Макросредства языка ассемблер (часть 1)

Нередко бывают ситуации, когда нужно изменить листинг программы непосредственно перед тем, как ассемблер будет транслировать программу. Примеров этому может быть много. Например, нужно объявить массив из десяти, ста, тысячи элементов и инициализировать его начальными значениями, равными своему индексу. Вручную писать X db 0,1,2,3..99 будет очень накладно, да и листинг программы забивать этими данными как-то неправильно. Или другой пример: допустим нужно сделать конструкцию, аналогичную #define в с, считающую сумму трёх слагаемых (напомню, что через EQU можно написать только то, что можно заранее предрасчитать). Или, чтобы, например, переменная, в зависимости от своего типа, корректно обнулялась. Если переменная типа двойного слова, то нужно произвести обнуление в два этапа: обнулить по очереди старшую и младшие части двойного слова. Либо необходимо в зависимости от определённого условия пропускать/добавлять кусок кода в программу. Всё вышеперечисленное реализуется в ассемблере благодаря макросам. Макросы упрощают разработку программ. Программа, написанная с использованием макроязыка, транслируется в два этапа. Сначала она переводится на, так сказать, чистый язык ассемблера, т.е. преобразуется к виду с чистым ассемблером, где нет никаких макросредств. Этот этап называется этапом макрогенерации, его осуществляет специальный транслятор – макрогенератор. На втором этапе полученная программа переводится на машинный язык. Этот этап ассемблирования осуществляется уже самим ассемблером.

Трансляция программы, написанной с использованием средств макроязыка, занимает больше времени, чем программа, написанная на «чистом» ассемблере. Макрогенератор и ассемблер могут взаимодействовать двояко. Во-первых, они могут действовать независимо друг от друга: сначала текст программы обрабатывает макрогенератор и только затем начинает работать ассемблер. Такой способ взаимодействия достаточно прост для понимания и для реализации и используется довольно часто, однако, у него есть серьёзный недостаток: макрогенератор, работая до ассемблера, не может воспользоваться информацией, извлекаемой из текста программы ассемблером. Например, в макросредствах нельзя использовать переменные, которые определяются с помощью директивы эквивалентности: N EQU 10. В данном случае N будет известна только ассемблеру и понять её макрос не сможет. Чтобы как-то взаимодействовать макроязыку с ассемблером, приходится усложнять синтаксис макросредств. Во-вторых, макрогенератор и ассемблер могут действовать совместно, чередуя свою работу: первым каждое предложение программы просматривает макрогенератор. Если это конструкция собственно макроязыка, то макрогенератор соответствующим образом преобразует её в группу предложений «чистого» языка ассемблера и тут же передает их на обработку ассемблеру, а если это обычное предложение «чистого» языка ассемблера, то макрогенератор сразу передаёт его ассемблеру. В таких условиях в конструкциях макроязыка уже можно ссылаться на объекты (например, константы) «чистого» языка ассемблера.

При «чередующемся» способе взаимодействия макрогенератора и ассемблера они фактически являются не двумя независимыми трансляторами, а частями одного транслятора, который принято называть макроассемблером.

Блоки повторения

Иногда, как было уже упомянуто в примерах выше, в некотором месте программы приходится выписывать несколько раз подряд один и тот же (ну или почти, один и тот же) фрагмент, и хотелось бы, чтобы мы сами выписывали этот фрагмент только один раз, а макрогенератор размножал его нужное число раз. Такая возможность предусмотрена в ЯА и, реализуется она с помощью блоков повторения (repeat blocks)

Блок повторения имеет следующую структуру:

<заголовок>
<тело>
ENDM

«Тело» тут – любое число любых предложений (в частности, ими могут быть снова блоки повторения), а ENDM – директива, указывающая на конец тела и всего блока повторений. Встречая в исходном тексте программы такой блок, макрогенератор подставляет вместо него в окончательную программу несколько копий тела. При дублировании тело может выписываться без каких-либо изменений, а может копироваться и с модификациями. Как именно происходит дублирование, сколько копий создаётся – всё это зависит от заголовка блока. Имеется три разновидности заголовка, в связи с чем, различают три варианта блока повторения:

REPT – блоки, IRP – блоки и IRPC – блоки.

REPT – блоки

Простейший блок повторения REPT выполняет ассемблирование участка программы заданное число раз. Этот тип блоков повторения записывается следующим образом:

REPT k
<тело>
ENDM

Тут k – константное выражение с неотрицательным значением. Это выражение должно быть таким, чтобы можно было вычислить его сразу (например, в нём не должно быть ссылок вперёд). Выражение должно иметь значение в виде 16-битового числа без знака и не может содержать внешних или неопределенных символов. Тело блока может включать в себя любые предложения языка. Вычислив значение k, макрогенератор создаёт k точных копий тела блока и подставляет их в окончательный текст программы.
Простейший пример:

```
REPT   10
       db      0
endm
```

Эквивалентно db 10 dup(0)
Например, по блоку

```
REPT 3
SHR ax, 1
ENDM
```

Будет построен следующий фрагмент окончательной программы”

```
SHR ax, 1
SHR ax, 1
SHR ax, 1
```

Другой пример:

```
N EQU 6
REPT N-4
DB 0,1
DW ?
ENDM
```

Преобразуется в:

```
N EQU 6
DB 0,1
DW ?
DB 0,1
DW ?
```

В блоках повторения довольно часто используется директива присваивания. Например, описать 100-байтовый массив X (из примера выше) можно так:

```
X db 0
K=0
REPT 99
K=K+1
DB K
ENDM
```

Точно так же можно описать, например, фрагмент кода, который содержит в себе массив, состоящий из ASCII-кодов прописных букв:

```
sym=’A’
symbols
rept 26
db sym
sym=sym+1
endm
```

При подключении к компьютеру измерительного или управляющего оборудования иногда возникает необходимость замедлить работу процессора при обращении к портам этого оборудования. Замедление осуществляется включением в текст программы одной или, если требуется, нескольких команд безусловного перехода на следующее предложение:

```
in al, 300h
jmp a
a: jmp b
b: jmp c
c: in al, 301h
```

Для того, чтобы не создавать много ненужных, в сущности, меток, такого рода предложения часто записывают следующим образом:

```
in al, 300h
jmp $+2
jmp $+2
jmp $+2
in al, 301h
```

Такой фрагмент можно оформить в виде блока повторения: rept 6 jmp $+2 endm. Для упрощения можно использовать инструкцию nop (отсутствие операции).

IRP-блоки

Блоки повторений, как и макроопределения, могут вызываться с параметрами. Для этого ипользуются директивы IRP/IRPC.

Блоки повторений типа IRP имеют следующий вид:

```
IRP p,<v1,v2, …, vn>
<тело>
ENDM
```

Тут p – некоторое имя, оно играет роль формального параметра и может использоваться в предложениях тела. Vi – это фактические параметры. Это любые тексты (возможно, и пустые), но, чтобы не было путаницы, они должны быть сбалансированы по кавычкам и не должны содержать запятые, точки с запятой и уголки вне кавычек. Но это не беда, так как в ассемблере, как и в ЯВУ также есть экранирующий символ. Параметры vi перечисляются через запятую, а вся их совокупность обязательно заключается в угловые скобки. Встречая такой блок, макрогенератор изменяет его на m копий тела, причём в i-ой копии все вхождения имени p заменяются на Vi. Например:

```
IRP REG, <AX,CX,SI>
PUSH REG
ENDM
```

Преобразуется в:

```
PUSH AX
PUSH CX
PUSH SI
```

Формальный параметр локализируется в теле блока (вне блока он будет недоступен) и может быть любым именем. Если оно совпадает с именем другого объекта программы, то в теле блока оно обозначает именно параметр, а не этот объект. Например:

```
IRP BX, <1,5>
ADD AX, BX
ENDM
```

Этот кусок преобразуется в:

```
ADD AX, 1
ADD AX, 5
```

Фактический параметр того макроса назывался «BX», но обозначал он совсем не регистр общего назначения. Также, в теле блока повторения заменяются только формальный параметр, другие имена (например, имена констант) переносятся в копии тела без изменений. Например:

```
N EQU 1
IRP P, <A,B>
P EQU N
ENDM
```

Это преобразуется в:

```
N EQU 1
A EQU N                 ;но не A EQU 1!
B EQU N
```

Можно использовать точно так же, как и REPT, для задания переменных:

```
IRP     arg,<0,1,2,3>
        db      arg
endm
```

преобразуется в

```
        db      0
        db      1
        db      2
        db      3
```

Хоть и похоже на rept, плюс требует описания всех параметров в списке, на самом деле является более гибким, так как rept ограничен формулой: вот такое с помощью rept не описать

```
IRP     arg,<5,15,8,2>
        db      arg
endm
```

преобразуется в

```
        db      5
        db      15
        db      8
        db      2
```

IRPC-блоки

Блоки этого типа записываются так:

```
IRPC p,s1…sk
<тело>
ENDM.
```

Тут p – формальный параметр, а вот si – это символы. Это могут быть любые символы, кроме пробелов, точек с запятой (с «;» начинается комментарий, а пробел заканчивает операнд. Если надо указать точке с запятой или пробел, то всю последовательность символов следует заключить в угловые скобки). Встречая IRPC блок, макрогенератор заменяет его на k копий тела блока (по одной на каждый символ), причём в i-й копии все вхождения параметра p будут заменены на символ si. Например:

```
IRPC D,17W
ADD AX, D
ENDM
```

Заменится на:

```
ADD AX, 1
ADD AX, 7
ADD AX, W
```

Следующий блок задает строку в памяти, располагая после каждого символа строки атрибут OFh (белый символ на черном фоне), так что эту строку впоследствии можно будет скопировать прямо в видеопамять.

```
irpc character,<строка символов>
db '&character&' ,OFh
endm
```

Макрооператоры

При использовании блоков повторения возникает ряд проблем с записью их формальных и фактических параметров. Эти проблемы решаются с помощью так называемых макрооператоров – операторов, разрешенных к применению только в конструкциях макроязыка.

Макрооператор &

Рассмотрим пример:

```
IRP **W**,<VAR1, VAR6>
W D**W** ?
ENDM
```

Ну никак тут не объяснить, что вместо W нужно подставить фактический параметр, а в DW, вместо второй буквы не надо ничего подставлять! Да и различаются фактические параметры тут только последним символом. Соответственно, лучше бы написать так:

```
IRP **W**,< 1, 6>
VAR**W** D**W** ?
ENDM
```

И тут получается неоднозначность. Для её устранения, непосредственно, перед символом, заменяющим фактические параметры, нужно подставить макрооператор &.

```
IRP **W**,< 1, 6>
VAR&**W** D**W** ?
ENDM
```

Оттранслируется так:

```
VAR1 DW ?
```

VAR6 DW ?

Можно использовать в комбинации с директивой REPT, чтобы задать некоторое число переменных и дать каждой из них уникальное имя. Например:

```
value=0
REPT   4
list&value db value
value=value+1
endm
```

Это оттранслируется в

```
list0     db       0
list1     db       1
list2     db       2
list3     db       3
```

По сути, этот макрооператор нужен для того, чтобы параметр, переданный в качестве операнда макроопределению или блоку повторений, заменялся значением до обработки ассемблером. Так, например, следующий макрос выполнит команду PUSH EAX, если его вызвать как PUSHREG A:

```
pushreg macro letter
push e&letter&x
endm
```

Иногда можно использовать только один амперсанд - в начале параметра, если не возникает неоднозначностей. Например, если передается номер, а требуется создать набор переменных с именами, оканчивающимися этим номером:

```
irp number, <1, 2, 3, 4>
msg&number db ?
endm
```

***Макрооператор* <>**

Как было сказано, фактические параметра IRP-блока не должны содержать “,<;>” , а во втором операнде IRPC-блока нельзя указывать пробелы и точки с запятой. Эти ограничения связаны с тем, что иначе возможна путаница: например, не понятно сколько вообще параметров есть у макроса. Чтобы можно было их использовать (группировать параметры), нужно их обрамить угловыми скобками.

```
IRP VAL,<<1,2>,3>
DB VAL
ENDM
IRPC S,<A;B>
DB ‘&S’
ENDM
```

Преобразуется в

```
DB 1,2
DB 3
```

Или

```
IRPC S,<A;B>
DB ‘&S’
ENDM
```

Преобразуется в

```
DB ‘A’
DB ‘;’
DB ‘B’
```

То есть он действует так, что весь текст, заключенный в эти скобки, рассматривается как текстовая строка, даже если он содержит пробелы или другие разделители. Этот макрооператор используется при передаче текстовых строк в качестве параметров для макросов. Другое частое применение угловых скобок - передача списка параметров вложенному макроопределению или блоку повторений.

Макрооператор !

Является экранирующим символом, который нужно подставить перед служебным (типа точки с запятой или угловой скобки), чтобы он означал не своё служебное значение, а часть параметра. Например:

```
IRP X, <A!>B, Hello world!>
DB ‘&X’
ENDM
```

Оттранслируется в

```
DB ‘A>B’
DB ‘Hello world!’
```

Используется аналогично угловым скобкам, но действует только на один следующий символ, так что, если этот символ - запятая или угловая скобка, он все равно будет передан макросу как часть параметра.

Микрооператор %

В макроязыке есть ещё один макрооператор, используемый при записи фактических параметров IRP-блоков (и макросов)

%<константное выражение> - вычисляет значение этого самого константного выражения. Здесь вложенность не допускается. Указывает, что находящийся за ним текст является выражением и должен быть вычислен. Обычно это требуется для того, чтобы передавать в качестве параметра в макрос не само выражение, а его результат.

```
K EQU 4
        IRP A,<K+1,%K+1,W%K+1>
        DW A
ENDM

        DW K+1
        DW 5
        DW W5
```

Макрооператор ;;

Если в теле блока повторения (и макроса) имеются комментарии, то они переносятся во все копии блока. Однако бывает так, что комментарий полезен при описании самого блока повторения, но не нужен в его копиях. В таком случае, следует начинать комментарий не с одной точки с запятой, а с двух. В отличие от обычных комментариев текст макрокомментария не попадает в листинг и в текст программы при подстановке макроса. Это сэкономит память при ассемблировании программы с большим количеством макроопределений.

```
IRP R,<AX,BX>
;;восстановление регистров
;восстановить R
POP R  ;;стек -> R
ENDM
```

Приведётся в:

```
;восстановить AX
POP AX
;восстановить BX
POP BX
```

Макросы

С помощью блока повторения один раз описывается некоторый фрагмент программы, который затем многократно копируется макрогенератором. Но блоки повторения можно использовать, только если их копии должны быть расположены рядом друг с другом. А что делать, если фрагмент программы должен повторяться, но в разных местах программы? В этом случае используются макросы: специальным образом описывается этот фрагмент программы (это и есть макрос) и ему дается имя, а затем в нужных местах программы выписывается ссылка на этот макрос (указывается его имя); когда макрогенератор просматривает текст программы и встречает такую ссылку, то он вместо неё подставляет в окончательный текст программы сам макрос – соответствующий фрагмент программы; и так делается для каждой ссылки на макрос, в каком бы место программы она ни встретилась. Описание макроса называется макроопределением, ссылка на макрос – макрокомандой, процесс замены макрокоманды на макрос – макроподстановкой, а результат такой подстановки – макрорасширением.

Макроопределения

Одно из самых мощных языковых средств ассемблера - макроопределения. Макроопределением (или макросом) называется участок программы, которому присвоено имя и который ассемблируется всякий раз, когда ассемблер встречает это имя в тексте программы. Макрос начинается директивой MACRO и заканчивается ENDM.

Описание макроса, т.е макроопределение, имеет следующий вид:

```
<имя макроса> MACRO <формальные параметры >
        <тело макроса>
ENDM
```

Директива MACRO - это заголовок макроопределения. В ней указывается имя и через запятую перечисляются формальные параметры, если необходимо.

Формальные параметры позволяют копировать макрос не в неизменном виде, а с изменениями. Те величины, которые необходимо будет изменить описываются формальными параметрами.

Замечание. Имена формальных параметров локальны по отношению к макросу, т.е. они могут совпадать с именами в основной программе, макрогенератор понимает их как параметры.

Завершает макроопределение директива ENDM. Не надо повторять имя макроса. Размещаться они могут в любом месте программы, но определение обязательно должно быть до первой ссылки на макрос либо в отдельном файле.

Макрокоманда - обращение к макроопределению. Или указание макрогенератору на то, что на указанном месте необходимо подставить тело макроопределения. Итак, одна макрокоманда заменяется на группу команд, поэтому она и называется макро (большая).

Синтаксис макрокоманды:

```
        <имя макроса>  [<фактические параметры>]
```

Замечание. Фактические параметры можно разделять запятыми или пробелами.

Формальные параметры макроопределения заменяются соответствующими фактическими параметрами макрокоманды.

i-тый фактический параметр соответствует i-тому формальному параметру

Число фактических параметров должно быть равно числу формальных параметров, если фактических параметров больше, то лишние игнорируются, если формальных больше, считается что в качестве недостающих фактических указаны пустые тексты.

Действия макрогенератора:

1) макрогенератор находит макроопределение с указанным именем
2) в его теле заменяет все формальные параметры фактическими
3) полученное макрорасширение подставляет в программу вместо макрокоманды

```
SUM MACRO X,Y      ;X = X+Y
        MOV AX, Y
        ADD X, AX
ENDM

VAR MACRO NM,TP,VL
        NM D&TP VL
ENDM
```

Теперь в программе можно использовать слово SUM/VAR, как если бы это было имя команды, и ассемблер заменит каждое такое слово на команды, содержащиеся в макроопределении.

Можно комбинировать макросы и блоки повторений.

```
define  macro   count
        value=0
        REPT   count
                db      value
        value=value+1
        endm
endm
```

Тогда следующие вызовы

```
define 4
define 3
оттранслируются в
        db      0
        db      1
        db      2
        db      3
        db      0
        db      1
        db      2

create  macro   argList
        IRP     arg,<argList>
                db      arg
        endm
endm
```

Тогда след. вызовы

```
create  <0,5,8>
define  3
create  <10,15,3,7>
```

преобразуются в (и так понятно).

С помощью блоков повторений и макросов можно даже описать аналог цикла for:

```
forloop macro start,stop,incr,statementList
        local   top,done
        push    ax
        mov     ax,start
top:    cmp     ax,stop
        jge     done
        IRP     statement,<statementList>
                statement
        endm
        add     ax,incr
        jmp     top
done:
        pop     ax
endm

xor bx,bx
xor cx,cx
forloop 20,5,-4,<<add bx,ax>,<sub cx,ax>>

        push    ax
        mov     ax,20
top:    cmp     ax,5
        jge     done1
        add     bx,ax
        sub     cx,ax
        add     ax,-4
        jmp     top1
done:
        pop     ax
```

То же самое можно записать с помощью специального оператора продолжения выражения \

```
xor bx,bx
xor cx,cx
forloop 20,5,-4,\
<\
```

```
        <add bx,ax>,\
        <sub cx,ax>\
>
```

В макросах также можно применять макрооператор &. Здесь он будет отделять формальный параметр от остальной строки, если нужно использовать формальный параметр вместе с какой-либо строкой. Используется, если формальный параметр является частью некоторого идентификатора, тогда последовательность символов, которая является формальным параметром отделяют от остальной строки символом &. Макрооператор & используется когда надо формальный параметр указать внутри строки.

```
defmas  macro type, len
                mas&type   d&type  len dup (?)
                endm

defmas b,10             ; после макроподстановки получим  masb  db 10 dup (?)
defmas w,20             ; после макроподстановки получим  masw  dw 20 dup (?)
```

Если в качестве фактического параметра макроса нужно передать пустую строку, она просто обособляется запятыми.