КОНСТРУИРОВАНИЕ ПРОГРАММ

Лекция № 05 Препроцессор NASM

+375 17 293 8039 (505a-5)

+375 17 320 7402 (ОИПИ НАНБ)

prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by/

Кафедра ЭВМ, 2022

2022.03.16

Оглавление

Препроцессор NASM

NASM содержит мощный макропроцессор, поддерживающий условное ассемблирование, многоуровневое включение файлов, две формы макро-определений (однострочные и многострочные) и механизм «контекстного стека», расширяющий возможности макро-определений (далее — макросы).

Все директивы препроцессора начинаются со знака %

- однострочные макросы;
- манипулирование строками;
- многострочные макросы;
- условное ассемблирование;
- циклы препроцессора;
- подключение других файлов (исходные файлы и зависимости);
- стек контекстов;
- директивы препроцессора для использования стека;
- Сообщения об ошибках, определяемых пользователем: %error, %warning, %fatal;
- %pragma: Установить опции;
- и многое другое.

Препроцессор сворачивает все строки, которые заканчиваются символом обратной косой черты (\), в одну строку.

Таким образом, выражение:

```
%define THIS_VERY_LONG_MACRO_NAME_IS_DEFINED_TO \
        THIS_VALUE
```

будет работать как однострочный макрос без символа обратной косой черты-новой строки.

```
%define THIS_VERY_LONG_MACRO_NAME_IS_DEFINED_TO THIS_VALUE
```

Однострочные макросы

`%define` — обычный макрос (позднее связывание)

Однострочные макросы описываются при помощи директивы препроцессора **`%define`**. Определения работают аналогично тому, как это имеет место в языке C. Например:

```
%define ctrl 0x1F &
%define param(a,b) ((a)+(a)*(b))
        mov byte [param(2,ebx)], ctrl 'D'
```

будет развернуто в

```
        mov byte [(2)+(2)*(ebx)], 0x1F & 'D'
```

Если однострочный макрос (1) содержит символы, вызывающие другой макрос (2), то развертывание первого осуществляется в процессе вызова (3), а не при определении (1):

```
%define a(x) 1+b(x)                    (1)
%define b(x) 2*x                       (2)
        mov ax, a(8)                   (3)
```

будет обработан как и ожидается

```
        mov ax, 1+2*8
```

несмотря на то, что при описании а макрос **`b`** еще не определен.
Это называется **поздним связыванием**.

Следует обратить внимание, что перед списком аргументов однострочного макроса нельзя ставить пробел. В противном случае это будет рассматриваться как развертывание. Например:

```
%define foo (a,b)    ; аргументы отсутсвуют, (a,b) — раскрытие
%define bar(a,b)     ; два аргумента, пустое раскрытие
```

Макросы, описываемые конструкцией **%define**, чувствительны к регистру — если определение имеет вид **%define foo bar**, то только **foo** будет развернуто в **bar**, а **Foo** или **F00** так и останутся нетронутыми.

```
%define foo bar
foo -> bar (развернется)
Foo -> Foo
F00 -> F00
```

При использовании вместо **%define** конструкции **%idefine** (от слова «insensitive») тот же **%idefine foo bar** будет развертывать в **bar** не только **foo**, но и **Foo**, **F00**.

```
%idefine foo bar
foo -> bar
Foo -> bar
F00 -> bar
```

Рекурсия

Существует механизм, следящий за рекурсивным развертыванием макросов, предотвращающий циклические ссылки и бесконечные циклы. Когда это случается, препроцессор будет развертывать только первое вхождение макроса:

```
%define a(x) 1+a(x)
        mov ax, a(3)
```

макрос **a(3)** будет развернут в

```
        mov ax, 1+a(3)
```

и дальнейшее развертывание производиться не будет.

Перегрузка макросов (overload)

Однострочные макросы можно *перегружать*:

```
%define foo(x)     1+x
%define foo(x,y)   1+x*y
```

Препроцессор, подсчитав передаваемые параметры, корректно обработает оба вызова макроса:

```
foo(3)     ; будет развернуто в 1+3;
foo(ebx,2) ; будет развернуто в 1+ebx*2.
```

Однако, если написать макрос без параметров

```
%define foo bar
```

все последующие описания **`foo`** будут запрещены.

Макрос без параметров не допускает описание макроса с тем же именем, но с параметрами, и наоборот.

Переопределение (redefinition) однострочных макросов

Переопределение однострочных макросов не запрещено — можно написать

```
%define foo    bar
```

и затем, в том же самом исходном файле, переопределить его как

```
%define foo    baz
```

Когда макрос **foo** будет вызван, он развернется в соответствии с самым поздним своим описанием.

Это особенно полезно при определении однострочных макросов, определенных с помощью директивы **%assign**.

(%) Замечания относительно NASM 2.15

В NASM 2.15 были добавлены следующие дополнительные функции:

Можно определить пустую строку вместо имени аргумента, если аргумент никогда не используется.

Например:

```
%define ereg(foo,) e %+ foo
        mov eax, ereg(dx, cx)
```

будет развернуто в

```
        mov eax, edx
```

Одна пара круглых скобок — это частный случай одного неиспользуемого аргумента:

```
%define myreg( ) eax
        mov edx, myreg( )
```

Результат развертывания:

```
        mov edx, eax
```

Это похоже на поведение препроцессора C.

Параметр может быть объявлен с префиксом

'=' — NASM будет после раскрытия вычислять аргумент как выражение;

'&' — параметр макроса после раскрытия будет преобразован в строку в кавычках;

'+' — жадный (поглощающий) параметр — он включает любые последующие запятые и параметры;

'!' — NASM не удаляет пробелы и фигурные скобки (полезно вместе с **'&'**).

Пример:

```
%define  xyzzy(=expr,&str) expr, str
%define  plugh(x) xyzzy(x,x)
         db plugh(3+5), `\0`       ; развернется в: db 8, "3+5", `\0`
```

```
plugh(3+5) -> xyzzy(3+5,3+5) -> 8, "3+5"
```

Однострочные макросы можно определить в процессе ассемблирования, используя параметр **-d** в командной строке.

При описании однострочных макросов можно сцеплять (concatenate) строки при помощи псевдо-оператора **%+**. Например:

```
%define _myfunc _otherfunc
%define cextern(x) _ %+ x
        cextern (myfunc)
```

После первого развертывания третья строка примет вид «**_myfunc**».

При дальнейшей обработке препроцессор развернет эту строку в «**_otherunc**».

%+ — оператор сцепления лексем (tokens) в однострочных макросах

Отдельные лексемы в однострочном макросе можно объединить, чтобы получить более длинные лексемы для последующей обработки. Это может быть полезно, если есть несколько похожих макросов, выполняющих аналогичные функции.

После **%+** требуется пробел, чтобы исключить его неоднозначность из синтаксиса **%+1**, используемого в многострочном макросе. Пример:

```
%define BDA_start 400h              ; Начало области данных BIOS (реальный режим)

struc   BDA_t                       ; струкутра области данных BIOS
        .COM1addr       RESW    1
        .COM2addr       RESW    1
        ; ..and so on
endstruc
```

Если нам нужно получить доступ к элементам **BDA** в разных местах, мы могли бы написать:

```
        mov     ax, BDA_start + BDA_t.COM1addr
        mov     bx, BDA_start + BDA_t.COM2addr
```

Если это будет использоваться во многих местах, то станет довольно уродливым и утомительным. Исходный код можно значительно уменьшить в размере, используя следующий макрос:

```
%define BDA(x)  BDA_start + BDA_t. %+ x  ; Доступ к переменным BIOS по их именам:
```

И вышеприведенный код можно переписать как:

```
        mov     ax, BDA(COM1addr)
        mov     bx, BDA(COM2addr)
```

Используя данную функцию, мы можем упростить ссылки на множество макросов и, соответственно, уменьшить количество ошибок ввода.

%xdefine — макрос раннего связывания

Чтобы ссылка на вложенный однострочный макрос раскрывалась во время определения вложенного макроса, в отличие от того случая, когда вложенный макрос раскрывается при вызове, нужен иной механизм, нежели тот, который предлагает **%define**.

Для этой цели используется **%xdefine** или его регистронезависимый аналог **%ixdefine**.

Предположим, есть следующий код:

```
%define  isTrue  1
%define  isFalse isTrue
%define  isTrue  0

val1:    db      isFalse ; -> isTrue -> 0

%define  isTrue  1

val2:    db      isFalse ; -> isTrue -> 1
```

В этом случае **val1** равно **0**, а **val2** равно **1**. Это потому, что когда однострочный макрос определяется с помощью **%define**, он раскрывается только при его вызове.

Поскольку **isFalse** расширяется до **isTrue**, расширение будет текущим значением **isTrue**. При первом вызове он равен **0**, а второй раз — **1**.

Если требуется, чтобы **isFalse** расширялся до значения, присвоенного встроенному макросу **isTrue** во время определения **isFalse**, необходимо использовать **%xdefine**.

```
%xdefine isTrue  1
%xdefine isFalse isTrue
%xdefine isTrue  0

val1:    db      isFalse ; -> 1

%xdefine isTrue  1

val2:    db      isFalse ; -> 1
```

Теперь каждый раз, когда вызывается **isFalse**, он раскрывается в **1**, поскольку это значение имел вложенный макрос **isTrue** во время определения **isFalse**.

%xdefine и **%ixdefine** поддерживают расширение аргументов точно так же, как **%define** и **%idefine**.

`%[...]` — косвенное обращение к макросу

Конструкция **`%[...]`** может использоваться для раскрытия макросов тогда, когда без нее раскрытия макроса могло бы не быть, в том числе и в именах других макросов.

Например, если есть набор макросов с именем **Foo16**, **Foo32** и **Foo64**, которые используются, соответственно, в случае 16- и, 32-х и 64-разрядного кода, для автоматического выбора между ними можно использовать встроенный макрос **`__?BITS?__`** и написать:

```
        mov ax, Foo%[__?BITS?__] ; The Foo value
```

Точно так же два оператора:

```
%xdefine Bar        Quux    ; Expands due to %xdefine
%define  Bar        %[Quux] ; Expands due to %[...]
```

по сути, имеют совершенно одинаковый же эффект.

`%[...]` сцепляется со смежными лексемами аналогично тому, как сцепляются параметры многострочных макросов.

%? and %?? — имя макроса

Для ссылки на само имя макроса внутри его раскрытия могут использоваться специальные символы `%?` и `%??`. Они работают как для однострочных, так и для многострочных макросов.

`%?` ссылается на имя макроса как оно было использовано при вызове.

`%??` ссылается на объявленное имя макроса.

Эти два параметра всегда одинаковы для макросов, учитывающих регистр, но для макросов, не учитывающих регистр, они могут отличаться. Например,

```
%imacro Foo 0               ; объявленное имя -- Foo
        mov %?, %??
%endmacro

        foo                 ; имя при вызове  -- foo
        F00                 ; имя при вызове  -- F00
```

будут раскрыты в следующие инструкции:

```
        mov foo, Foo
        mov F00, Foo
```

Для однострочных макросов можно использовать эти токены, если они определены вне любых многострочных макросов.

`%undef` - отмена определения макроса

Команда **`%undef`** удаляет однострочные макросы. Например, следующая последовательность:

```
%define     foo bar
%undef      foo
            mov eax, foo
```

будет развернута в инструкцию **`mov eax, foo`**, поскольку после **`%undef`** макрос **`foo`** больше не определен.

Отмена определения предопределенных макросов может быть осуществлена при помощи ключа '**`-u`**' командной строки NASM.

`%assign` — переменные препроцессора

Альтернативным способом определения однострочных макросов является использование директивы **`%assign`** (и ее нечувствительного к регистру эквивалента **`%iassign`**).

Данная директива используется для определения однострочного макроса без параметров, но включающего **числовое значение**.

Это значение может быть задано в форме выражения и обрабатывается оно только один раз при обработке директивы **`%assign`**.

Как и в случае **`%define`**, макрос, определенный при помощи **`%assign`**, может быть позднее переопределен, например следующая строка

```
%assign i i+1
```

увеличивает числовое значение макроса.

Выражение, передаваемое **`%assign`**, является критическим и должно на выходе давать просто число, а не различные перемещаемые ссылки наподобие адресов кода и данных.

Манипуляции со строками в макросах

Часто бывает полезно иметь возможность в макросах обрабатывать строки-литералы. NASM поддерживает несколько простых макросов для обработки строк, из которых могут быть построены более сложные операции.

Все строковые операторы определяют или переопределяют значение (строковое или числовое) для однострочного макроса. При создании строкового значения он может изменить стиль заключения в кавычки входной строки или строк, а также использовать т.н. escape-префикс `'\'` внутри строк в `'`-кавычках (одинарных), где он трактуется как обычный символ.

`%strcat` — объединение строк

Оператор **`%strcat`** сцепляет строки-литералы в кавычках и назначает их однострочному макросу. Например:

```
%strcat alpha "Alpha: ", '12" screen'
```

присвоит макросу **`alpha`** значение **`'Alpha: 12 "screen'`**. Аналогично:

```
%strcat beta '"foo"\', "'bar'"
```

присвоит макросу **`beta`** значение **`` `"foo"\\'bar' ``**.

Использование запятых для разделения строк необязательно.

Количество сцепляемых строк может быть больше двух.

%strlen — длина строки

Оператор **%strlen** назначает макросу длину строки. Например:

```
%strlen charcnt 'my string'
```

В этом примере **charcnt** получит значение 9, как если бы для присвоения значения использовалось **%assign**.

В этом примере **'my string'** представлена литералом, но она также могла быть однострочным макросом, который расширяется до строки, как в следующем примере:

```
%define sometext 'my string'
%strlen charcnt   sometext
```

Как и в первом случае, это приведет к тому, что **charcnt** будет присвоено значение 9.

%substr — извлечение подстроки

Отдельные буквы или подстроки в строках можно извлечь с помощью оператора **%substr**. Пример его использования более полезен, чем описание:

```
%substr mychar 'xyzw' 1       ; эквивалентно %define mychar 'x'
%substr mychar 'xyzw' 2       ; эквивалентно %define mychar 'y'
%substr mychar 'xyzw' 3       ; эквивалентно %define mychar 'z'
%substr mychar 'xyzw' 2,  2   ; эквивалентно %define mychar 'yz'
%substr mychar 'xyzw' 2, -1   ; эквивалентно %define mychar 'yzw'
%substr mychar 'xyzw' 2, -2   ; эквивалентно %define mychar 'yz'
```

Первый параметр — это создаваемый однострочный макрос, а второй — строка. Третий параметр указывает первый выбираемый символ, а необязательный четвертый параметр, которому предшествует запятая — это длина.

Следует обратить внимание, что первый индекс равен 1, а не 0, а последний индекс равен значению, которое **%strlen** присвоил бы этой строке.

Значения индекса вне диапазона приводят к пустой строке.

Отрицательная длина означает «до N-1 символа от конца строки», т.е. -1 означает до конца строки, -2 до одного символа до и т.д.

Многострочные макросы: `%macro`

Определение многострочного макроса

Определение многострочного макроса в NASM выглядит следующим образом:

```
%macro prologue 1        ; 1 – число параметров
          push ebp
          mov  ebp,esp
          sub  esp,%1
%endmacro
```

что определяет макрос **`prologue`**, который можно вызвать следующим образом:

```
myfunc:
          prologue 12
```

что в результате будет развернуто в три строки кода

```
myfunc:   push ebp
          mov  ebp,esp
          sub  esp,12
```

Число **`1`** после имени макроса в строке **`%macro`** определяет число параметров, которые ожидает получить макрос **`prologue`**.

Конструкция **`%1`** внутри тела макроса ссылается на первый передаваемый параметр.

В случае макросов, принимающих более одного параметра, ссылка на них осуществляется как **`%2`**, **`%3`** ...

Многострочные макросы, как и однострочные, чувствительны к регистру символов, для устранения чего используется альтернативная директива **%imacro**.

Параметры при вызове макроса перечисляются через запятую.

Если необходимо передать в многострочный макрос запятую в качестве составной части параметра, это можно сделать это, заключив параметр в фигурные скобки:

```
%macro foobar 2
%2:       db %1
%endmacro
          foobar 'a', letter_a    ; letter_a:  db 'a'
          foobar 'ab', string_ab  ; string_ab: db 'ab'
          foobar {13,10}, crlf    ; crlf:      db 13,10
```

Перегрузка многострочных макросов

Многострочные макросы, как и однострочные, могут быть перегружены путем определения одного и того же имени несколько раз с разным числом параметров.

В отличие от однострочных макросов, перегружать можно и макросы без параметров.

Для определения альтернативной формы функции-пролога, не выделяющей место в стеке, можно написать

```
%macro prologue 0
          push ebp
          mov  ebp, esp
%endmacro
```

В ряде случаев бывает удобным как бы «*перегрузить*» процессорные инструкции, например, если определить

```
%macro push 2
          push %1
          push %2
%endmacro
```

позже можно написать

```
          push ebx                ; эта строка не вызов макроса! -> Warning
          push eax, ecx           ; а это — вызов (push ecx)
```

По умолчанию NASM будет выдавать предупреждение при обработке первой из этих строк, так как **`push`** теперь определена как макрос и вызывается с числом параметров, определения для которого нет. Однако при этом будет сгенерирован корректный код — ассемблер всего лишь даст предупреждение.

Данное предупреждение можно отключить при помощи ключа командной строки **`-w-macro-params`**.

Локальные метки в макросах (18.03.2022)

NASM позволяет внутри определения многострочного макроса описать метки, которые останутся локальными для каждого вызова макроса — при многократном вызове макроса имена меток каждый раз будут изменяться.

Описание такой метки осуществляется при помощи префикса `%%`.

Например, можно описать инструкцию, выполняющую **RET** если установлен флаг **Z**:

```
%macro retz 0
          jnz %%skip
          ret
%%skip:
%endmacro
```

После этого можете вызывать этот макрос столько раз, сколько нужно, и при этом в каждом вызове вместо метки **`%%skip`** NASM будет подставлять разные имена.

NASM создает имена в форме

```
..@2345.skip
```

где число **2345** изменяется при каждом вызове макроса.

Префикс `..@` предотвращает пересечение имен локальных меток макросов с механизмом обычных локальных меток.

В связи с этим нужно избегать определения собственных меток в вышеприведенной форме (префикс `..@`, затем число, затем точка), иначе они могут совпасть с локальными метками макросов.

Поглощающие параметры макросов

Иногда полезно определить макрос, собирающий всю строку параметров в один параметр после извлечения одного или двух небольших параметров из начала очереди.

Пусть есть макрос, записывающий строку в файл, который можно вызвать, например, так:

```
writefile [filehandle],"Привет, мир!",13,10
```

NASM позволяет определить последний параметр макроса в качестве *поглощающего*.

Это означает, что если вызвать макрос с большим числом параметров, чем ожидалось, все «лишние» параметры вместе с разделительными запятыми присоединятся к последнему ожидаемому параметру:

```
%macro writefile 2+
          jmp %%endstr
%%str:    db  %2
%%endstr: mov dx, %%str
          mov cx, %%endstr - %%str
          mov bx, %1
          mov ah, 0x40
          int 0x21                ; DOS
%endmacro
```

то приведенный выше пример вызова макроса **`writefile`** будет работать как нужно — текст перед первой запятой **`[filehandle]`** используется в качестве первого параметра и развернется, когда встретится ссылка **`%1`**, весь последующий текст объединится в **`%2`** и расположится после **`db`**.

Поглощающая природа макроса указывается в NASM при помощи знака (**+**) после количества параметров в строке **`%macro`**.

Определение поглощающего макроса говорит NASM, как он должен разворачивать любое число параметров свыше явно указанного.

NASM также это учитывает при перегрузке макросов и не позволит определить другую форму **`writefile`**, принимающую, например, 4 параметра.

Приведенный выше макрос может быть реализован и обычным образом (не как поглощающий). В этом случае его вызов должен выглядеть так:

```
writefile [filehandle], {"Привет, мир!",13,10}
```

NASM поддерживает оба механизма помещения запятых в параметры макроса и позволяет программисту выбирать, какой предпочтительнее в каждом конкретном случае.

Диапазон параметров макроса

Если макрос может принимать разное количество параметров, это указывается в поле количества параметров как **x-y**, где **x** —минимальное количество параметров, **y** — максимальное. Таким обарзом, объявление **%macro foo 2-4** говорит о том, что при вызове макроса можно указать 2, 3 или 4 параметра.

Максимальное число параметров может быть неограниченным, что обозначается как *.

В этом случае полный набор параметров по умолчанию предусмотреть невозможно.

NASM позволяет раскрывать любое количество параметров с помощью специальной конструкции **%{x:y}**, где **x** — это первый индекс параметра, а **y** — последний. Любой индекс может быть отрицательным или положительным, но никогда не должен быть нулевым. Например:

```
%macro mpar 1-*
     db %{3:5}
%endmacro

mpar 1,2,3,4,5,6
```

будет развернуто в диапазон 3, 4, 5.

Более того, параметры можно поменять местами, чтобы макрос

```
%macro mpar 1-*
     db %{5:3}
%endmacro

mpar 1,2,3,4,5,6
```

развернулся в 5, 4, 3.

К параметрам также можно обращаться с помощью отрицательных индексов, и в этом случае NASM будет перечислять их в обратном порядке.

```
%macro mpar 1-*
     db %{-1:-3}
%endmacro

mpar 1,2,3,4,5,6
```

развернется в даипазон 6, 5, 4.

Следует обратить внимание, что NASM использует запятую для разделения раскрываемых параметров.

Между прочим, есть уловка — чтобы дать доступ к последнему аргументу, переданному макросу, следует использовать индекс **`%{-1:-1}`**.

Параметры макросов по умолчанию

NASM позволяет определять многострочный макрос с указанием ***диапазона*** допустимого количества параметров. Если эта возможность используется, можно задать значения по умолчанию для пропущенных параметров.

Например макрос **die** для завершения программы с выводом в **stderr** (дескриптор с номером 2) сообщения об ошибке :

```
%macro die 0-1 "Аварийно завершаемся :-("
          writefile 2,%1
          mov ax, 0x4c01
          int 0x21
%endmacro
```

Данный макрос (использующий ранее описанный макрос **writefile**) может быть вызван как с явно указанным сообщением об ошибке, помещаемым в выходной поток перед закрытием, так и без параметров. В последнем случае в качестве параметра будет подставлено сообщение по умолчанию, указанное при определении макроса.

Для макроса данного типа задается минимальное и максимальное число параметров. В вышеприведенном случае минимальное число — это параметры, требующиеся при вызове макроса, а для остальных задаются значения по умолчанию.

Если, например, определение макроса **foobar** начинается со строки

```
%macro foobar 1-3 eax,[ebx+2]
```

то он может быть вызван с числом параметров от одного до трех, при этом
- параметр **%1** всегда берется из строки вызова макроса;
- параметр **%2**, если не задан, примет значение **eax**;
- параметр **%3**, если не задан, — значение **[ebx+2]**.

Можно предоставить макросу дополнительную информацию, указав слишком много параметров по умолчанию:

```
%macro quux 1 something
```

По умолчанию это вызовет предупреждение. Когда вызывается **quux**, он получает не один, а два параметра. К **something** в этом случае можно обращаться, как **%2**.

Разница между передачей **something** таким образом и записью **something** в теле макроса заключается в том, что таким образом **something** вычисляется при определении макроса, а не при его раскрытии.

Значения по умолчанию при определении макроса задавать не обязательно — в таком случае они будут пустые. Это может быть полезно для макросов, принимающих различное число параметров, так как число реально передаваемых параметров вы можете указать при помощи конструкции **%0**.

Механизм параметров по умолчанию может комбинироваться с механизмом поглощающих параметров. В этом случае вышеприведенный макрос **die** можно сделать гораздо более мощным и полезным, просто изменив первую строку определения на

```
%macro die 0-1+ "Таки, аварийно завершаемся :-(",13,10
```

`%rep, %exitrep, %endrep` — препроцессорные циклы

Префикс TIMES в NASM удобен, но он не может быть использован в многострочных макросах, потому как обрабатывается уже после полного разворачивания последних.

Вследствие этого, NASM предусматривает другую форму циклов, работающих на уровне препроцессора, а именно **`%rep`**.

Директивы **`%rep`** и **`%endrep`** используются для заключения в них куска кода, который при этом реплицируется столько раз, сколько указано препроцессору.

`%rep` принимает числовой аргумент или выражение,

`%endrep` не принимает никаких аргументов.

```
%assign i 0
%rep 64
          inc word [table+2*i]
%assign i i+1
%endrep
```

Этот пример будет генерировать 64 инструкции **`INC`**, инкрементируя каждое слово в памяти от **`[table]`** до **`[table+126]`**.

Для образования более сложных условий окончания цикла или его принудительного завершения можно использовать директиву **`%exitrep`**.

Пример создает список всех чисел Фибоначчи, вписывающихся в размер 16 бит.

```
fibonacci:
%assign i 0
%assign j 1
%rep 100
%if j > 65535
%exitrep
%endif
          dw j
%assign k j+i
%assign i j
%assign j k
%endrep
fib_number equ ($-fibonacci)/2
```

В этом случае для **%rep** должно быть указано максимальное число повторов.

Это необходимо для того, чтобы NASM на стадии препроцессирования не впал в бесконечный цикл, что приводит обычно к быстрому исчерпанию памяти и невозможности запуска других приложений.

%0 — счетчик макро-параметров

Параметр **`%0`** возвращает числовую константу, представляющую собой количество передаваемых в макрос параметров.

Он может использоваться как аргумент для **`%rep`** с целью перебора всех параметров макроса.

%rotate — «вращение» параметров макросов

Командная оболочка U*x имеет команду **shift**, позволяющую «сдвигать» переданные скриптом оболочки аргументы (**$1**, **$2** и т.д.) так, что аргумент, имевший до этого ссылку **$2**, получает ссылку **$1**, а имевший ссылку **$1**, становится недоступен.

NASM обеспечивает подобный механизм при помощи команды **%rotate**.

Как следует из имени, эта команда отличается от команды **shift** тем, что параметры не теряются — «выталкиваемые» с левого конца списка аргументов, они появляются на правом, и наоборот.

%rotate вызывается с одним числовым аргументом (который может быть выражением). Параметры макроса «вращаются» влево на количество мест, указанных аргументом. Если аргумент отрицателен, параметры вращаются вправо. Например, пара макросов для сохранения и восстановления набора регистров может работать так:

```
%macro multipush 1-* ; принимает один и более параметров (1-*)
  %rep %0            ; и начинает макроцикл повторения
          push %1
    %rotate 1        ; сдвигаем параметры к следующему элементу
  %endrep            ; и переходим к его подстановке
%endmacro
```

Этот макрос вызывает инструкцию **PUSH** для каждого аргумента, слева-направо.

Удобно иметь и обратный макрос **multipop**, извлекающий регистры из стека.
Все это может быть реализовано при помощи следующего определения:

```
%macro multipop 1-*
  %rep %0
    %rotate -1
          pop %1
  %endrep
%endmacro
```

Данный макрос начинает работу с поворота своих аргументов вправо, поэтому исходный последний аргумент становится аргументом **%1**. Затем этот аргумент извлекается из стека, список аргументов снова поворачивается вправо и теперь аргументом **%1** становится аргумент, бывший в начале предпоследним. Таким образом, аргументы извлекаются из стека в обратном порядке.

Объединение параметров макросов

NASM может объединять параметры макросов с окружающим их текстом. Это позволяет при определении макроса объявлять, например семейства символов.

Например, если необходимо создать таблицу кодов клавиш вместе со смещениями в этой таблице, вы можете написать:

```
%macro keytab_entry 2
keypos_%1 equ $-keytab
          db %2
%endmacro
keytab:
          keytab_entry F1, 128 + 1
          keytab_entry F2, 128 + 2
          keytab_entry Return, 13
```

Это будет развернуто следующим образом:

```
keytab:
keypos_F1 equ $-keytab
          db 128 + 1
keypos_F2 equ $-keytab
          db 128 + 2
keypos_Return equ $-keytab
          db 13
```

Точно также можно присоединять текст к другому концу параметра, написав **`%1foo`**.

Если необходимо присоединить к параметру макроса цифру, например для определения меток **foo1** и **foo2** при передаче параметра **foo**, нельзя написать **%11**, так как это будет воспринято как одиннадцатый параметр макроса.

Вместо этого необходимо написать **%{1}1**, где первая единица, определяющая номер параметра макроса, будет отделена от второй, представляющей собой присоединяемый к параметру текст.

```
%macro foobar 1
%11         resb 8   ; не генерирует правильное имя
%{1}1       resb 8   ; генерирует правильное имя
%endmacro

section    .bss
foobar foo           ; foo1

section    .text
           mov  [foo1], eax  ; error: symbol `foo1' not defined
```

Объединение может быть применено к другим встраиваемым объектам препроцессора, таким как локальные метки макросов и контекстно-локальные метки.

В любом случае, неопределенность синтаксиса может быть разрешена путем заключения всего, находящегося после знака % и перед присоединяемым текстом, в фигурные скобки.

Например, **%{%label}_text** прицепит текст **text** к действительному имени локальной метки **%%label**.

Коды условий в качестве параметров макросов

NASM может *особым образом* обрабатывать параметры макросов, содержащие коды условий.

Во первых, можно ссылаться на макро-параметр **%1** при помощи альтернативного синтаксиса **%+1**, информирующего NASM о том, что этот параметр содержит код условия и заставляющего препроцессор сообщать об ошибке, если макрос вызывается с параметром, не являющимся действительным кодом условия.

Более полезной возможностью является ссылка на макро-параметр вида **%-1**, которую NASM будет разворачивать как обратный код условия. Так, макрос, описанный в разделе «Локальные метки в макросах (18.03.2022)»

```
%macro retz 0         ; Выйти, если ZF установлен
          jnz %%skip
          ret
%%skip:
%endmacro
```

может быть заменен макросом условного возврата более общего вида:

```
%macro retc 1           ; параметр может принимать любой символьный код
          j%-1 %%skip   ; из мнемоники инструкции Jcc
          ret
%%skip:
%endmacro
```

Данный макрос может быть теперь вызван, например, как **retc ne**, что будет развернуто в инструкцию условного перехода **JE**, или как **retc po**, что будет преобразовано в переход **JPE**.

Ссылка **%+1** на параметр может интерпретировать аргументы **CXZ** и **ECXZ** как правильные коды условий, однако если передать эти аргументы ссылке **%-1**, будет сообщено об ошибке, так как обратных кодов условий к таким параметрам не существует.

Коды условия

CC	**Условие**		
E / Z	**E**qual / **Z**ero	равно / ноль	ZF=1
NE / NZ	**N**ot **E**qual / **N**ot **Z**ero	не равно / не ноль	ZF=0
A / NBE	**A**bove / **N**ot (**B**elow or **E**qual)	выше / не (ниже или равно)	CF=0 & ZF=0
AE / NB	**A**bove or **E**qual / **N**ot **B**elow	выше или равно / не ниже	CF=0
B / NAE	**B**elow / **N**ot (**A**bove or **E**qual)	ниже/ не (выше или равно)	CF=1
BE / NA	**B**elow or **E**qual / **N**ot **A**bove	ниже или равно / не выше	CF=1 | ZF=1
G / NLE	**G**reater / **N**ot (**L**ess or **E**qual)	больше / не (меньше или равно)	ZF=0 & SF=OF
GE / NL	**G**reater or **E**qual / **N**ot **L**ess	больше или равно / не меньше	SF=OF
L / NGE	**L**ess / **N**ot **G**reater or **E**qual	меньше / не (больше или равно)	SF≠OF
LE / NG	**L**ess or **E**qual / **N**ot **G**reater	меньше или равно / не больше	ZF=1 | SF≠OF
C	**C**arry	перенос	CF=1
NC	**N**o **C**arry	отсутствие переноса	CF=0
O	**O**verflow	переполнение	OF=1
NO	**N**o **O**verflow	отсутствие переполнения	OF=0
S	**S**ign (negative)	знак	SF=1
NS	**N**o **S**ign (non-negative)	отсутствие знака	SF=0
P/PE	**P**arity / **P**arity **E**ven	паритет / четно	PF=1
NP/PO	**N**o **P**arity / **P**arity **O**dd	отсутствие паритета / нечетно	PF=0

Подавление развертывания макросов в листинге

Когда NASM генерирует из программы файл листинга, он обычно разворачивает многострочные макросы посредством записи макровызова и последующим перечислением каждой строки, полученной в результате развертывания. Это позволяет увидеть, как генерируются инструкции из макроса в реальный код.

Однако для некоторых макросов данное загромождение листинга не требуется.

NASM предусматривает для этой цели спецификатор **`.nolist`**, который можно включить в определение макроса с целью подавления развертывания макроса в файле листинга.

Спецификатор `.nolist` ставится сразу после количества параметров:

```
%macro foo 1.nolist
```

Или так:

```
%macro bar 1-5+.nolist a,b,c,d,e,f,g,h
```

иначе выйдет:

```
warning: too many default macro parameters in macro `foo' [-w+macro-defaults]
```

`%unmacro` — отмена определения многострочного макроса

Многострочные макросы можно удалить с помощью директивы **`%unmacro`**.

Однако, в отличие от директивы **`%undef`**, **`%unmacro`** принимает спецификации аргументов и удаляет только точные совпадения с данной спецификацией.

Например:

```
%macro foo 1-3
        ; Do something
%endmacro
%unmacro foo 1-3
```

удаляет ранее определенный макрос **`foo`**, но

```
%macro bar 1-3
        ; Do something
%endmacro
%unmacro bar 1
```

не удаляет макрос **`bar`**, так как спецификации его аргументов точно не совпадает с указанными.