Декларация данных.

Простые типы данных в ассемблере:

Синтаксис декларирования данных в ассемблере следующий:

<имя> <директива> <выражение>

<имя> - некоторое символическое имя метки или ячейки памяти в сегменте данных, используемое в программе;

<директива> - зарезервированное слово для указания типа объявляемых данных;

<выражение> - может содержать константу или «?» (неопределенное значение).

Определение последовательности повторяющихся данных:

выражение1 DUP (*выражение2*)

выражение2 = ?

ДИРЕКТИВА ОПРЕДЕЛЕНИЯ БАЙТА (DB)

Символьное выражение в директиве DB может содержать строку символов любой длины, вплоть до конца строки.

Числовое выражение в директиве DB может содержать одну или более однобайтовых констант.

Один байт выражается двумя шестнадцатеричными цифрами.

Наибольшее положительное шестнадцатеричное число в одном байте это 7f, все "большие" числа от 80 до ff представляют отрицательные значения.

В десятичном исчислении эти пределы выражаются числами +127 и -128.

ДИРЕКТИВА ОПРЕДЕЛЕНИЯ СЛОВА (DW)

Директива DW определяет элементы, которые имеют длину в одно слово (два байта).

Символьное выражение в DW ограничено двумя символами, которые ассемблер представляет в объектном коде так, что, например, 'pc' становится 'cp'.

Числовое выражение в DW может содержать одно или более двухбайтовых констант.

Два байта представляются четырьмя шестнадцатеричными цифрами.

Наибольшее положительное шестнадцатеричное число в двух байтах это 7fff; все "большие" числа от 8000 до ffff представляют отрицательные значения.

В десятичном исчислении эти пределы выражаются числами +32767 и -32768.

Для форматов директив DW, DD и DQ ассемблер преобразует константы в шестнадцатеричный объектный код, но записывает его в обратной последовательности.

Таким образом, десятичное значение 12345 преобразуется в шестнадцатеричное 3039, но записывается в объектном коде как 3930.

ДИРЕКТИВА ОПРЕДЕЛЕНИЯ ДВОЙНОГО СЛОВА (DD)

Директива DD определяет элементы, которые имеют длину в два слова (четыре байта).

Числовое выражение может содержать одну или более констант, каждая из которых имеет максимум четыре байта (восемь шестнадцатеричных цифр).

Наибольшее положительное шестнадцатеричное число в четырех байтах это 7fffffff; все "большие" числа от 80000000 до ffffffff представляют отрицательные значения.

В десятичном исчислении эти пределы выражаются числами +2147483647 и -2147483648.

Ассемблер преобразует все числовые константы в директиве DD в шестнадцатеричное представление, но записывает объектный код в обратной последовательности.

Таким образом, десятичное значение 12345 преобразуется в шестнадцатеричное 00003039, но записывается в объектном коде как 39300000.

Символьное выражение директивы DD ограничено двумя символами. Ассемблер преобразует символы и выравнивает их слева в четырехбайтовом двойном слове в объектном коде.

ДИРЕКТИВА ОПРЕДЕЛЕНИЯ УЧЕТВЕРЕННОГО СЛОВА (DQ)

Директива DQ определяет элементы, имеющие длину четыре слова (восемь байт).

Числовое выражение может содержать одну или более констант, каждая из которых имеет максимум восемь байт или 16 шестнадцатеричных цифр.

Наибольшее положительное шестнадцатеричное число - это семерка и 15 цифр f.

Для получения представления о величине этого числа, покажем, что шестнадцатеричное 1 и 15 нулей эквивалентен следующему десятичному числу:

1152921504606846976

Обработка ассемблером символьных строк в директиве DQ аналогично директивам DD и DW.

ДИРЕКТИВА ОПРЕДЕЛЕНИЯ ДЕСЯТИ БАЙТ (DT)

Директива DT определяет элементы данных, имеющие длину в десять байт.

Назначение этой директивы связано с "упакованными десятичными" числовыми величинами.

По директиве DT генерируются различные константы, в зависимости от версии ассемблера.

Константные выражения.

Символьные строки.

Содержимое строки отмечается одиночными кавычками, например, 'pc' или двойными кавычками - "pc".

Ассемблер переводит символьные строки в объектный код в обычном формате ASCII.

Символьная строка определяется только директивой DB, в которой указывается более двух символов в нормальной последовательности слева направо.

Следовательно, директива DB представляет единственно возможный формат для определения символьных данных.

Целые числа:

[*знак*]*последовательность_цифр* [*признак_системы_счисления*]

***Признак системы счисления (СС)*:**

B -двоичная,

Q или **O** - восьмеричная,

D - десятичная,

H - шестнадцатеричная(если число начинается с цифр A..F, то впереди должна быть вставлена цифра 0).

Например:

42q, 00100010b, 22h, 34d, 34, 0A34h

По умолчанию в ASM86 считается обычно десятичная система счисления.

Умалчиваемая система счисления может быть переопределена директивой **.RADIX.**

Например:

.RADIX 8

Числовые константы используются для арифметических величин и для адресов памяти.

Для описания константы кавычки не ставятся.

Ассемблер преобразует все числовые константы в шестнадцатеричные и записывает байты в объектном коде в обратной последовательности - справа налево.

Десятичный формат.

Десятичный формат допускает десятичные цифры от 0 до 9 и обозначается последней буквой d, которую можно не указывать, например, или 125d.

Шестнадцатиричный формат.

Шестнадцатиричный формат допускает шестнадцатеричные цифры от 0 до f и обозначается последней буквой h. Так как ассемблер полагает, что с буквы начинаются идентификаторы, то первой цифрой шестнадцатеричной константы должна быть цифра от 0 до 9.

Двоичный формат.

Двоичный формат допускает двоичные цифры 0 и 1 и обозначается последней буквой b.

Восьмеричный формат.

Восьмеричный формат допускает восьмеричные цифры от 0 до 7 и обозначается последней буквой q или o, например, 253q.

Десятичный формат с плавающей точкой.

Этот формат поддерживается только ассемблером Masm.

При записи символьных и числовых констант следует помнить, что, например, символьная константа, определенная как db '12', представляет символы ASCII и генерирует шестнадцатеричное 3132, а числовая константа, определенная как DB 12, представляет двоичное число и генерирует шестнадцатеричное 0c.

Программный счетчик ассемблера.

Для определения относительной позиции в сегменте данных или кода ассемблер использует адресный счетчик.

Например:

Смещение	Имя	Операция	Операнд	Адресный счетчик
00	flda	dw	2542h	02
02	fldb	db	36h	03
03	fldc	dw	212eh	05
05	fldd	dd	00000705h	09

Для изменения значения адресного счетчика и соответственно адреса следующего определяемого элемента используется директива **org.**

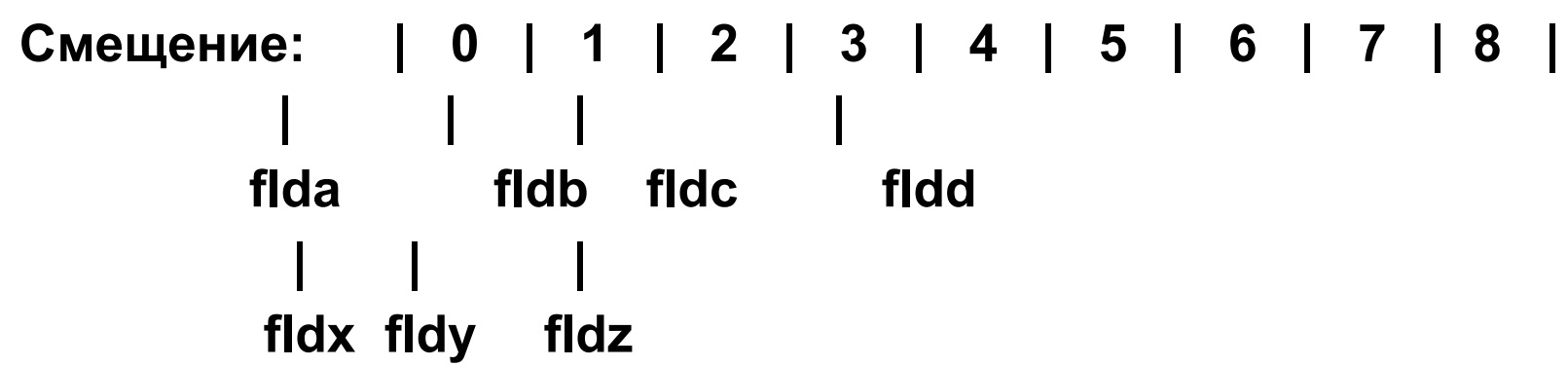
Формат директивы:

ORG *выражение*

Рассмотрим следующие элементы данных, определенные непосредственно после поля fldd:

Смещение	Имя	Операция	Операнд	Адресный счетчик
		org	0	00
00	fldx	db	?	01
01	fldy	dw	?	02
03	fldz	db	?	04
		org	$+5	09

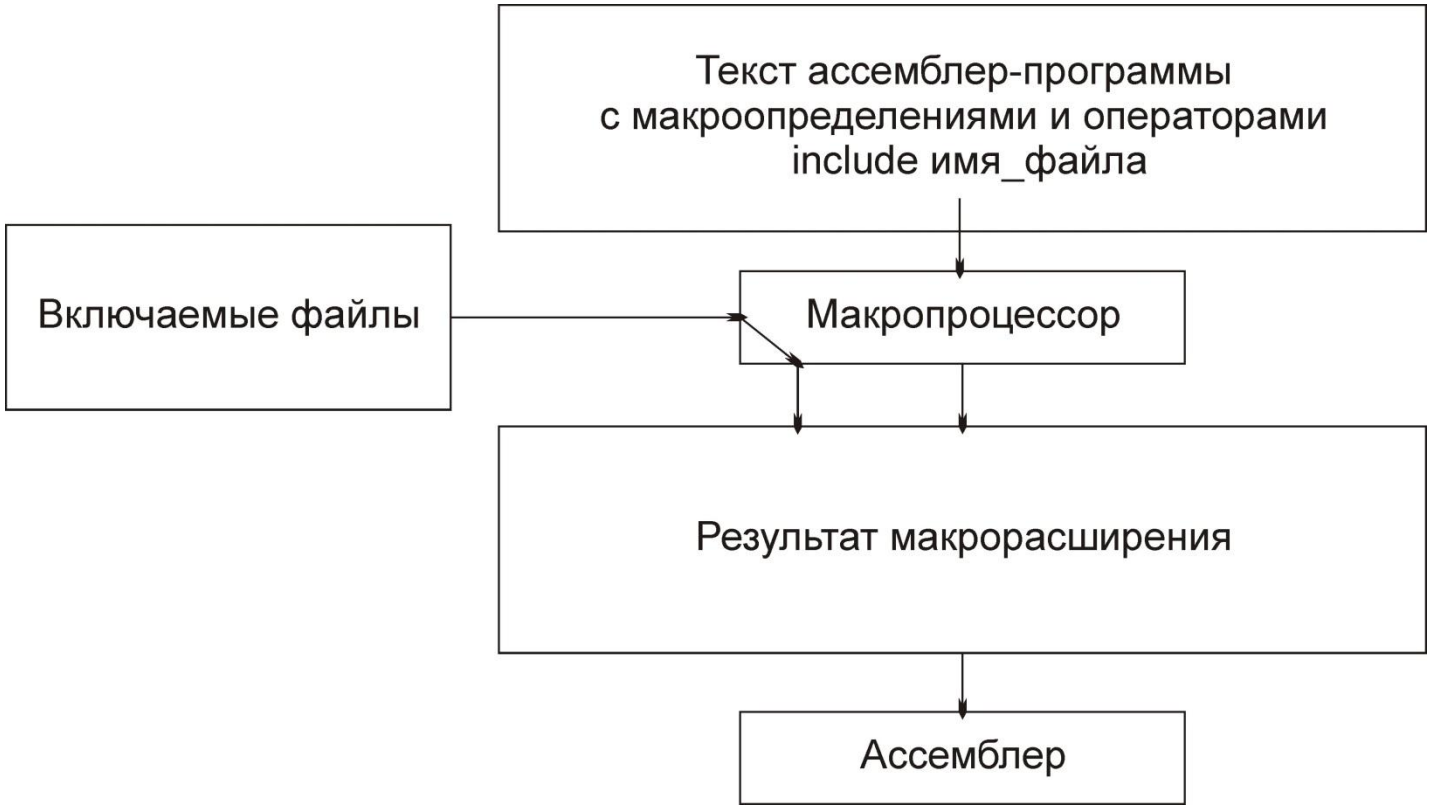
Первая директива org возвращает адресный счетчик в нулевое значение.
Поля fldx, fldy и fldz определяют те же области памяти, что и поля flda, fldb и fldc:



Операнд $+5 устанавливает адресный счетчик равным 04 + 5 = 09 (также как и после определения fldd.

```
mov ax,floc          ;Одно слово
mov al,fldz          ;один байт
```

Работа с макросами.



Для каждой закодированной команды ассемблер генерирует одну команду на машинном языке.

Но для каждого закодированного оператора компиляторного языка pascal или с генерируется один или более (чаще много) команд машинного языка.

В этом отношении можно считать, что компиляторный язык состоит из макрооператоров.

Использование макрокоманд позволяет:

1. упростить и сократить исходный текст программы;
2. сделать программу более понятной;
3. уменьшить число возможных ошибок кодирования.

Простое макроопределение.

Макроопределение должно находиться до определения сегмента.

Директива macro указывает ассемблеру, что следующие команды до директивы endm являются частью макроопределения.

Директива endm завершает макроопределение.

Команды между директивами macro и endm составляют тело макроопределения.

Имена, на которые имеются ссылки в макроопределении должны быть определены где-нибудь в другом месте программы.

Описанная макрокоманда может использоваться в кодовом сегменте там, где необходимо.

Когда ассемблер анализирует команду <имя_макроса>, он сначала просматривает таблицу мнемокодов и, не обнаружив там соответствующего элемента, проверяет макрокоманды.

Так как программа содержит определение макрокоманды с нужным именем, по которому она вызывается в сегменте кода, ассемблер подставляет тело макроопределения, генерируя необходимые команды - макрорасширение.

Программа может использовать макрокоманду любое число раз, и каждый раз для таких макрокоманд ассемблер генерирует одинаковые макрорасширения.

Пример1:

```
init1 macro
assume
    cs:cseg,ds:dseg,ss:stack;es:dseg
push ds
sub ax,ax
push ax
mov ax,dseg
mov ds,ax
mov es,ax
endm
```

Использование параметров в макрокомандах.

Для того, чтобы макрокоманда была более гибкой и могла принимать любые имена сегментов, необходимо определить имена сегментов, как формальные параметры:

Пример2:

```
init2 macro    csname,dsname,ssname ;Формальные параметры
assume         cs:csname,ds:dsname,ss:ssname;es:dsname
push ds
sub ax,ax
push ax
mov ax,dsname
mov ds,ax
mov es,ax
endm
```

Формальные параметры в макроопределении указывают ассемблеру на соответствие их имен любым аналогичным именам в теле макроопределения.

Формальные параметры могут иметь любые правильные ассемблерные имена, не обязательно совпадающими именами в сегменте данных.

Теперь при использовании макрокоманды init2 необходимо указать в качестве параметров действительные имена трех сегментов в соответствующей последовательности.

Пример:

макрокоманда содержит три параметра, которые соответствуют формальным параметрам в исходном макроопределении.

Макроопределение: init2 macro csname,dsname,ssname (форм. параметры)
| | |
Макрокоманда: init2 cseg, dseg, stack (параметры)

Так как ассемблер уже определил соответствие между формальными параметрами и операторами в макроопределении, то теперь ему остается подставить параметры макрокоманды в макрорасширении:

1. Параметр 1: cseg ставится в соответствие с csname в макроопределении. Ассемблер подставляет cseg вместо csname в директиве assume.

2. Параметр 2: dseg ставится в соответствие с dsname в макроопределении. Ассемблер подставляет dseg вместо двух dsname: в директиве assume и в команде mov.

3. Параметр 3: stack ставится в соответствие с ssname в макроопределении. Ассемблер подставляет stack вместо ssname в директиве assume.

Ассемблер не распознает регистровые имена и имена, определенные в области данных, как таковые. В одной макрокоманде может быть определено любое число формальных параметров, разделенных запятыми, вплоть до 120 колонки в строке.

Комментарии в макросах.

Директива **comment** или символ «**;**» указывают на строку комментария.

Так как по умолчанию в листинг попадают только команды генерирующие объектный код, то ассемблер не будет автоматически выдавать и комментарии, имеющиеся в макроопределении.

Если необходимо, чтобы в расширении появлялись комментарии, следует использовать перед макрокомандой директиву **.lall** ("**list all**" - выводить все), которая кодируется вместе с лидирующей

точкой:

.lall

<имя_макроса> [<формальные_параметры_макроса>]

Также существует – (**;;**) - признак подавления вывода комментария в листинг.

По умолчанию в ассемблере действует директива **.xall**, которая выводит в листинг только команды, генерирующие объектный код.

Можно запретить появление в листинге ассемблерного кода в макрорасширениях, особенно при использовании макрокоманды в одной программе несколько раз директивой **.sall** ("suppress all" - подавить весь вывод), которая уменьшает размер выводимого листинга, но не оказывает никакого влияния на размер объектного модуля.

Директивы управления листингом **.lall, .xall, .sall** сохраняют свое действие по всему тексту программы, пока другая директива листинга не изменит его.

Эти директивы можно размещать в программе так, чтобы в одних макрокомандах распечатывались комментарии, в других - макрорасширения, а в третьих подавлялся вывод в листинг.

Использование макрокоманд в макроопределениях.

Макроопределение может содержать ссылку на другое макроопределение.

Пример:

Рассмотрим простое макроопределение dos21, которое заносит в регистр ah номер функции dos и выполняет int 21h:

```
dos21    macro    dosfunc
mov ah,dosfunc
int  21h
endm
```

Для использования данной макрокоманды при вводе с клавиатуры необходимо закодировать:

```
lea dx,namepar
dos21    0ah
```

Предположим, что имеется другое макроопределение, использующее функцию 02 в регистре ah для вывода символа:

```
disp macro    char
mov ah,02
mov dl,char
int  21h
endm
```

Для вывода на экран, например, звездочки достаточно закодировать макрокоманду disp '*'. Можно изменить макроопределение disp, воспользовавшись макрокомандой dos21:

```
disp macro    char
mov dl,char
dos21    02
endm
```

Теперь, если закодировать макрокоманду disp в виде disp '*', то ассемблер сгенерирует следующие команды:

```
mov dl,'*'
mov ah,02
int  21h
```

Директива local.

Для обеспечения уникальности генерируемых в каждом макрорасширении имен используется директива local, которая кодируется непосредственно после директивы macro, даже перед комментариями.

Общий формат имеет следующий вид:

```
local  dummy-1,dummy-2, ;Формальные параметры
```

Использование библиотеки макроопределений.

Возможна каталогизация собственных макрокоманд в библиотеке, используя любое описательное имя, например, **MACRO.LIB**:

```
INIT MACRO CSNAME, DSNAME, SSNAME
.
.
ENDM
PROMPT   MACRO    MESSGE
.
.
ENDM
```

После этого для использования любой из каталогизированных макрокоманд вместо MACRO определения в начале программы следует применять **директиву INCLUDE**:

```
INCLUDE   <путь>MACRO.LIB
.
.
INITCSEG,DATA,STACK
```

В этом случае ассемблер обращается к файлу MACRO.LIB и включает в программу оба макроопределения INIT и PROMPT.

Для обеспечения обработки директивы INCLUDE только в первом проходе (а не в обоих) можно использовать следующую конструкцию:

```
IF1
INCLUDE   <путь>MACRO.LIB
ENDIF
```

IF1 и ENDIF – **условные директивы**.

Директива IF1 указывает ассемблеру на необходимость доступа к библиотеке только в первом проходе трансляции.

Директива ENDIF завершает IF-логику. Таким образом, копия макроопределений не появится в листинге - будет сэкономлено и время, и память.

Расположение директивы INCLUDE не критично, но она должна появиться ранее любой макрокоманды из включаемой библиотеки.

Директива очистки.

Директива INCLUDE указывает ассемблеру на включение всех макроопределений из специфицированной библиотеки.

Пример:

Библиотека содержит макросы INIT, PROMPT и DIVIDE, хотя программе требуется только INIT.

Директива PURGE позволяет "удалить" нежелательные макросы PROMPT и DIVIDE в текущем ассемблировании:

```
IF1
    ;Включить всю библиотеку
INCLUDE <путь>MACRO.LIB
    ENDIF
    ;Удалить ненужные макросы
PURGE   PROMRT,DIYIDE
...
    ;Использование
    ;оставшейся макрокоманды
INIT  CSEG,DATA,STACK
```

Директива PURGE действует только в процессе ассемблирования и не оказывает никакого влияния на макрокоманды, находящиеся в библиотеке.

Конкатенация (&).

Символ амперсанд **(&)** указывает ассемблеру на **сцепление** (**конкатенацию**) текста или символов.

Следующая макрокоманда MOVE генерирует команду MOVSB или MOVSW:

```
MOVE    MACRO   TAG
REP  MOVS&TAG
ENDM
```

Теперь можно кодировать макрокоманду в виде MOVE B или MOVE W.

В результате макрорасширения ассемблер сцепит параметр с командой MOVS и получит REP MOVSB или REP MOVSW.

Директивы повторения: REPT, IRP, IRPC.

Заставляют ассемблер повторить блок операторов, завершаемых директивой ENDM.

Эти директивы не обязательно должны находиться в макроопределении, но если они там находятся, то одна директива ENDM требуется для завершения повторяющегося блока, а вторая ENDM – для завершения макроопределения.

Директивы повторения: REPT, IRP, IRPC.

REPT: Повторение.

Приводит к повторению блока операторов до директивы ENDM в соответствии с числом повторений, указанным в выражении:

REPT выражение

Пример 1:

Происходит начальная инициализация значения N=0 и затем повторяется генерация db n пять раз:

```
n = 0
rept 5
n =  n + 1
db n
endm
```

В результате будут сгенерированы пять операторов db от db 1 до db 5. Директива rept может использоваться таким образом для определения таблицы или части таблицы.

Пример 2:

Генерация пяти команд movsb, что эквивалентно rep movsb при содержимом cx равном 05:

```
rept 5
movsb
endm
```

IRP: Неопределенное повторение.

Приводит к повторению блока команд до директивы endm. Основной формат:

IRP dummy,

Аргументы, содержащиеся в угловых скобках, представляют собой любое число правильных символов, строк, числовых или арифметических констант.

Ассемблер генерирует блок кода для каждого аргумента.

Пример 3:

Ассемблер генерирует DB 3, DB 9, DB 17, DB 25 и DB 28:

```
IRP  N,<3, 9, 17, 25, 28>
DB   N
ENDM
```

IRPC: Неопределенное повторение символа.

Приводит к повторению блока операторов до директивы ENDM. Основной формат:

IRPC dummy,string

Ассемблер генерирует блок кода для каждого символа в строке "string".

Пример 4:

Ассемблер генерирует DW 3, DW 4 ... DW 8:

```
IRPC  N,345678
DW  N
ENDM
```

Условные директивы.

Условные директивы наиболее полезны внутри макроопределений, но не ограничены только этим применением.

Каждая **директива IF** должна иметь спаренную с ней **директиву ENDIF** для завершения IF-логики и возможную **директиву ELSE** для альтернативного действия:

IFxx (условие)

.

.

ELSE (не обязательное действие)

.

.

ENDIF (конец IF-логики)

Обработка данных директив макроассемблером заключается в вычислении логического выражения – условия и включения в объектный модуль либо блок до директивы ELSE, если условие истинно, либо блок до директивы ENDIF, если условие ложно.

Отсутствие директивы ENDIF вызывает сообщение об ошибке: "Undeterminated conditional" (незавершенный условный блок).

Если проверяемое условие истинно, то ассемблер выполняет условный блок до директивы ELSE или при отсутствии ELSE - до директивы ENDIF.

Если условие ложно, то ассемблер выполняет условный блок после директивы ELSE, а при отсутствии ELSE вообще обходит условный блок.

IF выражение	Если выражение не равно нулю, ассемблер обрабатывает операторы в условном блоке.
IFE выражение	Если выражение равно нулю, ассемблер обрабатывает операторы в условном блоке.
IF1 (нет выражения)	Если осуществляется первый проход ассемблирования то обрабатываются операторы в условном блоке.
IF2 (нет выражения)	Если осуществляется второй проход операторы ассемблирования, то обрабатываются в условном блоке.
IFDEF идентификатор	Если идентификатор определен в программе или объявлен как EXTRN, то ассемблер обрабатывает операторы в условном блоке.
IFNDEF идентификатор	Если идентификатор не определен в программе или не объявлен как EXTRN, то ассемблер обрабатывает операторы в условном блоке.
IFB <аргумент>	Если аргументом является пробел, ассемблер обрабатывает операторы в условном блоке. Аргумент должен быть в угловых скобках.
IFNB <аргумент>	Если аргументом является не пробел, то ассемблер обрабатывает операторы в условном блоке. Аргумент должен быть в угловых скобках.
IFIDN <арг-1>,<арг-2>	Если строка первого аргумента идентична строке второго аргумента, то ассемблер обрабатывает операторы в условном блоке. Аргументы должны быть в угловых скобках. **Учитывает различие строчных и прописных букв.**
IFIDNI <арг 1>, <арг 2>	Если строка первого аргумента идентична строке второго аргумента, то ассемблер обрабатывает операторы в условном блоке. Аргументы должны быть в угловых скобках. **Игнорирует различие строчных и прописных букв.**
IFDIF <арг-1>,<арг-2>	Если строка первого аргумента отличается от строки второго аргумента, то ассемблер обрабатывает операторы в условном блоке. Аргументы должны быть в угловых скобках. **Учитывает различие строчных и прописных букв.**
IFDIFI <арг-1>,<арг-2>	Если строка первого аргумента отличается от строки второго аргумента, то ассемблер обрабатывает операторы в условном блоке. Аргументы должны быть в угловых скобках. **Игнорирует различие строчных и прописных букв.**

Пример:

Для DOS INT 21H все запросы требуют занесения номера функции в регистр AH, в то время как лишь некоторые из них используют значение в регистре DX.

Следующее макроопределение учитывает эту особенность:

```
DOS21    MACRO    DOSFUNC,DXADDRES
MOV AH,DOSFUNC
IFNB
MOV DX,OFFSET DXADDRES
ENDIF
INT 21H
ENDM
```

Использование DOS21 для простого ввода с клавиатуры требует установки значения 01 в регистр AH:

```
DOS21    01
```

Ассемблер генерирует в результате команды MOV AH,01 и INT 21H. Для ввода символьной строки требуется занести в регистр AH значение 0AH, а в регистр DX - адрес области ввода:

```
DOS21 0AH,IPFIELD
```

Ассемблер генерирует в результате обе команды **MOV и INT 21H.**

Директива выхода из макроса EXITM.

Макроопределение может содержать условные директивы, которые проверяют важные условия.

Если условие истинно, то ассемблер должен прекратить дальнейшее макрорасширение.

Для этой цели служит **директива EXITM**:

```
IFxx [условие]
.
.
.
EXITM
.
.
ENDIF
```

Как только ассемблер попадает в процессе генерации макрорасширения на директиву EXITM, дальнейшее расширение прекращается и обработка продолжается после директивы ENDM.

Можно использовать EXITM для прекращения повторений по директивам REPT, IRP и IRPC даже если они находятся внутри макроопределения.

Вложенность директив условной трансляции.

TASM допускает вложенность условных директив компиляции.

Для этого предоставляется набор дополнительных директив формата ELSEIFxxx, которые заменяют последовательность подряд идущих ELSE и IFxxx в структуре:

Пример 1:

```
IFxxx
…
    ELSE
    IFxxx
…
ENDIF
ENDIF
```

Эту последовательность директив можно заменить следующей последовательностью директив:

```
IFxxx
…
    ELSEIFxxx
…
ENDIF
```

Пример 2:

```
show macro rg
ifdifi <al> <rg>
goto M_al
elseifdifi <ah>, <rg>
goto M_ah
else
exitm
endif
:M_al
…
:M_ah
…
endm
```

Директивы генерации ошибок.

Предназначены для обнаружения различных ошибок в программе, таких как неопределенные метки или пропуски параметров макроса.

Данные директивы можно разделить на два типа по принципу работы:

1. ***безусловные директивы***, генерирующие ошибку трансляции без проверки каких-либо условий;
2. ***условнее директивы***, генерирующие ошибку трансляции после проверки определенных условий.

Безусловная генерация пользовательской ошибки.

Директива **ERR** (**.ERR**).

Данная директива приводит к генерации ошибки на этапе трансляции и удалению объектного модуля.

Она очень эффективна при ее использовании с директивами условной компиляции или в теле макрокоманды с целью отладки.

Пример:

```
show macro rg
ifdifi <al>, <rg>
goto M_al
else
ifdifi <ah>, <rg>
goto M_ah
else
.err
endif
endif
…
endm
```

Если потом в сегменте кода вызвать макрокоманду show с фактическими параметром, отличных от имен регистров ah или al, будет сгенерирована ошибка компиляции, сам процесс компиляции прекращен и, естественно, объектный модуль создан не будет.

Условная генерация пользовательской ошибки.

Директивы .ERRB (ERRIFB) и .ERRNB (ERRIFNB).

Синтаксис директив:

.ERRB (ERRIFB) <имя_формального_аргумента> - генерация пользовательской ошибки, если <имя_формального_аргумента> пропущено;

.ERRNB (ERRIFNB) <имя_формального_аргумента> - генерация пользовательской ошибки, если <имя_формального_аргумента> присутствует.

Применяются для генерации ошибки трансляции в зависимости от того, задан или нет при вызове макрокоманды фактический аргумент, соответствующий формальному аргументу в заголовке макроопределения с именем <имя_формального_аргумента>. Их обычно используют для проверки задания параметров при вызове макроса.

Директивы .ERRDEF (ERRIFDEF) и .ERRNDEF (ERRIFNDEF).

Синтаксис директив:

.ERRDEF (ERRIFDEF) символическое_имя - если указанное имя определено до выдачи этой директивы в программе, то генерируется пользовательская ошибка;

.ERRNDEF (ERRIFNDEF) символическое_имя – если указанное символическое_имя не определено до момента обработки транслятором данной директивы, то генерируется пользовательская ошибка.

Данные директивы генерируют ошибку трансляции в зависимости от того, определено или нет некоторое символическое_имя в программе

Директивы .ERRDIF (ERRIFDIF) и .ERRIDN (ERRIFIDN).

Синтаксис директив:

.ERRDIF (ERRIFDIF) <строка_1>, <строка_2> - директива, генерирующая пользовательскую ошибку, если две строки посимвольно не совпадают. Строки могут быть символическими именами, числами или выражениями и должны быть заключены в угловые скобки. Учитывает различие прописных и срочных букв;

.ERRIDN (ERRIFIDN) <строка_1>, <строка_2> - директива, генерирующая пользовательскую ошибку, если строки посимвольно идентичны. Учитывает различие прописных и срочных букв.

Для игнорирования различных строчных и прописных букв, существуют аналогичные директивы:

.ERRIFDIFI <строка_1>, <строка_2> - то же, что и ERRIFDIF, но игнорируется различие строчных и прописных букв;

.ERRIFIDNI <строка_1>, <строка_2> - то же, что и ERRIFIDN, но игнорируется различие строчных и прописных букв**.**

Директивы .ERRE (ERRIFE) и .ERRNZ (ERRIF).

Синтаксис директив:

.ERRE (ERRIFE) константное_выражение – директива вызывает пользовательскую ошибку, если константное_выражение ложно (равно нулю). Вычисление константного_выражения должно приводить к абсолютному значению, и это выражение не может содержать компонентов, являющихся ссылками вперед.

.ERRNZ (ERRIF) константное_выражение – директива вызывает пользовательскую ошибку, если константное_выражение истино (не равно нулю). Вычисление константного_выражения должно приводить к абсолютному значению, и это выражение не может содержать компонентов, являющихся ссылками вперед.

Константные выражения в условных директивах.

Константные выражения должны:

1. давать абсолютный результат;
2. содержать только компоненты, являющиеся ссылками назад, а не вперед;

Кроме этого константные выражения могут приводить к чисто логическому результату с помощью операторов отношений, выражающих отношение двух значений или константных выражений.

В контексте условных директив вместе с операторами отношений можно рассматривать и логические операторы.

Так результатом работы и тех и других может быть одно из двух значений:

- **истина** – число, которое сдержит двоичные единицы во всех разрядах;
- **ложь** – число, которое содержит двоичные нули во всех разрядах.

Константные выражения в условных директивах.

Оператор отношения	Синтаксис	Результат
EQ (equal) - равно	выражение_1 **EQ** выражение_2	Истина, если выражение_1 равно выражение_2
NE (not equal) – не равно	выражение_1 **NE** выражение_2	Истина, если выражение_1 не равно выражение_2
LT (less than) – меньше	выражение_1 **LT** выражение_2	Истина, если выражение_1 меньше выражение_2
GT (greater than) – больше	выражение_1 **GT** выражение_2	Истина, если выражение_1 больше выражение_2
GE (greater or equal) – больше или равно	выражение_1 **GE** выражение_2	Истина, если выражение_1 больше или равно выражение_2
NOT – логическое отрицание	**NOT** выражение	Истина, если выражение ложно; ложь, если выражение истинно
AND – логическое И	выражение_1 **AND** выражение_2	Истина, если выражение_1 и выражение_2 истины
OR – логическое ИЛИ	выражение_1 **OR** выражение_2	Истина, если выражение_1 и выражение_2 истины
XOR – исключающее ИЛИ	выражение_1 **XOR** выражение_2	Истина, если выражение_1 = (NOT выражение_2)

Дополнительное управление трансляцией.

TASM предоставляет средства для вывода текстового сообщения во время трансляции программы – директивы **DISPLAY** и **%OUT**.

С их помощью можно при необходимости следить за ходом трансляции.

Пример:

```
display недопустимые_аргументы_макрокоманды
…
%out недопустимое_имя_регистра
```

В результате обработки этих директив на экран будут выведены тексты сообщений.

Если эти директивы использовать совместно с директивами условной компиляции, то, к примеру, можно отслеживать путь, по которому осуществляется трансляция исходного текста программы.