enum \_ENUM <0,1,2,3>

…

* xor ax,ax;ax==0
* mov ah,my\_enum.LEFT

**Объединения в ассемблере.**

Объединение (union) —  одна и та же область памяти, используемая как разные типы данных. Естественно, в таком случае размер объединения будет равен размеру наибольшего из значений и не равна сумме длин всех запоминаемых, как в структуре. Тип данных создавался для Си, как способ экрномии памяти компьютера (сейчас — не актуально, но ранее активно использовался в написании кода, в том числе и сетевого характера, поэтому применяется и сейчас для совместимости).

* MY\_UNION union  
  \_word dw ?  
  \_byte db ?  
  MY\_UNION ends
* \_union MY\_UNION <1234h>

…

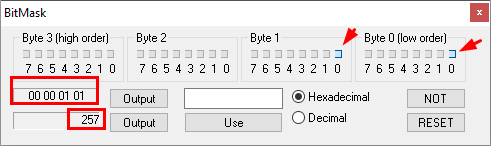
После этого \_union.\_word=1234h, а \_union.\_byte=34h.

* xor ax,ax;ax==0
* mov ah,\_union.\_byte

**Записи с битовыми полями (запись).**

Бит — единица данных, может содержать значение 1 или 0. Записи с битовыми полями (records) используют эту возможность.

Каждое битовое поле имеет заданную длину (в битах) и начальное значение. Размер данных типа записи равен сумме длин всех полей

[](http://assembler-code.com/wp-content/uploads/2020/04/record-asm.png)Запись с битовыми полями (запись) — 32 бита.

Опять таки — удобно, экономит место и вычислительное время. Например, чтобы задать цвет точки в изображении (совокупность различных оттенков красного, зелёного, синего (RGB) или свойств окна в операционке Windows.

Не будем подробно разбирать тему, приведём пример кода.

* BitMask RECORD f0:4=1,f1:4=1,f2:4=0,f3:4=0

…

* xor ax,ax;ax==0
* mov ax,BitMask;ax==257

**Работающий ассемблерный код с примерами.**

Ниже приведён пример кода — GBLSTRUC.COM (вставили дополнительные строки в goblin.com). Прогоните готовый исполняемый файл через дизасемблер и дебагер при желании разобраться в вопросе поглубже.