17. Структура программы на языке ассемблера. Модули. Сегменты.

Программа на ассемблере может состоять из нескольких файлов - **модулей**. При компиляции (трансляции) каждый модуль превращается в объектный файл. Далее при компоновке объектные файлы объединяются в один исполняемый файл.

Структура программы

1. модули (файлы исходного кода) - если в программе несколько модулей, только один может содержать начальный адрес.
2. сегменты (описание блоков памяти)
3. составляющие программного кода
    1. команды процессора
    2. инструкции описания структуры, выделения памяти, макроопределения
4. формат строки программы - строки имеют следующий вид:

    `метка команда/директива операнды ; комментарий`

Метка может быть любой комбинацией букв английского алфавита, цифр и символов (`_`, `?`, `@`, `$`), но не может начинаться с цифры. Метка может быть пустой.

Поле **команды** может содержать команду процессора, которая транслируется в исполняемый код, или директиву, которая не приводит к появлению кода, а управляет работой самого ассемблера.

> Пример директивы: `END` - означает конец программы, `SEGMENT` - означает начало сегмента. Начинаются с символа `.`

Поле **операндов** содержит требуемые командой или директивой операнды. То есть нельзя указать операнды, но не указать команду или директиву.

Поле **комментария** начинается с символа `;` и продолжается до конца строки. Может быть пустым. Поле комментария не анализируется ассемблером.

Сегменты

Каждая программа, написанная, на любом языке программирования, состоит из одного или нескольких сегментов.

1. **Сегмент кода** - содержит команды процессора, которые выполняются при работе программы.
2. **Сегмент данных** - содержит данные, с которыми работает программа.
3. **Сегмент стека** - содержит данные, используемые для организации стека.

Описание сегмента

Сегмент программы описывается директивой `SEGMENT` и заканчивается директивой `ENDS`. Все пять операндов директивы `SEGMENT` являются необязательными.

```
имя_сегмента segment readonly выравниваение тип разряд 'класс'
...
имя_сегмента ends
```

1. **имя_сегмента** - метка, которая будет использоваться для получения сегментного адреса, а так же для комбинирования сегментов в группы.
2. **readonly** - определяет, что сегмент будет доступен только для чтения. Поддерживает только *(только ли?)* MASM.
3. **выравнивание** - указывает ассемблеру и компоновщику, с какого адреса может начинаться сегмент. Значения этого операнда:
    - `byte` - с любого адреса
    - `word` - с четного адреса
    - `dword` - с адреса, кратного 4
    - `para` - с адреса, кратного 16 (значение по умолчанию)
    - `page` - с адреса, кратного 256
4. **тип** - определяет один из возможных типов комбинирования сегментов:
    - `public` - все такие сегменты с одинаковым именем, но разными классами объединяются в один сегмент
    - `private` - сегмент такого типа не объединяется с другими сегментами
    - `stack` - то же самое, что и `public`, но должен использоваться для сегментов стека, потому что при загрузке программы, стек получается как объединение всех сегментов типа `stack`
    - `common` - сегменты с одинаковым именем также объединяются в один, но не последовательно, а по одному и тому же адресу. Следовательно, длина суммарного сегмента будет равна не сумме длин объединенных сегментов, а длине максимального. Можно формировать оверлейные программы
    - `at` - выражение указывает, что сегмент должен располагаться по фиксированному абсолютному адресу в памяти. Результат выражения, использующегося в качества операнда для `at`, равен этому адресу, деленному на 16. Например, segment at 40h - сегмент начинается по абсолютному адресу 0400h. Такие сегменты обычно содержат только метки, указывающие на области памяти, которые могут понадобиться
5. **класс** - любая метка, взятая в одинарные кавычки. Сегменты одного класса располагаются в памяти последовательно. Если класс не указан, то используется класс `DATA`.

Директива ASSUME

ASSUME *регистр*: *имя сегмента*

- Не является командой
- Нужна для контроля компилятором правильности обращения к переменным

Модель памяти

.model *модель, язык, модификатор*

- TINY - один сегмент на всё
- SMALL - код в одном сегменте, данные и стек - в другом
- COMPACT - допустимо несколько сегментов данных
- MEDIUM - код в нескольких сегментах, данные - в одном
- LARGE, HUGE
- Язык - C, PASCAL, BASIC, SYSCALL, STDCALL. Для связывания с ЯВУ и вызова подпрограмм.
- Модификатор - NEARSTACK/FARSTACK
- Определение модели позволяет использовать сокращённые формы директив определения сегментов.

Конец программы и точка входа

```
...
END start
```

- start - имя метки, объявленной в сегменте кода и указывающее на команду, с которой начнётся исполнение программы.
- Если в программе несколько модулей, только один может содержать начальный адрес.