Лабораторная работа 2. ОРГАНИЗАЦИЯ ВНЕШНИХ ПОДПРОГРАММ НА ЯЗЫКЕ АССЕМБЛЕРА

В некоторых случаях часть программы на языке высокого уровня удобнее записать непосредственно на языке ассемблера, например, для доступа к ресурсам операционной системы, ускорения работы критичного по времени выполнения участка программы и т.д.

В языке, например, Turbo Pascal для этой цели служат внешние подпрограммы, объявляемые в разделе описания процедур и функций следующим образом:

```
{$L filename}
procedure name(список_формальных_параметров);
external;
...
function name(список_формальных_параметров): тип;
external;
...
```

Заголовки описаний внешних подпрограмм составляются по обычным правилам, за исключением прибавления ключевого слова external. По заголовкам транслятор проверяет соответствие типов параметров и генерирует код для вызова подпрограмм. Перед объявлением внешней подпрограммы необходимо указать с помощью ключа {$L} в каком файле находится ее объектный код. В данном примере предполагается, что все используемые внешние подпрограммы размещаются в файле filename.obj, где расширение obj принимается по умолчанию. Можно записать несколько ключей, если подпрограммы размещаются в разных файлах. Причем не является обязательным построение объектного кода только транслятором с языка ассемблера. Через внешние подпрограммы Turbo Pascal допускает возможность связи с подпрограммами на любом языке программирования, если его транслятор создает объектный файл и поддерживает приводимый далее способ передачи параметров подпрограмм.

В процессе составлении внешней подпрограммы важно знать, как представляются ее параметры в памяти ЭВМ, и спо-

соб их передачи. Turbo Pascal имеет семь встроенных порядковых типов: byte, word, shortint, integer, longint, boolean, char и адресный тип pointer. Диапазон значений этих типов показан в табл. 1

Таблица 1

Характеристики типов данных

Тип	Диапазон	Размер	Примечание
byte	0..255	1 байт	байт без знака
word	0..65535	2 байта	слово без знака
shortint	-128..+127	1 байт	байт со знаком
integer	-32768..+32767	2 байта	слово со знаком
longint	-2147483648.. ..+2147483647	4 байта	двойное слово со знаком
boolean	false..true	1 байт	false=0, true=1
char	chr(0)..chr(255)	1 байт	совместим с byte
pointer	0000:0000.. ffff:000f	4 байта	совместим со ссылочными типами, nil = 0000:0000

Примечание: для типа longint младшие разряды размещаются в младших адресах памяти; для типа pointer в памяти размещается сначала смещение, а затем сегмент.

В ходе работы с символьными данными часто используется тип string, в формате которого нулевой байт является динамической (реальной) длиной строки, в последующих байтах находятся символы строки. Статическая длина зависит от объявления, но не больше 255 символов.

Переменные перечисленных выше типов могут быть параметрами внешних подпрограмм или результатами внешних функций. В принципе, Turbo Pascal не ограничивает использование переменных и любых других типов в качестве параметров, но не все типы разрешены для результатов функций.

В Turbo Pascal существует три вида параметров подпрограмм: параметр-значение, параметр-переменная с явным указанием типа, параметр-переменная, не имеющий типа.

Формальный параметр-значение считается локальной переменной подпрограммы с той разницей, что его инициализация выполняется вне подпрограммы перед ее вызовом. Изменение такого параметра внутри подпрограммы не оказывает влияния на соответствующий фактический параметр. Если фактический

параметр задан в виде выражения, то перед вызовом подпрограммы оно вычисляется. Параметр-переменная используется, когда требуется вернуть некоторое значение в вызывающую программу. Для этого подпрограмме передается адрес размещения фактического параметра, позволяющий его изменять. Тип формального параметра-переменной может отсутствовать в объявлении подпрограммы, тогда транслятор не будет проверять соответствие типа, а лишь сгенерирует адрес указанной переменной при обработке вызова. Пример такого объявления – procedure Clear (var Address). При ссылке на фактический параметр-переменную, являющийся элементом массива, полем записи и т.д., ее адрес вычисляется перед вызовом подпрограммы.

Вне зависимости от размера параметра-переменной подпрограмме всегда передается адрес, а для параметра-значения принято, если размер формального параметра 1, 2 или 4 байта, то передается значение, в противном случае – адрес копии параметра.

Фактические параметры всегда передаются подпрограмме через стек в порядке их перечисления в заголовке. Если размер параметра – один байт, то он расширяется до слова, старший байт которого неопределен. Для параметров длиной в двойное слово старшее слово помещается в стек перед младшим. Примеры заголовков подпрограмм и организация передачи их параметров приведены далее.

Тип вызова подпрограммы near или far задается ключем {$F}, требуемое значение которого указывается перед заголовком.

Пример 1. Заголовок подпрограммы:

```
{$F-}
procedure Sum(A: longint; var B: longint; C: boolean);
```

Формирование параметров:

```
mov   ax,HighA; старшее слово A
push  ax
mov   ax,LowA; младшее слово A
push  ax
mov   ax,seg B; сегмент B
push  ax
mov   ax,offset B; смещение B
```

```
push  ax
mov   al,C; значение C
push  ax
call  Sum
```

Стек после вызова (near-вызов):

Содержимое	ss:sp	Содержимое	ss:sp
IP выхода	+0	seg B	+6
c	+2	Low A	+8
offset B	+4	High A	+10

Пример 2. Заголовок подпрограммы:

```
{$F+}
function Del(S: string; C: char): string;
```

Формирование параметров:

```
; резервирование места для результата Res
mov   ax,seg Res; сегмент Res
push  ax
mov   ax,offset Res; смещение Res
push  ax
; создание копии S в строке S1
mov   ax,seg S1; сегмент копии
push  ax
mov   ax,offset S1; смещение копии
push  ax
mov   al,C; значение символа C
push  ax
call  far Del; far-вызов
```

Стек после вызова (far-вызов):

Содержимое	ss:sp	Содержимое	ss:sp
IP выхода	+0	seg S1	+8
CS выхода	+2	offset Res	+10
C	+4	seg Res	+12
offset S1	+6		

По окончании работы подпрограмма удаляет все переданные параметры из стека (см. одно исключение ниже) и возвращает управление основной подпрограмме.

Результат функции возвращается в регистрах процессора: байты – в регистре al, слова – в регистре ax, двойные – слова в паре регистров ax_dx (старшее слово – в dx, младшее – ax). Если результат функции указатель, то dx – сегмент, а ax – смещение. Для функций с результатом типа string резервируется место под результат, а его адрес передается в качестве первого параметра. Функция не должна изменять этот параметр и удалять его из стека после окончания работы.

Для доступа к параметрам внутри подпрограммы обычно используют регистр базы bp. Например, Turbo Pascal генерирует код подпрограммы по следующей схеме:

```
Name proc  near/far
push  bp          ; сохранение bp
mov   bp,sp       ; настройка bp на вершину стека
sub   sp,VarSize ; резервирование места в стеке
    ; для локальных переменных (если они есть),
    ; где VarSize = число слов
 ...; основной код подпрограммы
mov   sp,bp       ; восстановление  sp (удаление
                  ; локальных переменных)
pop   bp          ; восстановление bp
ret   ParSize     ; выход с удалением параметров
Name endp   ;  из стека, ParSize = число байтов
```

Тогда для near-подпрограммы параметры начинаются с [bp+4] (за счет присутствия bp в стеке), а far-подпрограммы [bp+6]. Напомним, что адресация через bp по умолчанию использует сегмент стека ss.

Требуемые подпрограмме локальные переменные можно размещать в стеке, сегменте данных или сегменте кода. Однако Turbo Pascal имеет некоторые ограничения на способ определения переменных в сегменте данных, поэтому рекомендуется этот сегмент не использовать.

Специальный вид подпрограмм представляют рекурсивные подпрограммы, т.е. подпрограммы, вызывающие сами себя. Раз-

личают два типа рекурсии: прямая, когда подпрограмма содержит вызов самой себя непосредственно в ее теле; косвенная, когда подпрограмма вызывается одной из вызываемых ею программ. Тип рекурсии практически не влияет на реализацию кода подпрограммы.

Не каждый язык программирования допускает написание рекурсивных программ: возможность рекурсивного вызова обеспечивают языки, использующие динамическое распределение памяти под локальные переменные. Традиционно локальные переменные размещаются в сегменте стека, поэтому каждый прямой рекурсивный вызов приводит к порождению одинаковой по размеру области стека (фрагмент стека), отличающийся от предыдущей значениями фактических параметров и адресом выхода:

ss:sp

адрес выхода	локальные переменные	фактические параметры

При косвенном вызове происходит перемешивание фрагментов стека подпрограмм, встречающихся в цепи вызова. Количество фрагментов стека, порождаемых в процессе рекурсивных вызовов, называют глубиной рекурсии, а максимально возможное количество фрагментов в доступной части сегмента стека – глубиной стека.

Для построения кода рекурсивной подпрограммы в Turbo Pascal используется приведенная выше схема. Учитывается лишь необходимость формирования в теле подпрограммы параметров ее рекурсивного вызова по обычным правилам. Проиллюстрируем сказанное реализацией кода рекурсивной функции вычисления факториала целого числа (ошибки переполнения не контролируются), специально дополненной ненужной в алгоритме локальной переменной X и операторами над ней:

```
function Fact(I: integer): integer;
      var  X: boolean;
      begin
         if I <= 1 then
            begin
            Fact := 1; X := true
            end
```

```
         else
            begin
            X := false; Fact := Fact(I-1)*I
            end
 end { Fact } ;

i equ word ptr [bp+4]; адрес параметра I в стеке
x equ byte ptr [bp-2]; адрес переменной X в стеке
fact   proc  near
       push  bp   ; сохранение bp
       mov   bp,sp; настройка bp на вершину стека
       sub   sp,2; выделение места в стеке для X
       cmp   i,1 ; if i<=1"  - i<=1?
       jle   exit1; then - да, выход с Fact=1
                  ; else - нет, еще вызов
       mov   x,0  ; X:=false;
       mov   ax,i ; ax:=значение i
       dec   ax   ; I-1
       push  ax     ; засылка параметра в стек
       call  Fact   ; Fact(I-1)
       mul   i  ; Fact(I-1)*I (dx:ax:=[bp+4]*ax)
       jmp   exiti; выход с ax=Fact=текущий
                        ; результат
exit1: mov   ax,1   ; Fact:=1
       mov   x,1    ; X:=true
exiti: mov   sp,bp ; восстановление sp (удаление X)
       pop   bp    ; восстановление bp
       ret   2   ; выход с удалением параметра I
fact   endp      ;  из стека
```

Проектирование рекурсивных подпрограмм должно сопровождаться оценкой глубины стека и правильным составлением условия окончания рекурсии. Ошибки при оценке этих параметров приводят к переполнению стека. Обычно подпрограмме выделяется только часть от общего сегмента стека, но, если предполагается большая глубина рекурсии либо подпрограмме требуется много места под локальные переменные, то необходимо позаботиться о переключении сегмента стека путем изменения регистра ss. Такие проблемы возникают и при рекурсивном программировании на языках высокого уровня, но доступ к управлению распределением памяти там обычно ограничен, и в каче-

стве промежуточного решения можно рекомендовать вынесение локальных переменных из подпрограммы и уменьшение количества ее параметров. Это несколько сокращает затраты стека, но не всегда решает проблему.

В ходе написания внешних подпрограмм для Turbo Pascal необходимо соблюдать соглашение о сохранении регистров bp, sp, ss и ds при выходе из подпрограммы. Несоблюдение соглашения приведет к нарушению работы операционной системы. Все остальные регистры можно изменять.

В общем случае внешняя ассемблерная подпрограмма включает:

– механизмы доступа к параметрам и возврата результатов;

– реализацию кода подпрограммы с учетом необходимости сохранения значений некоторых регистров;

– механизмы распределения памяти под локальные переменные и выхода из подпрограммы с удалением динамически распределяемых переменных и параметров;

– описание точки входа в подпрограмму с помощью псевдооператора public, что делает возможным обращение к подпрограмме из других модулей путем объявления ее имени в псевдооператоре extrn.

Правила оформления модуля с внешней подпрограммой не отличаются от применяемых при написании текста обычной программы с той разницей, что в псевдооператоре end не должна указываться метка точки входа.

Рассмотрим примеры реализации внешних подпрограмм.

Пример 1. Написать две внешние near-подпрограммы, первая из которых выполняет перевод прописной латинской буквы в строчную, вторая – строчной в прописную. Подпрограммы объединить в одном объектном модуле.

Заголовки подпрограмм:

```
{$F-}
procedure LoCase(var C: char); external;
function  UpCase(C: char): char; external.
```

Тексты подпрограмм:

```
code    segment byte public
        assume  cs:code
```

```
        public  LoCase, UpCase; определение общих имен
; procedure LoCase(var C: char),
; переводит прописную латинскую букву в строчную
LoCase  proc    near
varC    equ     dword ptr[bp+4]; адрес параметра C
        push    bp   ; сохранение bp
        mov     bp,sp; настройка bp на вершину стека
        les     bx,varC; es:[bx]:=адрес C
        mov     al,es:[bx]; al:=символ
        cmp     al,'a'    ; символ меньше "a"?
        jb      exit      ; да, пропустить
        cmp     al,'z'    ; символ больше "z"?
        ja      Exit      ; да, пропустить
        and     al,0dfh; перевод: прописная
                        ; в строчную
        mov     es:[bx],al ; символ:=al
Exit:   pop     bp         ; восстановление bp
        ret     4          ; выход с удалением
                 ; адреса C (4 байта) из стека
LoCase  endp

; function UpCase(C: char): char,
; переводит строчную латинскую букву в прописную,
; результат возвращается в регистре al
UpCase  proc    near
C       equ     word ptr[bp+4]; адрес параметра C
        push    bp            ; сохранение bp
        mov     bp, ; настройка bp на вершину стека
        mov     ax,C  ; ah:=?, al:=символ
        cmp     al,'A'; символ меньше "A"?
        jb      Exit1 ; да, пропустить
        cmp     al,'Z'; символ больше "Z"?
        ja      Exit1 ; да, пропустить
        or      al,' '; перевод: строчная
                       ; в прописную
Exit1:  pop     bp     ; восстановление bp
        ret     2      ; выход с удалением
                        ; параметра C
UpCase  endp
code    ends
        end
```

Пример 2. Написать внешнюю far-функцию, возвращающую строку, в которой все вхождения символа Ch1 заменены на символ Ch2.

Заголовок:

```
{$F+}
function Change(S: string; Ch1, Ch2: char): string;
external;
```

Текст подпрограммы:

```
code    segment byte public
        assume  cs:code,ds:code
        public  Change

; function Change(S: string; Ch1, Ch2: char):
;string, возвращает строку, в которой все вхождения
;Ch1 заменены на Ch2
Change  proc far
; адреса параметров в стеке:
Ch2     equ  byte ptr [bp+6]  ;  параметр Ch2
Ch1     equ  byte ptr [bp+8]  ;  параметр Ch1
S       equ  dword ptr [bp+10];  адрес строки S
Res     equ  dword ptr [bp+14];  адрес строки
                              ; результата
        push  bp      ; сохранение bp
        mov   bp,sp   ; настройка bp на вершину
                      ; стека
        push  ds      ; сохранение ds
        les   di,Res  ; es:di:=адрес результата
        lds   si,S    ; ds:si:=адрес исходной
                      ; строки
        cld; очистка флага направления (инкремент)
        lodsb; al:=(ds:[si]), si:=si+1 (al - длина S)
        stosb; (es:[di]):=al, di:=di+1 (запись длины)
        mov   ch,0; подготовка cx в качестве счетчика
        mov   cl,al; количества символов строки S
        jcxz  Exit; выход, если S - пустая строка (cx=0)
Repeat: lodsb  ; считать в al очередной символ S
        cmp   al,Ch1; символ равен Ch1?
        jne   Save; нет, сохранить без изменений
        mov   al,Ch2  ; да, заменить на Ch2
```

```
Save:   stosb; записать очередной символ результата Res
        loop  Repeat  ; повторять, пока есть
                      ; символы в S (cx>0)
Exit:   pop   ds      ; восстановить ds
        pop   bp      ; восстановить bp
        ret 8; выход с удалением параметров Ch1,
          ; Ch2 и адреса S (Res удалять нельзя!)
Change  endp
code    ends
        end
```

Отладку программ на языке Turbo Pascal с внешними подпрограммами на языке ассемблера лучше всего выполнять с помощью отладчика Turbo Debugger, откомпилировав программу с ключами {$L+} и {$D+}, подключающими к exe-файлу отладочную информацию. Отладчик позволяет отлаживать Turbo Pascal-программы как на уровне операторов языка, так и на уровне команд ассемблера, эффективно организуя диалог с пользователем. Другие отладчики не дадут возможности определить размещение кодов операторов, подпрограмм и т.д.

Текст Pascal-программы:

```
{$F+}
{$L Change.obj}
function Change(S: string; Ch1, Ch2: char): string;
external;
      var str1,str2:string;
          ch1,ch2:char;
      begin
            writeln('Введите строку');
            readln(str1);
            writeln('Введите символ, который нужно
заменить');
            readln(ch1);
writeln('Введите символ, которым нужно заменить');
            readln(ch2);
            str2:=Change(str1,ch1,ch2);
            writeln('Результат:  ',str2)
      end.
```

Оформленные показанным выше способом подпрограммы можно использовать при написании программ на языке ассемблера и создании библиотек объектных модулей. Например, следующая программа организует вызов внешней подпрограммы Change (см. пример 2) и выводит на экран результат ее работы, имитируя оператор языка Pascal write (Change(S,'1','2',)). Сегменты стека и данных не выделяются, так как данные размещаются в сегменте кода, а сегмент стека используется системный:

```
code    segment
        assume  cs:code,ds:code
        extrn   Change:far; описание внешней
                            ; far-процедуры
Example: mov     ax, cs; настройка сегмента данных
       mov     ds,ax  ; на сегмент кода
; подготовка параметров вызова Change(S,'1','2')
        push    ds; сегмент строки результата Res
        mov     ax, offset Res
        push    ax; смещение Res
        push    ds; сегмент исходной строки S
        mov     ax, offset S
        push    ax; смещение S
        mov     al, '1'; параметр Ch1 = '1'
        push    ax
        mov     al, '2'; параметр Ch2 = '2'
        push    ax
        call    Change ; far-вызов Change
; чтение адреса результата функции Change из стека
        pop     bx   ; bx:=смещение Res
        pop     ds   ; ds:=сегмент  Res
        mov     ch, 0; подготовка в cx длины
                     ; строки Res
        mov     cl, [bx]
        jcxz    Exit; выход, если строка Res пуста
Write:  inc     bx; продвижение указателя символа
        mov     dl, [bx]; dl:=очередной символ Res
        mov     ah, 2; вывод символа из dl
        int     21h  ; на экран средствами DOS
        loop    Write; цикл по длине строки Res
Exit:   mov     ax, 4c00h; завершение программы
```

```
        int     21h     ;  через функцию DOS

S       db      7, 'Ch1;Ch2', 248 dup(?)
Res     dw      128 dup(?)

code    ends
           end     Example
```

Построение исполняемого кода программы выполняется командами:

tasm change;

tasm example;

tlink example+change.

Задание к лабораторной работе

Составить внешнюю подпрограмму для варианта задания, соответствующего порядковому номеру студента в списке группы. Подпрограмму оформить в отдельном модуле. Для проведения отладки подпрограммы необходимо также составить основную программу на языке ассемблера, вызывающую составленную подпрограмму как внешнюю. Тип вызова выбирается самостоятельно.

1. function HexL(L: longint): string. Возвращает шестнадцатеричное символьное представление числа L.

2. function BinaryL(L: longint): string. Возвращает двоичное символьное представление числа L.

3. function OctalL(L: longint): string. Возвращает восьмеричное символьное представление числа L.

4. function Long2Str(L: longint): string. Возвращает десятичное символьное представление числа L.

5. function Str2Long(S: string; var L: longint): boolean. Переводит символьное представление числа S в длинное целое L. Возвращает true, если формат числа в S правильный (например, не содержится недопустимых символов), в противном случае – false.

6. function StrUpCase(S: string): string. Возвращает строку, в которой все строчные русские и латинские буквы заменены на прописные. Для перевода русских букв используются их поряд-

ковые номера: буквам "А".."Я", "а".."п", "р".."я" соответствуют десятичные номера 128..159, 160..175, 224..239.

7. function StrLoCase(S:string): string. Возвращает строку, в которой все прописные русские и латинские буквы заменены на строчные. Перевод аналогичен используемому в задании 6.

8. function DelBlanks(var S: string; Len: byte): boolean. Равномерно удаляет пробелы между словами в строке S до получения длины строки Len. Для этого S циклически просматривается и после каждого слова в ней удаляется 1 пробел (нельзя удалять последний пробел между словами). Если больше нет возможности удалять, а строка все еще длиннее, чем Len, то функция завершается с результатом false. Аналогично при длине S меньше или равной Len. При неуспешном завершении результат функции равен true.

9. function PadCh(S: string; C: char; Len: byte): string. Возвращает строку, в которой S смещена влево, а остаток строки заполнен символами C. Для этого знаки C включаются справа от конца S до тех пор, пока общая длина строки не станет равной Len. Если S длиннее, чем Len, то строка не изменяется. Если S пустая строка, то возвращается строка из Len символов C.

10. function LeftPadCh(S: string; C: char; Len: byte):string. Возвращает строку, в которой S смещена вправо, а начало строки заполнено символами C. Знаки C включаются слева от начала строки, пока общая длина строки не станет равной Len. Остальное аналогично заданию 9.

11. function TrimLead(S: string): string. Возвращает строку, в начале которой удалены все цифры 0 и символы с кодом меньше пробела.

12. function TrimTrail(S: string): string. Возвращает строку, в конце которой удалены все цифры 0 и символы с кодом меньше пробела.

13. function DeleteCh(S: string; C: char): string. Возвращает строку, в которой удалены все вхождения символа C.

14. function CenterCh(S: string; C: char; Len: byte):string. Возвращает строку длиной Len, в которой содержимое S сцентрировано с помощью символов C. Для этого слева и справа от S равномерно добавляются символы C до получения длины Len. Остальное аналогично заданию 9.

15. function CompString(S1, S2: string): byte. Возвращает 0, если S1=S2, 1 – если S1>S2, 2 – если S1<S2. Сравнение строк производится следующим образом: если длины строк неодинаковы, то результатом будет сравнение длин; если длины одинаковы, то строки сравниваются посимвольно, а результатом будет либо значение первого несравнения символов, либо S1=S2, когда достигнут конец строки. Заметим, что результат сравнения не всегда совпадает с принятым для строк в Turbo Pascal.

16. function Pos(SubS, S: string): byte. Возвращает позицию первого вхождения подстроки SubS в строку S или 0, если вхождений нет.

17. procedure Delete(var S: string; Start, Len: byte). Удаляет в строке S символы с позиции Start и длинной Len. Если Start, больше длины S, то ничего не изменяется.

18. procedure Insert(SubS: string; var S:string; Start:byte). Вставляет подстроку SubS в строку S, начиная с позиции Start. Если Start больше длины S, то ничего не изменяется.

19. function Copy(S: string; Start, Len: byte): string. Возвращает подстроку длиной Len, выделенную из строки S, начиная с позиции Start. Если Start больше длины S, то возвращается пустая строка.

20. function InsBlanks(S: string; Len: byte): string. Возвращает строку длиной Len, в которой равномерно добавлены пробелы. Для этого строка циклически просматривается и после каждого слова в ней добавляется по одному пробелу до тех пор, пока не будет равенства длины строки и Len. Если длина S сразу больше или равна Len либо S – пустая строка, то ничего не изменяется.

Примечание: пустой называется строка, не содержащая ни одного символа, байт длины которой равен нулю.

Порядок выполнения работы

1. Изучить основные сведения по работе.

2. В соответствии с заданием составить внешнюю подпрограмму на ассемблере и основные программы на ассемблере и языке высокого уровня. В основной программе производится ввод данных, вызов подпрограммы и вывод результата. Вывод несимвольных результатов подпрограммы организуется в про-

извольной, но понятной форме. Подготовить 2-3 варианта исходных данных для отладки программы.

2. Выполнить ввод, трансляцию, построение кода программы и получить результаты ее работы для подготовленных вариантов исходных данных. При необходимости воспользоваться отладчиком.

Содержание отчета

1. Цель работы.

2. Текст задания, схема реализации, блок-схема или программа на языке высокого уровня.

3. Содержимое стека после вызова подпрограммы для объяснения адресации параметров.

4. Тексты основной программы и подпрограммы.

4. Результаты работы программы.

5. Выводы по работе.

Контрольные вопросы

1. Каковы достоинства и недостатки способа передачи параметров подпрограммы через стек?

2. Приведите варианты реализаций способов доступа к параметрам подпрограммы без применения регистра bp.

3. Покажите, как можно выполнить выход из подпрограммы с удалением ее параметров из стека, не используя параметры в команде ret.

4. Поясните разницу между near- и far-типами команд call, ret и jmp.

5. Покажите на примерах способы передачи параметров разных типов.

6. Напишите код инициализации параметров подпрограммы с заголовком function Dummy(var I: integer; B: boolean; P: pointer): string, для вызова Dummy(X, true, nil).

7. Напишите текст внешней подпрограммы, заполняющей произвольную переменную V количеством Count нулевых байтов. Заголовок подпрограммы: procedure Fill(var V; Count: word).

8. В чем отличия реализации рекурсивных и нерекурсивных подпрограмм? Приведите схемы распределения памяти для вызовов таких подпрограмм.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Андреева А.А. Основы программирования персонального компьютера на языке ассемблера: лабораторный практикум. Чебоксары. Изд-во Чуваш. ун-та, 2013. 84 с.
2. Брэй Б. Микропроцессоры Intel: 8086/8088…80486, Pentium: пер. с англ. СПб: BHV-Петербург, 2005. 1328 с
3. Зубков С.В. Ассемблер для DOS, Windows и UNIX. М.: ДМК-Пресс, 2013. 638 с.
4. Юров В.И. Assembler: учебник для вузов. СПб.: Питер, 2011. 640 с.
5. Юров В.И. Assembler: практикум: учеб. Пособие для вузов. СПб.: Питер, 2007. 400 с.