

**Методические указания к лабораторной работе № 9
по дисциплине “Системное Программирование”**

"Разработка и использование макрокоманд Ассемблера"

СОДЕРЖАНИЕ

1. Цель и задачи лабораторной работы.....	3
2. Макросы и их применение	3
3. Цели и задачи ЛР № 9	31
4. Требования к выполнению лабораторной работы №9	31
5. Перечень вариантов ЛР по группам.....	35
6. Перечень вариантов для заполнения массива	35
7. Перечень вариантов для сильных и продвинутых студентов	36
8. Контрольные вопросы по лабораторной работе.....	36
9. Требования к оформлению отчета по ЛР	38
10. Литература по ЛР СП	38
11. Пример программы с макрокомандой	39
12. Шаблон отчета по ЛР № 9.....	43

1. Цель и задачи лабораторной работы

Целью выполнения лабораторной работы является изучение языка и возможностей Макроассемблера, способов написания и использования макрокоманд, приемов их отладки и тестирования. В работе студенты разрабатывают собственные макрокоманды, проверяют их работу, получают навыки создания систем макрокоманд и их отладки. По результатам работы оформляется отчет. Данная ЛР выполняется студентами в полном варианте (с дополнительными требованиями) и простом варианте (1 и 2-й пункты задания). Полный вариант выполняется сильными студентами, что учитывается при сдаче зачета по курсу и защите КР.

2. Макросы и их применение

2.1. Назначение макросов

Макрокоманды – это очень интересный механизм для разработки программ на языке Ассемблер. Макрокоманды или иначе макросы (на жаргоне программистов) обрабатываются на этапе компиляции и позволяют сделать программу более наглядной и обозримой. Макросы могут использоваться для создания собственного языка программирования из команд, записанных пользователем. В программе на языке Ассемблер Вы можете встретить:

- Машинные команды,

- Директивы Ассемблера
- Макрокоманды и
- Комментарии.
- Для использования макросов в программах на языке Ассемблер нужно:
- Описать макрокоманду, дать определения макроса.
- Выполнить вызов макрокоманды (макровывозов).
- Проверить правильность работы макрокоманды (проверить правильность генерации макрорасширения).

Рассмотрим отдельно эти действия.

2.2. Описание макросов

Описание макроса или макроопределение задается в следующем формате:

```
<имя макроса> MACRO <список формальных параметров макросов>  
...  
<тело макрокоманды>  
...  
ENDM
```

Тело макрокоманды состоит из обычных машинных команд, записанных в правилах синтаксиса Ассемблера, директив языка Ассемблер, команд включающих формальные параметры макроса и специальных директив макрокоманд. Список формальных параметров описывается так:

```
<список формальных параметров макросов>:=<формальный параметр> |  
    <формальный параметр> , <список формальных параметров макросов >  
<формальный параметр>:= <имя параметра>
```

Пример простой макрокоманды для печати символа на дисплей показан ниже:

```
PUTCHAR MACRO SIMBOL ; Заголовок макроса с одним параметром  
    PUSH AX  
    PUSH DX  
    MOV DL, SIMBOL ; Использование параметра макроса  
    MOV AH, 02H ; Константа  
    INT 021H  
    POP DX  
    POP AX  
ENDM
```

Данная макрокоманда сохраняет предварительно в стеке регистры AX и DX, выводит символ на экран дисплея и восстанавливает значения запомненных регистров. Хотя данная макрокоманда выглядит универсальной, при ее вызове возможны ошибки. Например, при задании в качестве параметра регистра AX (например, макровывоз – **PUTCHAR AX**) компилятор Ассемблера выдаст ошибку при обработке команды в макросе **MOV DL, SYMBOL**, так как размерности регистров не согласованы (DL – 1 байт и AX – два байта соответственно). Описания макрокоманд могут быть, несомненно, более сложными: из одной макрокоманды можно вызывать другие макрокоманды, создавая, таким образом, системы макрокоманд, которые могут подключаться из библиотек.

2.3. Параметры макросов и макровывозы

Параметры, задаваемые при описании макрокоманды (определении макрокоманды) называются формальными, а параметры, задаваемые при использовании макрокоманды (макровывозе) называются фактическими. Вызов макрокоманды или макровывозов задается так:

```
<имя макрокоманды> <список фактических параметров>  
<список фактических параметров макросов>:=< фактический параметр>|  
    < фактический параметр> , <список фактических параметров макросов >  
< фактический параметр>:= <имя параметра> | <пусто>
```

Как видно из формального описания, при макровывозе параметры могут быть пропущены, при этом запятая должна оставаться на месте. Вызов макрокоманды для предыдущего макроопределения может быть таким:

```
PUTCHAR AL ; Фактический параметр регистр  
...  
PUTCHAR VAR ; Фактический параметр переменная  
...  
PUTCHAR 'A' ; Фактический параметр константа
```

Если при вызове макрокоманды задается несколько параметров, то допускается пропуск параметров (запятая при этом остается):

```
COMPARE VAR1, VAR2, VAR3 ; Параметры все заданы
...
COMPARE VAR1, , VAR3 ; Второй параметр пропущен
...
COMPARE , VAR2, VAR3 ; Первый параметр пропущен
...
COMPARE VAR1, VAR2 ; Третий параметр пропущен
```

Если параметры при вызове опущены, то в самой макрокоманде нужно предусмотреть проверку наличия конкретного фактического параметра, для чего используются специальные директивы условной компиляции.

2.4. Параметры по умолчанию

При описании макрокоманды могут быть заданы значения параметров по умолчанию. Это выполняется следующим образом (в дополнение к формальному описанию заголовка макрокоманды):

```
<формальный параметр>:= <имя параметра> |
<имя параметра> = <значение по умолчанию><имя параметра> = REQ
```

Такая возможность существует не во всех макроассемблерах. Приведем пример вызова макрокоманды с параметрами по-умолчанию:

```
PUTCHAR MACRO SIMBOL=REQ , CR=<PER>
```

В этом примере первый параметр является обязательным (задано служебное слово **REQ** – require, требуется), а второй параметр получает значение по умолчанию, которое используется в том случае, когда в макровывозе параметр не задан.

2.5. Макрорасширения

После компиляции на место макровывоза вставляются команды, полученные на основе макрорасширения (вставки текстового фрагмента). Эти команды должны строго соответствовать синтаксису языка Ассемблер, его семан-

тике, а также логике работы задуманной программы. Макрорасширение вставляется автоматически и его можно, при желании, включить в листинг программы. Для включения в листинг команд макрорасширения используются специальные директивы различного назначения:

- **.LALL** - получение в листинге всех макрорасширений
- **.SALL** - подавление в листинге всех макрорасширений
- **.XALL** – восстановление характеристик вывода листинга, заданного по умолчанию.
- **.LIST** – включает печать исходного текста программы.
- **.XLIST** – выключает всю печать исходного текста программы.

Макрорасширения для наших примеров получим в следующем виде:

```
PUTCHAR AL
1 PUSH AX
1 PUSH DX
1 MOV DL, AL ; Регистр
1 MOV AH, 02H
1 INT 021H
1 POP DX
1 POP AX
```

В первой строке макрорасширения показан уровень расширения. В нашем случае это “1”. Если в самой макрокоманде встречается другой макровывод, то эти строки макрорасширения будут помечены “2”, и так далее.

```
PUTCHAR VAR
1 PUSH AX
1 PUSH DX
1 MOV DL, VAR ; Переменная
1 MOV AH, 02H
1 INT 021H
1 POP DX
1 POP AX
```

В случае, когда при вызове макрокоманды передается ошибочный параметр, например, регистр **AX** (нужен 1 байт в регистр DL, а передается 2 байта):

```
PUTCHAR AX
1 PUSH AX
1 PUSH DX
1 MOV DL, AX
```

```
***** ERROR *****  
1 MOV AH , 02H  
1 INT 021H  
1 POP DX  
1 POP AX
```

Компилятор Ассемблера фиксирует ошибку, так как регистры (DL и AX) разной длины. При передаче параметра однобайтовой константы ошибки не возникает:

```
PUTCHAR 'A'  
1 PUSH AX  
1 PUSH DX  
1 MOV DL, 'A' ; Константа  
1 MOV AH , 02H  
1 INT 021H  
1 POP DX  
1 POP AX
```

2.6. Директивы написания макрокоманд

Помимо специальных директив описания макрокоманд (MACRO и MEND) в тексте программы на Ассемблере и в самих макрокомандах могут использоваться следующие директивы макрогенерации:

- **REPT** – директива циклического повторения текста по счетчику.
- **IRP** - директива циклического повторения текста по списку параметров.
- **IRPC** – директива циклического повторения по списку символов (строке).
- **EXITM** – директива завершения работы макрокоманды или цикла
- **LOCAL** – директива описания локальных переменных и меток макроса.
- **IF, IFE, ELSE, ENDIF** – директивы условной компиляции (логическое условие).
- **IF1, IF2, ELSE, ENDIF** – директивы условной компиляции (проходы компилятора).
- **IFDEF , IFNDEF, ELSE, ENDIF** – директивы условной компиляции (объявление переменных этапа компиляции).

- **IFB, IFNB, ELSE, ENDIF** – директивы условной компиляции (проверка наличия параметра макрокоманды).
- **IFIDN, IFIDIF, ELSE, ENDIF** - директивы условной компиляции (сравнение переменных этапа компиляции).
- Служебные символы: “%” , “&” , “!” , “;:” – используются для написания макрокоманд.

Рассмотрит ниже назначение и возможности основных директив описания макрокоманд.

2.7. Циклическая компиляция

Директивы циклической компиляции (**REPT, IRP, IRPC**) позволяют организовать циклы генерации текста программы.

Директива **REPT** имеет следующий синтаксис:

```
REPT <выражение этапа компиляции>  
<тело цикла>  
ENDM
```

Параметр <выражение этапа компиляции> определяет число повторений цикла. Тело цикла может содержать любые команды и операторы языка Ассемблер, включая макрокоманды. Цикл генерации будет повторяться заданное число раз. Это число вычисляется до начала цикла и не может изменяться в цикле. Простейший пример цикла:

```
REPT 3 ; Константа  
DB 0  
ENDM  
1 DB 0  
1 DB 0  
1 DB 0
```

Если параметром циклического оператора является переменная этапа компиляции **N**, то получим аналогичное макрорасширение:

```
N = 3  
REPT N ; Переменная  
DB N  
ENDM  
1 DB 3  
1 DB 3  
1 DB 3
```

Если задано выражение, то получим макрорасширение:

```
N = 3  
REPT N - 1 ; Выражение  
DB N  
ENDM  
1 DB 3  
1 DB 3
```

В теле цикла могут быть использованы выражения этапа компиляции:

```
K = 3  
N = 65 ; код 'A'  
REPT K  
DB N  
N = N + 1 ; В цикле изменяем переменную  
ENDM  
1 DB 65  
1 N = N + 1  
1 DB 66  
1 N = N + 1  
1 DB 67
```

Допустимо также использование вложенных циклов:

```
K = 2  
N = 3  
REPT K ; Цикл  
  REPT N ; Вложенный цикл  
  DB 0  
ENDM  
ENDM  
1 REPT N  
1 DB 0  
1 ENDM  
2 DB 0 ; Уровень вложенности 2  
2 DB 0  
2 DB 0  
1 REPT N  
1 DB 0  
1 ENDM  
2 DB 0  
2 DB 0  
2 DB 0
```

Подчеркнутым шрифтом выделены результаты первого прохода компиляции (уровень 1), а окончательные результаты расширения даны на втором уровне.

Директива **IRP** имеет следующий синтаксис:

IRP <переменная цикла>, <<список фактических параметров > >

<тело цикла>

ENDM

<список фактических параметров >:= <параметр> |

<параметр> , <список фактических параметров >

Директива **IRP** повторяет цикл столько раз, сколько указано фактических параметров в угловых скобках (отмечу, что в данном случае это терминальные символы). На каждой шуге цикла значение переменной цикла, являющейся формальным параметром этого цикла последовательно принимает значения из перечня параметров. Пример использования директивы **IRP** приведен ниже:

IRP p, <AX, BX, CX, DX> ; Четыре регистра
XOR p, p
ENDM

```
1 XOR AX, AX
1 XOR BX, BX
1 XOR CX, CX
1 XOR DX, DX
```

Жирным шрифтом текста показан код на макроязыке, а обычным макрорасширения. Другой пример иллюстрирует задание выражений в списке фактических параметров для директивы **IRP**. Здесь используются специальные символы и вспомогательные переменные этапа компиляции (N и W4), которым предварительно задаются значения (3 и 5 соответственно, с помощью присваивания и директивы эквивалентности).

```
N = 3
W4 EQU 5
IRP A, <N+1, %N + 1, W%N+1, %N, N, %W4> ; Пять выражений
DW A
ENDM
1 DW N+1
```

```
1 DW 4
1 DW W4
1 DW 3
1 DW N
1 DW 5
```

Служебный символ “%” позволяет взять значение переменной в выражении, об этих символах мы поговорим ниже. На втором уровне выделена подстановка значения переменной этапа компиляции W4.

Директива IRPC имеет следующий синтаксис:

IRPC <переменная цикла>,<Строка символов >

<тело цикла>

ENDM

<Строка символов > := <символ> | <Строка символов >

Для этого циклического оператора макроассемблера, переменной цикла (**D**) присваивается значение символа из строки перечисления и с каждым значением переменной из списка цикл выполняется. Число повторений цикла равно числу символов в строке. Пример использования циклического оператора:

```
IRPC D, 123456789 ; Девять символов цикла
V1&D DB D
ENDM
1 V11 DB 1
1 V12 DB 2
1 V13 DB 3
1 V14 DB 4
1 V15 DB 5
1 V16 DB 6
1 V17 DB 7
1 V18 DB 8
1 V19 DB 9
```

Формальный параметр “D” является одновременно переменной цикла. Для каждого символа из строки генерируется описание в виде байтовой переменной. В этом примере используется служебный символ макрокоманд “&”, который позволяет сливать имена переменных.

2.8. Локальные метки и переменные макрокоманд

При разработке макрокоманд могут использоваться вспомогательные метки и переменные. Так как макрокоманда может быть вызвана несколько раз, то при совпадении имен будет выдана ошибка и компиляция остановится. Для автоматической индексации таких имен используется специальный механизм локальных переменных. Для его использования внутри макроопределения задается директива макрокоманд LOCAL, которая имеет следующий синтаксис:

```
LOCAL <локальное имя> {< локальное имя > ... }
```

В качестве локального имени могут использоваться локальные переменные или локальные метки. Компилятор заменяет эти имена служебными идентификаторами (??dddd, где d – десятичная цифра) и обеспечивает сквозную нумерацию во всем исходном модуле. Так исключается дублирование описаний. Покажем применение локальных меток и переменных на примере. Пусть имеется макроопределение:

```
print MACRO CH, CR
LOCAL loc_per , loc_met
MOV DL, CH
MOV loc_per , DL
CALL DISPL
    IFIDN <CR>,<PER>
CALL crlf
ENDIF
JMP loc_met
loc_per db 0
loc_met:
ENDM
```

В этом макроопределении объявлена локальная метка (**loc_met**) и переменная (**loc_per**). При вызове макрокоманды мы получим в листинге следующие макрорасширения:

```
print '5', PER
1    MOV DL, '5'
1    MOV ??0000 , DL
1    CALL DISPL
1    CALL crlf
1    JMP ??0001
1    ??0000 db 0
```

1 **??0001:**

В следующем вызове этой же макрокоманды **print** в том же модуле программы Ассемблера метки и переменные будут проиндексированы (вместо **??0000** будет **??0002**):

```
print '7', PER
1    MOV DL, '7'
1    MOV ??0002 , DL
1    CALL DISPL
1    CALL crlf
1    JMP ??0003
1    ??0002 db 0
1    ??0003:
```

Общее число локальных меток и переменных во всех макрорасширениях одного исходного модуля, не может превышать 9999.

2.9. Служебные символы макрокоманд

Специальные служебные символы применяются для написания макрокоманд для достижения следующих целей:

- “%” - данный символ позволяет получить значение числовой или текстовой переменной этапа компиляции и вставить его в макрорасширение. Он работает в макрокомандах и циклах этапа компиляции.
- “&” - позволяет склеить строку символов и переменную этапа компиляции или параметр макроопределения (или наоборот).
- “!” - используется в аргументе для указания Ассемблеру, что символ, следующий за ним, является литералом, а не именем (параметра, переменной этапа компиляции).
- “<” “>” – используется для задания текста, который передается в макрорасширения целиком, даже если он содержит пробелы и другие знаки операций.

- “;” - используются для включения комментария в текст макроопределения. Эти комментарии не передаются в макрорасширения.

Рассмотрим примеры. Служебные символы “%” и скобки “<” “>”. Если переменная символьная. Знак процента позволяет взять значение переменной этапа компиляции, вне зависимости от того как она задается. Для оператора EQU мы получим:

```
L EQU <'&text'>
IRP D,<%L, L >
  DB D
ENDM
  1    DB '&text'
  1    DB L
```

При использовании числовой переменной, задаваемой присваиванием, получим следующее расширение:

```
L = 2
IRP D,<%L, L >
  DB D
ENDM
  1    DB 2
  1    DB L
```

Служебный символ “&”. Опишем макрокоманду для генерации макрокоманд MOVSB и MOVSW. Проверим ее вызовы двумя разными параметрами.

```
; Описание макрокоманды
MOVE MACRO TAG
REP MOVS&TAG
  ENDM
;; Вызов макрокоманды и расширение
MOVE W
1 REP MOVSW
MOVE B
1 REP MOVSB
```

Служебный символ “&” используется для слияния константы и формального параметра макрокоманды (TAG). Следующая макрокоманда иллюстрирует использование почти всех служебных символов (“&”, “%”, “%”).

```
TESTPR MACRO PAR1 , PAR2 , V ; Три параметра в заголовке
LOCAL MET, MSG
; Вызов по параметру
MOV AH , 09H
MOV DX , OFFSET PAR1
INT 21H
```

```
;; Вызов по локальному параметру
MOV AH , 09H
MOV DX , OFFSET MSG
INT 21H
JMP MET
MSG DB '&PAR2 ---','!&V ,&V , %V,!%V'
MET:
ENDM
```

Параметры PAR1 и PAR2 вставляются в макрорасширение по значению.

Пример вызова:

```
MSG5 DB 'TEST$',10,13
...
.LALL
TESTPR MSG5 , <Пример текста !%V> , 1 ; Вызов макрокоманды
1 ; Вызов по параметру
1     MOV AH , 09H
1     MOV DX , OFFSET MSG5
1     INT 21H
1     MOV AH , 09H
1     MOV DX , OFFSET ??0008
1     INT 21H
1     JMP ??0007
1     ??0008 DB 'Пример текста %V ---','!1 ,1 , %V , !%V'
1     ??0007:
```

В этом примере используются при вызове макрокоманды угловые скобки, для передачи в макрокоманду текста с несколькими пробелами. В примере видно, что параметры &V будет заменены (в примере на 1). Параметр в тексте %V должен в передаваемом тексте (PAR2) быть помечен знаком “!”, в противном случае будет отмечена ошибка (он будет интерпретирован как литерал). Значение &PAR1 и просто PAR1 в этом примере являются эквивалентными (проверьте!).

2.10. Переопределение и удаление макрокоманд

В процессе одной макрогенерации можно переопределить макрос. Для этого достаточно задать новое макроопределение с таким же именем. Например:

```
;; Описание первого макроса
A MACRO V1
    INC V1
ENDM
```

```
;; Вызов первого макроса A  
A AH
```

Расширение:

```
1 INC AH  
;; Описание/переопределение второго макроса A  
A MACRO V1,V2  
  INC V1  
  DEC V2  
ENDM  
;; Вызов второго макроса A  
A AH,AL
```

Расширение:

```
1 INC AH  
1 DEC AL
```

Директива `PURGE` исключает макрос из списка доступных на данный момент макрогенерации:

```
;; Удаление макроса A и дальнейшая попытка его запустить  
PURGE A  
A AX
```

Расширение:

...

Расширения нет, так как данный макрос выключен, а в окне ошибок появляется предупреждение.

2.11. Вложенные вызовы макрокоманд

Из одних макрокоманд можно вызывать другие макрокоманды. Такие вызовы называются вложенными. Использование вложенных макрокоманд позволяет сделать настройки программы более гибкими и сократить размеры макрокоманд. Рассмотрим пример. Первая макрокоманда заполняет поле значением параметра.

```
; Макрокоманда выделения памяти с заполнением выражением (VAR)  
FIELD MACRO VAR  
DB &VAR  
ENDM
```

Во второй макрокоманде организован цикл заполнения массива (`NAME`). Натуральными числами в порядке возрастания, начиная с базового значения (`BASE`), с определенным шагом (`STEP`) и определенного размера (`SIZEM`).

```
; Натуральные числа в порядке возрастания  
MAS MACRO SIZEM , NAME , BASE , STEP  
FCount = BASE  
&NAME DB BASE  
REPT SIZEM - 1  
FIELD %FCount + STEP ; Вызов макрокоманды для описания и заполнения поля  
FCount = FCount + 1  
ENDM  
ENDM
```

При вызове макрокоманды мы получим следующее расширение:

```
MAS 10 , T0 , 3 , 5 ; 5 шаг данных  
1 T0 DB 3  
3 DB 8  
3 DB 9  
3 DB 10  
3 DB 11  
3 DB 12  
3 DB 13  
3 DB 14  
3 DB 15  
3 DB 16
```

Макрокоманда генерирует массив T0 размером в 10 элементов, начиная с 3 с шагом 5.

2.12. Директивы условной компиляции

Директивы условной компиляции позволяют включать или не включать текст в исходную программу в зависимости от разнообразных условий, а также выбирать альтернативный вариант включения текста . Перечень групп директив (они разделены по типам условий) приведен ниже:

- **IF1**, **IF2**, **ELSE**, **ENDIF** – директивы условной компиляции, выделяют разные проходы исходного текста программы компилятором (первый, второй).
- **IF**, **IFE**, **ELSE**, **ENDIF** – директивы условной компиляции, основанные на логическом условии этапа компиляции.
- **IFDEF** , **IFNDEF**, **ELSE**, **ENDIF** – директивы условной компиляции, базирующиеся на возможном объявлении (описании, задании) переменных этапа компиляции.

- **IFB, IFNB, ELSE, ENDIF** – директивы условной компиляции, применительно к параметрам макрокоманд, включающие проверку наличия или отсутствия параметров макрокоманды.
- **IFIDN, IFIDIF, ELSE, ENDIF** - директивы условной компиляции, выполняющие сравнение переменных этапа компиляции.

Рассмотрим кратко возможности директив условной компиляции. В общем виде для всех директив задается следующая конструкция:

```
IFXX <аргументы условного оператора>  
    <директивы для включения в текст программы при истинности условий>  
ENDIF
```

Обозначение **IFXX** заменяет все разновидности директив условной компиляции.

Директивы IF1, IF2 включают текст в исходную программу, если выполняется первый (IF1) или второй (IF2) проход компилятора. Обычно это делается при подключении библиотек макрокоманд, так как повторное включение приводит к ошибкам. Например:

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

Макробибблиотека **MACRO.LIB** для нашего примера будет включена в текст исходного модуля только на первом проходе компилятора.

Директивы IF, IFE, ELSE, ENDIF включают текст в исходную программу, на основе логических условий этапа компиляции. Если условие выполнено (равно нулю для IFE), то текст в исходную программу включается. Если условие не выполнено (не равно нулю для IF), то текст в исходную программу включается. Конструкция **ELSE** позволяет задать альтернативу по этому условию. Синтаксис и семантика директив задан ниже:

```
IF <логическое выражение>
```

<операторы условного блока>**ENDIF**

Если значение логического выражения не нулевое (истина в Ассемблере), операторы условного блока включаются в исходный модуль программы.

IFE < логическое выражение>**<операторы условного блока>****ENDIF**

Если значение логического выражения равно нулю, операторы условного блока включаются в исходный модуль программы.

Значения истина и ложь в Ассемблере определяются так:

FALSE (0000h) - ложь

TRUE (FFFFh) - истина

В общем виде конструкции всех условных директив можно описать и так, с учетом возможности альтернативных включений текста:

IFXX <аргументы>

<директивы для истинного значения условия>

[ELSE

<директивы для ложного значения условия>

]

ENDIF

Рассмотрим простые примеры условных директив:

```
IF N EQ 5
    print 'A', PER
ELSE
    print 'B', PER
ENDIF
```

Если переменная этапа компиляции N равна 5, то выполняется обращение к макрокоманде print с параметром 'A', в противном случае с параметром 'B'.

```
IFIDN <&CR>, <PER>
    CALL LFCR
ENDIF
```

Если параметр макрокоманды (CR) установлен в значение <PER>, то вызывается процедура перевода строки и возврата каретки.

```
IFNDEF COUNT
```

```
COUNT = 0  
ENDIF
```

Если переменная этапа компиляции COUNT пока не определена, то она определяется с начальным значением равным нулю.

```
IFB V  
    JMP MET  
ENDIF
```

Если параметр макрокоманды V не задан при вызове (BLANK - пустой), то генерируется команда безусловной передачи управления на метку.

Надеюсь, что более полную информацию по условной компиляции вы найдете в литературе, документации и справочниках по ассемблеру и макроассемблеру (см. список литературы). Практическое использование таких директив Вы должны продемонстрировать при выполнении 9-й ЛР.

2.13. Директива EXITM

Обычно работа макрокоманды завершается в тот момент, когда встретилась последняя для рассмотрения директива **ENDM** макроопределения. Существует и другая возможность завершить работу макрокоманды (как процедуры этапа компиляции) – это использование директивы EXITM. Если встретилась такая директива, то дальнейший режим обработки макровызова завершается и Ассемблер переключается на обработку команды, следующей за данным макровыводом. В одном макроопределении может быть несколько выходов, которые выполняются директивой EXITM. Например, если после проверки параметра мы обнаруживаем ошибку, то можем сделать выход из макроопределения. Например:

```
...  
IF <PAR> NE <TEST>  
...  
    JMP MET  
ELSE  
    EXITM  
...
```

ENDIF

...

2.14. Отладка макрокоманд

Отладка макрокоманд является трудоемкой задачей, так как нет специальных отладчиков для этого. Результат работы макрокоманд можно увидеть только в листинге программы с включенным режимом макрорасширений (см. выше). По результатам макрорасширений можно судить о правильности их работы. При необходимости можно выдать сообщение на стандартный вывод (дисплей или окно ошибок) собственное пользовательское сообщение. Это выполняется директивой %OUT. Например:

```
%OUT Отладка макрокоманды print
```

В окно ошибок будет помещено это сообщение. Если такая директива встретится в макрокоманде, то будет напечатано это сообщение. Для инициации (форсированном выполнении прерывания компиляции) ошибки используется другая директива - .ERR. Например:

```
.ERR  
%OUT Ошибка в макрокоманде print
```

В окно ошибок будет помещено это сообщение, в текст листинга также вставиться сообщение и компиляция останавливается.

2.15. Сравнение макросов и процедур

По сравнению с процедурами макрокоманды имеют, как минимум, следующие четыре преимущества:

1. Макрокоманды являются более гибкими по сравнению с процедурами. Они позволяют задавать параметры, пропускать их при вызове макрокоманд, выполнять условную компиляцию.
2. Макрокоманды выполняются с более высокой скоростью (более эффективной), так как не тратится время на команды вызова, а все действия по настройке выполняются на этапе компиляции.
3. Макрокоманды легко упаковать в библиотеку и подключать их в программу.
4. Программа, построенная на основе макрокоманд, является более наглядной и понятной.

Недостатками макрокоманд по сравнению с процедурами является:

1. Программа, построенная на основе макрокоманд, имеет большой размер, так как при каждом вызове макрокоманды выполняется макрорасширение. В некоторых случаях это может стать существенным недостатком.
2. Макрокоманды труднее отлаживать, так как трудно создать специальный отладчик, так как все выполняется на этапе компиляции.

В целом, можно сделать вывод, что если рационально использовать комбинацию механизмов макрокоманд и процедур, то можно получить эффективный и наглядный код даже на языке Ассемблера.

2.16. Макрокоманды – процедуры этапа компиляции

Для тех, кто хорошо освоил механизм макрокоманд, приходит понимание того, что механизм макрокоманд и условной компиляции программ очень мощный и эффективный механизм для создания программных систем. Главное понять, что макрокоманды, по сути, являются процедурами этапа компиляции. Так как существует возможность вложенного вызова макрокоманд, возмож-

ность определения переменных этапа компиляции и использования библиотек, то возникает возможность описания собственного языка специального назначения. Действительно существуют специализированные системы программирования, целиком построенные на основе макроассемблера. Например, система имитационного моделирования GPSS в первых своих версиях была реализована по этой схеме. Важно, что трудозатраты на такие системы значительно ниже, чем при создании системы программирования с нуля.

2.17. Примеры программы с макросами

В следующем тексте работающей программы с макросами, которые я привожу без пояснений, вы найдете много полезных фрагментов для выполнения 9-й ЛР.

```
mycode segment 'code'
  assume cs:mycode, ds:mycode
  assume ss: stseg
  ; Макрокоманда выделения памяти с заполнением выражениями
  FCount = 1
  Fparam = 1
  FIELD MACRO VAR
    DB &VAR
  ENDM
  ;;; Натуральные числа в порядке возрастания
  MAS MACRO SIZEM , NAME , BASE
    FCount = BASE
    &NAME DB BASE
    REPT SIZEM - 1
    FIELD %FCount + 1
    FCount = FCount + 1
  ENDM
  ENDM
  ;;; Натуральные числа в порядке убывания
  MASM MACRO SIZEM , NAME , BASE , STEP
    FCount = BASE
    &NAME DB BASE
    REPT SIZEM - 1
    FIELD %FCount - 1
    FCount = FCount - STEP
  ENDM
  ENDM
```

; Арифметическая прогрессия

```
MASN MACRO SIZEM , NAME , BASE , STEP
  FCount = 1
  &NAME DB BASE
  REPT SIZEM - 1
  FIELD %FCount * STEP + BASE
  FCount = FCount + 1
  ENDM
ENDM
```

; Геометрическая прогрессия

```
MASP MACRO SIZEM , NAME , BASE , STEP
  FCount = 1
  &NAME DB STEP + BASE
  REPT SIZEM - 1
  FParam = 1
  REPT Fcount + 1
  FParam = Fparam * STEP
  ENDM
  FIELD %FParam + BASE
  FCount = FCount + 1
  ENDM
ENDM
```

; Макрокоманда суммирования

```
SUM MACRO NAME , SIZE , VSUM
  LOCAL VM
  LOCAL VERR
  PUSH CX
  PUSH AX
  PUSH DX
  PUSH SI
  MOV CX , SIZE
  CMP CX , 0
  JB VERR
  XOR AX , AX
  MOV SI,0
  XOR DX , DX
  VM: MOV DL , NAME[SI]
  ADD AX, DX
  INC SI
  LOOP VM
  MOV VSUM , AX
  MOV DX, AX
  CALL PRINTDEC
  VERR: POP SI
  POP DX
  POP AX
  POP CX
  ENDM
```

;Макрокоманда для произведения

```
PRSTR MACRO NAME , SIZE , KSTR
  LOCAL MC
  LOCAL MC1
  LOCAL MFIN
```

```
    LOCAL MPROD
    PUSH CX
    PUSH SI
;; НУ цикла
;; Перевод строки
    mov dl, 0Ah
    mov ah, 2
    int 21h
    mov dl, 0Dh
    mov ah, 2
    int 21h
    MOV CX , SIZE
    MOV SI , 0
;; Цикл по массиву
MC:  PUSH CX
    MOV CX , KSTR
;; Цикл по строке
MC1: MOV DL, NAME[SI]
    CALL PRINTB
    INC SI
    CMP SI , SIZE
    JNE MPROD
;; Последний перевод
    mov dl, 0Ah
    mov ah, 2
    int 21h
    mov dl, 0Dh
    mov ah, 2
    int 21h
    JMP MFIN
MPROD:
    LOOP MC1
    POP CX
;; Перевод строки
    mov dl, 0Ah
    mov ah, 2
    int 21h
    mov dl, 0Dh
    mov ah, 2
    int 21h
    LOOP MC
;; Конец Цикла
MFIN:
    POP SI
    POP CX
    ENDM
;;Макрокоманда печати массива в десятичном виде по столбцам
PRCOL MACRO NAME , SIZE , KCOL
    LOCAL MC
    LOCAL MC1
    LOCAL MFIN
    LOCAL MPROD
    LOCAL TEMP
```

```
LOCAL OBXOD
LOCAL MTEMP
PUSH CX
PUSH SI
;; НУ цикла
;; Перевод строки
mov dl, 0Ah
mov ah, 2
int 21h
mov dl, 0Dh
mov ah, 2
int 21h
;; Определение числа строк
MOV CX , SIZE/KCOL
MOV SI , SIZE MOD KCOL
CMP SI , 0
JE MTEMP
ADD CX, 1
MTEMP: MOV TEMP , CX
;;
MOV SI , 0
;; Цикл по массиву
MC: PUSH CX
MOV CX , KCOL
PUSH SI
;; Цикл по строке
MC1: MOV DL, NAME[SI]
CMP SI , SIZE
JGE MPROD
CALL PRINTB
ADD SI , TEMP
LOOP MC1
;;
MPROD:POP SI
INC SI
POP CX
;; Перевод строки
mov dl, 0Ah
mov ah, 2
int 21h
mov dl, 0Dh
mov ah, 2
int 21h
LOOP MC
;; Конец Цикла
MFIN:
POP SI
POP CX
JMP OBXOD
TEMP DW 0
OBXOD:
ENDM
;
```

```
main proc
;Занесение регистра DS
    PUSH CS
    POP DS
;-----
;-----
    JMP MEND1
; Описания и генрация массивов
.XALL
    MAS 10 , T0 , 3
.SALL
    MASM 5 , T1 , 100
    MASN 5 , T2 , 1 , 5
.XALL
    MASP 5 , T3 , 10 , 3
.SALL
    PRSTR T0 , 5 , 3
; PRCOL T0 , 5 , 3
; выход с ожиданием
    VAR DW 0
; Запрос символа с клавиатуры
; Цикл Суммирования
MEND1:
    SUM T0 , 10 , VAR
    MOV DX , VAR
    CALL printdec
    MOV DL , 131
    CALL printb
    PRSTR T0 , 20 , 5
; PRCOL T0 , 27 , 6
    mov ah, 08h
    int 21h
    mov al, 0
; Выход в ДОС
    mov ah, 4ch
    int 21h
main endp
;-----
;-----
; Печать числа
; DX - число
printdec proc
; Перевод числа
; 10000
    PUSH CX
    PUSH SI
    PUSH AX
; Начальные условия цикла перевода
    MOV SI , 0
    MOV CX , 4
; DX:AX - делимое
    MOV DDWORK , 10000
; Цикл перевода
CICLE:
```

```

MOV AX, DX
MOV DX, 0
DIV DDWORK
; AX - остаток , DX - частное
XCHG DX, AX
; Коррекция для вывода символа
ADD dl, 30h
CMP dl, 30h
JNE MC1
MOV dl, 20h
MC1:  MOV CHI[SI], dl
      INC SI
; DX - остаток , AX - частное
XCHG DX, AX
; Новый делитель
PUSH DX
PUSH AX
MOV AX, DDWORK
MOV DX, 0
DIV D10
MOV DDWORK, AX
POP AX
POP DX
LOOP CICLE
; На конец цикла
; Коррекция для вывода символа
ADD dl, 30h
MOV CHI[SI], dl
; Вывод строки
MOV AH, 09h
MOV DX, offset STR
int 021h
POP AX
POP SI
POP CX
ret
printdec endp
; Перевод десятичное и печать байтовой переменной
; без перевода строки DL - байт
printb proc
PUSH CX
PUSH SI
PUSH AX
; Начальные условия цикла перевода
MOV SI, 0
MOV CX, 2
; DX:AX - делимое
MOV DDWORK, 100
MOV DH, 0
; Цикл перевода
CICLE1:
MOV AX, DX
MOV DX, 0

```

```
    DIV DDWORK
; AX - остаток , DX - частное
    XCHG DX , AX
; Коррекция для вывода символа
    ADD dl , 30h
    CMP dl , 30h
    JNE MC11
    MOV dl , 20h
MC11:  MOV CHIB[SI] , dl
    INC SI
; DX - остаток , AX - частное
    XCHG DX , AX
; Новый делитель
    PUSH DX
    PUSH AX
    MOV AX , DDWORK
    MOV DX , 0
    DIV D10
    MOV DDWORK , AX
    POP AX
    POP DX
    LOOP CICLE1
; На конец цикла
; Коррекция для вывода символа
    ADD dl , 30h
    MOV CHIB[SI] , dl
; Вывод строки
    MOV AH , 09h
    MOV DX , offset CHIB
    int 021h
    POP AX
    POP SI
    POP CX
    ret
printb endp
;;
; - mas - size - kcol
;
DDWORK DW 0
D10  DW 10
DD10  DW 10000
DD11  DW 1000
DD12  DW 100
DD13  DW 10
STR  DB 'Результат = '
CHI  DB ' ', 10, 13, '$'
CHIB DB ' ', '$'
;-----
mycode ends
stseg segment stack 'stack'
    dw 256 dup(0)
stseg ends
```

```
;-----  
end main
```

3. Цели и задачи ЛР № 9

Задание на лабораторную работу заключается в разработке программы на языке макроассемблера, в которой студенты дают описание собственных макрокоманд по темам определенным в вариантах и демонстрируют их правильное использование на конкретных примерах. Должны быть разработаны макрокоманды с параметрами, включающие условную компиляцию, локальные переменные и вложенные макрокоманды. В ходе работы студенты разрабатывают: макрокоманду вывода текстовых данных, макрокоманду описания массива с конкретным заполнением (по варианту), макрокоманду расчета агрегированных характеристик массива (сумма, произведение и т.д.) и макрокоманду вывода на печать результатов работы программы в виде таблицы по колонкам или строкам.

4. Требования к выполнению лабораторной работы №9

При разработке программы с макрокомандами и их отладке, студент должен выполнить следующие задания:

1. Разработать простую макрокоманду вывода символа на экран дисплея. Задается один параметр – выводимый символ. Символ выводиться из однобайтовой переменной или однобайтового регистра, отличающегося от регистра **DL**. Сохранить и восстановить регистры в теле макроса, которые используются в данной макрокоманде. Макрокоманду назвать **PRINTC**. Продемонстрировать

в основной программе использование макрокоманды с: константой ('A'), переменной (**VAR**) и регистром (**BL**).

2. Разработать Макрокоманду **вывода на экран дисплея одного символа** (на консоль) с двумя параметрами: выводимый символ (1-й параметр) и признак перевода строки после введенного символа (это 2-й параметр, для него нужно указать специальную текстовую константу "**PER**"). Для вывода символа и перевода строки на экране использовать процедуры, разработанные ранее в цикле ЛР по ассемблеру(в 3ЛР): **DISPL** и **CRLF**. В основной процедуре нужно продемонстрировать использование (макровызов) созданной макрокоманды во всех режимах, включая и пример ошибочного задания параметров. При наличии ошибки при вызове макрокоманды выдается оператором сообщение макроассемблера **%OUT** ("Ошибка параметра макрокоманды") с включением режима ошибок - **.ERR**. Контроль пустого параметра выполнять условным оператором **IFNB** или **IFB**. Макрокоманду назвать **PRINT**. Данное задание является общим для всех студентов и не имеет вариантов. Контроль правильности генерации макрокоманд проверяется по листингу, выдаваемому макроассемблером. Для управления листингом нужно освоить и использовать псевдооператоры управления листингом макроассемблера: **.LALL** , **.SALL**, **.XALL**.

3. Разработать макрокоманду генерации **описания и заполнения массива** (назвать макрокоманду **MAS**). В зависимости от варианта массив может быть байтовым (**DB**) или двухбайтовым (**DW**). Макрокоманда должна иметь четыре параметра: размер массива (числовой), название массива (уникальное имя), базовое значение для заполнения (опорное число) и параметр заполнения массива (шаг прогрессии). В зависимости от варианта заполнение массива вы-

полняется: натуральными числами, возрастающими по значению; натуральными числами, убывающими по значению; последовательностью арифметической прогрессии; последовательностью геометрической прогрессии. Для заполнения массива можно использовать вложенные макрокоманды (см. пример выше) и целые макропеременные (целые переменные декларируются с помощью начального присваивания - "=", а константы с помощью EQU). Числа для заполнения массива должны быть сгенерированы на этапе макропроцессора, быть положительными и возможные значения размерности должны быть не менее 10. Продемонстрировать использование макрокоманды описания массивов для разных случаев параметров, в том числе и при ошибочном задании параметров. При разработке макроопределения использовать операторы макроассемблера: "=", "%" и "&". При написании макрокоманды MAS необходимо использовать оператор REPT. Название массива передается в качестве параметра и должно устанавливаться у первого элемента массива.

Примечание:

Результат генерации массива должен быть примерно таким:

```
1 NAME DB 1
1     DB 2
1     DB 3
...
```

Значения переменных заполнения и их диапазон изменения зависят от варианта.

В примере показано заполнение натуральными числами с базовым значением 1.

4. Разработать макрокоманду **подсчета** агрегированных характеристик сгенерированного массива (см. п.3). В макрокоманде нужно предусмотреть следующие параметры: название массива (сгенерированного массива), размерность для подсчета (число элементов), переменная, в которую возвращается результат вычисления (например сумма). В зависимости от варианта, в макрокоманде

де выполняется подсчет: **суммы** заданного числа переменных массива, их **произведения**, числа **нечетных** элементов массива или числа элементов, в которых **бит четности** для числа равен нулю (не путать с четными и нечетными переменными). При организации цикла в макрокоманде использовать локальные метки и переменные (оператор **LOCAL**). Предположить для массива из слов (DW), что результат расчета сможет также поместиться в переменную типа слово (DW). В случае переполнения результата возвращать "-1" (для суммы и произведения). При выполнении макрокоманды сохранять все используемые регистры и восстанавливать их при ее завершении (PUSH, POP). Результат расчета распечатать в десятичном виде, для чего нужно воспользоваться процедурами перевода и печати, разработанными в предыдущих лабораторных работах по Ассемблеру.

5. Разработать макрокоманду **распечатки** массива чисел. Распечатка выполняется в десятичном формате, для этого используются процедуры из предыдущих лабораторных работ по ассемблеру (для перевода в десятичный формат печати). В зависимости от варианта распечатка выполняется по строкам, с указанием числа чисел в строке и по колонкам, с указанием числа столбцов. Макрокоманда должна содержать следующие параметры: название массива, число элементов, которые должны быть распечатаны, параметры распечатки (число строк или столбцов). При распечатке между отдельными числами должны быть размещены пробелы, лидирующие нули также должны заменяться на пробелы. При организации циклов должны определяться локальные метки и переменные (LOCAL).

Примечание:

Распечатка по строкам предположительно имеет вид (число чисел в строке - 5, а размер массива - 8):

1 2 3 4 5
6 7 8

Распечатка по столбцам имеет вид (число столбцов - 6, размер массива - 11):

1 3 5 7 9 11
2 4 6 8 10

6. Продемонстрировать использование в программе операторы макроассемблера: **IRP** и **IRPC** (на основе примеров теоретического раздела).
7. Дополнительные требования. Разработать систему макрокоманд по выбору из вариантов представленных ниже.

5. Перечень вариантов ЛР по группам

Для различных групп выделяются особенности выполнения задания лабораторной работы, которые представлены ниже в таблице.

№	Группа	Переменные массива	Печать массива
1.	ИУ5-41	DB	По строкам
2.	ИУ5-42	DW	По строкам
3.	ИУ5-43	DB	По столбцам
4.	ИУ5-44	DW	По столбцам
5.	СУЦ	DB	По строкам

6. Перечень вариантов для заполнения массива

№ Вар. По журн.	Подсчет	Заполнение
1 - 7	Суммы элементов массива	Натуральные числа в порядке возрастания (1,2,3, ...)
8 - 14	Числа нечетных элементов массива	Арифметическая прогрессия целых чисел (задается разность прогрессии)

15 - 21	Произведения элементов массива	Натуральные числа в порядке убывания (10,9,8, ...)
21- 30	Числа элементов массива, для которых вычисленные бит четности равен 0	Геометрическая прогрессия целых чисел (задается знаменатель прогрессии)

7. Перечень вариантов для сильных и продвинутых студентов

Варианты выбираются по желанию, но не более одного варианта в группе.

№ Вар. доп.	Содержание системы макрокоманд	Примечание
1.	<u>Система</u> макрокоманд для структурного программирования	Макросы: ветвления (IF-ELSE), цикла (FOR), переключателя (SWITCH),
2.	<u>Система</u> макрокоманд ввода и вывода данных	С учетом перевода и форматирования данных для вывода.
3.	<u>Система</u> макрокоманд для вызова процедур с параметрами.	Предусмотреть вызов по значению и по адресу.
4.	<u>Система</u> макрокоманд для взаимного перевода данных из разных форматов друг в друга	Строковые, целые (DB, DW, DD) и действительные
5.	<u>Система</u> макрокоманд для работы с файлами	Предусмотреть все операции для последовательных файлов
6.	<u>Система</u> макрокоманд для работы с записями	Должна быть удобная работа с записями для доступа к полям и выборки данных
7.	<u>Система</u> макрокоманд для работы с датами и временем в разных форматах	Предусмотреть упаковку и распаковку двоично-десятичных форматов.

8. Контрольные вопросы по лабораторной работе

1. На каком этапе компиляции работает макропроцессор?
2. В каких случаях выгоднее использовать макрокоманды по сравнению с процедурами?
3. В каких случаях выгоднее использовать процедуры по сравнению с макрокомандами?
4. Каковы основные преимущества и недостатки макрокоманд?
5. Для чего нужен оператор LOCAL?
6. Какую операцию выполняет оператор макроассемблера "&"?
7. Какую операцию выполняет оператор макроассемблера "%"?
8. Что нужно сделать для отмены или включения печати макрорасширений в распечатке ассемблера?
9. Можно ли из одной макрокоманды вызывать другую макрокоманду с параметрами?
10. Как определить локальную переменную в макрокоманде (не метку)?
11. Для чего используются операторы: REPT, IRP, IRPC?
12. Как завершить выполнение макрокоманды, не достигая ее конца?
13. Как можно изменить значение текстовой переменной этапа компиляции?
14. Как можно изменить значение целочисленной переменной этапа компиляции?
15. Как и куда можно вывести сообщение об ошибках этапа компиляции, определяемых пользователем?
16. Как можно подключить библиотеку внешних макрокоманд?
17. Как можно связать выполнение различных макрокоманд?
18. Что такое условная компиляция? Приведите примеры операторов.
19. Перечислите основные операторы макроассемблера.

20. Можно ли считать, что макрокоманда – это процедура, которая выполняется на этапе компиляции программы?!

9. Требования к оформлению отчета по ЛР

В отчет по лабораторной работе должно входить (см. шаблон ниже):

1. Титульный лист (группа, ФИО, вариант)
2. Постановка задачи (общие требования и требования варианта)
3. Блок-схемы макрокоманд программы.
4. Листинг программы с расширениями вызовов макрокоманд.
5. Результаты запуска программы при проверке работы макрокоманд.
6. Перечень основных ошибок, которые возникали и были исправлены при отладке макрокоманд. Отсутствие перечня ошибок, или копирование его у других студентов, для меня дает дополнительную информацию о самостоятельности работы над заданием лабораторной работы конкретного студента.

10. Литература по ЛР СП

1. К-Г.Финогенов Основы языка Ассемблера.— М.: Радио и связь, 2001, 288 с.
2. П.И.Рудаков, К.Г.Финогенов “Язык ассемблера: Уроки программирования” – М.: ДИАЛОГ-МИФИ, 2001 г., 640с.
3. К.Г. Финогенов “Самоучитель по системным функциям MSDOS”- М.,РиС,Энтроп, 1995 г. 382с.
4. В.Н. Пильщиков “Программирование на языке ассемблера IBM PC ” – М.:: “ДИАЛОГ-МИФИ”, 1996. – 288с.
5. Жуков А.В., Авдюхин А.А. “Ассемблер” – СПб.: БХВ - Петербург, 202. 448 с.

6. Скэнлон Л. “Персональные ЭВМ IBM PC. Программирование на языке ассемблера.” -М.,РиС,1991 г.

7. Р.Джордейн “Справочник программиста персональных компьютеров типа IBM PC”- М.,ФиС, 1991г.

8.2. Список литературы по дисциплине СП, представленный на сайте (www.sergebolshakov.ru).

9. “Методическое пособие для выполнения лабораторных работ по дисциплине СП”, представленное на сайте (www.sergebolshakov.ru)..

11. Пример программы с макрокомандой

Пример программы с макрокомандой печати символа с переводом строки дан ниже. Здесь показаны макрокоманды похожие на те, которые вам нужно сделать в первом во втором пункте задания ЛР №9. Описание макрокоманды и макровыводы (их всего три) выделены жирным шрифтом. Исходный текст программы дан ниже:

```
mycode segment 'code'
assume cs:mycode, ds:mycode
assume ss: stseg
; Макрокоманда печати символа с переводом строки
; 1-й параметр= символ, второй параметр признак перевода "PER"
PRINT MACRO ch, CR
    mov dl, ch
    mov ah, 2
    int 21h
IFIDN <CR>,<PER>
    mov dl, 0Ah
    mov ah, 2
    int 21h
    mov dl, 0Dh
    mov ah, 2
    int 21h
ELSE
    IFNB <CR>
    .ERR
```

```

        %OUT ERROR Ошибка параметра перевода строки
    ENDIF
ENDIF
ENDM
; Макрокоманда печати одного символа
VARC DB 'C'
PRINTC MACRO SIMB
    PUSH DX
    PUSH AX
    MOV DL , SIMB
    MOV AH , 02H
    INT 21H
    POP AX
    POP DX
ENDM
main proc
;Занесение регистра DS
    PUSH CS
    POP DS
;-----
;-----
    PRINT '5', PER
; Ошибка в команде -> print 'A', AAA
; При использовании вызова, расположенного выше генерируется ; ошибка пользователя
; Корректные вызовы
    print 'A'
    PRINT ' ', PER
; Корректные вызовы другой команды печати символа
PRINTC 'A'
    MOV AL , 'B'
PRINTC AL
PRINTC VARC
; выход с ожиданием
; Запрос символа с клавиатуры
    mov ah, 08h
    int 21h
    mov al, 0
; Выход в ДОС
    mov ah, 4ch
    int 21h
main endp
;-----
mycode ends
;-----
stseg segment stack 'stack'
    dw 256 dup(0)
stseg ends
;-----
end main

```

Листинг программы, полученный при ее компиляции, приведен ниже. Жирным шрифтом я выделил макрорасширения, а размером шрифта и жирным шрифтом макровыводы:

```

0000          mycode segment 'code'
              assume cs:mycode, ds:mycode
              assume ss: stseg
              ; Макрокоманда печати символа с переводом строки
              ; 1-й параметр= символ, второй парметр признак перевода "PER"
              PRINT MACRO ch, CR
                  mov dl, ch
                  mov ah, 2
                  int 21h
                  IFIDN <CR>, <PER>
                      mov dl, 0Ah
                      mov ah, 2
                      int 21h
                      mov dl, 0Dh
                      mov ah, 2
                      int 21h
                  ELSE
                      IFNB <CR>
                          .ERR
                          %OUT ERROROR Ошибка параметра перевода строки
                      ENDIF
                  ENDIF
              ENDM
              ; Макрокоманда печати одного символа
0000 43      VARC DB 'C'
              PRINTC MACRO SIMB
                  PUSH DX
                  PUSH AX
                  MOV DL , SIMB
                  MOV AH , 02H
                  INT 21H
                  POP AX
                  POP DX
              ENDM
0001          main proc
              ;Занесение регистра DS
0001 0E      PUSH CS
0002 1F      POP DS
              ;;;;;;;;;;;;;;;;;;
              PRINT '5', PER
0003 B2 35      1      mov dl, '5'
0005 B4 02      1      mov ah, 2
0007 CD 21      1      int 21h
0009 B2 0A      1      mov dl, 0Ah
000B B4 02      1      mov ah, 2
000D CD 21      1      int 21h
000F B2 0D      1      mov dl, 0Dh
0011 B4 02      1      mov ah, 2
0013 CD 21      1      int 21h
              ; Ошибка в команде -> print 'A', AAA
              ; При использовании вызова, расположенного выше генерируется ;
ошибка пользователя
              ; Корректные вызовы

```



```

0000 0100[                dw 256 dup(0)
           0000
           ]

0200                stseg ends
                ;-----
                end main

```

Если макрокоманда вызвана с ошибочным параметром, в тексте примера этот вызов закомментирован, то макроассемблер сформирует сообщение об ошибке на этапе работы макропроцессора. Здесь внесена ошибка параметра “AAA” вместо “PER”:

```

...
; Ошибка -
    print 'A', AAA
...

```

Пример ошибки при вызове нашего макроса и сообщения об ошибке в листинге:

```

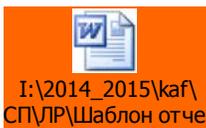
; Ошибка -
                                print 'A', AAA
0014 B2 41                1        mov dl, 'A'
0016 B4 02                1        mov ah, 2
0018 CD 21                1        int 21h
                                1        .ERR
**Error** MAC1S.asm(33) PRINT(13) User error
...

```

И сообщения об ошибке в окне командной строки (см. %OUT в макросе):
 ERRORR Ошибка параметра перевода строки
 Error MAC1S.asm(33) PRINT(13) User error

При получении нормального представления текста на русском языке, нужно согласовать кодировки символов для печати и редактирования. Программа откомпилирована и отлажена в среде TASM. Компиляция, редактирование связей и отладка данной программы выполняется также, как и для других ЛР по Ассемблеру. Смотрите пособие и методические указания к ЛР.

12. Шаблон отчета по ЛР № 9



Шаблон отчета в виде документа с полями форматирования