Программирование на Ассемблер
к.т.н., доц. Красов А.В.
Лекция 2

Факультет	МТС
Курс	3
Семестр	6
Форма контроля	зачет
Лекции	14 часов
Лабораторные работы	12 часов

Автор курса

к.т.н., доцент Красов Андрей Владимирович
директор УИЦ ИТТ, доцент кафедры ИБТС
Куратор специальности
201800 «Защищенные системы связи»

Синтаксис ассемблера

На рис. 2.1 представлены синтаксические диаграммы ассемблера.

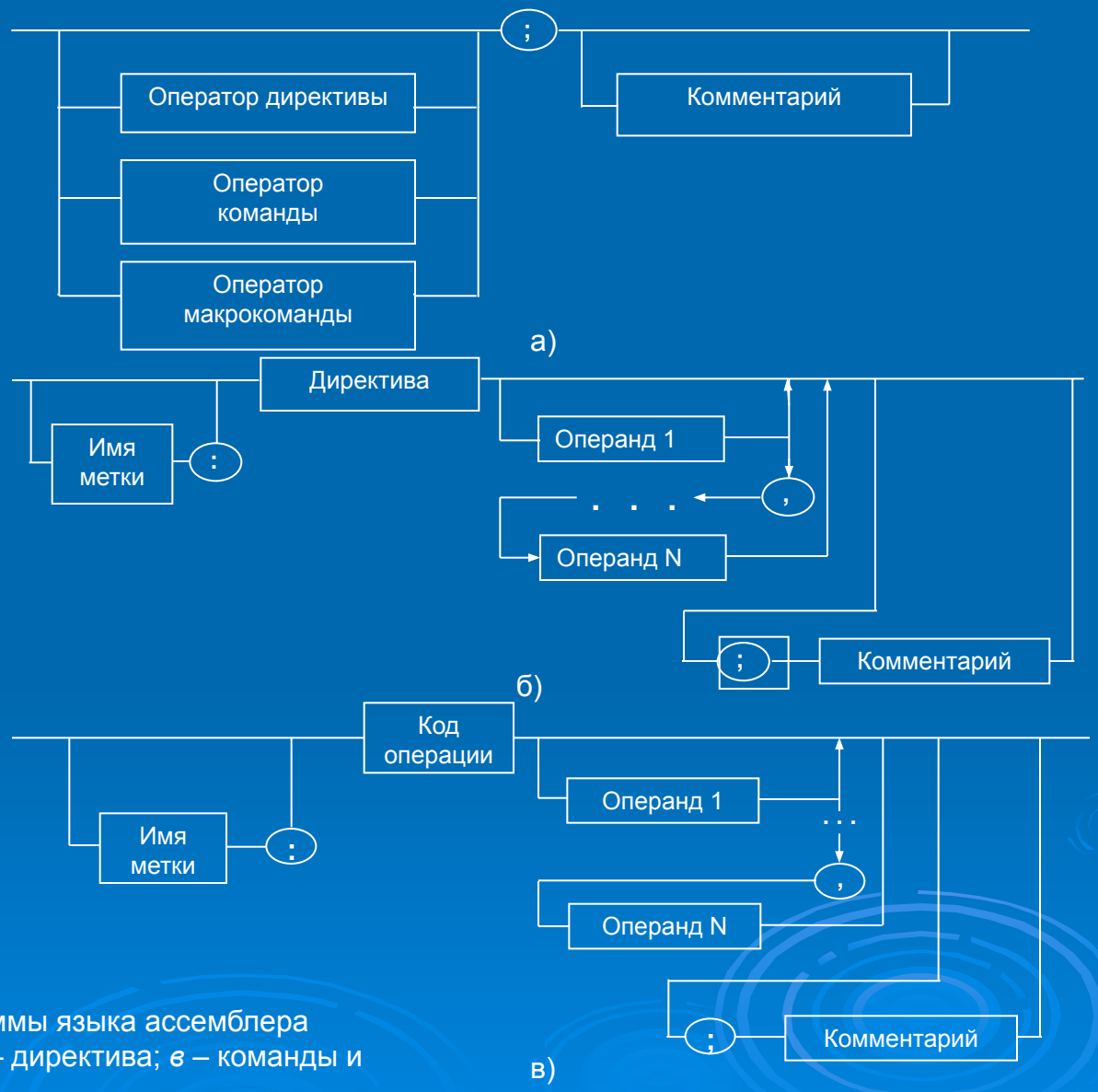

Рис. 2.1. Синтаксические диаграммы языка ассемблера
а – предложение ассемблера; *б* – директива; *в* – команды и макрокоманды

Набор регистров

Синтаксические диаграммы задают все правила формирования программы на языке ассемблера.

Непосредственные операнды

Непосредственным операндом называется число, строка или выражение имеющие фиксированное значение, оно может быть заданно конкретным значением в поле операнда или определено через equ или ‘=’. Например:

r equ 13

e = r - 2

mov al, r

mov al, 13

значения **13**, **r**, **e**, в приведенном фрагменте являются непосредственными операндами.

Адресные операнды

Адресные операнды задают физическое расположение операнда в памяти. Синтаксическая диаграмма адресных операндов представлена на рис. 2.2.

Например:

mov ax, ss:0013h

приведенный оператор записывает слово из регистра **ax** по адресу, старшая часть которого хранится в регистре **ss**, а младшая имеет значение 0013h.

Рис. 2.2. Синтаксическая диаграмма адресных операндов

Перемещаемые операнды

Перемещаемые операнды являются именами переменных или меткам инструкций. В отличие от адресных операндов их значение изменяется в зависимости от значения сегментной составляющей адреса.
Например:

```
data segment
prim  dw  25 dup (0)
. . .
code segment
. . .
lea  si,  prim
```

конкретное физическое значение физического адреса переменной prim будет известно только после загрузки программы.

Счетчик адреса

Счетчик адреса позволяет задавать относительные адреса. Для обозначения текущего значения счетчика адреса используется символ **$**.
Например:

```
jmp  $+3
cld
mov  al, 2
```

в приведенном фрагменте управление передается на оператор mov, минуя оператор cld, имеющий длину 1 байт.

Арифметические операторы

Синтаксическая диаграмма арифметических операторов представлена на рис. 2.3. К бинарным арифметическим операциям относятся операции ‘*”, ‘/’, ‘+’, ‘–‘ и ‘mod’; к унарным арифметическим операциям относятся операции ‘–‘ и ‘+’.

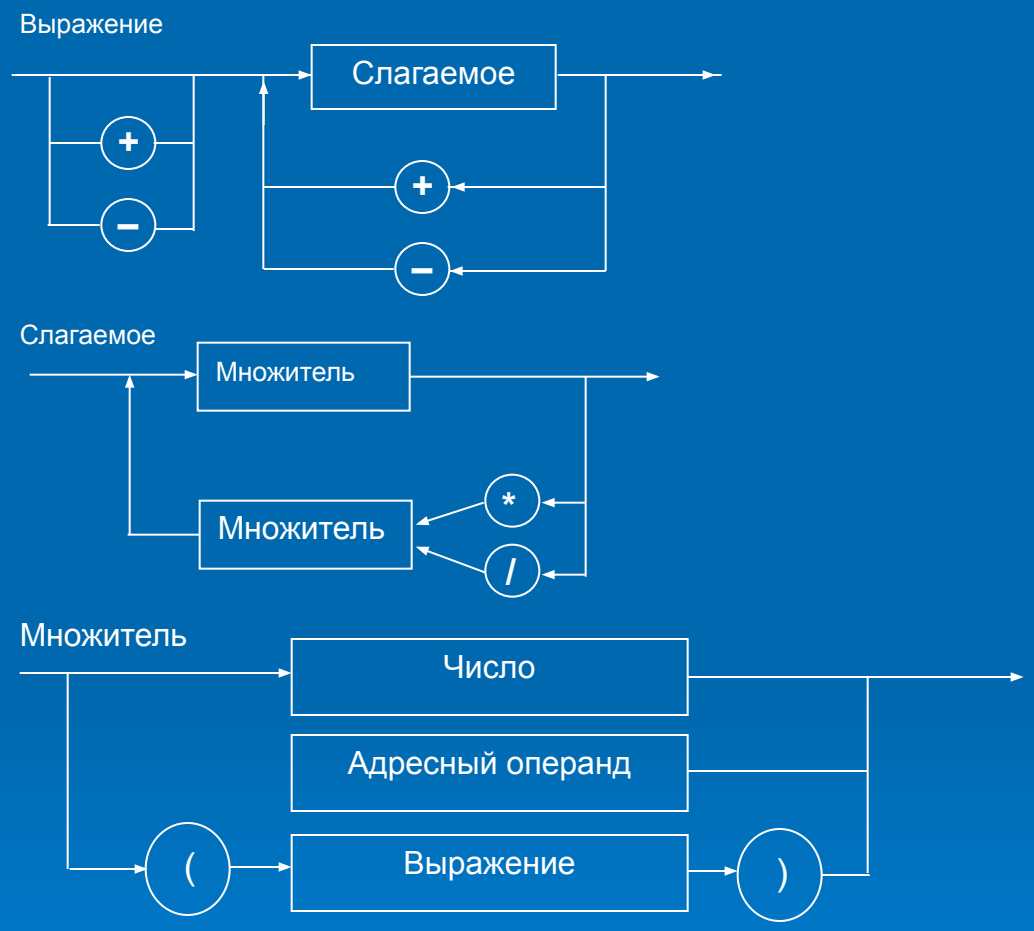

Рис. 2.3. Синтаксические диаграммы арифметических операторов

Операторы сдвига

Операторы сдвига выполняют сдвиг числа на указанное количество разрядов влево или вправо. Синтаксическая диаграмма оператора сдвига представлена на рис. 2.4.

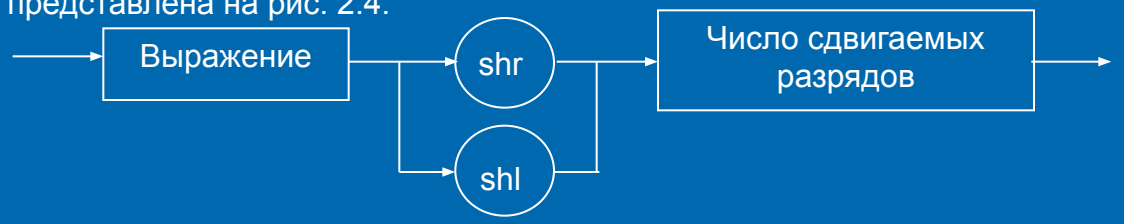

Рис. 2.4. Синтаксическая диаграмма операторов сдвига

Операторы сравнения

Оператор сравнения предназначен для формирования логических выражений. Значение “Да” сответствует числу 1, “Нет” – числу 0. Синтаксическая диаграмма оператора сравнения представлена на рис. 2.5.

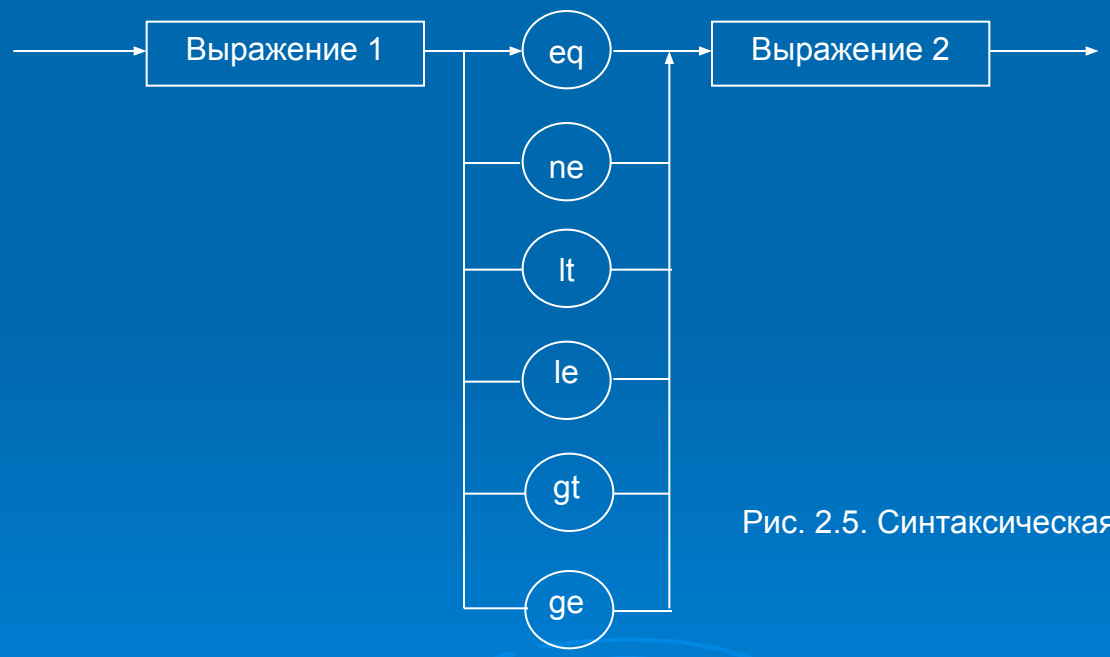

Рис. 2.5. Синтаксическая диаграмма оператора сравнения

Соответствие операторов ассемблера математической записи представлено в таблице 2.1.

Таблица 2.1.Операторы сравнения

Оператор	Eq	Ne	Lt	Le	Gt	Ge
Знак	=	≠	<	≤	>	≥

Логические операторы

Логические операторы выполняют над аргументами побитовые операции. Синтаксические диаграммы логических операторов представлены на рис. 2.6.

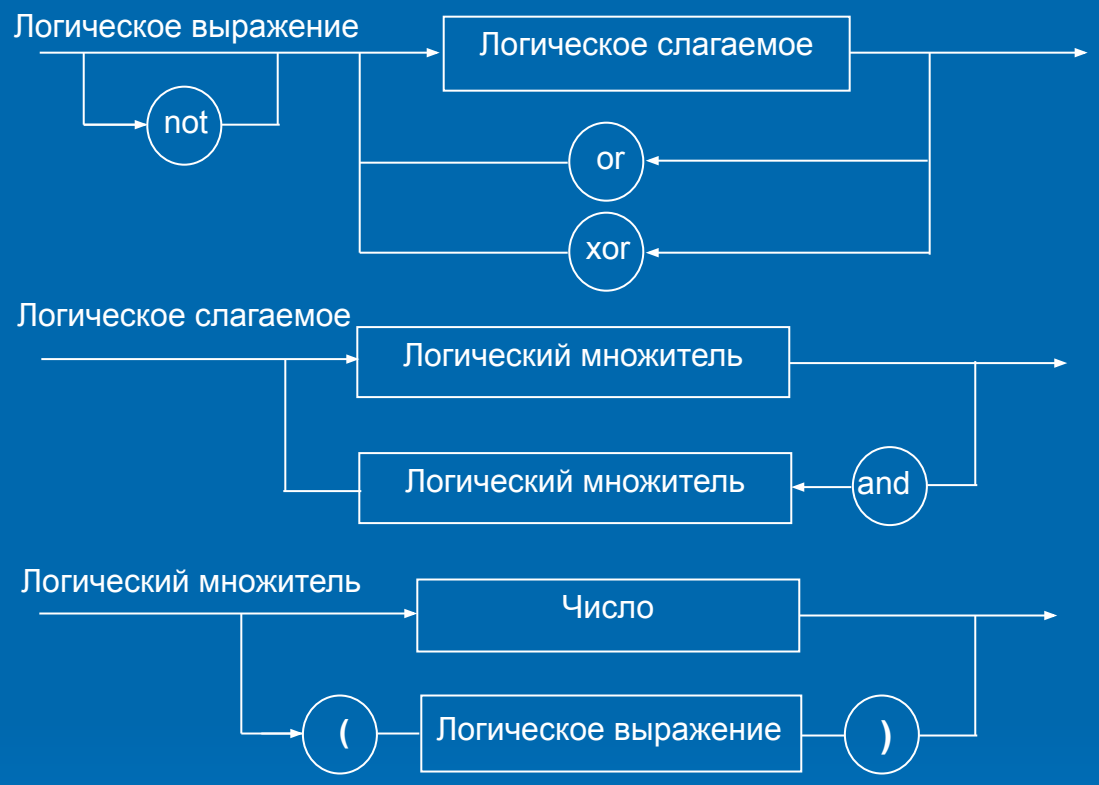

Рис. 2.6. Синтаксические диаграммы логических операторов

Индексный оператор

Индексный оператор позволяет организовать работу с массивами. В операции используются данные, размещенные по адресу заданному именем переменной плюс смещение, заданное в квадратных скобках. Синтаксическая диаграмма, иллюстрирующая работу индексного оператора, представлена на рис. 2.7.

Рис. 2.7. Синтаксическая диаграмма индексного оператора

Оператор преобразования типа

Оператор ptr позволяет преобразовать тип переменной или типа адресации. Возможно использование следующих значений типов: byte, word, dword, qword, tbyte и два указателя на способ адресации: near, far. Синтаксическая диаграмма оператора преобразования типа представлена на рис. 2.8.

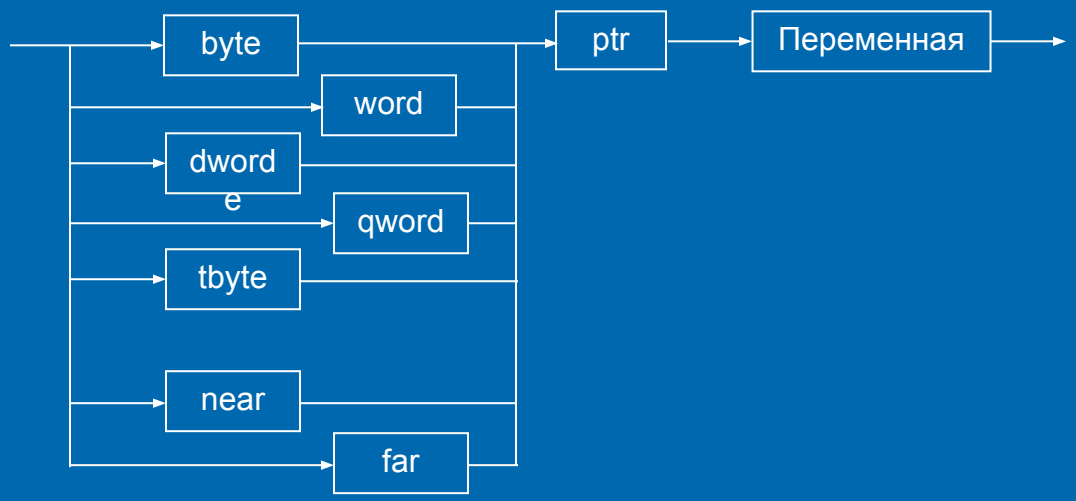

Рис. 2.8. Синтаксическая диаграмма оператора преобразования типа

Например:
mov al, byte ptr d_wrd+1 ;пересылка второго байта из двойного слова

Операторы получения сегментной составляющий адреса и смещения
Оператор SEG позволяет получить значение сегмента, а offset – смещения для указанного адреса (см. 2.2 и 1.3).
Например:
mov ex, seg prim
mov dx, offset prim
после выполнения данных операторов в паре регистров ex: dx будет полный адрес переменный prim.

Директивы сегментации

Программа на ассемблере может работать с шестью сегментами: кода, сетка и четырьмя сегментами данных. Для простых программ содержащих только с одним сегментом кода, сетка и данных возможно применение упрощенной модели сегментации.

model small
.stack (размер)
.data
описание переменных
.code
main proc
тело программы
main endproc
End

Назначение директив приведено в таблице 2.2.

Таблица 2.2. Назначение директив сегментации

Директива	Описание
model	Выбор модели памяти. Поддерживаются следующие модели памяти: **tyne, small, medium, compact и large**. Информация о моделях памяти представлена в табл. 2.4.
.data	Описание переменных программы.
.stack	Описание стека. Используется: **.stack** <размер>
.code	Сегмент для размещения операторов программы.
.fardata	Описание переменных, размещаемых за пределами текущего сегмента.

Идентификатор model создает следующие служебные переменные, которые Вы множите использовать в своих программах. Назначение переменных представлено в табл. 2.3.

Таблица 2.3. Служебные переменные

Имя	Назначение
@code	Физический адрес сегмента кода.
@data	Физический адрес сегмента данных.
@fardata	Физический адрес сегмента данных.
@curseg	Физический адрес сегмента неинициализированных данных типа far
@stack	Физический адрес сегмента стека.

Таблица 2.4. Модели памяти

Модель памяти	Тип кода	Тип данных	Описание
Tyne	Near	near	Модель памяти, используемая в .com файлах. Сегмент кода совпадает с сегментом данных.
Small	Near	near	Наиболее часто используемая модель памяти для программирования на ассемблере. Все данные объединены в один сегмент.
Medium	Far	near	Каждый программный модуль размещается в своем сегменте, при этом используются длинные адреса. На данные выделяется один сегмент.
Compact	Near	far	Программа размещается в одном сегменте. Данные могут размещаться в различных сегментах, при этом используется длинные адреса.
Large	Far	far	Каждый программный модуль размещается в своем сегменте, везде используются длинные адреса.

Использование сегментов в различных моделях памяти проиллюстрировано на рис. 2.9.

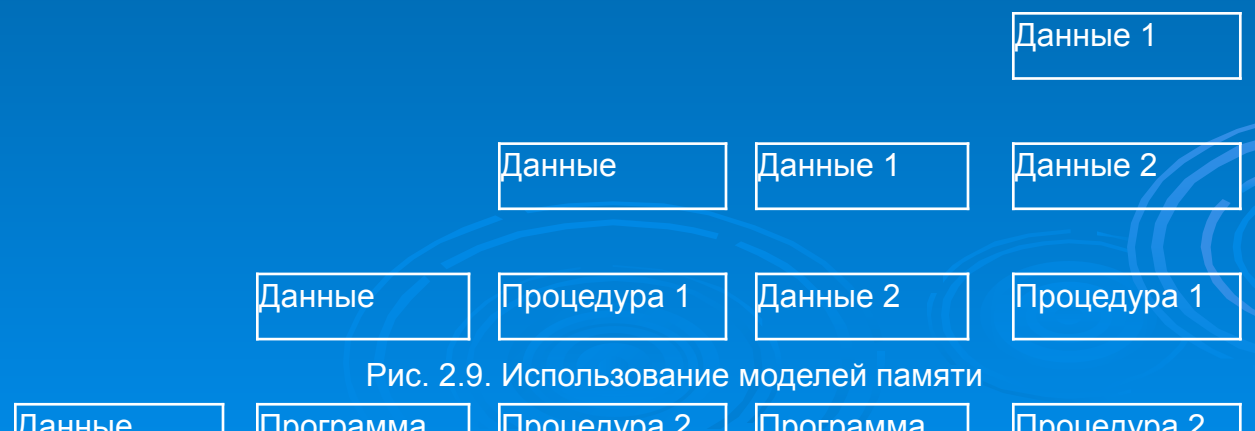

Рис. 2.9. Использование моделей памяти

Описание простых типов данных

Для описания простых данных используются директивы резервирования памяти. Синтаксическая диаграмма описания простых переменных приведена на рис. 2.10.
Значение количества памяти резервируемое при описании переменных приведена на в табл. 2.5.

Таблица 2.5. Резервирование памяти

Имя типа	db	dw	dd	dp	df	dq	dt
Кол-во памяти	1 Байт	2 Байта	4 Байта	6 Байт	6 Байт	8 Байт	10 Байт

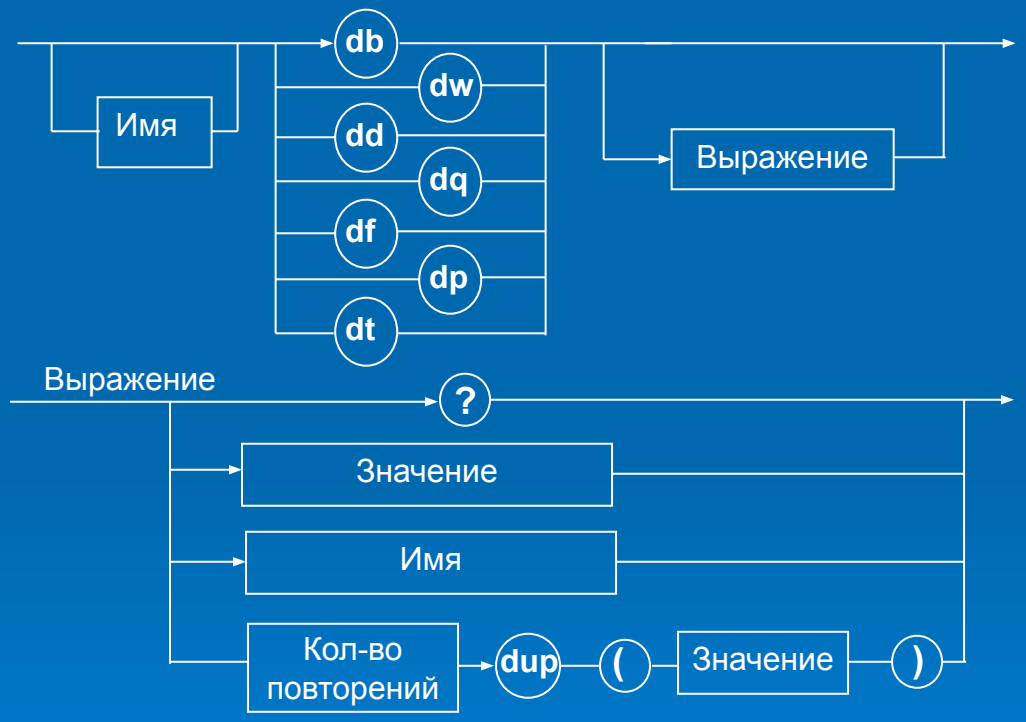

Рис. 2.10. Синтаксическая диаграмма описания простых переменных

При работе с переменными необходимо учитывать следующее, **младший байт размещается всегда по младшему адресу**.
Например:

```
model small
.stack 100h
.data
test1  db  12h
test2  db  10
test3  db  10  dup ('  ')
test4  db  10  dup (?)
srt1  db  'строка$'
```

символ ‘?’ означает что значение ячеек не будет определено.
символ ‘$’ означает что значение ячеек не будет определено.

Организация ввода вывода

Для вывода на экран сообщения, возможно, использовать прерывание 21h.
Вывод строки на экран:

```
mov  ah,  09h  ;        поместить в регистр ah номер функции прерывания 21h
mov  dx,  offset str1  ;      в регистр dx помещается указатель на строку
int  21h  ;             вызов прерывания 21h
```

Вывод символа на экран (выводимый символ находится в регистре dl):

```
mov  ah,  02h ;         поместить в регистр ah номер функции прерывания 21h
int  21h ;              вызов прерывания 21h
```

Ввод символа с клавиатуры (введенный символ находится в регистре al):

```
mov  ah,  01h  ;        поместить в регистр ah номер функции прерывания 21h
int  21h  ;             вызов прерывания 21h
```

Пример программы

Программа выводит на экран строку **"Введите символ: ".**

После ввода допустим символа "a" на экране появится надпись **"Введенный символ: a".**

```
model small
.stack 100h
.data
str0 db 13,10,'$'
str1 db 'Введите символ: $'
str2 db 'Введенный символ: $'
t1 db 1 dup(' '),'$'
.code
start:
mov  ax,@data
mov  ds,ax

mov  ah,09h
mov  dx, offset str1          ; вывод строки str1 на экран
int  21h

mov  ah,01h           ; ввод символа с клавиатуры в регистр al
int  21h

mov  t1,al            ; перенос значения регистра al в переменную t3

mov  ah,09h
mov  dx,  offset str0         ; вывод строки str0 на экран
int  21h

mov  dx,  offset str2         ; вывод строки str2 на экран
int  21h

mov  ah,02h
mov  dl,t1            ; вывод символа из регистра dl на экран
int  21h

mov  ah,4ch           ; завершение работы программы
int  21h
end start
```