Макросредства ассемблера часть 2
План: макроопределения, макрокоманды, директивы local и exitm, условное ассемблирование.
<Имя макроса> MACRO <формальный параметры>
<тело>
<ENDM>
Макроопределение

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

Макрокоманда

```
SUM A, ES:B

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

Формальные параметры перечисляются через запятую или разделяются пробелами, пустая строка обособляется запятыми.
Отличие от команд и директив:
-имя макроса придумывается самим автором
-Параметры могут отличаться друг от друга как запятыми, так и пробелами
-В качестве параметра может быть указан любой текст (даже пустой)
-Текст должен быть сбалансирован по кавычкам и угловым скобкам
-Все пробелы, запятые и точки с запятой должны быть внутри угловых скобок или кавычек
Пример: требуется сократить написание конструкций
If (x<y) goto L;

```
IF_LESS MACRO X,Y,L
        MOV AX, X
        CMP AX, Y
        JL L
ENDM

MOV DX, A
        IF_LESS A,B,M1
        MOV DX, B
M1:     IF_LESS DX,C,M2
        MOV DX, C
M2:
```

Пример: требуется вычислить NOD несколько раз подряд

```
MOV AX, A
MOV BX, B
CALL NOD
MOV CX, AX
MOV AX, A
MOV BX, D
CALL NOD
ADD CX, AX
```

Определяем макрос

```
CALL_NOD MACRO X,Y
        MOV AX, X
        MOV BX, Y
        CALL NOD
ENDM
```

```
CALL_NOD A, B
MOV CX, AX
CALL_NOD C,D
ADD CX, AX
```
Отличие от подпрограмм:
-Оттранслированный макрос подставляется во все места, где происходил его вызов
-После работы макроса не требуется осуществлять возврат
-Макрос можно переопределить или уничтожить

Сравнительный анализ процедур и макросредств

Повторяющиеся действия (фрагменты) в программе можно описать и как процедуру, и как макроопределение. При этом в обоих случаях повторяющийся участок кода описан только один раз, а обращаемся к нему с помощью одной команды (вызов процедуры или макрокоманда). Но...

После трансляции процедура так и останется описанной один раз, а тело макроопределения подставится во все места вызова и тем самым увеличит размер программы.

Вывод 1. Применение процедур делает код более компактным, т.е. **экономим память**

Но при обращении к процедуре

а) выполняется засылка параметров в регистры или стек,
б) запоминается адрес возврата
в) осуществляется переход,
г) по окончании работы процедуры восстанавливается адрес возврата,
д) очищаются регистры или стек и т.п.

Итак, при работе процедуры тратится время на переходы и передачу параметров во время выполнения программы.

!!! При замене макрокоманд на макрорасширения тоже тратится время, но это происходит на этапе трансляции, а не во время выполнения программы.

Вывод 2. Применение макросредств **экономит время** выполнения программы**.**

Поэтому в программах критических по времени следует применять макросредства, а если необходимо экономить память следует применять процедуры.

Если в повторяющемся участке кода много команд (т.е. большой фрагмент) лучше описать его как процедуру. Если же небольшую группу команд описать процедурой, то число вспомогательных команд по ее вызову и передаче параметров станет сравнимым с числом команд самой процедуры, ее время выполнения станет на много больше.

Вывод 3. **Большие участки кода** рекомендуется описывать **как процедуры, а маленькие - как макроопределения.**

!!! Еще одно отличие использования макросредств и процедур заключается в том, что параметрами процедур могут быть только операнды команд, а параметрами макрокоманд могут быть любые последовательности символов, в том числе и сами команды.

Сравнительные примеры:

Разумеется, можно оформить те же участки кода в виде процедур и вызывать их командой CALL - если процедура вызывается больше одного раза, этот вариант программы займет меньше места, но вариант с макроопределением станет выполняться быстрее, так как в нем не будет лишних команд CALL и RET. Однако скорость выполнения - не главное преимущество макросов. В отличие от процедур макроопределения могут вызываться с параметрами, следовательно, в зависимости от ситуации, включаемый код будет немного различаться, например:

```
s_mov macro register1, register2
push register1
pop register2
endm
```
Теперь можно использовать S_MOV вместо команды MOV для того, чтобы скопировать значение из одного сегментного регистра в другой.

Определение макроса через макрос:

```
ARR MACRO X, N
        X DB N DUP (?)
ENDM
```

```
ARR2 MACRO X1, X2, K
        ARR X1, <K>
        ARR X2, <K>
ENDM
```

ARR2 A,B,20 ➔ ARR A, <20> ➔ A DB 20 DUP (?)
ARR B, <20> B DB 20 DUP (?)

Макрооператоры - операторы, которые применяются только в конструкциях макроязыка. Используются при записи формальных и фактических параметров макрокоманд. Параметрами макрокоманд могут быть и сами команды, и любые последовательности символов.
Макрооператор &
Указывает границу параметра, чтобы выделить его из окружающего теста
Используется, если формальный параметр является частью некоторого идентификатора, тогда последовательность символов, которая является формальным параметром отделяют от остальной строки символом &

```
Пример 1.    defmas  macro type, len
             mas&type   d&type  len dup (?)
             endm
        .data
        defmas b,10          ; после макроподстановки получим  masb  db 10 dup (?)
        defmas w,20          ; после макроподстановки получим  masw  dw 20 dup (?)
```

Существует три варианта размещения макроопределений:
в начале исходного текста программы, до кода и данных с тем, чтобы не ухудшать читаемость программы. Этот вариант следует применять в случаях, если определяемые макрокоманды актуальны только в пределах одной этой программы;
в отдельном файле. Этот вариант подходит при работе над несколькими программами одной проблемной области. Чтобы сделать доступными эти макроопределения в конкретной программе, необходимо в начале исходного текста этой программы записать директиву include ИмяФайла, например:
include show.inc ;сюда вставляется текст файла show.inc
третий способ далее

Если какие-то строки макроопределения должны содержать метки (например, с целью организации циклов), то обозначения меток следует объявить локальными с помощью оператора local. В этом случае ассемблер, генерируя макрорасширения, будет создавать собственные обозначения меток, не повторяющиеся при повторных вызовах одной и той же макрокоманды. Оператор local перечисляет метки, которые будут применяться внутри макроопределения, чтобы не возникало ошибки «метка уже определена» при использовании макроса более одного раза или если та же метка присутствует в основном тексте программы (в зависимости от программы-ассемблера, метки-операнды команды LOCAL могут оказаться недоступными для использования в других местах программы, то есть вне текущего макроса - в другом макросе или в участках программы без макросов). Операнд для LOCAL - метка или список меток, которые будут использоваться в макросе.

```
delay macro
local point
mov CX,200
point: loop point
endm
```

Макрос delay создает задержку фиксированной длительности. Если в текст программы включить две макрокоманды delay

…
delay
…
delay

то их макрорасширения, подставленные в текст программы, будут выглядеть следующим образом:

…
mov CX, 20000
??0000: loop ??0000
…
mov CX, 20000
??0001: loop ??0001

При повторных подстановках макроопределения транслятор заменяет обозначение метки point на различающиеся обозначения ??0000, ??0001 и т.д., обеспечивая тем самым правильное выполнение команд циклов и переходов.

```
MD MACRO R1,R2
        LOCAL M, M1
M:      CMP R1,R2
        JB M1
        SUB R1,R2
        JMP M
M1:
        ENDM
```

Директива exitm: используется для досрочного выхода из макроопределения или блока повторения. Например, следующее макроопределение не выполнит никаких действий, то есть не будет расширено в команды процессора, если параметр не указан:

```
pushreg macro reg
ifb <reg>
exitm
endif
push reg
endm
```

Директива GOTO имя_метки переводит процесс генерации макроопределения в другое место, прекращая тем самым последовательное разворачивание строк макроопределения. Метка, на которую передается управление, имеет специальный формат:

:имя_метки

Директива purge: удаление макроса/макросов.

PURGE <имя макроса> {, <имя макроса>}

Отменяет определенный ранее макрос. Эта директива часто применяется сразу после INCLUDE, включившей в текст программы файл с большим количеством готовых макроопределений.

Ещё один способ объявления макросов - в макробиблиотеке. Универсальные макрокоманды, которые используются практически во всех программах целесообразно записать в так называемую макробиблиотеку. Сделать актуальными макрокоманды из этой библиотеки можно также с помощью директивы include. Недостаток этого способа в том, что в исходный текст программы включаются абсолютно все макроопределения. Для исправления ситуации можно использовать директиву purge, в качестве операндов которой через запятую перечисляются имена макрокоманд, которые не должны включаться в текст программы. К примеру:

include iomac.inc
purge outstr,exit
В данном случае в исходный текст программы перед началом трансляции вместо строки include iomac.inc вставит строки из файла iomac.inc. Но вставленный текст будет отличаться от оригинала тем, что в нем будут отсутствовать макроопределения outstr и exit.

Существует два класса условных директив: директивы условного ассемблирования и условные директивы генерации сообщений об ошибке.
В большинстве языков программирования присутствуют средства, позволяющие игнорировать тот или иной участок программы в зависимости от выполнения условий, например: в языке С это осуществляется директивами препроцессора #if, #ifdef, #ifndef и т. д. Ассемблер тоже предоставляет такую возможность.
Основные классы директив – IF, ELSEIF и ERR.
Условное ассемблирование: IF, IFE, IFIDN, IFDIF, IFB, IFNB.
<IF-директива>
<фрагмент-1>
ELSE
<фрагмент-2>
ENDIF
Или
<IF-директива>
<фрагмент-1>
ENDIF
Каждая директива условного ассемблирования IFxxx задает конкретное условие, при вычислении которого получается истинное (true) или ложное (false) значение. Если условие имеет значение true, то выполняется ассемблирование и помещение в объектный файл блока ассемблируемого кода "тело_условия_true". Если при вычислении условия получается значение false, то Турбо Ассемблер пропускает "тело_условия_true" и не включает его в объектный файл. При наличие директивы ELSE, если условие имеет значение false, то ассемблируется и выводится в объектный файл блок "тело_условия_false". Если условие имеет значение true, то этот блок игнорируется. Условный блок завершается директивой ENDIF.
Два описанных блока кода являются взаимноисключающими: в объектный файл включается либо "тело_условия_true", либо "тело_условия_false", но не оба блока одновременно. Кроме того, если вы используете форму IFxxx.ELSE.ENDIF, один из блоков будет обязательно включаться в объектный файл. Если используется форма IFxxx.ENDIF, то "тело_условия_true" может включаться или не включаться в объектный файл, в зависимости от условия.
При использовании вложенных директив IF и ELSE директива ELSE всегда соответствует ближайшей директиве IF.
Вы можете использовать директивы условного ассемблирования ELSEIFxxx как сокращенную форму, когда требуется использовать несколько директив IF. Директива ELSEIFxxx представляет собой эквивалент директивы ELSE, за которой следует вложенная директива IFxxx, но дает более компактный код. Например:

```
IF mode EQ 0
  …
ELSEIF mode LT 5
  …
ELSE
  …
ENDIF
```

сравните его со следующим:

```
IF mode EQ 0
  …
ELSE
 IF mode LT 5
```

…
ELSE
…
ENDIF
ENDIF
Вне оператора IFxxx директиву ELSEIFxxx использовать нельзя.

Обобщаем:
if выражение
...
endif
Если значение выражения - ноль (ложь), весь участок программы между IF и ENDIF игнорируется.
Директива IF может также сочетаться с ELSE и ELSEIF:
if выражение
...
else
...
endif
Если значение выражения - ноль, ассемблируется участок программы от ELSE
до ENDIF, в противном случае - от IF до ELSE.
if выражение1
...
elseif выражение2
...
elseif выражение3
...
else
...
endif
Так, если, например, выражение 2 не равно нулю, будет ассемблироваться участок программы между первой и второй директивой ELSEIF Если все три выражения равны нулю, ассемблируется фрагмент от ELSE до ENDIF. Данная структура директив может использоваться в частном случае аналогично операторам switch/case языков высокого уровня, если выражения - проверки некоторой константы на равенство.
Кроме общих директив IF и ELSEIF ассемблеры поддерживают набор специальных команд, каждая из которых проверяет специальное условие:

IF, IFE
IF <константное выражение>
IFE <константное выражение>
<константное выражение> != 0
<константное выражение> == 0
Т.е. IFE/ELSEIFE проверяют, если выражение ложно (равно нулю), в отличие от IF, который проверяет его истинность.

```
DEBUG EQU 1
MOV X, AX
IF DEBUG
        PRINTF X
ENDIF
MOV BX, 0EFH
……..
```

Соответствующие директивы ошибок – ERRIF, ERRIFE.

IFIDN, IFDIF (сравниваются тексты в параметрах)
Эти и следующие директивы применяются в макроопределениях для проверки параметров.
IFIDN <t1>, <t2>
IFDIF <t1>, <t2>
t1, t2 – любые тексты
IFIDN <a+b>, <a+b> true
IFIDN <a+b>, <a> false
IFIDN <a+b>, <a+B> false
IFDIF <arg1>,<arg2>/ELSEIFDIF <arg1>,<arg2> - если аргументы отличаются (с различием больших и маленьких букв);
IFDIFI <arg1>,<arg2>/ELSEIFDIFI <arg1>,<arg2> - если аргументы отличаются (без различия больших и маленьких букв);
IFIDN <arg1>,<arg2>/ELSEIFIDN <arg1>,<arg2> - если аргументы одинаковы (с различием больших и маленьких букв);
IFIDNI <arg1>,<arg2>/ELSEIFIDNI <arg1>,<arg2> - если аргументы одинаковы (без различия больших и маленьких букв).
Тексты должны обязательно быть заключены в «уголки»

```
MM MACRO R1,R2,T
   LOCAL L
   IFDIF <R1>,<R2>      ;;R1 и R2 - разные регистры?
   CMP R1,R2
   IFIDN <T>,<MAX>      ;;T=MAX?
   JGE L                ;;да - поместить JGE L в макрорасширение
   ELSE
   JLE L                ;;нет - поместить JLE L
   ENDIF
   MOV R1,R2
L:
   ENDIF
   ENDM
```

IFB, IFNB (проверка параметра на пустоту)
IFB <t>
IFNB <t>

```
DEF MACRO X,V
        IFB <V>          ;;параметр V задан пустой?
                X DB ?
        ELSE
                X DB V
        ENDIF
ENDM
```

IFB <аргумент>/ELSEIFB <аргумент> - если значение аргумента - пробел;
IFNB <аргумент>/ELSEIFNB <аргумент> - если значение аргумента - не пробел;

IFDEF, IFNDEF – определен ли идентификатор.
IFDEF метка/ELSEIFDEF метка - если метка определена;
IFNDEF метка/ELSEIFNDEF метка - если метка не определена;

Директивы условного ассемблирования эффективно используются в макросах. Например: напишем макрос, выполняющий умножение регистра AX на число, причем, если множитель - степень двойки, то умножение будет выполняться более быстрой командой сдвига влево.

```
fast_mul macro number
if number eq 2
shl ax, 1
elseif number eq 4
shl ax, 2
elseif number eq 8
shl ax, 3
...
elseif number eq 32768
shl ax, 15
else
mov dx, number
mul dx
endif
endm
```

Этот макрос можно усложнить, применяя различные команды и их комбинации, но в нынешнем виде он чрезмерно громоздкий. Проблема решается с помощью блоков повторений.

Директивы условного ассемблирования позволяют вам генерировать во время ассемблирования сообщения об ошибках при наступлении определенных событий. Ассемблер выводит сообщение об на экран и в файл листинга и предотвращает создание объектного файла. То есть, иногда директивы условного ассемблирования используются для того, чтобы прервать ассемблирование программы, если обнаружилась какая-нибудь ошибка.

Для таких случаев предназначены директивы условной генерации ошибок.

Директива ERRxxx генерирует при удовлетворении определенных условий сообщения пользователя об ошибке. Она имеет следующий общий синтаксис:

ERRxxx [аргументы] [сообщение]

В этом случае директива ERRxxx представляет какую-либо из директив условной генерации сообщения об ошибке (такие как ERRIFB и т.д.).

"Аргументы" представляют аргументы, которые могут потребоваться в директиве для вычисления условия. Некоторые директивы требуют выражения, другие требуют символьного выражения, а некоторые - одно или два текстовых выражений. Некоторые из директив вовсе не требуют аргументов.

Если указано "сообщение", то оно задает необязательное сообщение, которое выводится с ошибкой. Сообщение должно быть заключено в кавычки (' или ").

Директивы генерации сообщений об ошибке генерируют пользовательское сообщение об ошибке, которое выводится на экран и включается в файл листинга (если он имеется) в месте расположения директивы в исходном коде. Если директива задает сообщение, оно выводится на той же строке непосредственно за ошибкой. Например, директива:

ERRIFNDEF foo "foo не определено!"

если идентификатор foo не определен при обнаружении ошибки, приведет к генерации ошибки:

User error: "foo не определено!"

Безусловными директивами генерации сообщений об ошибке являются директивы ERR. Эти директивы всегда генерируют ошибку и не требуют аргументов, хотя могут содержать необязательное сообщение.

Аналогично командам условного ассемблирования, существуют модификации команды ERR:

ERRE выражение - ошибка, если выражение равно нулю (ложно);

ERRNZ выражение - ошибка, если выражение не равно нулю (истинно);

ERRDEF метка - ошибка, если метка определена;

ERRNDEF метка - ошибка, если метка не определена;
ERRB <аргумент> - ошибка, если аргумент пуст;
ERRNB <аргумент> - ошибка, если аргумент не пуст;
ERRDIF <arg1>,<arg2> - ошибка, если аргументы различны;
ERRDIFI <arg1>,<arg2> - ошибка, если аргументы отличаются (сравнение не различает большие и маленькие буквы);
ERRIDN <arg1>,<arg2> - ошибка, если аргументы совпадают;
ERRIDNI <arg1>,<arg2> - ошибка, если аргументы совпадают (сравнение не различает большие и маленькие буквы).