??????

Типы данных

db	байт
dw	слово
dd	двойное слово
dq	двойное длинное слово

Для строк в конце добавляется завершающий 0 (NULL).

Массив:

```
nums dq 11, 12, 13, 14, 15, 16, 17 ; семь 8-байтных чисел
```

Как и всегда, хранится адрес первого элемента.

times определяет массив одинаковых элементов.

```
numb:  times 10 db 2 ; десять чисел, каждое из которых равно 2, 1-байтные
```

Упрощенный вариант выделения памяти для массива (начальные 0):

resb	выделяет некоторое количество байт
resw	выделяет некоторое количество слов (2-х байтовых чисел)
resd	выделяет некоторое количество двойных слов (4-х байтовых чисел)
resq	выделяет некоторое количество четверных слов (8-х байтовых чисел)

Пример:

```
buffer resb 10
```

Значение переменной получаем при [], по умолчанию адрес переменной. В Windows лексика сложнее.

Структура:

Простое определение - через метку и смещение

```
section .data
; условная структура
person:
    db "Alice",10    ; имя
    dq 34           ; возраст

; смещение компонентов в структуре
NAME_OFFSET equ 0
AGE_OFFSET equ 6
section .text
_start:

    mov rsi, person     ; в RSI - адрес строки
    mov rdi, 1          ; в RDI - дексриптор вывода в стандартный поток (консоль)
    mov rdx, AGE_OFFSET    ; в RDX - длина строки
    mov rax, 1        ; в RAX - номер функции для вывода в поток
    syscall              ; вызываем функцию Linux

    mov rdi, [rsi + AGE_OFFSET]  ; в RDI - возраст
```

Другой способ:

```
struc имя_структуры

    поле_1:      тип_поля_1    размер_поля_1
    поле_2:      тип_поля_2    размер_поля_2
    .....................................
    поле_N:      тип_поля_N   размер_поля_N

endstruc
```

Пример:

```
struc person
    .id:        resd 1  ; 4 байта (d=double word)
    .name:      resb 20 ; 20 байт (b=byte)
    .age:       resw 1  ; 2 байта (w=word)
endstruc
```

```
person.id = 0
person.name = 4 (потому что .id занял 4 баи˘та)
person.age = 24 (потому что .name занял 20 баи˘т после .id)
person_size = 26 (общии˘ размер: 4 + 20 + 2)
```

Т е при использовании struc не нужно самому высчитывать адреса меток.

Создание экземпляра (выделение памяти)
Для хранения данных обычно применяются две секции - .bss (для неинициализированных данных) и .data, то соответственно мы можем создавать инициализированные и неинициализированные экземпляры структуры.

Неинициализированный экземпляр (в секции .bss) Это самый простой способ, использующий метку _size:

```
section .bss
    person1: resb person_size  ; Выделить 26 байт под один экземпляр
    person2: resb person_size  ; Выделить еще 26 байт
```

Инициализированный экземпляр (в секции .data)
Для создания экземпляра с начальными значениями используются макросы ISTRUC, AT и IEND.

```
section .data
    tom:
        istruc person      ; Начало экземпляра структуры person
            at person.id,   dd 101           ; в поле .id число 101
            at person.name, db "Tom", 0  ; в .name строка "Tom", 0
            ; (Оставшиеся байты .name будут неявно заполнены нулями)
            at person.age, dw 2            ; в поле .age число 2
        iend                ; Завершение экземпляра структуры person
```

Стоит отметить, что поля структуры должны быть объявлены в том же порядке, в котором они были указаны в определении структуры.

Доступ к полям структуры осуществляется путем сложения базового адреса экземпляра структуры с меткой-смещением нужного поля:

Пример доступа к полям экземпляра tom (из .data):

```
; Поместить ID в EAX
mov eax, [tom + person.id]   ; EAX = 101
```

```
; Поместить возраст в BX
mov bx, [tom + person.age]  ; BX = 2

; Получить адрес имени (например, для вызова функции)
lea esi, [tom + person.name]
; Теперь ESI указывает на строку "Tom"
```

Преобразование разрядности.

При несоответствии разрядности регистра и памяти желательно точно определять, что делать.

```
...
section .data
nums db 1, 2, 0, 0, 0, 0, 0, 0
...
movzx rax, byte [nums]
```

- byte: преобразует в байт
- word: преобразует в слово
- dword: преобразует в двойное слов
- qword: преобразует в четверное слово

Точка определения данных:

section .text	Должны определяться либо до первой инструкции, либо после последней инструкции. Только константы.
section .data	Наиболее логичная точка размещения. Но все данные в этой секции размещаются в бинарнике и затем копируются в ОП.
section .rodata	Раздел только для чтения. Отличие от констант в том, что занимают память. Константы подставляются во время компиляции. Нельзя сделать массив констант.
section .bss	Логичнее размещать здесь неизвестные сначала данные, resb/... Не занимается память в бинарнике,

Косвенная адресация.

Обращение по некоторому адресу: [base + (index * scale) + offset] Компоненты:

base	базовый регистр, который содержит некоторый адрес. Это может быть 64-разрядный или 32-разрядный регистр общего назначения или регистр RSP
index	индексный регистр, который содержит некоторый индекс относительно адреса в базовом регистре. В качестве индексного регистра также могут выступать 64-разрядный или 32-разрядный регистр общего назначения или регистр RSP
scale	множитель, на который умножается значение индексного регистра. Может принимать значения 1, 2, 4 или 8
offset	может представлять 32-разрядное значение в виде числа или имени переменной. Это может быть 64-разрядный регистр общего назначения или регистр RSP

Revision #3
Created 9 November 2025 16:28:55 by Admin
Updated 9 November 2025 17:30:39 by Admin