**ПЛАН ЗАНЯТИЯ**

**Дисциплина:** МДК.01.04 Системное программирование

**Преподаватель:** Галузин А.Б.

**Курс:** 4

**Группа:** П-40

**Специальность:** 09.02.07 Информационные системы и программирование

**Дата:** 23.09.24

**Время проведения:** 1 пара

**Тема:** Директивы сегментации.

**Цель занятия:**

**дидактическая:** изучить директивы сегментации ассемблера

**развивающая**: развивать абстрактное мышление, логику

**Вид занятия** лекция

**Литература**

Юров В.И. Assembler. Учебник для вузов. - 2-ое изд. – СПб.: Питер, 2003, стр. 103.

**Интернет-ресурсы:**

https://mf.grsu.by/UchProc/livak/b\_osnovy/oal\_5.htm

**ЗАДАНИЕ**: законспектировать лекцию с учетом контрольных вопросов.

**КОНСПЕКТ ЛЕКЦИИ**

План

1. Сегменты и директивы сегментации.
2. Стандартные директивы сегментации.
3. Упрощенные директивы определения сегмента**.**

**1. Сегменты и директивы сегментации**

Физически сегмент представляет собой область памяти, занятую командами и (или) данными, адреса которых вычисляются относительно значения в соответствующем сегментном регистре.

Каждая программа содержит 3 типа сегментов:

* *Сегмент кода* – содержит машинные команды для выполнения. Обычно первая выполняемая команда находится в начале этого сегмента, и операционная система передает управление по адресу данного сегмента для выполнения программы. Регистр сегмента кодов (CS) адресует данный сегмент.
* *Сегмент данных* – содержит данные, константы и рабочие области, необходимые программе. Регистр сегмента данных (DS) адресует данный сегмент.
* *Сегмент стека* — содержит адреса возврата как для программы (для возврата в операционную систему), так и для вызовов подпрограмм (для возврата в главную программу), а также используется для передачи параметров в процедуры. Регистр сегмента стека (SS) адресует данный сегмент. Адрес текущей вершины стека задается регистрами SS:ESP.

Функциональное назначение сегмента несколько шире, чем простое разбиение программы на блоки кода, данных и стека. Сегментация является частью более общего механизма, связанного с концепцией модульного программирования. Она предполагает унификацию оформления объектных модулей, создаваемых компилятором, в том числе с разных языков программирования. Это позволяет объединять программы, написанные на разных языках. Именно для реализации различных вариантов такого объединения и предназначены директивы сегментации.

*Вывод*:

- программа на языке Assembler в соответствии с особенностями архитектуры компьютера (микропроцессора) состоит из сегментов;

- физически сегмент представляет собой область (блок) памяти, занятую командами и/или данными;

- адреса сегментов (адрес начала сегмента) хранятся в соответствующих сегментных регистрах;

- адреса команд/данных вычисляются относительно начала сегмента.

Микропроцессор имеет шесть сегментных регистров, посредством которых может одновременно работать:

* с одним сегментом кода;
* с одним сегментом стека;
* с одним сегментом данных;
* с тремя дополнительными сегментами данных.

Для описания сегментов предназначены директивы сегментации.

Директивы сегментации подразделяются на

1) стандартные (поддерживаются всеми трансляторами Assembler);

2) упрощенные (поддерживаются транслятором TASM).

**2. Стандартные директивы сегментации**

**Директива SEGMENT**

Стандартно сегменты на языке Assembler описываются с помощью директивы SEGMENT.

Синтаксическое описание сегмента представляет собой следующую конструкцию

имя SEGMENT [выравнивание] [комбинирование] [класс] [размер]

<тело сегмента>

имя ENDS

Рассмотрим основные параметры директивы SEGMENT.

***Атрибут выравнивания сегмента*** (тип выравнивания) сообщает компоновщикуу о том, что нужно обеспечить размещение начала сегмента на заданной границе. По умолчанию тип выравнивания имеет значение PARA:

* BYTE—выравнивание не выполняется. Сегмент может начинаться с любого адреса памяти;
* WORD — сегмент начинается по адресу, кратному двум, то есть последний (младший) значимый бит физического адреса равен 0 (выравнивание по границе слова);
* DWORD — сегмент начинается по адресу, кратному четырем, то есть две последних (младших) значимых бита равны 0 (выравнивание по границе двойного слова);
* PARA — сегмент начинается по адресу, кратному 16, то есть последняя шестнадцатиричная цифра адреса должна быть 0h (выравнивание по границе параграфа);
* PAGE — сегмент начинается по адресу, кратному 256, то есть две последних шестнадцатиричных цифры должны быть 00h (выравнивание по границе страницы);
* MEMPAGE — сегмент начинается по адресу, кратным 4 Кбайт, то есть три последних шестнадцатиричных цифры должны быть 000h.

**Атрибут комбинирования сегментов** (комбинаторный тип) сообщает компоновщикуу, как нужно комбинировать сегменты разных модулей, которые имеют одно и то же имя. По умолчанию атрибут комбинирования принимает значение PRIVATE. Возможные значения атрибута комбинирования сегмента:

* PRIVATE — сегмент не будет объединяться с другими сегментами с тем же именем вне данного модуля;
* PUBLIC — принуждает компоновщик объединить все сегменты с одинаковым именем. Новый объединенный сегмент будет целым и непрерывным. Все адреса объектов будут вычисляться относительно начала этого нового сегмента;
* COMMON — располагает все сегменты с одним и тем же именем по одному адресу, то есть все сегменты с данным именем перекрываются. Размер полученного в результате сегмента будет равен размеру наибольшего сегмента;
* AT xxxx — располагает сегмент по абсолютному адресу параграфа (параграф — область памяти, объем которой кратный 16, последняя шестнадцатиричная цифра адреса параграфа 0h). Абсолютный адрес параграфа задается выражением ххх . Компоновщик располагает сегмент по заданному адресу памяти, учитывая атрибут комбинирования. Все метки и адреса в сегменте отсчитываются относительно заданного абсолютного адреса;
* STACK — определение сегмента стека. Принуждает компоновщик объединить все одноименные сегменты и вычислять адреса в этих сегментах относительно регистра SS. (аналогично типу PUBLIC за исключением того, что регистр SS является стандартным сегментным регистром для сегментов стека). Регистр SP устанавливается на конец объединенного сегмента стека.

**Атрибут класса сегмента** (тип класса) — заключенная в кавычки строка, которая помогает компоновщику определить нужный порядок прохождения сегментов при сборке программы из сегментов нескольких модулей, объединяет вместе все сегменты с одним и тем же именем класса.

**Атрибут размера сегмента**. Для процессоров i80386 и выше сегменты могут быть 16- или 32-разрядными. Это влияет на размер сегмента и порядок формирования физического адреса внутри него. Возможные значения атрибута:

* USE16 — сегмент допускает 16-разрядную адресацию. При формировании физического адреса может использоваться только 16-разрядное смещение. Такой сегмент может содержать до 64 Кбайт кода или данных;
* USE32 — сегмент должен быть 32-разрядным. При формировании физического адреса может использоваться 32- разрядное смещение. Такой сегмент может содержать до 4 Гбайт кода или данных.

**Директива ASSUME**

С помощью директивы ASSUME можно сообщить транслятору, какой сегмент, к какому сегментному регистру «привязан», или, говоря боле точно, в каком сегментном регистре хранится адрес сегмента.

Формат директивы ASSUME:

ASSUME <сегментный регистр>:<имя сегмента>

Директивы SEGMENT и ASSUME - стандартные директивы сегментации.

Для простых программ, которые содержат по одному сегменту кода, данных и стека легче использовать упрощенные описания соответствующих сегментов.

Трансляторы MASM и TASM предоставляют возможность использования упрощенных директив сегментации (вместо SEGMENT).

*Замечание*

* Стандартные и упрощенные директивы сегментации не исключают друг друга.
* Стандартные директивы используются, когда программист хочет получить полный контроль над размещением сегментов в памяти и их комбинированием с сегментами других модулей.
* Упрощенные директивы целесообразно использовать

1) для простых программ

2) программ, предназначенных для связывания с программными модулями, написанными на языках высокого уровня (это позволяет компоновщику эффективно связывать модули разных языков за счет стандартизации связей и управления).

**3. Упрощенные директивы определения сегмента**

Рассмотрим упрощенные директивы определения сегмента (для режима MASM):

*.CODE [имя]*

Директива предназначена для определения начала или продолжения сегмента кода. Возможно определение нескольких сегментов данного типа.

.DATA

Директива предназначена для определения начала или продолжения сегмента инициализированных данных. Также используется для определения данных типа near.

.STACK [размер]

Директива предназначена для определения начала или продолжения сегмента стека. Параметр [размер] задает размер стека.

.CONST

Директива предназначена для определения начала или продолжения сегмента постоянных данных (констант).

.DATA?

Директива предназначена для определения начала или продолжения сегмента неинициализированных данных. Также используется для определения данных типа near

.FARDATA [имя]

Директива предназначена для определения начала или продолжения сегмента инициализированных данных типа far. Возможно определение нескольких сегментов данного типа.

.FARDATA? [имя]

Директива предназначена для определения начала или продолжения сегмента неинициализированных данных типа far. Возможно определение нескольких сегментов данного типа.

Совместно с упрощенными директивами сегментации используется директива указания модели памяти *MODEL*, которая частично управляет размещением сегментов и выполняет функции директивы *ASSUME*, т.е. связывает сегменты с сегментными регистрами (поэтому при использовании упрощенных директив сегментации директиву ASSUME можно не использовать).

Упрощенный формат директивы MODEL

*MODEL* [<модификатор>] <модель памяти> [др. параметры]

Обязательным параметром директивы MODEL является «модель памяти». Этот параметр определяет модель сегментации памяти для программного модуля. Возможные значения параметра «модель памяти»:

*TINY* - Код и данные объединены в одну группу с именем DGROUP. Используется для создания программ формата com.

*SMALL* - Код занимает один сегмент, данные объединены в одну группу с именем DGROUP. Эту модель обычно используют для большинства программ на языке Assembler.

*MEDIUM* - Код занимает несколько сегментов, по одному на каждый объединяемый программный модуль. Все ссылки на передачу управления типа far. Данные объединены в одной группе, все ссылки на них типа near.

*COMPACT* - Код в одном сегменте; ссылка на данные типа far.

*LARGE* - Код в нескольких сегментах. Каждый объединяемый в одну программу модуль хранится в отдельном сегменте кода.

Параметр «модификатор» директивы MODEL уточняет особенности использования выбранной модели памяти.

Возможные значения параметра «модификатор модели памяти»:

use16 - сегменты выбранной модели используются как 16-битные

use32 - сегменты выбранной модели используются как 32-битные

dos - программа предназначена для работы в ОС MS-DOS

Другие параметры используются при написании программ на разных языках программирования.

Пример.

Для большинства программ на ассемблере используют директиву

MODEL SMALL

Перечислим идентификаторы, создаваемые директивой MODEL:

@code - физический адрес сегмента кода

@data - физический адрес сегмента данных типа near

@fardata - физический адрес сегмента данных типа far

@fardata? - физический адрес сегмента неинициализированных данных типа far

@curseg - имя текущего сегмента

@stack - физический адрес сегмента стека

**Контрольные вопросы**

1. Что собой представляет сегмент в программах на ассемблере?
2. Какие типы сегментов содержит программа на ассемблере? Опишите их.
3. Где хранятся адреса сегментов?
4. На какие две группы делятся директивы сегментации? В чем их отличие?
5. Какие директивы относятся к стандартным?
6. Опишите синтаксис директивы SEGMENT с подробным описанием параметров.
7. Какие директивы относятся к упрощенным? Опишите их назначение и синтаксис.