Лекция 13

3.3 Макрокоманды

Для коротких фрагментов программ их организация в виде подпрограмм (процедур) может быть неэффективной. Например, добавление содержания двух слов и результат записывается в третье слово может быть реализовано так:

```
ADD_WORDS PROC
MOV AX, BX
ADD AX, CX
RET
ADD  WORDS ENDP
```

До этого добавляются три команды передачи параметров - две до вызова процедуры и одна после:

```
MOV BX, ARG1
MOV CX, ARG2
CALL ADD_WORDS
MOV SUM, AX
```

Как видим, на две команды нужно дополнительно еще 5 команд, из которых вызов процедуры реализуется как 2 команды **PUSH** и **JMP**. Следовательно, такая реализация неэффективна. Лучше просто записать 5 команд в определенном месте.

Но в макроассемблере существует для этого случая еще одно средство - макрокоманды. Макрокоманды позволяют обращаться к группе команд как к одной.

Для этого они предварительно должны быть записаны в виде макроопределения.

Макроопределение состоит из:

- Заглавие;
- Тело;
- Заключительная строка

Заглавие состоит из имен, ключевого слова MACRO и списка формальных параметров.

Имя MACRO список параметров

Тело - это последовательность команд макроассемблера.

Список формальных параметров может отсутствовать. Если формальные параметры отмечены, то тело макроопределения их использует.

Заключительная строка макроопределения имеет вид:

ENDM ; без указания имени макроопределения.

Следовательно, приведенный выше фрагмент может быть реализован в виде макроопределения:

```
ADD_WORDS MACRO ARG1, ARG2, SUM
MOV AX, ARG1
ADD AX, ARG2
MOV SUM, AX
ENDM
```

Макроопределения размещаются в любом месте программы, но к вызову макрокоманды.

Обращение к макрокоманде имеет вид:

Имя список факт.

В нашем случае:

```
ADD_WORDS TERM1, TERM2, COST
```

Ясно, что таких обращений в программе может быть несколько. Как выполняется макрокоманда? На место вызова макрокоманды вставляется тело макроопределения с заменой формализированных параметров фактическими.

То есть, вместо одной строки обращения будет размещено 3 строки:

```
MOV AX, TERM1
ADD AX, TERM2
MOV COST, AX
```

В результате этого транслятор создает макрорасширение в каждом месте, где был вызов макрокоманды. То есть, программа будто расширяется, увеличивается. После этого макроопределения не нужно и может быть изъято. Дальше скомпонованная программа выполняется последовательно без переходов к другим частям. То есть, скорость выполнения будет высокой, но использование макрокоманд приводит к увеличению объема программы.

Таким образом видим, что вызов макрокоманды это есть **НЕ** указание микропроцессору, а директива транслятору.

Макрокоманды динамические: за счет изменения параметров можно и объекты, и сами действия. Для процедур можно изменять лишь объекты.

Макрокоманды выполняются быстрее процедур, нет потребности в переходах и возвращениях. Но использование макрокоманд увеличивает объем памяти: в теле программы макрорасширения дублируются столько раз, сколько были вызваны. Процедура же в памяти записывается один раз.

Макроопределение можно записать в библиотеку и использовать при разработке новых программ.

3.3.1 Псевдооператоры макроассемблера

Разделяются на 4 группы:

1. общего назначения
2. повторение
3. условные
4. управление листингом

3.3.1.1. Псевдооператоры общего назначения

Директива **LOCAL**

Пусть, есть макроопределение, где какое-то слово уменьшается до 0, чем реализуется определенная задержка.

```
DELAY_1 MACRO COUNT
LOCAL NEXT
.....
PUSH CX
MOV CX, COUNT
NEXT: LOOP NEXT
POP CX
ENDM
```

Если к этой макрокоманде обращаются в программе несколько раз, то в каждом макрорасширении появится метка NEXT, что недопустимо, потому что каждая метка должна быть уникальной.

Для того, чтобы в каждом расширении создавались уникальные метки, нужно применить директиву **LOCAL**.

После этого первый раз будет создана метка ??0000, второй -- ??0001, третий -- ??0002 и т.д. Директива **LOCAL** размещается сразу за заглавием макрорасширения.

3.3.1.2. Псевдооператоры повторения

Предназначенные для повторения нескольких команд, которые начинаются заглавием и заканчиваются словом **ENDM**.

Существует 4 вида таких псевдооператоров:

1. **WHILE;**
2. **REPT;**
3. **IRP;**
4. **IRPC**

Директивы **WHILE** и **REPT** применяют для повторения определенное количество раз некоторой последовательности строк.

WHILE константное_выражение
последовательность_строк
ENDM

При использовании директивы **WHILE** макрогенератор транслятора будет повторять **последовательность_строк** до тех пор, пока значение **константное_выражение** не станет равно *нулю*. Это значение вычисляется каждый раз перед очередной итерацией цикла повторения (то есть значение **константное_выражение** должно подвергаться изменению внутри **последовательность_строк** в процессе макрогенерации).

REPT константное_выражение
последовательность строк
ENDM

Директива REPT, подобно директиве WHILE, повторяет последовательность_строк столько раз, сколько это определено значением **константное_выражение** и автоматически уменьшает на единицу значение **константное_выражение** после каждой итерации.

Например:

```
;Использование директив повторения
;prg_13_3.asm
def_sto_1 macro id_table,ln:=<5>
;макрос резервирования памяти длиной len.
;Используется WHILE
 id_table label byte
  len=ln
  while len
  db 0
  len=len-1
 endm
endm
 def_sto_2 macro id_table,len
;макрос резервирования памяти длиной len
  id_table label byte
  rept len
  db 0
 endm
endm

data segment para public 'data'
 def_sto_1 tab_1, 10
 def_sto_2 tab_2, 10
data ends
end
```

```
   21   0000                    data segment para public 'data'
   22                            def_sto_1  tab_1, 10
1  23  0000                     tab_1 label byte
2  24  0000  00                 db 0
2  25  0001  00                 db 0
2  26  0002  00                 db 0
2  27  0003  00                 db 0
2  28  0004  00                 db 0
2  29  0005  00                 db 0
2  30  0006  00                 db 0
2  31  0007  00                 db 0
2  32  0008  00                 db 0
2  33  0009  00                 db 0
   34                            def_sto_2  tab_2, 10
1  35  000A                     tab_2 label byte
2  36  000A  00                 db 0
2  37  000B  00                 db 0
2  38  000C  00                 db 0
2  39  000D  00                 db 0
2  40  000E  00                 db 0
2  41  000F  00                 db 0
2  42  0010  00                 db 0
2  43  0011  00                 db 0
2  44  0012  00                 db 0
2  45  0013  00                 db 0
   46                          data ends
```

IRP формальный_аргумент, <строка_символов_1, ..., строка_символов_N>
последовательность_строк
ENDM

Действие данной директивы заключается в том, что она повторяет **последовательность_строк** ***N*** раз, то есть столько раз, сколько **строк_символов** заключено в угловые скобки во втором операнде директивы **IRP**. Повторение **последовательности_строк** сопровождается заменой в ней **формального_аргумента** строкой символов из второго операнда.

Так, при первой генерации **последовательности_строк формальный_аргумент** в них заменяется на **строка_символов_1**.

Если есть **строка_символов_2**, то это приводит к генерации второй копии **последовательности_строк**, в которой **формальный_аргумент** заменяется на **строка_символов_2**. Эти действия продолжаются до **строка_символов_N** включительно.

Например:

```
IRP ini,<1,2,3,4,5>
 db ini
endm
```

Макрогенератором будет сгенерировано следующее макрорасширение:

```
db 1
db 2
db 3
db 4
db 5
```

IRPC формальный_аргумент, строка_символов
последовательность строк
ENDM

Действие данной директивы подобно **IRP**, но отличается тем, что она на каждой очередной итерации заменяет **формальный_аргумент** очередным символом из **строка_символов**.

Понятно, что количество повторений **последовательность_строк** будет определяться количеством символов в **строка_символов**.

Например:

```
irpc char,HELLO
      db char
endm
```

В процессе макрогенерации эта директива развернется в следующую последовательность строк:

```
DB 'H'
DB 'E'
DB 'L'
DB 'L'
DB 'O'
```

```
DB H
DB E
DB L
DB L
DB O
```

Нужно отметить, что псевдооператоры повторения несколько отличаются от команды **LOOP** и префикса **REP**. **LOOP** выполняется подобно процедуре, то есть, управление передается на начало цикла. Директивы повторения выполняются подобно макрокомандам: дублируется фрагмент соответствующее число раз.

3.3.1.3. Условные псевдооператоры

Макроассемблер поддерживает несколько условных директив, которые полезны внутри макроопределений. Каждая директива **IF** должна иметь ее соответствующую директиву **ENDIF** для завершения условного блока и возможно внутри **ELSE**.

То есть, структура условного блока:

IFxxx логическое_выражение_или_аргументы
фрагмент_программы_1
ELSE
фрагмент_программы_2
ENDIF

Всего имеется **10** типов условных директив компиляции. Их логично попарно объединить в четыре группы:

1. Директивы **IF** и **IFE** —по результату вычисления логического выражения;
2. Директивы **IFDEF** и **IFNDEF** —по факту определения символического имени;
3. Директивы **IFB** и **IFNB** —по факту определения фактического аргумента при вызове макрокоманды;
4. Директивы **IFIDN**, **IFIDNI**, **IFDIF** и **IFDIFI** —по результату сравнения строк символов.

IF(E) логическое_выражение
фрагмент_программы_1
ELSE
фрагмент_программы_2
ENDIF

Обработка этих директив макроассемблером заключается в вычислении логического_выражения и включении в объектный модуль **фрагмент_программы_1** или **фрагмент_программы_2** в зависимости от того, в какой директиве **IF** или **IFE** это выражение встретилось:

- если в директиве **IF** логическое выражение *истинно*, то в объектный модуль помещается **фрагмент_программы_1**.

Если логическое выражение *ложно*, то при наличии директивы **ELSE** в объектный код помещается **фрагмент_программы_2**. Если же директивы **ELSE** нет, то вся часть программы между директивами **IF** и **ENDIF** игнорируется и в объектный модуль ничего не включается. Кстати сказать, понятие истинности и ложности значения **логического_выражения** весьма условно. Ложным оно будет считаться, если его значение равно нулю, а истинным — при любом значении, отличном от нуля.

- директива **IFE** аналогично директиве **IF** анализирует значение логического_выражения. Но теперь для включения **фрагмент_программы_1** в объектный модуль требуется, чтобы **логическое_выражение** имело значение “*ложь*”.

Директивы IF и IFE очень удобно использовать при необходимости изменения текста программы в зависимости от некоторых условий.

```
<1>...
<2>debug equ 1
<3>...
<4>.code
<5>...
<6>if debug
<7> ;любые команды и директивы ассемблера
<8> ;(вывод на печать или монитор)
<9>endif
```

IF(N)DEF символическое_имя
фрагмент_программы_1
ELSE
фрагмент_программы_2
ENDIF

Данные директивы позволяют управлять трансляцией фрагментов программы в зависимости от того, определено или нет в программе некоторое символическое_имя. Директива **IFDEF** проверяет, описано или нет в программе символическое_имя, и если это так, то в объектный модуль помещается **фрагмент_программы_1**. В противном случае, при наличии директивы **ELSE**, в объектный код помещается **фрагмент_программы_2**.

Если же директивы **ELSE** нет (и **символическое_имя** в программе не описано), то вся часть программы между директивами **IF** и **ENDIF** игнорируется и в объектный модуль не включается.

Действие **IFNDEF** обратно **IFDEF**. Если **символического_имени** в программе нет, то транслируется **фрагмент_программы_1**. Если оно присутствует, то при наличии **ELSE** транслируется **фрагмент_программы_2**. Если **ELSE** отсутствует, а **символическое_имя** в программе определено, то часть программы, заключенная между **IFNDEF** и **ENDIF**, игнорируется.

Пример:

Рассмотрим ситуацию, когда в объектный модуль программы должен быть включен один из трех фрагментов кода. Какой из трех фрагментов будет включен в объектный модуль, зависит от значения некоторого идентификатора sw:

1. если sw = 0, то сгенерировать фрагмент для вычисления выражения y = x*2**n;
2. если sw = 1, то сгенерировать фрагмент для вычисления выражения y = x/2**n;
3. если sw не определен, то ничего не генерировать.

```
IFNDEF sw   ;если sw не определено, то выйти из макроса
 EXITM
else        ;иначе — на вычисление
 movcl,n
 ife sw
  sal x,cl   ;умножение на степень 2 сдвигом влево
 else
  sar x,cl ;деление на степень 2 сдвигом вправо
 endif
ENDIF
```

IF(N)B аргумент
фрагмент_программы_1
ELSE
фрагмент_программы_2
ENDIF

Данные директивы используются для проверки фактических параметров, передаваемых в макрос. При вызове макрокоманды они анализируют значение аргумента, и в зависимости от того, равно оно пробелу или нет, транслируется либо **фрагмент_программы_1**, либо **фрагмент_программы_2**. Какой именно фрагмент будет выбран, зависит от кода директивы:

Директива **IFB** проверяет *равенство* аргумента *пробелу.* В качестве аргумента могут выступать имя или число.

- Если его значение равно пробелу (то есть фактический аргумент при вызове макрокоманды не был задан), то транслируется и помещается в объектный модуль **фрагмент_программы_1**.
- В противном случае, при наличии директивы **ELSE**, в объектный код помещается **фрагмент_программы_2**. Если же директивы **ELSE** нет, то при равенстве аргумента пробелу вся часть программы между директивами **IFB** и **ENDIF** игнорируется и в объектный модуль НЕ включается.

Действие **IFNB** обратно **IFB**. Если значение аргумента в программе НЕ РАВНО ПРОБЕЛУ, то транслируется **фрагмент_программы_1**.

Пример: строки в макроопределении, которые будут проверять, указывается ли фактический аргумент при вызове соответствующей макрокоманды:

```
Show macro reg
 IFB
  display 'не задан регистр'
  exitm
 ENDIF
...
endm
```

Если теперь в сегменте кода вызвать макрос *show* без аргументов, то будет выведено сообщение о том, что не задан регистр и генерация макрорасширения будет прекращена директивой **exitm**.

IFIDN(I) аргумент_1, аргумент_2
фрагмент_программы_1
ELSE
фрагмент_программы_2
ENDIF

В этих директивах проверяются **аргумент_1** и **аргумент_2** как строки символов. Какой именно код — **фрагмент_программы_1** или **фрагмент_программы_2** — будет транслироваться по результатам сравнения, зависит от кода директивы.

Парность этих директив объясняется тем, что они позволяют учитывать, либо не учитывать различие строчных и прописных букв. Так, директивы **IFIDNI** и **IFDIFI** *игнорируют* это различие, а **IFIDN** и **IFDIF** — учитывают.

Директива **IFIDN(I)** сравнивает символьные значения **аргумент_1** и **аргумент_2**.

Если результат сравнения положительный (>0), то **фрагмент_программы_1** транслируется и помещается в объектный модуль. В противном случае, при наличии директивы **ELSE**, в объектный код помещается **фрагмент_программы_2**.

Если же директивы **ELSE** нет, то вся часть программы между директивами **IFIDN(I)** и **ENDIF** игнорируется и в объектный модуль не включается.

IFDIF(I) аргумент_1, аргумент_2
фрагмент_программы_1
ELSE
фрагмент_программы_2
ENDIF

Действие **IFDIF(I)** обратно **IFIDN(I)**.

Если результат сравнения отрицательный (<0) ***(строки не совпадают***), транслируется **фрагмент_программы_1**. В противном случае все происходит аналогично рассмотренным ранее директивам.

Как мы уже упоминали, эти директивы удобно применять для проверки фактических аргументов макрокоманд.

Пример: Проверим, какой из регистров — **al** или **ah** — передан в макрос в качестве параметра (проверка проводится без учета различия строчных и прописных букв):

```
make_signed_word   macro   signed_byte
IFDIFI   <al>, <signed_byte>        ; убедиться, что операндом НЕ является al
        mov   al, signed_byte
ENDIF
        cwb
        endm
```

3.3.1.4. Операции в макроопределениях

Есть **4** операции в макроопределениях.

1. ;; -- комментарий, который не включается в листинг;
2. & -- конкатенация;
3. Переопределение;
4. Отмена.

& -- конкатенация, позволяет задавать модифицированные метки и операторы;

Например:

```
DEF_TABLE MACRO SUFFIX, LENGTH
TABLE& SUFFIX DB LENGTH DUP(?)
ENDM
```

```
DEF_TABLE A,5
……..
TABLE A DB 5 DUP(?)
```

Если отметить вызов DEF_TABLE A,5, то создастся расширение
TABLE A DB 5 DUP(?)

Переопределение. Если в программе **описать** макрокоманду **с тем же именем**, которое было раньше у определенной макрокоманды, то предыдущее ее определение уже НЕ действует:

```
A MACRO Y
 INC Y
 INC BX
ENDM
A BX
A MACRO X, Z
 CMP X, 0
 CMP BH, 0
 JE Z
 JE EL
ENDM
A BM, EL
```

Отмена. Макроопределение можно уничтожить директивой
PURGE <имя макроса>
Из этого момента обращаться к отмеченной макрокоманде нельзя.

3.3.1.5. Использование библиотек макрокоманд

Кроме макрокоманд, определенных в программе, можно использовать макрокоманды из библиотеки MACRO.LIB, или создать свои собственные и туда записать. Тогда не нужно наводить определения, а вызывать их из библиотеки с помощью директивы:

INCLUDE <имя файла>

Лучше это сделать в условной директиве поскольку макроопределение считывается на первом проходе транслятора. Если какие-то макроопределения ненужны, то их можно удалить:

```
IF1
 INCLUDE MACRO.LIB
ENDIF
```

```
PURGE MAC1, MAC2, MAC3
```

Всего есть возле **40** стандартных макрокоманд в виде .ASM файла. Они выполняют много функций, аналогичных функциям DOS или BIOS.

Директиву INCLUDE можно использовать и для включения других файлов

Вместо этой директивы ассемблер тидставить весь текст файла

INCLUDE A: MACROS.TXT