Лекция 5

1.5 Особенности формирования машинных команд

1.5.1. Формат машинной команды

По мнемонике команды транслятор создает машинную команду. Она может занимать от 1 до 6 байт в зависимости от того, какие режимы адресации применяются. Смещение и непосредственные данные также записываются в команду. Если они в границах -128 +127, то занимают байт, иначе - слово.

Кратчайшие команды - в один байт, для тех команд, для которых операнд определяется самой командой (команды для работы с битами регистра флагов или **CLC** - очистить бит флага CF).

Самые длинные команды возникают тогда, когда в них используется 16-битовое смещение (DISP) или 16-битовые непосредственные данные (DATA).

Поэтому для разных команд и форматы команд будут разными, то есть какие коды и где они размещаются.

Рассмотрим формат двухоперандной команды. В первом байте записывается код операции, во втором - режимы адресации. Остальные байты - под смещение (DISP) или непосредственные данные (DATA).

КОП	**S**	**W**

MOD	**REG**	**R/M**

В первом байте, кроме кода операции, есть еще 2 однобитовых индикатора: **W** - определяет, над какой единицей данных выполняется команда. **W** = 1 - над словом, **W** = 0 - над байтом.

Бит **S** показывает, чем является тот регистр - операндом-источником (**S** = 0) или приемником (**S** = 1).

Регистры кодируются таким образом:

КОП \ **W**	**W = 1**	**W = 0**
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

Сегментные регистры кодируются так:

00	DS
01	CS
10	SS
11	ES

Режим адресации второго операнда определяется кодами в полях **MOD** и **R/M** (register-memory).

Поле **MOD** определяет, что именно закодировано в поле **R/M**. Если **MOD** = 00 -- смещения НЕТ; **MOD** = 01 - смещение БАЙТ, **MOD** = 10 - смещение СЛОВО. Когда же **MOD** = 11, то значат, что в поле R/M записан код второго регистра.

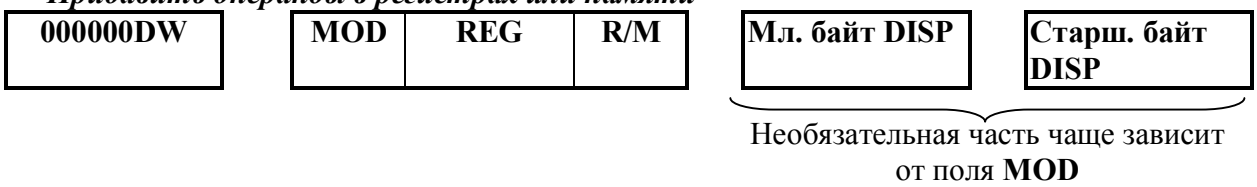
Таблица

Коды режимов адресации

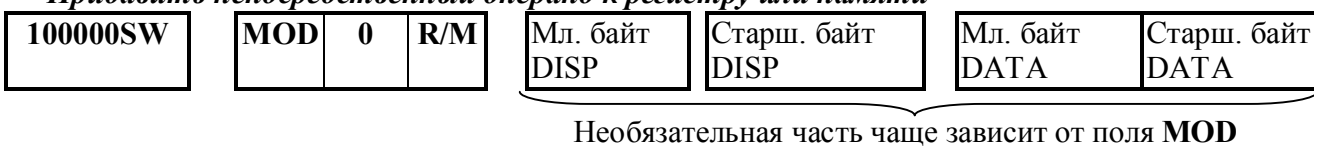
R/M \ MOD	00	01	10
000	[BX] + [SI]	[BX + SI] + DISP 8	[BX + SI] + DISP 16
001	[BX] + [DI]	[BX + DI] + DISP 8	[BX + DI] + DISP 16
010	[BP + SI]	[BP + SI] + DISP 8	[BP + SI] + DISP 16
011	[BP + DI]	[BP + DI] + DISP 8	[BP + DI] + DISP 16
100	[SI]	[SI] + DISP 8	[SI] + DISP 16
101	[DI]	[DI] + DISP 8	[DI] + DISP 16
110	disp 16	[BP] + DISP 8	[BP] + DISP 16
111	[BX]	[BX] + DISP 8	[BX] + DISP 16

Особенностью МП является то, что для одной команды может быть несколько кодов и форматов в зависимости от режима адресации или операндов, которые там используются. Например, для команды **ADD** (сложить):

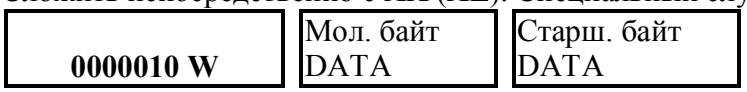
Прибавить операнды в регистрах или памяти



Прибавить непосредственный операнд к регистру или памяти



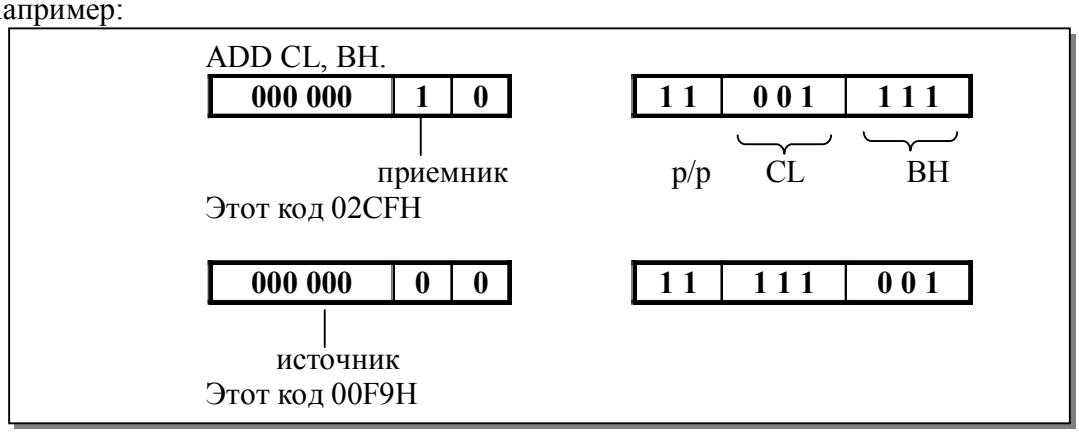
Сложить непосредственно с **AX** (**AL**). Специальный случай для аккумулятора:



То есть, команда **ADD** имеет 3 варианта кода и разные форматы. Как видно, в формате команды существует лишь одно поле для кодировки ячейки памяти. Поэтому в двухадресных командах может быть лишь один операнд - ячейка памяти, а не два. Следовательно, сложить содержание одной ячейки ко второй за одну команду нельзя.

Иногда одна и та же команда может быть закодирована транслятором по-разному.

Например:



Поля из 6 битов для кодировки всех команд недостаточно. Поэтому некоторые команды объединяются в группу и в первом байте кодируется группа команд, а команда образуется во втором байте. Например, форматы команд с непосредственной адресацией:

регистр - непосредственный операнд

КОП	S	W

1 1	КОП	REG

необязательная часть

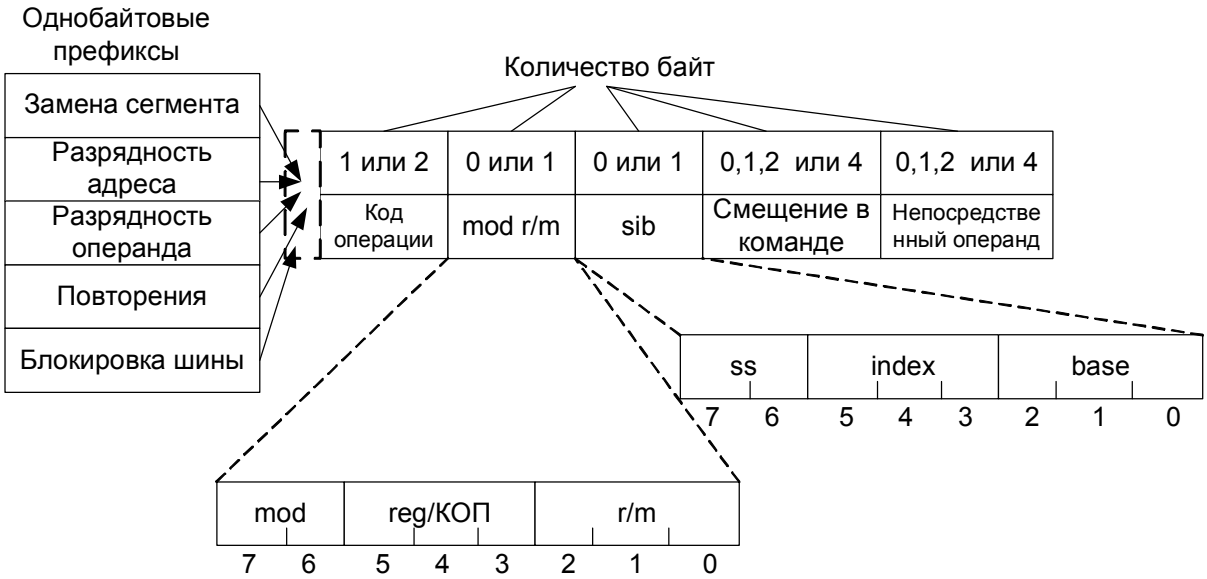
память - непосредственный операнд

КОП	S	W

1 1	КОП	MEM

Кроме отмеченных частей в команде может быть несколько префиксов, что может увеличить размер до 15 байт (рис. 1.5.1).

Формат машинной команды



Замена сегмента:
2eh — замена сегмента cs;
36h — замена сегмента ss;
3eh — замена сегмента ds;
26h — замена сегмента es;
64h — замена сегмента fs;
65h — замена сегмента gs

Разрядность адреса:
уточняет разрядность адреса (32 или 16-разрядный)

Разрядность операнда:
66h — 16-разрядный;
67h — 32-разрядный

Префикс повторения:
- ***безусловные*** (rep — 0f3h), заставляющие повторяться цепочечную команду некоторое количество раз;
- ***условные*** (repe/repz — 0f3h, repne/repnz — 0f2h), которые при зацикливании проверяют некоторые флаги, и в результате проверки возможен досрочный выход из цикла

Код операции (КОП)

Обязательный элемент, описывающий операцию, выполняемую командой. Многим командам соответствует несколько кодов операций, каждый из которых определяет нюансы выполнения операции.

Байт modr/m - режим адресации

Значения этого байта определяет используемую форму адреса операндов:
mod = 00 - поле смещение в команде отсутствует
mod = 01 - поле смещение в команде присутствует
mod = 11 - операндов в памяти нет: они находятся в регистрах
reg/КОП - определяет либо *регистр*, находящийся в команде на месте первого операнда, либо возможное *расширение кода операции*
r/m *используется совместно с полем mod и определяет либо регистр, либо базовые и индексные регистры*

Байт масштаб-индекс-база (байт sib)

расширяет возможности адресации операндов:
ss – в нем размещается масштабный множитель для индексного компонента index
index — используется для хранения номера индексного регистра
base — используется для хранения номера базового регистра

Поле смещения в команде

8, 16 или 32-разрядное целое число со знаком, представляющее собой, полностью или частично, значение эффективного адреса операнда

Поле непосредственного операнда

Необязательное поле, представляющее собой 8, 16 или 32-разрядный непосредственный операнд. Наличие этого поля, отражается на значении байта mod r/m.

Рис. 1.5.1. Формат машинной команды

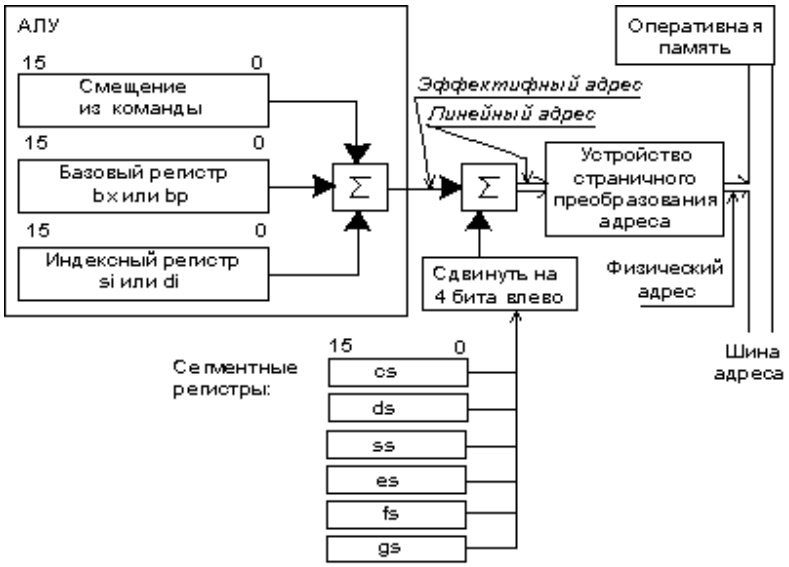


Рис. 1.5.2. Механизм формирования физического адреса в реальном времени

1.5.2. Операторы

Программа на языке Ассемблер состоит из отдельных строк-операторов, которые описывают выполняемые операции. Оператором может быть *команда*, или *псевдооператор* (директива).

Команда - это условно (символьное) обозначение машинных команд. То есть, она отмечает ЭВМ что ей нужно выполнить. Команды после трансляции превращаются в машинные коды.

В отличие от команды, *псевдооператоры* или *директивы* имеют вспомогательный характер. Они сообщают транслятору что сделать с командами. Как правило, они не превращаются в машинные команды. Команды выполняются во время вычислений, а псевдооператоры - во время трансляции.

Каждая команда может иметь 4 поля:

[Метка:] Мнемокод [Операнд] [, Операнд] [; Комментарий]

Из них обязательным является лишь поле мнемокода, все другие частично или полностью могут отсутствовать.

Например:

```
COUNT: MOV AX,DI; переслать DI в аккумулятор
```

Метка присваивает имя команде. На эту метку можно передавать управление с других мест программы, то есть, на нее могут ссылаться другие команды.

Метка может складываться не больше как из 31 символа и заканчивается ":". Сюда могут входить большие или малые латинские буквы от A к Z, или от a к z (большие и малые буквы не различаются), цифры от 0 до 9 и специальные знаки ? . @ _ $. Метка может начинаться любым символом, кроме цифры. Знаки (пропуски) препинаний нельзя включить, за исключением подчеркиваний.

Поле мнемокода - содержит мнемокод или мнемонику команды от 3 до 6 символов.

Операнды, как отмечалось, имеют свое название: *приемник и источник*. После выполнения команды изменяется приемник, а источник остается неизменным. Операнды отделяются запятой.

Комментарии - все, что стоит справа от «**;**». Могут быть в строке, а также занимать целую строку. Поля отделяются пропуском.

Псевдооператоры - руководят работой транслятора, а не микропроцессора. С их помощью определяют сегменты и процедуры (подпрограммы), дают имена командам и элементам данных, резервируют рабочие места в памяти.

Псевдооператоры также могут состоять из 4-х полей:

[Идентификатор] Псевдооператор [операнд] [; коментарий]

Поля в скобках могут отсутствовать. В Макроассемблере насчитывается до 60 псевдооператоров. Рассмотрим наиболее распространенные.

1.5.3. Псевдооператоры данных

Их можно разделить на **5** групп:

1. Определение идентификаторов

Позволяют присвоить символическое имя выражения, константе, адресу, другому символическому имени. После этого можно использовать это имя.

Напрмер:

```
K          EQU   1024; константа
TABLE      EQU   DS: [BP] [SI]; адрес
SPEED      EQU   RATE ; синоним
COUNT      EQU   CX; имя регистра
```

Операндом в этом псевдооператоре в общем случае может быть выражение:

```
DBL  SPEED EQU 2*SPEED
```

То есть, в выражение могут входить некоторые более простые арифметические и логические действия. Если в выражение входит константа, то по умолчанию она считается десятичной 256.

Шестнадцатиричные константы - справа буква H (2FH). Когда такая константа начинается с буквы, то слева нужно ставить 0 - 0FH, а не FH;

Восьмиричные константы - с буквой Q: 256Q;

Двоичные - с буквой B - 01101B.

Ясно, что в отличие от языков высокого уровня, здесь выражение выполняется во время трансляции транслятором.

Кроме псевдооператора EQU можно употреблять “ = ”.

```
CONST = 59;
CONST = 98;
CONST = CONST + 2.
```

Такое имя может задавать лишь числовую константу, а не символьную и какую-то другую. Это имя можно переопределять, а определенныюо через **EQU** - НЕЛЬЗЯ.

2. Определение данных

Когда ячейка используется для хранения данных, ей можно присвоить имя с помощью псевдооператоров **DB**, **DW**, **DD** (Define Byte, Word, Double Word).

Общая структура:

[имя] псевдооператор выражение [,]

[,] - один или несколько выражений через запятую.

```
MAX DB 57
WU_MAX DW 5692
A_TABLE DW 25 -592, 643, 954 - массив
```

Когда значение переменной предварительно неизвестно, а будет использовано для результатов, то в поле выражения нужно поставить «**?**».

```
LAMBDA DW ?
```

Для текста можно использовать псевдооператор **DB**.

```
POLITE DB ‘Введите данные опять’.
```

Если записываются одинаковые данные, то можно использовать операцию **DUP** (duplicate) - повторить.

```
BETA DW 15 DUP(0)     или     GAMA DW 3 DUP (4DUP(0))
```

Ясно, что эту же операцию можно применить для резервирования памяти:

```
ALPHA DW 20 DUP (?)
```

Переменные можно использовать не только для хранения данных, но и адресов. Если какая-то переменная имеет название THERE, тогда псевдооператоры

```
        THERE DB 123
……….……….……….……
NEAR_THERE DW THERE
NEAR_THERE DD THERE
```

Под именем NEAR_THERE записывают 16-битовый адрес THERE, то есть, ее смещение, а под именем FAR_THERE - 32-битовый адрес THERE, то есть, сегмент + смещение.

3. Псевдооператоры определения сегмента и процедуры

Как отмечалось, программа может состоять из нескольких сегментов: кода, данных, стека, дополнительного сегмента.

Для раздела программы на сегменты используются псевдооператоры **SEGMENT** и **ENDS**. Их структура:

Имя SEGMENT [атрибуты]

……

Имя ENDS

Например:

```
DATASEG SEGMENT
      A DW 500
      B DW -258
  SQUARE DB 54, 61, 95, 17
DATASEG ENDS
```

В сегменте данных определяются имена данных и резервируется память для результатов.

У псевдооператора **SEGMENT** могут быть 3 атрибута:

Имя SEGMENT [выравнивание] [объединение] [класс] [размер сегмента]

Выравнивания определяет, из каких адресов можно размещать сегмент. Часто это есть атрибут **PARA**, который значит, что сегмент нужно разместить из границы параграфа, то есть, начальный адрес кратен 16. По умолчанию принимается PARA.

Может быть PAGE, PARA, WORD, BYTE.

Объединения определяет способ обработки сегмента при компоновке.

PRIVATE - по умолчанию, сегмент должен быть отделен от других сегментов.

PUBLIC - все сегменты с одинаковым именем и классом загружаются в смежной области. Все они будут иметь *один начальный адрес*.

STACK - для компоновника это аналогично PUBLIC. В любой программе должен быть один сегмент с атрибутом STACK.

Класс - этот атрибут может иметь любое правильное имя, ограниченное апострофами. Атрибут используется компоновщиком для обработки сегментов с одинаковыми именами и классами. Часто используются имена ‘CODE’ и ’STACK’.

```
STACK SEG SEGMENT PARA STACK ‘STACK’
    MAS DW 20 DUP (?)
STACK SEG ENDS.
```

Как отмечалось, процессор использует регистр **CS** для адресации сегмента кода, **SS** - сегмента стека, **DS** - данных, **ES** - дополнительный.

Поскольку транслятор не понимает текста, то ему нужно сообщить назначение каждого сегмента. Для этого существует псевдооператор **ASSUME**, который имеет вид:

ASSUME SS:имя стека, DS:имя данных, CS:имя кода.

Например:

```
ASSUME SS:STACK SEG, DS:DATA SEG, CS:CODE SEG.
```

Если какой-то сегмент не используется, то его можно пропустить, или записать:

DS:NOTHING

Эта директива лишь сообщает *транслятору* что с чем связать. Но не загружает соответствующие адреса в регистры.Это должен сделать сам программист в программе.

Директива реализуется в сегменте кодов вначале.

Сегмент кодов может включать одну или несколько процедур. Отдельная процедура начинается псевдооператором PROC:

имя PROC [атрибут]

имя ENDP и заканчивается

```
имя ENDP
```

Если процедура предусматривает возвращение к точке вызова, то перед **ENDP** должна стоять директива **RET** (Return From Procedure). Тогда процедура становится подпрограммой.

```
CALC PROC
.......
        RET
CALC ENDP
```

Атрибутом процедуры может быть **NEAR** (близкий) и **FAR** (далекий).

NEAR-процедура может быть вызвана только из этого сегмента.

Процедуру с атрибутом **FAR** можно вызывать из любого сегмента команд. Основная процедура должна иметь атрибут FAR.

1.5.4. Программные сегменты

В программе может быть несколько сегментов с одинаковым именем. Считается, что это - один сегмент, который по каким-то причинам записан частями.

Параметры директивы **SEGMENT** нужны для больших программ, которая состоит из нескольких файлов. Для небольшой программы, которая составляет один файл, эти параметры ненужны, кроме некоторых случаев.

Пусть, имеем такой ряд сегментов:

```
A SEGMENT
   A1 DB 400h DUP(?)
   A2 DW 8
A ENDS
;
B SEGMENT
   B1 DW A2
   B2 DD A2
B ENDS
;
C SEGMENT
ASSUME ES:A, DS:B, CS:C
L: MOV AX, A2
   MOV BX, B2
……….
C ENDS
```

Размещаются сегменты на границе параграфа, то есть, из адреса, кратного 16. Если А размещено 1000h, то он займет место к 1402h. Следующий адрес - 1403h не кратная 16, потому сегмент В разместится из адреса 1410h. Под сегмент В будет отведено 6 байт, а сегмент С разместится из адреса 1420h.

Значение имени сегмента

Значением имени сегмента является номер, соответствующий сегменту памяти, то есть, первые 16 битов начального адреса данного сегмента. То есть, **А** будет иметь значение 1000h, **В** - 1410h, **С** - 1420h. Храня в тексте имя сегмента, ассемблер будет замещать его на соответствующую величину. Например:

```
MOV BX, B    отвечает    MOV BX, 1410h
```

Сравните MOV BX, A2.

Следовательно, в языке ассемблер имена сегментов являются константными выражениями, а не адресными. Поэтому команда запишет к BX адрес 1410h, а не содержание слова по этому адресу.

Начальная загрузка сегментных регистров

Директива **ASSUME** отмечает, с какими сегментными регистрами нужно связать сегментные регистры. Регистры **DS** и **ES** должен загрузить начальными адресами сам программист.

Пусть, регистр **DS** нужно установить на начало сегмента **B**. Поскольку имя сегмента является константой, то непосредственно в **DS** ее переслать нельзя, а нужно через регистр общего назначения:

```
MOV AX, B
MOV DS, AX
```

Аналогично загружается и регистр **ES**.

Регистр **CS** загружать не нужно. Это выполнит сама операционная система перед тем, как передавать управление программе. Относительно регистра стека **SS** существует две возможности:

- во-первых, это можно сделать так, как и для регистров **DS** и **ES**. Но здесь нужно записать начальный адрес в регистр **SS**, а в указатель стека **SP** – количество байт под стек
- во-вторых - это можно поручить операционной системе. Для этого в соответствующей директиве **SEGMENT** нужно отметить атрибут **STACK**.

1.6. Структура программы

Взаимное расположение сегментов программы может быть произвольным. Если сегменты данных расположены после сегмента кода, то в сегменте кода есть ссылка по адресам сегмента данных. То есть, сразу нельзя определить смещение этих адресов. Следовательно, будем иметь ссылку вперед.

Потому для уменьшения количества ссылок вперед рекомендуется сегменты данных и стека размещать перед сегментом кода.

Относительно сегмента стека **SS**, то если даже программа и не использует стек, создать такой сегмент в программе нужно. Потому что стек программы используется операционной системой при обработке прерываний, например, при нажатии клавиш. Рекомендовано размер стека -- 128 байт. Поэтому, программа на языке ассемблер имеет такую структуру:

```
Title EXAMPLE   ;заголовок
; сегмент данных
dat segment
mas dw 1-3,56,91
res dw 10 dup(?)
dat ends
; сегмент стека
st segment stack ‘stack’
dw 128 dup(?)
st ends
; сегмент кода
cod segment ‘code’
      ASSUME DS:dat, SS:st, CS:cod
beg proc far
      <операторы>
     ret
beg endp
cod ends
      end beg
```

В общем случае в сегменте данных можно размещать и команды, а в сегменте кода - данные. Но лучше этого не делать, потому что возникнут проблемы с сегментацией.

END - конец программы. Если программа - из одного файла, то в этой строке добавляется имя начального выполняемого адреса - beg. Если из нескольких файлов, то только в одной программе отмечается этот адрес. В других - лишь end.

1.6.1. Упрощено описание сегментов

Есть два транслятора из ассемблера - **MASM** фирмы Microsoft, и **TASM** фирмы Borland. Последний может работать в режиме MASM и IDEAL.

Для простых программ, которые состоят из одного сегмента данных, одного сегмента стека, одного сегмента кода есть возможность упростить директивы описания сегмента. Для этого сначала наводится директива **masm** или **ideal** режима работы транслятора **TASM**. Дальше отмечается модель памяти **Model small**; какая частично выполняет функции директивы **ASSUME**.

Например:

```
Masm ; режим работы транслятора TASM
model small ; модель памяти
.data ; заглавие сегмента данных
mas dw 10 DUP (?)
.stack ; заглавие сегмента стека
db 256 dup (?)
.code ; заглавие сегмента кода
main proc
      mov ax @data ; адрес сегмента стека к ax
      mov ds, ax
; текст программы
      mov ax, 4c00h    ; то же, что и RET
       int 21h
main endp
end main
```

Таблица

Упрощенные директивы определения сегмента

Режим MASM	Режим IDEAL	Описание
.code [имя]	Codeseg[имя]	Начало или продолжение сегмента кода
.data	Dataseg	Начало или продолжение сегмента иниц-х данных
.const	Const	Начало или продолжение сегмента констант
.data?	Udataseg	Начало или продолжение сегмента неиниц-х данных
.stack [размер]	Stack[размер]	Начало сегмента стека
.fardata[имя]	Fardata[имя]	Инициализиров-е данные типа FAR
.fardata?[имя]	Ufardata[имя]	Неинициализиров-е данные типа FAR

Идентификаторы, которые создает директива **MODEL**:

@ code -- физический адрес (смещение) сегмента кода
@ data - физический адрес (смещение) сегмента данных
@ fardata -- физический адрес (смещение) сегмента данных типа far
@ fardata? -- физический адрес (смещение) сегмента неинициализированных данных far
@ curseg -- физический адрес (смещение) сегмента неинициализированных данных типа far
@ stack -- физический адрес (смещение) сегмента стека

Модели памяти

Модель	Тип кода	Тип данных	Назначение модели
TINY	Near	Near	Код и данные в одной группе DGROUP для создания .com-программ
SMALL	Near	Near	1 сегмент кода. Данные в одну группу DGROUP
MEDIUM	Far	Near	Код занимает ***n*** сегментов, по одному в каждом модуле. Все передачи управления типа **far**. Данные в одной группе; все ссылки на них – типа **near**
COMPACT	Near	Far	Код в 1 сегменте; ссылка на на данные типа **far**
LARGE	Far	Far	Код в ***n*** сегментах, по одному на каждый объединенный модуль

Модификатор директивы **MOD**el позволяет определить некоторые особенности выбраной модели памяти:

Use 16 - 16-битовые сегменты

Use 32 - 32-битовые сегменты

DOS - программа в MS DOS

Полное и упрощенный описание не исключают друг друга. Но в полном больше возможностей.