Лекция 16

Язык Ассемблер, трансляция программ с языка Ассемблер в машинный код, структура программы, описание сегментов, указание констант, объявление данных

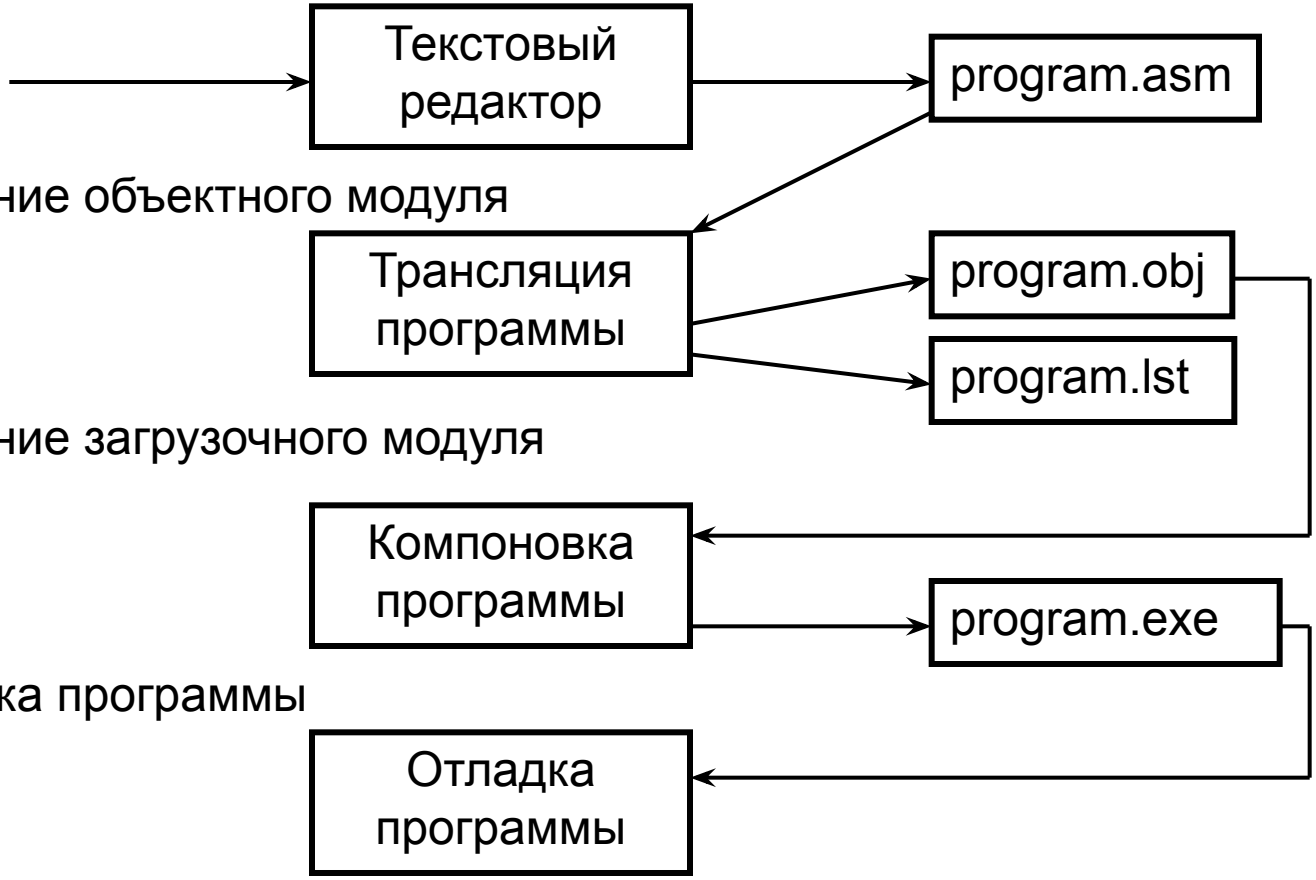
Трансляторы языка Ассемблер

Основные представители:

- TASM – Turbo Assembler
- MASM – Macro Assembler
- FASM – Flat Assembler
- NASM – Native Assembler

Процесс разработки программ на Ассемблере

1. Ввод исходного текста программы



2. Создание объектного модуля

3. Создание загрузочного модуля

4. Отладка программы

Язык Ассемблер

- Язык Ассемблер является символическим аналогом машинного языка. Программа, написанная на Ассемблере, должна отражать все особенности архитектуры микропроцессора: организацию памяти, способы адресации операндов, правила использования регистров и т.д.

- Программа на Ассемблере представляет собой совокупность блоков памяти, называемых сегментами памяти. Программа может состоять из одного или нескольких таких блоков.

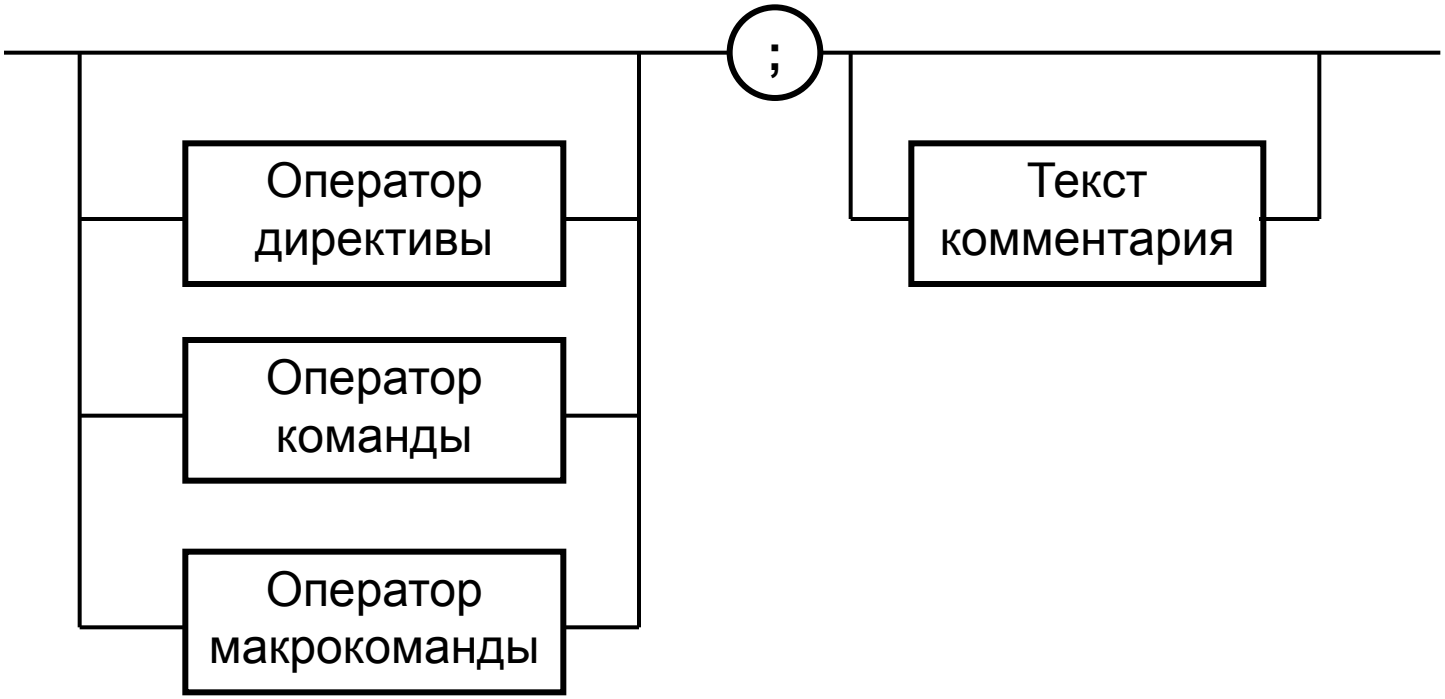
- Сама программа состоит из предложений Ассемблера.

Язык Ассемблер

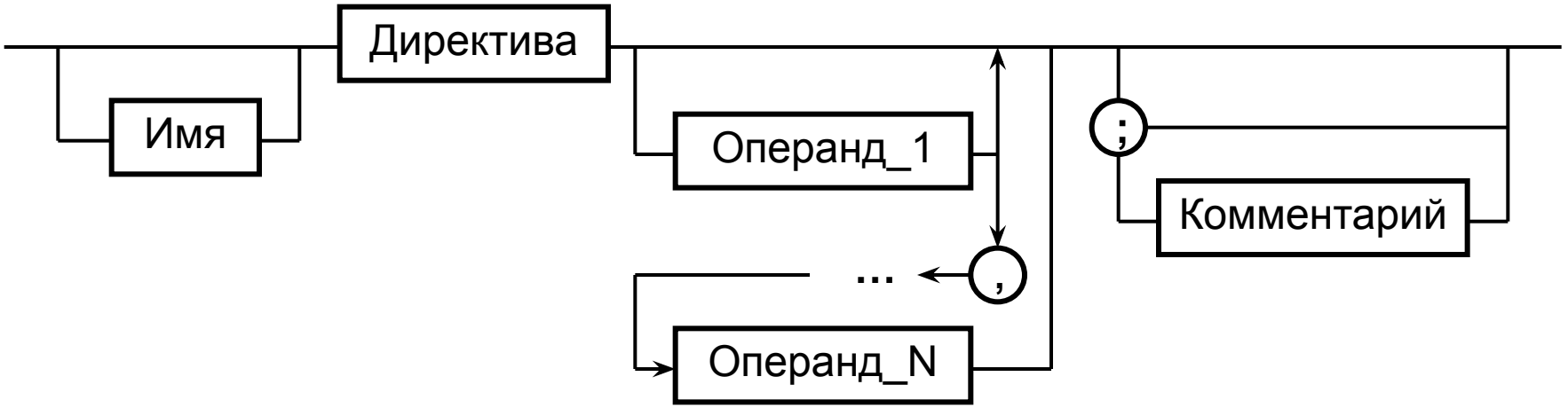
Предложения языка Ассемблер бывают четырех типов:

- Команды (или инструкции) – символические аналоги машинных команд.
- Макрокоманды – оформляемые определенным образом предложения текста программы, замещаемые во время трансляции другими предложениями.
- Директивы – указания транслятору на выполнение некоторых действий.
- Строки комментариев – текст, игнорирующийся транслятором.

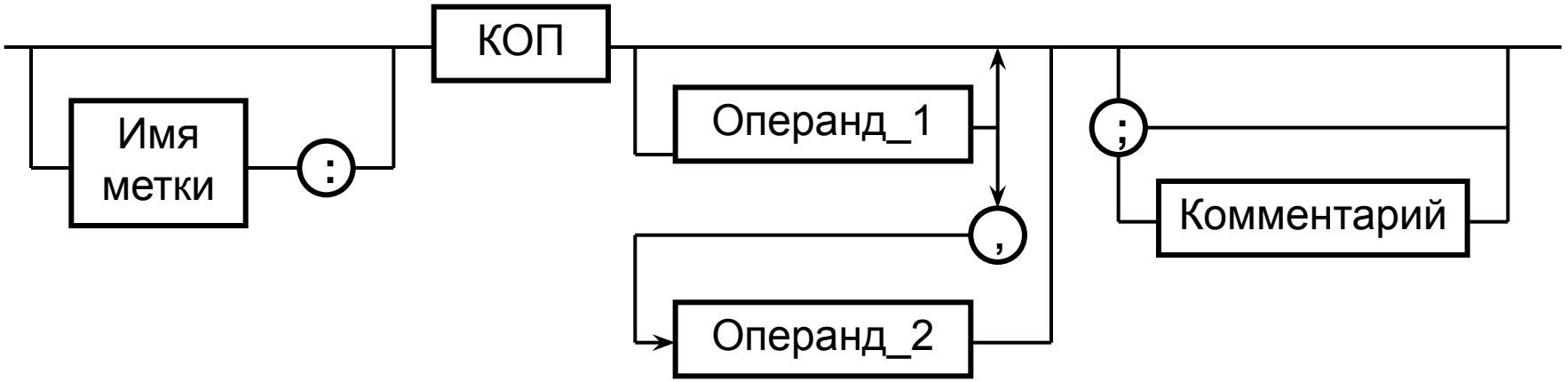
Формат предложения



Формат директив



Формат команд и макрокоманд



- Имя метки – идентификатор, значением которого является адрес первого байта того предложения исходного текста программы, которое он обозначает.
- Имя – идентификатор, отличающий данную директиву от других директив.
- Код операции или директива – это мнемоническое обозначения соответствующей машинной команды, макрокоманды или директивы транслятора.
- Операнды – части команды, макрокоманды или директивы ассемблера, обозначающие объекты, над которыми производятся действия.

Синтаксис языка Ассемблер

Допустимыми символами при написании текста программ являются:

- все латинские буквы;
- цифры;
- знаки: ?, @, $, _, &;
- разделители: , . [ ] ( ) < > { } + / * % ! ‘ “ ? \ = # ^

Синтаксис языка Ассемблер

Предложения Ассемблера формируются из лексем, представляющих собой синтаксически неразделимые последовательности допустимых символов языка, имеющие смысл для транслятора. Лексемами являются:

- Идентификаторы – последовательности допустимых символов, использующиеся для обозначения таких объектов программы, как коды операций, имена переменных и названия меток.
- Цепочки символов – последовательности символов, заключенные в одинарные или двойные кавычки.
- Целые числа в двоичной, десятичной или шестнадцатеричной системах счисления:

10000011b – двоичная система счисления
123 – десятичная система счисления
2Ah – шестнадцатеричная система счисления
0D4h – шестнадцатеричная система счисления

Виды операндов

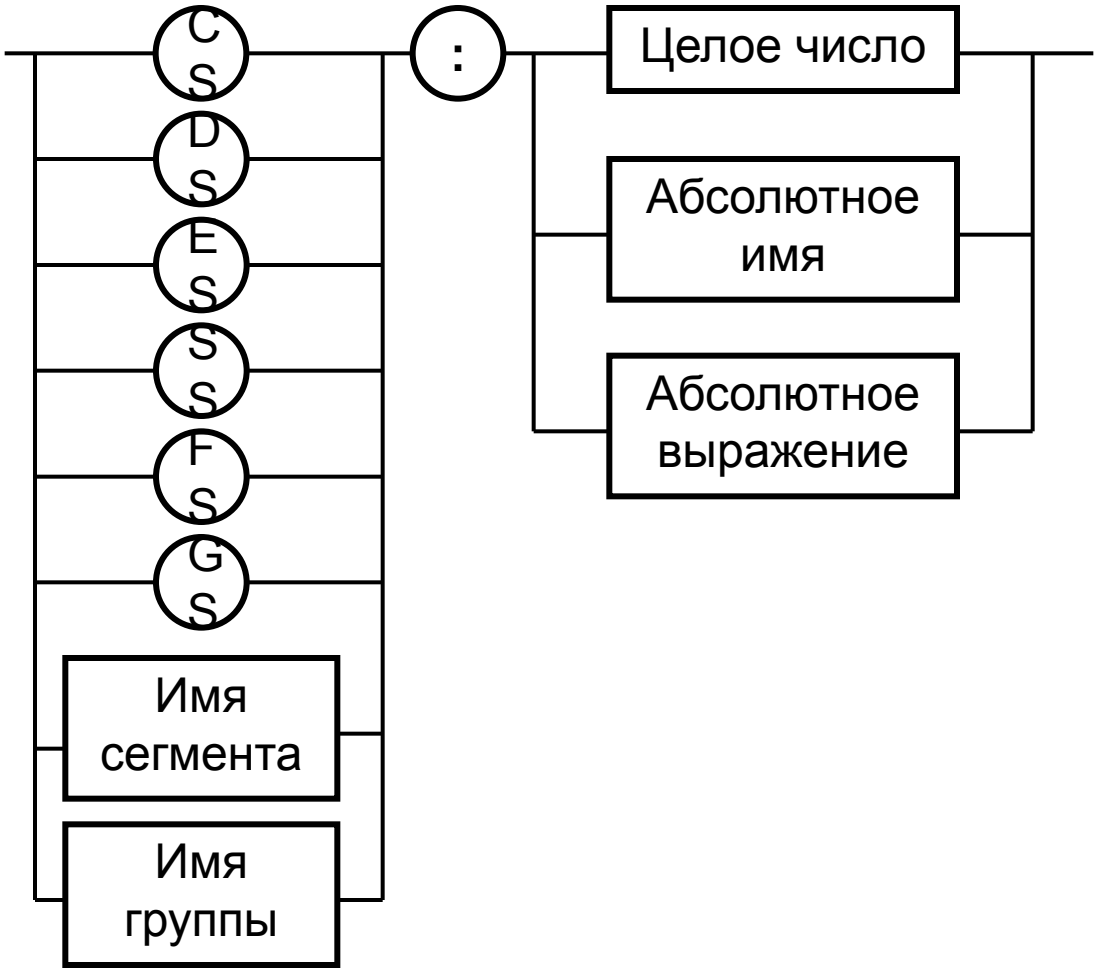
- Постоянные (непосредственные) операнды
- Адресные операнды.
- Перемещаемые операнды.
- Счетчик адреса.
- Регистровый операнд.
- Базовый и индексный операнды.
- Структурные операнды.
- Записи.

Постоянные (непосредственные) операнды

Постоянным (непосредственным) операнд – число, строка, имя или выражение имеющее некоторое фиксированное значение. Имя должно быть определено операторами equ или =.

```
val equ 10
num = val – 5
    mov     ax, val    ;mov  ax, 10
    mov     ax, num    ;mov  ax,5
    mov     ax, 10
```

Адресные операнды




```
mov    ax, ds:0000h
```

Перемещаемые операнды

Перемещаемые операнды – любые символьные имена, представляющие некоторые адреса в памяти. Эти адреса могут обозначать местоположение в памяти некоторой инструкции (если операнд – метка) или данных (если операнд – имя области памяти в сегменте данных).

```
Data SEGMENT
values          db   10 dup(0)
…
Code SEGMENT
…
        jmp next
…
next:       lea  si, values
…
```

Счетчик адреса

Счетчик адреса – специфический вид операнда, обозначаемый знаком $. Когда транслятор встречает в исходной программе этот символ, то он подставляет вместо него текущее значение счетчика адреса.

```
jmp     $+3
nop
mov     al, 10
```

Остальные операнды

- Регистровый операнд – это просто имя регистра.
- Базовый и индексный операнды – используются при реализации косвенной, индексной или их комбинаций и расширений.
- Структурные операнды – используются для доступа к конкретному элементу структуры.
- Записи (аналогично структурному типу) используются для доступа к битовому полю некоторой записи.

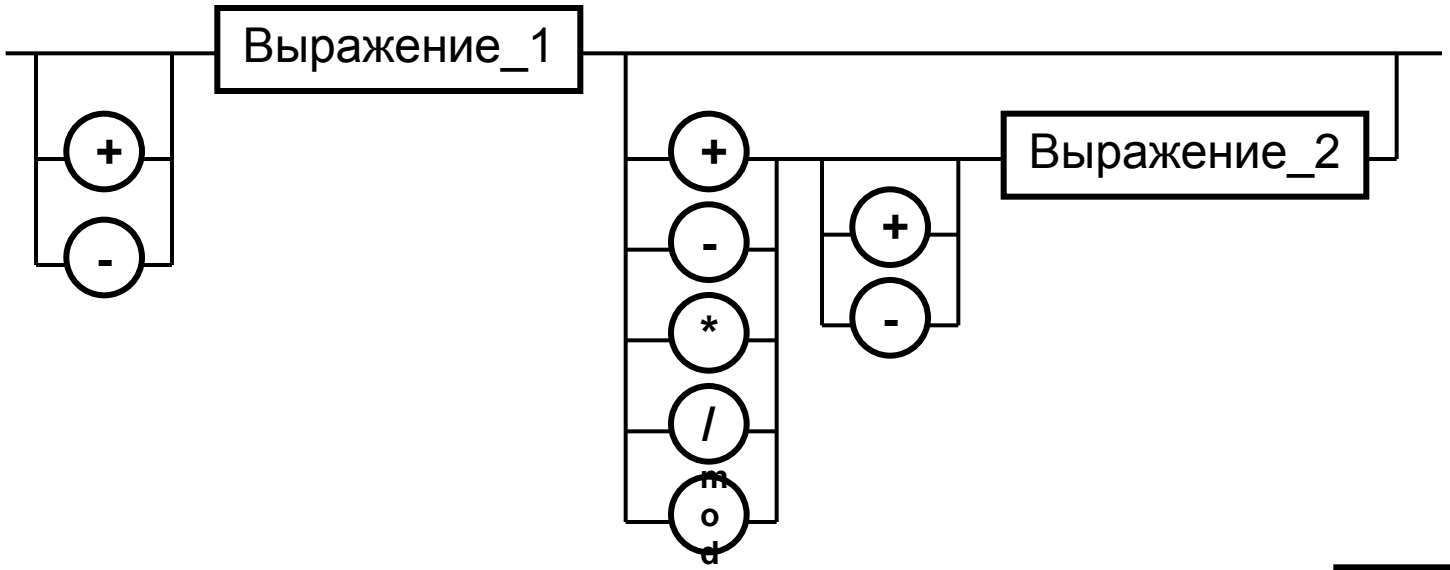
Операторы языка Ассемблер

Делятся на следующие виды:

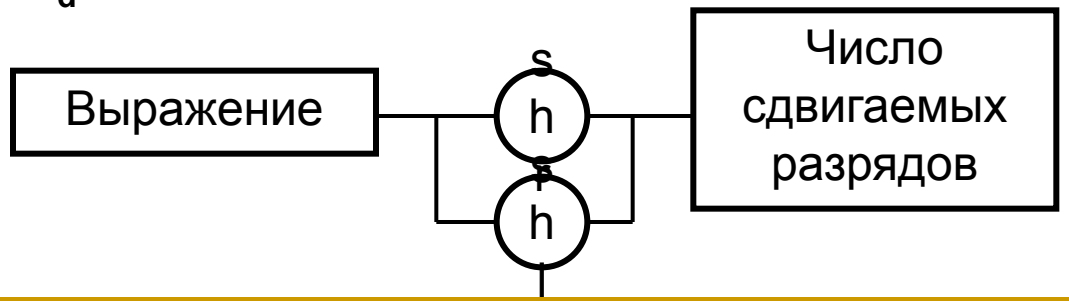
- Арифметические операторы,
- Операторы сдвига,
- Операторы сравнения,
- Логические операторы,
- Индексный оператор,
- Оператор переопределения типа,
- Оператор переопределения сегмента,
- Оператор именования типа
- Оператор получения сегментной составляющей адреса
- Оператор получения смещения выражения

Операторы языка Ассемблер

Арифметические операторы

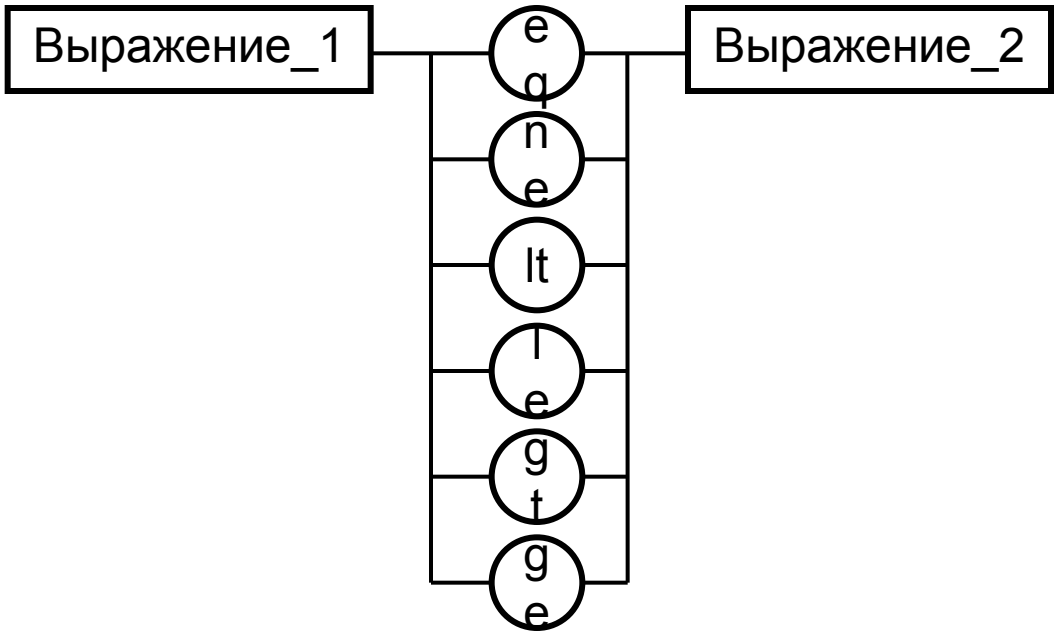


Операторы сдвига



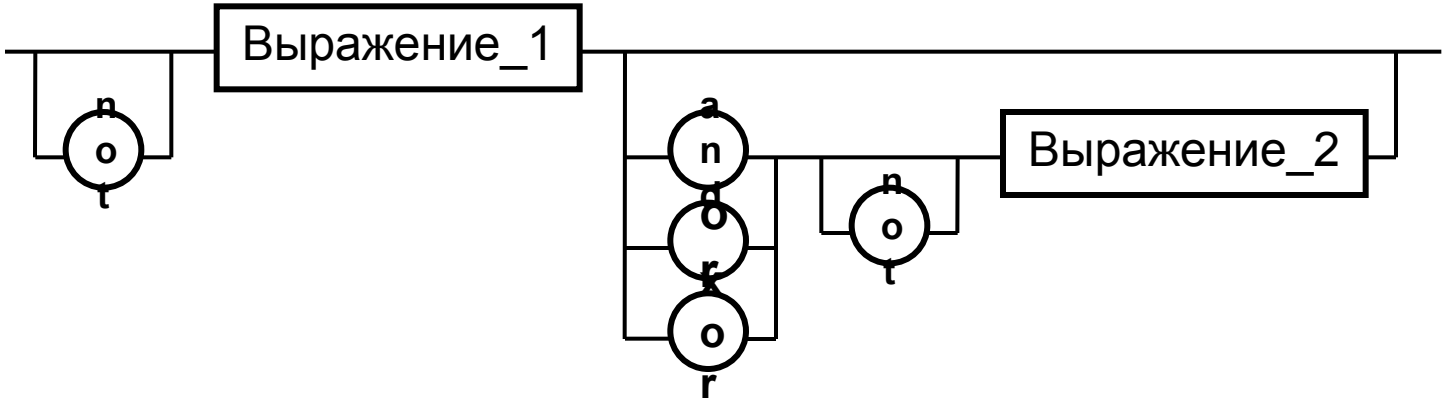
Операторы языка Ассемблер

Операторы сравнения

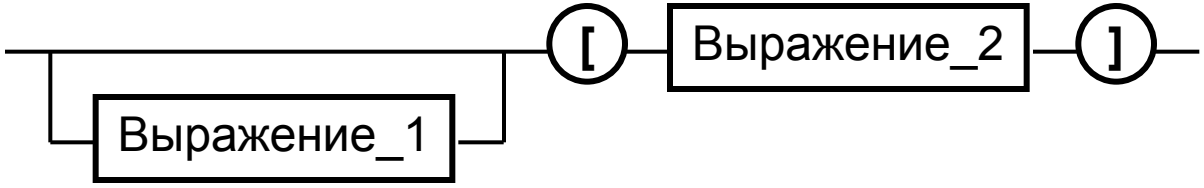


Операторы языка Ассемблер

Логические операторы



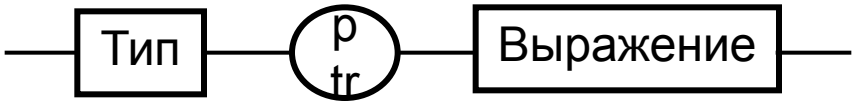
Индексный оператор



Операторы языка Ассемблер

Оператор переопределения
типа ptr

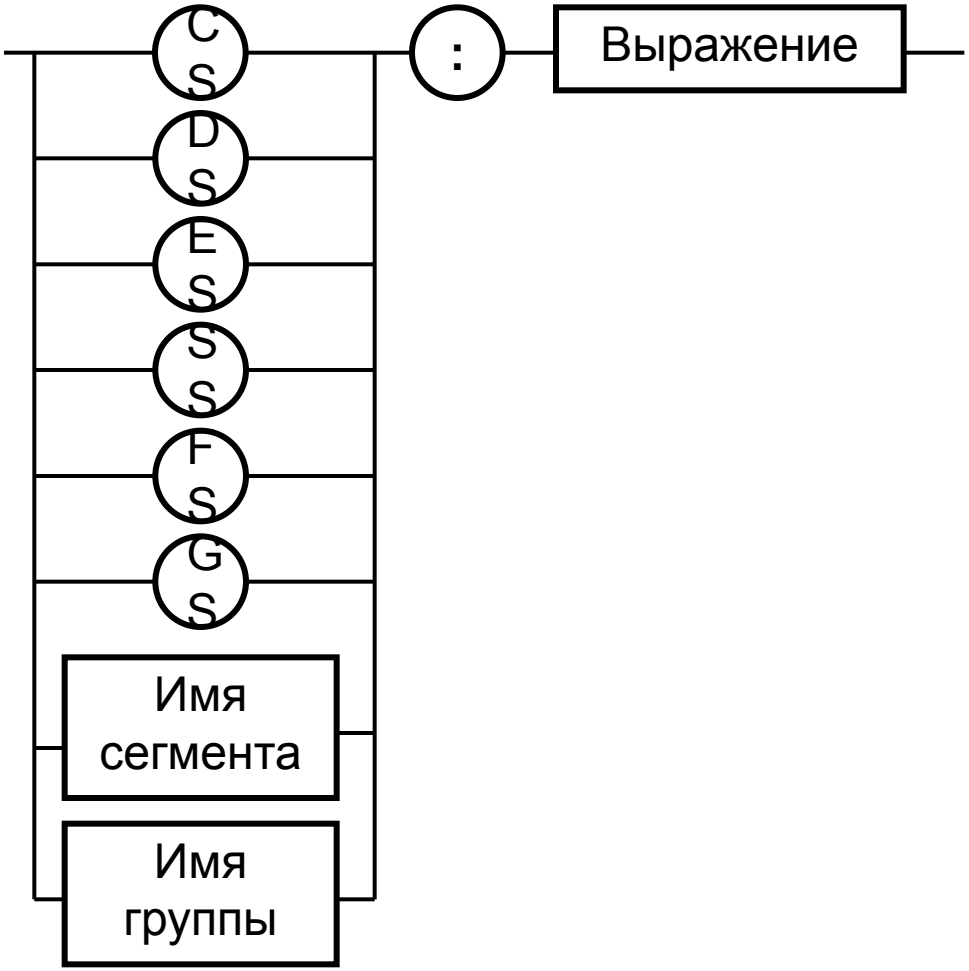
```
val dd  0
…
     mov    al, byte ptr val+1
```



Операторы языка Ассемблер

Оператор переопределения сегмента:

```
.code
     jmp     metka
val dw 100
metka:
…
     mov    al, cs:val
```

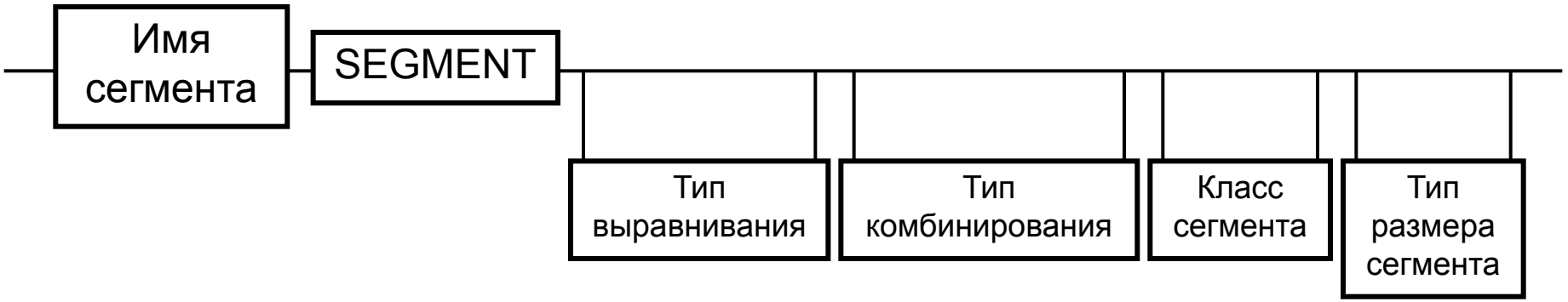


Операторы языка Ассемблер

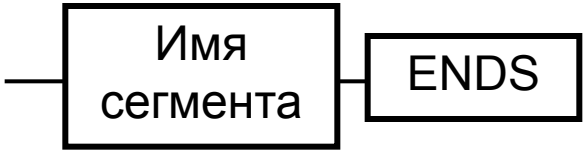
Оператор получения сегментной составляющей адреса seg
Оператор получения смещения выражения offset

```
.data
value       dw      5
…
.code
…
    mov     ax, seg value
    mov     es, ax
    mov     bx, offset value
    mov     ax, es:[bx]
```

Описание сегментов



…
Директивы ассемблера
Команды ассемблера
Макрокоманды ассемблера
Строки комментариев
…



Описание сегментов

Выравнивание сегмента:

- BYTE – выравнивание не выполняется
- WORD – сегмент начинается по адресу, кратному двум
- DWORD - сегмент начинается по адресу, кратному четырем
- PARA – сегмент начинается по адресу, кратному шестнадцати (по умолчанию)
- PAGE - сегмент начинается по адресу, кратному 256
- MEMPAGE - сегмент начинается по адресу, кратному 4Кбайт

Описание сегментов

Атрибут комбинирования сегментов:

- PRIVATE – сегмент не будет объединятся с другими сегментами с тем же именем вне данного модуля
- PUBLIC – заставляет компоновщик соединить все сегменты с одинаковым именем
- COMMON – располагает все сегменты с одним и тем же именем по одному адресу
- AT XXXX – располагает сегмент по абсолютному адресу параграфа
- STACK – определение сегмента стека

Атрибут класса сегмента – это заключенная в кавычки строка, помогающая компоновщику определить соответствующий порядок следования сегментов при сборке программы из сегментов нескольких модулей.

Описание сегментов

Атрибут размера сегмента:

USE16 – это означает, что сегмент допускает 16-разрядную адресацию.

USE32 – сегмент будет 32-ухразрядным

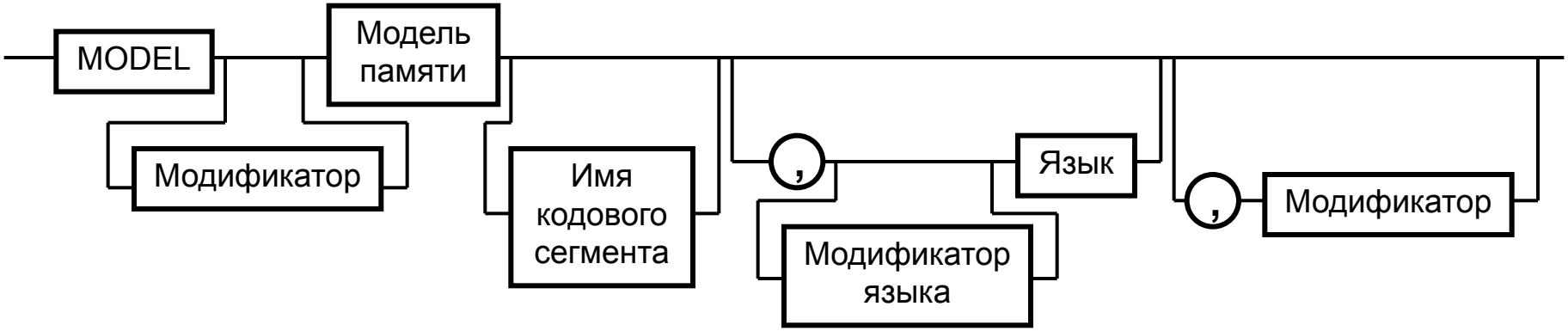
Директива SEGMENT не содержит информации о функциональном назначении сегмента (код, данные или стек). Указание этого назначения осуществляется с помощью директивы ASSUME в следующем виде:

ASSUME <имя-сегментного-регистра> ‘:’ <имя-сегмента>

Пример:

ASSUME cs:Code, ds:Data, ss:Stack

Директива MODEL



Директива MODEL предназначена для управления моделью памяти программы. Эта директива позволяет использовать упрощенные директивы сегментации.

Упрощенные директивы определения сегмента

Режим MASM	Режим IDEAL	Назначение
.CODE [имя]	CODESEG [имя]	Начало или продолжение сегмента кода
.DATA	DATASEG	Начало или продолжение сегмента инициализированных данных
.CONST	CONST	Начало или продолжения сегмента константных данных
.DATA?	UDATASEG	Начало или продолжение сегмента неинициализированных данных
.STACK [размер]	STACK [размер]	Начало или продолжение сегмента стека модуля
.FARDATA [имя]	FARDATA [имя]	Начало или продолжение сегмента инициализированных данных типа far
.FARDATA? [имя]	UFARDATA [имя]	Начало или продолжение сегмента неинициализированных данных типа far

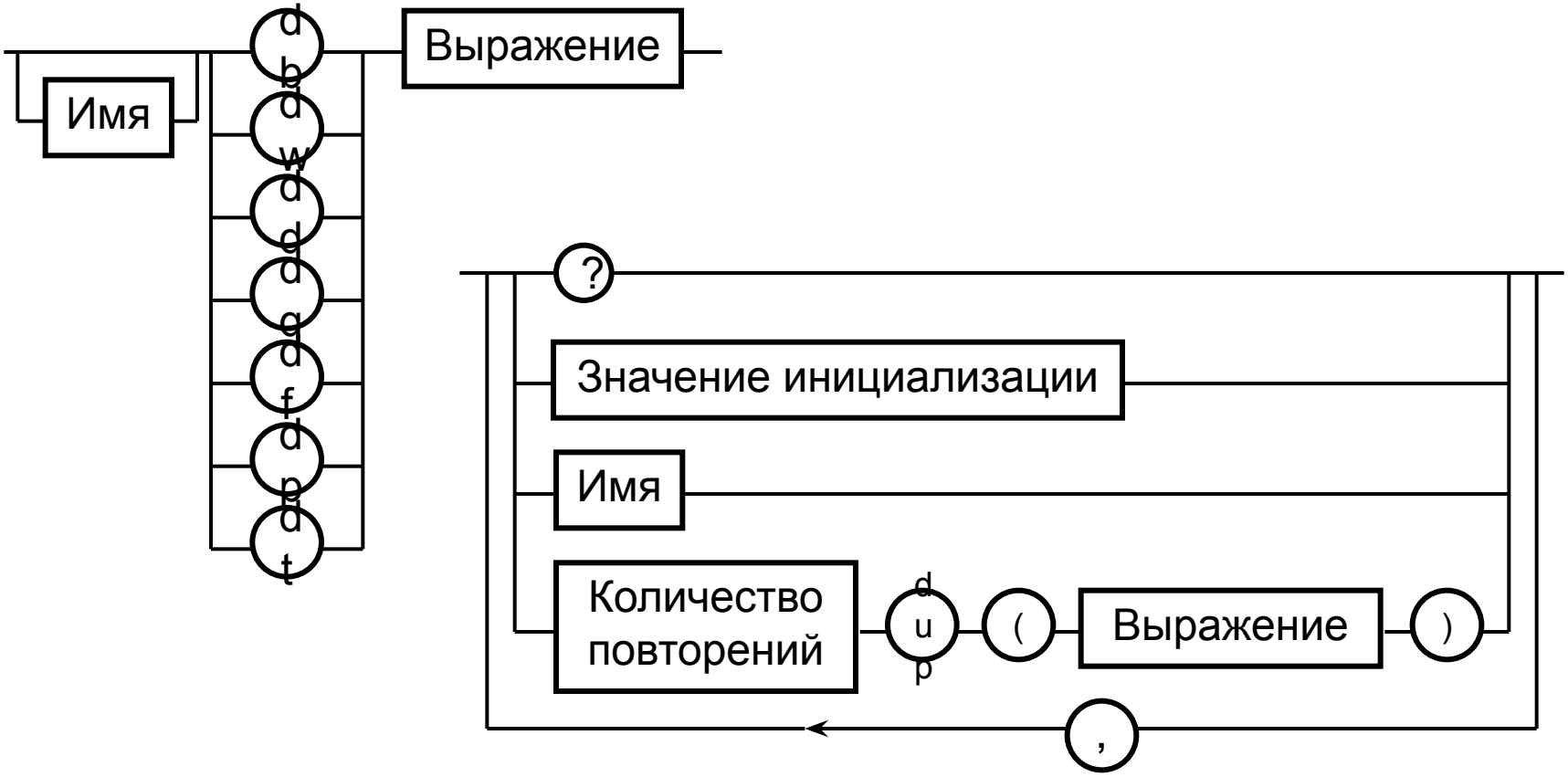
Идентификаторы, создаваемые директивой MODEL

Имя идентификатора	Значение переменной
@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	Физический адрес сегмента неинициализированных данных типа far
@stack	Физический адрес сегмента стека

Модели памяти

Модель	Тип кода	Тип данных	Назначение модели
TINY	near	near	Код и данные объединены в одну группу DGROUP
SMALL	near	near	Код занимает один сегмент, данные объединены в одну группу с именем DGROUP
MEDIUM	far	near	Код занимает несколько сегментов, по одному на каждый объединяемый программный модуль
COMPACT	near	far	Код в одном сегменте, данные в нескольких
LARGE	far	far	Код и данные в нескольких сегментах
FLAT	near	near	Код и данные в одном сегменте (плоская модель памяти)

Простые типы данных



Простые типы данных

Обозначения:

- ? – показывает, что значение не определено;
- Значение инициализации – значение элемента данных, которое будет занесено после загрузки программы;
- Выражение – итеративная конструкция;
- Имя – некоторое символическое имя метки или ячейки данных.

Типы данных:

- db – 1 байт
- dw – 2 байта
- dd – 4 байта
- dq – 8 байт
- df – 6 байт
- dp – 6 байт
- dt – 10 байт

Простые типы данных

Примеры:

```
Mess    db ‘Hello world!’, 0
Value   dw   1400
Array   dd 20 dup(?)
```

Пример COM программы для MS-DOS

```
.386
model tiny                  ;Указание модели памяти
Code segment use16          ;Начало описания сегмента кода
ASSUME cs:Code, ds:Code     ;Ассоциация регистров с сегментом
    org   100h              ;Генерация смещения на 256 байт
start:                      ;Метка начала программы
    push cs                 ;Запись регистра CS в стек
    pop  ds                 ;Загрузка регистра DS значением из стека
    mov  dx, offset mess        ;Помещение в DS смещения строки mess
    mov  ah, 09h            ;Запись в AH номера функции вывода строки
    int  21h                ;Вызов сервиса MS-DOS
    int  20h                ;Завершение COM программы в MS-DOS
mess    db  'Hello world!','$'  ;Объявление строки
Code ends                   ;Завершение описания строки
end start
```

Пример EXE программы для MS-DOS

```
.386
model small                     ;Указание модели памяти
Stack SEGMENT STACK use16           ;Объявление сегмента стека
    ASSUME ss:Stack                 ;Ассоциация регистра SS с сегментом стека
    DB    100h dup(?)               ;Резервирование 256 байт под стек
Stack ENDS                      ;Завершение описания сегмента стека
Data SEGMENT use16                  ;Объявление сегмента данных
    ASSUME ds:Data                  ;Ассоциирование регистра DS с сегментом данных
mess db    'Hello world!','$'   ;Объявление строки
Data ENDS                       ;Завершение описания сегмента данных
Code SEGMENT use16                  ;Объявление сегмента кода
    ASSUME cs:Code                  ; Ассоциирование регистра CS с сегментом кода
start:                      ;Метка начала программы
    mov    ax, seg mess             ;Загрузка в AX адреса сегмента строки mess
    mov    ds, ax                   ;Запись в DS значения AX
    mov    dx, offset mess          ;Запись в DX смещения строки mess
    mov    ah, 09h                  ;Запись в AH номера функции вывода строки
    int    21h                  ;Вызов сервиса MS-DOS
    mov    ax, 4c00h                ;Запись в AX функции завершения программы
    int    21h                  ;Завершение EXE программы в MS-DOS
Code ENDS                       ;Завершение описания сегмента данных
end start
```

Пример EXE программы для Windows

```
include \masm32\include\masm32rt.inc      ;Подключение библиотеки
            ;Объявление сегмента неинициализированных данных
.data?
    value dd ?                  ;Объявление переменной без инициализации
            ;Объявление сегмента инициализированных данных
.data
    item dd 0                   ; Объявление переменной с инициализацией

.code                           ;Объявление сегмента кода
start:                      ;Метка начала программы
    call    main                ;вызов процедуры main
    inkey                       ;вызов макроса ожидания нажатия клавиши
    exit                        ;вызов макроса завершения программы
main proc                       ;объявление процедуры main
    cls                         ;вызов макроса очистки экрана
    print "Hello World!",13,10          ;вызов макроса вывода сообщения
    ret                         ;команда выхода из процедуры
main endp                       ;конец описания процедуры
end start
```