Московский государственный технический университет им. Н.Э.Баумана

Г. С. Иванова

ПРОГРАММИРОВАНИЕ НА АССЕМБЛЕРЕ ПЭВМ

**Методические указания к лабораторным работам
по курсу “Системное программирование”**

Часть 1

Машинные команды ассемблера
Приемы программирования

МОСКВА 1999

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ 2

1. АННОТАЦИЯ 3

2. ОБРАБОТЧИКИ ПРЕРЫВАНИЙ 4

2.1. Введение 4
2.2. Основные сведения о системе прерываний IBM совместимых ЭВМ 4
2.3. Классификация прерываний 6
2.4. Структура обработчиков прерываний. Модификация области векторов прерываний 7
2.5. Пользовательские обработчики прерываний 8
2.6. Резидентные программы 10
2.7. Примеры резидентных обработчиков прерываний на ассемблере 11
2.8. Примеры обработчиков прерываний, написанных на Турбо Паскале и Турбо Си 14
2.8.1. Средства разработки обработчиков прерываний на Турбо Паскале 14
2.8.2. Средства разработки обработчиков прерываний на Турбо Си 15
2.9. Предотвращение повторной загрузки резидентного обработчика прерываний в память. Передача параметров резидентам 18
2.10. Удаление резидентного обработчика прерывания из памяти 21
2.11. Предотвращение зависания при использовании функций DOS 24

3. СВЯЗЬ РАЗНОЯЗЫКОВЫХ МОДУЛЕЙ 26

3.1. Основные положения 26
3.2. Связь модулей, написанных на Турбо Паскале и Ассемблере 26
3.2.1. Передача управления между модулями. Размещение области параметров в стеке 26
3.2.2. Некоторые внутренние форматы данных Турбо Паскаля 27
3.2.3. Передача параметров по значению и ссылке. Возврат результатов процедур и функций 28
3.2.4. Особенности оформления и компоновки программного продукта, состоящего из модулей на Турбо Паскале и ассемблере 28
3.2.5. Примеры программ 29
3.3. Связь модулей, написанных на Турбо C++(Borland C++) и Ассемблере 34
3.3.1. Особенности передачи управления между модулями, написанными на Си 34
3.3.2. Передача параметров. Возврат результатов 36
3.3.3. Особенности компоновки программного продукта, состоящего из модулей на Турбо C++ и ассемблере 37
3.3.4. Определение глобальных и внешних имен 38
3.3.5. Примеры программ 39

4. ЛИТЕРАТУРА 43

1. Аннотация

Настоящие методические указания предназначены для студентов, изучающих курсы “Системное программирование” и “Вычислительная техника и информационная технология”, и содержат сведения, необходимые для написания резидентных обработчиков прерываний и программ, содержащих модули, написанных на Турбо Паскале, Турбо C++ и ассемблере.

В первой части определены базовые понятия и основные принципы функционирования системы прерываний ПЭВМ, приведены структуры инсталлирующих программ и обработчиков для различных случаев. Рассмотрены наиболее важные проблемы, возникающие при написании обработчиков прерываний: предотвращение повторной загрузки, передача параметров резидентам, выгрузка из памяти и проблемы, связанные с использованием функций DOS. Все рассмотренные случаи иллюстрируются примерами. Дополнительно описаны средства Турбо Паскаля и Турбо C++, позволяющие писать обработчики прерываний на этих языках, и приведены соответствующие примеры.

Во второй части приводятся сведения, необходимые для создания программ, включающих модули, написанные на Турбо Паскале, Турбо C++ и ассемблере: описаны способы передачи параметров и управления, используемые Турбо Паскалем и Турбо C++, внутренние представления данных и структуры сегментов, создаваемые компиляторами. Все рассмотренные случаи иллюстрируются примерами.

Все приведенные примеры отлажены на ЭВМ i486 с использованием Turbo Pascal 7.0 и Borland C++ 3.1.

2. Обработчики прерываний

2.1. Введение

Для IBM совместимых ЭВМ понятие прерывания является одним из базовых. Через систему прерываний в них реализованы доступ к внешним устройствам, взаимодействие со схемами контроля ЭВМ, управление процессом выполнения программы со стороны операционной системы, выполнение сервисных функций DOS и многое другое. Операционная система MS DOS позволяет пользователю разрабатывать собственные программы, которые могут вызываться через прерывания, дополняя и даже заменяя стандартные программы MS DOS.

2.2. Основные сведения о системе прерываний IBM совместимых ЭВМ

Прерыванием называется временное переключение процессора на выполнение некоторой заранее определенной последовательности команд, после завершения, которой процесс выполнения программы возобновляется.

Прерывания в IBM совместимых ЭВМ (см. Рисунок 1.1) могут инициироваться как специальными сигналами микропроцессора (внутренние), так и внешними сигналами, например, от внешних устройств (внешние).

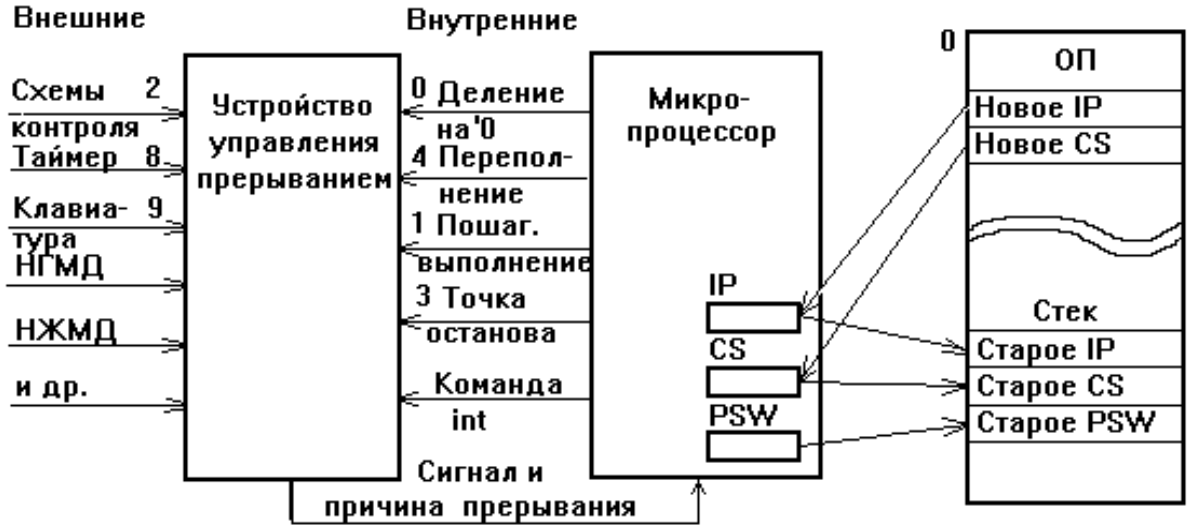

Рисунок 2.A. Система обработки прерываний.

Внешние сигналы на прерывание поступают на специальные микросхемы - контроллеры прерываний 8259A, имеющие 8 уровней приоритета. Начиная с PC AT, микроЭВМ включает два контроллера, что позволяет увеличить количество уровней приоритета до 16. Уровни приоритета определяются аппаратно в зависимости от места подключения внешнего устройства.

Как правило, самый высокий приоритет имеет системный таймер, затем следует клавиатура, что обеспечивает оперативное управление микроЭВМ, далее идут прочие внешние устройства. Завершают список обычно накопители на гибких магнитных дисках и устройство печати.

Прерывания, поступающие через контроллеры прерываний, также называют *аппаратными* в отличие от остальных, носящих название *программных.*

Выполнение всех прерываний, кроме прерывания 2, можно запретить, установив в 0 флаг IF флажкового регистра. Прерывание 2 в связи с этим носит название *немаскируемого* и используется для того, чтобы выполнять обработку сигналов, наличие которых нельзя игнорировать, например, сигнала о недопустимом изменении напряжения в сети питания.

Адреса программ-обработчиков прерываний хранятся в специальной области основной памяти, которая обычно располагается с 0-го адреса и занимает 1кБ, то есть может содержать адреса 256 программ, (каждый адрес занимает 4 байта). Местоположение адреса нужного обработчика в области векторов прерываний определяется по типу (номеру) прерывания:

A=4*N, где N - номер прерывания.

При наличии сигнала прерывания выполняется следующая последовательность действий:

1) проверяется установка флажка IF (для немаскируемых прерываний этот пункт игнорируется): 1- прерывания разрешены, 0 - прерывания запрещены;

2) если прерывание разрешено, то после завершения выполнения текущей команды слово состояния программы(PSW), хранящееся во флажковом регистре микропроцессора, значение сегментного регистра кодов(CS) и значение счетчика команд(IP) заносятся в стек;

3) из области векторов прерываний в регистры CS и IP помещается адрес программы обработки прерываний.

После чего начинается выполнение программы обработки прерывания.

По завершению этой программы значения PSW, CS и IP восстанавливаются из стека, и выполнение прерванной программы возобновляется.

2.3. Классификация прерываний

Использование большинства прерываний определяется конкретным программным обеспечением и соответственно для каждой ПЭВМ будет своим. Однако существуют номера прерываний, закрепленные за соответствующими функциями.

I. ***Прерывания микропроцессора(0H-6H)*****:**

0 - прерывание от схем контроля микропроцессора - “Деление на 0”;

1 - прерывание специального режима работы микропроцессора, устанавливаемого, если флажок TF=1 - “Пошаговое выполнение”;

2 - немаскируемое прерывание;

3 - прерывание микропроцессора, осуществляемого при обнаружении адреса останова - “Точка останова”;

4 - инициируется по команде INTO, используемой после выполнения арифметической операции - “Переполнение”;

5 - печать содержимого экрана - инициируется нажатием клавиши Print Screen.

II. ***Прерывания микроконтроллера прерываний(7H-0FH, 70H-77H)*****:**

8 - прерывание от таймера;

9 - прерывание от клавиатуры;

0BH - COM2;

0CH - COM1;

0EH - прерывание от НГМД (дискеты);

0FH - прерывание от печатающего устройства;

70H - прерывание от часов реального времени;

76H - прерывание от НЖМД (жесткий диск);

III. ***Процедуры BIOS (10H-1AH, 33H, 41H):***

10H - управление дисплеем;

11H - определение конфигурации ПЭВМ;

12H - определение объема памяти ПЭВМ;

13H - управление дисковой памятью;

14H - управление асинхронной связью;

16H - управление клавиатурой;

17H - управление печатающим устройством;

1AH - управление часами реального времени.

IV. ***Процедуры пользователя (1BH и 1CH):***

1BH - возможность подключения при обнаружении Ctrl-Break;

1CH - возможность подключения к обработке кванта таймера.

V. ***Указатели системных таблиц (1DH-1FH, 41H):***

1DH - таблица параметров видео;

1EH - таблица параметров дискеты;

1FH - таблица символов для графического режима;

41H - таблица параметров жесткого диска.

VI. ***Прерывания DOS (20H- 3FH):***

20H - нормальное завершение программы и возврат управления DOS;

21H - вызов диспетчера функций DOS;

22H - адрес пользовательской программы обработки нормального завершения программы;

23H - адрес пользовательской программы обработки завершения по Ctrl-Break;

24H - адрес пользовательской программы обработки завершения по ошибке;

25H - абсолютное чтение секторов с диска;

26H - абсолютная запись секторов на диск;

27H - завершение программы с сохранением в памяти.

VII. Прерывания, зарезервированные для пользователей (60H-66H, 0F0H-0FEH).

Для определения конкретной конфигурации прерываний можно использовать, например, программу SYSINFO пакета Norton Utilities.

2.4. Структура обработчиков прерываний. Модификация области векторов прерываний

Программа обработки прерывания в общем случае имеет следующую структуру:

```
<имя>          proc    far
               <сохранение содержимого регистров>
               <обработка>
               <восстановление содержимого регистров>
                       iret    ; возврат управления с восстановлением PSW,CS,IP
<имя>          endp
```

Для подключения обработчика прерываний необходимо поместить его адрес в область векторов прерываний, а после завершения использования - восстановить старое содержимое вектора. Фрагменты программы, выполняющие эти операции приведены ниже.

```
                title   inter
code            segment
                assume  cs:code
old_adress      dw      2 dup (?)   ;   область для сохранения старого вектора
;
main            proc
                .....
                push    ES
                mov     AH,35H      ;   вызов функции DOS для получения
                mov     AL,0FEH     ;   старого значения вектора
                int     21H         ;   прерывания
                mov     old_adress,ES   ;   сохранение старого
                mov     old_adress+2,BX ;   значения вектора
                push    DS
                mov     DX,offset user  ;   вызов функции DOS
                mov     AX,seg user     ;   для записи адреса
                mov     DS,AX           ;   пользовательской
                mov     AH,25H          ;   программы обработки
                mov     AL,0FEH         ;   прерывания в область
                int     21H             ;   векторов прерываний
                pop     DS
                int     0FEH    ;   вызов прерывания
                push    DS
                mov     DX,old_adress+2 ;   вызов функции DOS
                mov     AX,old_adress   ;   для восстановления
                mov     DS,AX           ;   старого адреса
                mov     AH,25H          ;   в области векторов
                mov     AL,0FEH         ;   прерываний
                int     21H             ;
                pop     DS
                pop     ES
                ........
main            endp
user            proc    far
                .......
                iret
user            endp
code            ends
                end     main
```

В данном примере для получения и записи значений векторов прерывания использованы специальные функции прерывания 21H DOS (25H и 35H). Прямая запись и прямое чтение области векторов прерываний не желательны, так как прямое обращение по данным адресам в некоторых операционных системах может привести к зависанию системы.

2.5. Пользовательские обработчики прерываний

В простейшем варианте пользовательские обработчики прерываний могут использоваться для вызова процедур, хотя такое их использование вряд ли целесообразно. Гораздо чаще пользовательские обработчики прерываний используются для замены или дополнения стандартных программ обработки прерываний, как программных, так и аппаратных. При этом возможны различные варианты подключения пользовательских программ.

I. Используется только пользовательская программа обработки прерывания.

В этом случае обработчик строится по схеме, приведенной в разделе 2.4, и завершается командой IRET. При программировании **аппаратных** прерываний перед IRET необходимо записать две команды, при выполнении которых сбрасываются флаги запрещения выполнения аппаратных прерываний более низкого уровня приоритета (если пишется обработчик прерывания уровней 8..15, то добавляется третья команда, выполняющая те же действия для второго микроконтроллера): **mov AL,20H**

```
out     20H,AL
out     A0H,AL
```

II. Пользовательская программа выполняет предобработку прерывания и передает управление уже имеющейся программе обработки (см. рисунок 2.b).

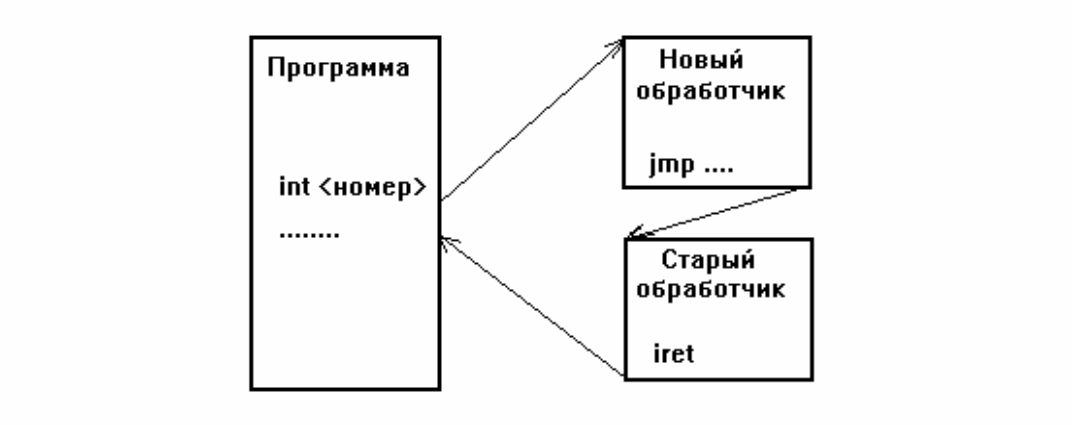

Рисунок 2.B. Вызов старого обработчика прерываний после нового.

В этом случае пользователь, записывая адрес своей программы в область векторов прерываний, должен сохранить старое значение адреса в своей программе и после выполнения необходимых действий передать по нему управление:

```
jmp     far CS:old_adress
```

Следовательно, команда IRET в программе пользователя в этом случае отсутствует.

III. Пользовательская программа вызывает старый обработчик прерываний из себя (см.рисунок 2.c).

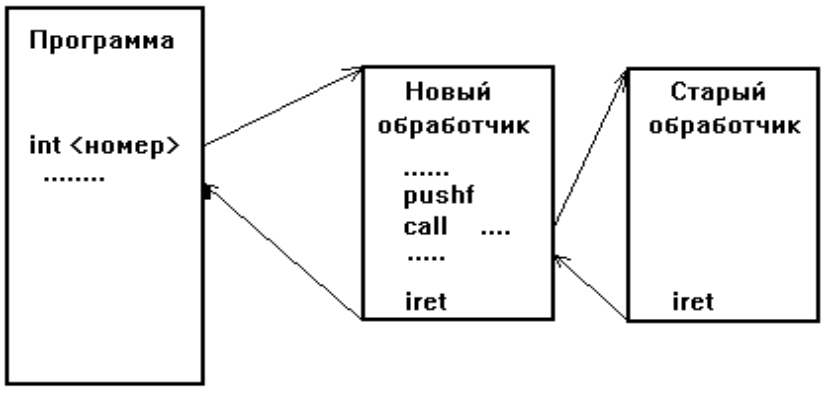

Рисунок 2.C. Вызов старого обработчика прерываний из нового.

В этом случае необходимо согласовать возврат управления из старого обработчика прерываний по IRET и в завершении выдать свой собственный IRET:

```
.....
pushf
call    far CS:old_adress
```

```
.......
iret
```

Согласование возврата по IRET выполняется за счет помещения в стек содержимого флажкового регистра перед вызовом старого обработчика прерывания. Таким образом, в стеке оказываются значения PSW, CS и адрес следующей команды. При выполнении IRET старого обработчика эти значения восстанавливаются во флажковый регистр, CS и IP, и продолжается выполнение пользовательской программы обработки прерывания.

Примечание. При желании можно использовать более сложный способ подключения пользовательской обработки, не требующий изменения соответствующего вектора. В этом случае любое место старого обработчика прерываний копируется в специальную область нового обработчика, и на это место пишется вызов нового обработчика. Получив управление, новый обработчик выполняет необходимые операции, затем выполняет скопированные команды и возвращает управление старому обработчику. Данный метод получил название “врезка" и в основном используется в вирусах и антивирусах.

При написании обработчиков прерываний следует иметь в виду, что при прерывании автоматически выдается команда CLI, устанавливающая в 0 флаг IP флажкового регистра, т.е. запрещающая маскируемые прерывания. Поэтому, если обработчик не меняет область векторов прерываний и не содержит фрагментов, выполнение которых жестко рассчитано по времени, получив управление желательно выдать команду STI, разрешив, таким образом, остальные прерывания.

2.6. Резидентные программы

В большинстве случаев пользовательские обработчики прерываний необходимо резидентно сохранять в памяти в течение всего сеанса времени работы ПЭВМ. Общая схема работы в этом случае выглядит следующим образом. Сначала запускается специальная программа - *инсталлятор*, которая содержит программу-обработчика и специальную часть, выполняющую корректировку нужного вектора, и завершающуюся без удаления из памяти. Затем работают программы, использующие данный обработчик. Для удаления пользовательского обработчика прерывания из памяти обычно используется перезагрузка.

Ниже приводятся структуры инсталляторов COM и EXE-файлов.

Для COM-файлов:

```
                org         100H
begin:          jmp         short  set_up  ;   передача управления инсталлятору
; обработчик
user            proc        far
            <обработка >
                iret
finish          equ         $
user            endp
; инсталлятор
set_up          mov         DX,offset  user              ;    вызов функции DOS
                mov         AL,<номер прерывания>  ;    для записи нового
                mov         AH,25H                       ;    вектора
                int         21H                          ;
                .......
                lea         DX,finish          ;  выход с сохранением процедуры
                int          27H               ;  обработчика в памяти
```

Для COM-файлов инсталляция выполняется достаточно просто, так как префикс программного сегмента (PSP) и сама программа находятся в одном сегменте (см. Рисунок 1.4 а).

Для EXE-файлов:

```
 ; резидент
user            proc        far
            <обработка>
                iret
finish          equ         $
user            endp
; инсталлятор
set_up          proc        near
                push        DS
                mov         AX,0
                push        AX
                mov         DX,offset  user
                mov         AX,seg  user
                mov         DS,AX
                mov         AL,<номер прерывания>
                mov         AH,25H
                int         21H
                mov         DX,finish+100H
                mov         byte ptr ES:1,27H
                ret
```

При инсталляции EXE-файлов, определяя размер резидентной части, приходится учитывать размер области PSP (100H) (см. Рисунок 1.4 б), так как адрес finish считается относительно CS и не учитывает PSP.

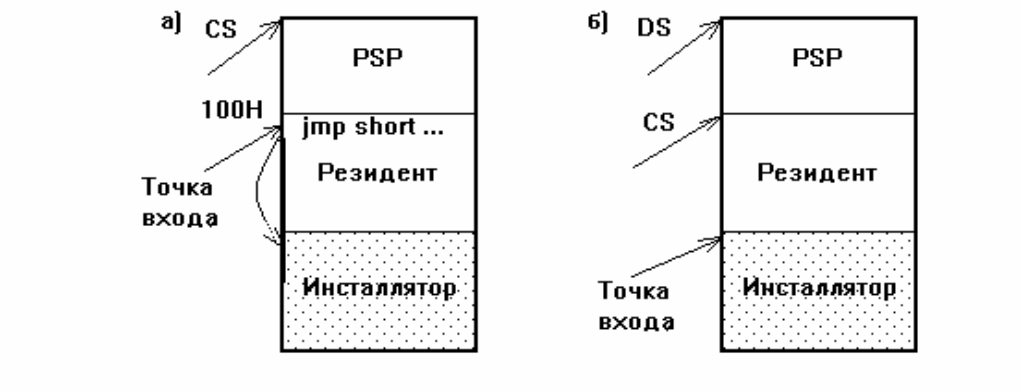

Рисунок 2.D. Структура программ инсталляции а) COM-файла, б) EXE-файла.

2.7. Примеры резидентных обработчиков прерываний на ассемблере

Пример 1. Резидентный обработчик прерывания **int 1CH**. Через каждые 10 обращений к таймеру выводит напрямую в видеопамять символ ASCII. Резидент загружается из COM-файла.

```
code        segment     byte public
            org         100h
            assume      cs:code,ds:code
begin:      jmp         start
tik         db          0       ; счетчик "тиков"
nch         db          0       ; номер символа из таблицы ASCII
;  резидентный обработчик
process:    cli
            inc         tik     ;  увеличение счетчика тиков на 1
            cmp         tik,10  ; есть 10 тиков ?
            jl          a1   ; если нет, то переход на завершение обработки
            push        ES
            push        AX
            mov         AX,0b800h   ;    запись адреса видеобуфера
            mov         ES,AX       ;    в сегментный регистр ES
            mov         tik,0   ;  обнуление счетчика тиков
            inc         nch     ;   переход к следующему символу таблицы
            mov         AL,nch
            mov         ES:[0],AL    ;    вывод символа
            mov         ES:[1],1eH   ;    вывод атрибута
            pop         AX
            pop         ES
a1:         sti
            iret
; инсталлятор
start:      mov         AX,251cH      ;    запись адреса обработчика в
            lea         DX,process    ;    область векторов
            int         21H           ;     прерываний
            lea         DX,start      ;    выход с сохранением программы
            int         27H           ;    обработки в памяти
code        ends
            end
```

Пример 2. Резидентный перехватчик прерывания **int 9H**. Обнаруживает одновременное нажатие клавиш <Ctrl+Alt+Home> и устанавливает режим "озвучания" нажатия клавиш. При повторном нажатии той же комбинации сбрасывает режим "озвучания". После выполнения всех операций возвращает управление старому обработчику прерывания. Резидент загружается из COM-файла. При загрузке адреса в область векторов прерываний используется "прямая" запись без использования функции DOS. При передаче управления старо-

му обработчику используется машинный код команды **jmp** и следующее за ним поле, где был размещен адрес старого обработчика прерывания.

```
                title           prer
code            segment
                assume          CS:code
                org             100h
start:          jmp             init
;   резидентный обработчик
tsr_9:          push            AX
                in              AL,60H  ;       читаем scan-код  из порта 60H
                push            BX
                push            CX
                push            DX
                push            DS
                cmp             AL,47H   ;  код клавиши  <Home>?
                jne             check       ; по “нет” переход на обработку
                xor             BX,BX               ;      загрузка байта
                mov             DS,BX               ;       состояния
                mov             AL,DS:[0417H]   ;       клавиатуры
                mov             AH,AL     ;  сохранение байта состояния
                and             AL,0cH; проверка на нажатие  <Ctrl+Alt>
                cmp             AL,0cH
                jne             check       ; по “нет” переход на обработку
                xor             byte ptr CS:flag,1  ; установка/сброс  режима “озвучания”
check:          cmp             byte ptr CS:flag,0  ; режим “озвучания” включен?
                je              exit   ; по “нет” переход на конец
                mov             DX,800   ;
                in              AL,61H   ;
                and             AL,0feH   ;       п р о г р а м м а
n_cycl:         or              AL,2        ;
                out             61H,AL    ;
                mov             CX,150     ;       о з в у ч а н и я
cycl_u:         loop            cycl_u       ;
                and             AL,0fdH   ;
                out             61H,AL    ;       н а ж а т и я
                mov             CX,150     ;
cycl_d:         loop            cycl_d       ;
                dec             DX          ;       к л а в и ш и
                jnz             n_cycl       ;
exit:           pop             DS
                pop             DX
                pop             CX
                pop             BX
                pop             AX
                db      0eaH    ; Передаем управление родному обработчику
original        dw              ?    ;  адрес старого обработчика
                dw              ?    ;   9-го прерывания
flag            db              0       ;  флаг  режима "озвучания”
mes             db              'tsr:',0ah,0dh,24h  ; сообщение о загрузке резидента
;   инсталлятор
init:           push            ES              ;  получение
                mov             AX,3509H    ;  старого значения
                int             21H             ;   вектора 9-го прерывания
                mov             original,BX         ;   сохранение этого вектора
                mov             original+2,ES      ;    в “теле” обработчика
                pop             ES
                xor             AX,AX               ;   “прямая” запись
```

```
                mov     DS,AX           ;  нового адреса в
                lea     AX,tsr_9        ;  область
                cli                     ;  векторов
                mov     DS:[0024H],AX   ;  прерываний
                mov     DS:[0026H],CS   ;
                sti                     ;
                push    CS              ;  выдача сообщения
                pop     DS              ;        о
                lea     DX,mes          ;    загрузке
                mov     AH,9            ;   в память
                int     21H             ;   резидента
                lea     DX,init         ;  завершение и выход с сохранением
                int     27H             ;  обработчика прерываний в памяти
code            ends
                end
```

2.8. Примеры обработчиков прерываний, написанных на Турбо Паскале и Турбо Си

Несмотря на то, что наиболее эффективные обработчики прерываний пишутся на языке Ассемблера, многие языки высокого уровня включают средства для написания таких программ.

2.8.1. Средства разработки обработчиков прерываний на Турбо Паскале

В Турбо Паскале 5.0 и выше такими средствами являются специальные процедуры и функции и описания типов, хранящиеся в библиотеке DOS. Так для организации доступа к регистрам микропроцессора Турбо-Паскаль предлагает использовать специальный тип, определенный в модуле DOS:

```
Type Registers=record
      case integer of
        0: (AX,BX,CX,DX,BP,SI,DI,DS,ES,FLAGS:word);
        1: (AL,AH,BL,BH,CL,CH,DL,DH:byte)
                  end;
```

В этом же модуле находятся тексты специальных процедур:

GetIntVec(<номер прерывания>:byte,< адрес процедуры>:pointer) - процедура, которая позволяет получить адрес обработчика, указанного прерывания.

SetIntVec(<номер прерывания>:byte,<адрес процедуры>:pointer) - процедура, которая позволяет установить новый адрес обработчика указанного прерывания.

Keep (<код возврата>:word) - процедура, которая завершает выполнение программы, сохраняя ее копию в памяти. Для уменьшения резидентной части можно использовать директиву **$M** компилятора. Первый параметр директивы определяет размер стека (не менее 1024 байт), а второй и третий соответственно минимальный и максимальный размеры динамической области (если в программе не используются динамически распределяемые переменные, то можно задавать нули).

Intr(<номер прерывания>:byte,<регистры>:Registers) - процедура вызова прерывания по номеру.

Кроме этого, процедура-обработчик должна оформляться специальным образом: она должна иметь следующий заголовок

Procedure <имя>(Flags,CS,IP,AX,BX,CX,DX,SI,DI,ES,DS,BP:word);interrupt; и компилироваться в режиме формирования дальних адресов.

Для организации прямого доступа к памяти по указанному адресу могут использоваться специальные массивы **Mem, MemW** и **MemL**, которые работают соответственно с байтами, словами и двойными словами. В качестве индексов этих массивов используются абсолютные адреса, состоящие из сегмента и смещения, например: **Mem[$0000:$1000]**.

В случае крайней необходимости можно вставить в текст шестнадцатиричный код машинной команды, используя команду **Inline**, а, начиная с версии 6.0, можно напрямую вписывать в текст команды Ассемблера, используя служебное слово **asm.**

Пример 1. Программа, перехватывающая прерывание **int 9H**. Перехватив прерывание, программа передает управление старому обработчику, затем организует необходимые проверки и выполняет звуковое сопровождение нажатия клавиш <**Ctrl+Alt**>. Нажатие клавиш <**R-Shift+Ctrl**> восстанавливает старое значение вектора прерывания.

```
{$M 1024,0,0}      (* Управление резервированием памяти *)
Program Resident_Example;
Uses CRT,DOS;
Var  Int9:procedure;        { адрес старого обработчика прерываний}
     Logr:boolean;          { признак  прерывания прерывания}
{$F+}
   Procedure PressButton(Flags,CS,IP,AX,BX,CX,DX,SI,DI,ES,DS,BP:word);interrupt;
      Var   B:byte;  { байт состояния клавиатуры}
            I:word;  { счетчик}
      Begin
          Inline($9C);     { PUSHF}
          Int9;                { Вызов системной обработки}
          B:=Mem[$0000:$0417];
          If ((B and 12)=12) and (Logr) then    {CTRL-ALT}
              begin
```

```
                    Logr:=False;
                    I:=500;
                    While  I<= 1000 do
                          begin
                             Sound(I);
                             Delay(1);
                             Inc(I,20);
                          end;
                    NoSound;
                    Logr:=true;
                end;
          If (B and 5)=5 then          {R-SHIFT--CTRL}
          SetIntVec($9,Addr(Int9));
 end;
 {$F-}
 Begin
     GetIntVec($9,@Int9);
     SetIntVec($9,Addr(PressButton));
     Logr:=true;
     SwapVectors;
     Keep(0);
end.
```

2.8.2. Средства разработки обработчиков прерываний на Турбо Си

Турбо Си также содержит все необходимые средства для разработки обработчиков прерываний. При этом описания всех необходимых функций, констант и типов переменных находятся в <dos.h>.

Для доступа к памяти по заданному адресу может использоваться функция **MK_FP,** которая формирует дальний адрес из сегмента и смещения, а для доступа к портам - функции **inp** и **outp**.

Доступ к регистрам осуществляется через псевдорегистры: **_AX,_AL,_AH,_BX** и т.д.

Для того чтобы сохранить программу в памяти после ее завершения используется функция **void keep(unsigned retcode, unsigned memsize);** где первый параметр - код завершения, а второй - размер резидентной части программы в параграфах(1 параграф=16 байт). Для определения необходимого объема памяти необходимо учитывать, что Турбо Си (модель памяти small) строит исполняемую программу по следующей схеме:

PSP - префикс программного сегмента (256 байт);
CODE - сегмент кода программы;
DATA - сегмент данных программы;
HEAP - область размещения динамических данных (устанавливается максимальной);
STACK - область стека (по умолчанию 4 КБайта).

Таким образом, размер программы определяется как разность адресов вершины стека и начала PSP, т.е. V= (SS:SP- PSP:0)/16+1, где добавляемая единица учитывает вероятную не кратность размера программы 16-ти.

Для уменьшения размеров HEAP и STACK можно использовать глобальные переменные unsigned **_heaplen** и unsigned **_stklen**, которые определяют размеры этих областей в байтах, но необходимо удостовериться, что заданных областей для программы будет достаточно.

Пример 1. Определение размера резидентной части программы:

```
# include <dos.h>
# include <stdio.h>
# include <stdlib.h>
enum{ERROR=-1,OK};
unsigned _stklen=256;        //  определение размера стека
unsigned _heaplen=1024;      //  определение размера динамической области
char far *tsrstack;          //   адрес стека
char far *tsrbottom;         //   адрес области PSP
int main(void)
    {
```

```
unsigned tsrsize;      //    размер программы в параграфах
tsrstack=(char far *)MK_FP(_SS,_SP);
tsrbottom=(char far *)MK_FP(_psp,0);
tsrsize=((tsrstack-tsrbottom)>>4)+1;
printf("Размер программы = %4d параграфа",tsrsize);
keep(OK,tsrsize);
return ERROR;
}
```

При запуске данная программа выдает на экран свой размер в параграфах:

Размер программы = 154 параграфа (или 2464 байта).

Реальный размер резидента - 2736 байт, так как еще 272 байта займет копия среды DOS, создаваемая при запуске любой программы. Адрес этой области находится в слове, расположенном со смещением +2CH относительно начала PSP. Для того чтобы освободить эту область достаточно указать: **freemem(peek(_psp,0x2C));** но тогда ваша программа не будет обнаруживаться как резидентный процесс детекторами.

Получение значения вектора прерывания и его изменение осуществляется с помощью функций:

void interrupt(*getvect(int <номер>))(); - получить вектор и

void setvect(int <номер>,void interrupt(*<имя>)()); - записать адрес нового обработчика прерывания в вектор.

Ключевое слово **interrupt,** так же как и в Турбо Паскале используется для описания функций-обработчиков прерываний, что предполагает сохранение и восстановление всех регистров и возврат управления командой **iret**.

Пример 2. Программа перехватывает прерывание 9H, читает scan-код из порта 60H, вызывает старый обработчик прерываний и генерирует звук, частота которого соответствует номеру нажатой клавиши. При инсталляции обработчика читается и фиксируется старое значение вектора прерывания, на его место заносится новое значение, которое в свою очередь заменяется старым при нажатии клавиши ESC.

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>

void interrupt (*old_int0x09)(...);   //  переменная - адрес старого обработчика прерываний
void interrupt new_int0x09(...)       //  заголовок нового обработчика прерываний
   {
     int i;
     static odd=0;
     odd=!odd;   // используется для предотвращения выдачи звука при прерывании прерывания
     i=inp(0x60);    // обращение к порту 60H
     (*old_int0x09)();    // вызов старого обработчика прерываний
     if (odd) return;
     sound(i<<4);     //  звук частотой i*16
     delay(500);      //   задержка
     nosound();       //   отключение звука
   }
void main(void)
   {
     old_int0x09=getvect(0x09);      //  получить старое значение вектора
     setvect(0x09,new_int0x09);      //   записать новое значение вектора
     while(getch()!=27);
     setvect(0x09,old_int0x09);      //  восстановить старое значение вектора
    }
```

При необходимости функция может описываться с параметрами, так как при входе в обработчик прерывания содержимое всех регистров сохраняется в стеке. Порядок описания параметров должен соответствовать порядку нахождения их значений в стеке: **void interrupt <имя>(unsigned bp, unsigned di, unsigned si, unsigned es, unsigned dx, unsigned cx, unsigned bx, unsigned ax, unsigned ip, unsigned cs, unsigned flags,...);**. Однако, к сожалению, при написании инсталлятора в этом случае следует учитывать используемый компилятор, так как функция **setvect** Турбо C++ и Borland C++ из-за контроля типа параметров функции в C++ не по-

зволяет заносить в область векторов прерываний адрес функции с параметрами. Для того чтобы обойти это ограничение, нужный адрес можно записать напрямую, вызвав функцию 25H прерывания int 21H.

Пример 3. Обработчик прерывания int 16H, выполняющий ту же операцию, что и в Примере 2.

Вариант 1(Для Турбо Си):

```
#include <dos.h>
#include <conio.h>
void interrupt (*old_int0x16)();
#pragma warn -par /* Отключение сообщения о неиспользуемых параметрах*/
 void interrupt  new_int0x16(unsigned int bp,unsigned int di,
                             unsigned int si,unsigned int ds,
                             unsigned int es,unsigned int dx,
                              unsigned int cx,unsigned int bx,
                              unsigned int ax,unsigned int ip,
                              unsigned int cs,unsigned int fl)
   {
      int i;
     i=ax>>8;              //       запись в i содержимого AH
      _AX=ax;              //        восстанавливаем значение AX из стека
      (*old_int0x16)();    //         вызываем старый обработчик
       ax=_AX;             //      записываем в стек результаты для
       fl=_FLAGS;          //      вызывающей программы
       if (i==0||i==0x10){                    //    при чтении кода клавиши
                  sound((ax>>8)<<4);      //    выдаем звуковой сигнал
                    delay(500);
                   nosound();
                  }
   }
 #pragma warn +par
 void main(void)
      {
        old_int0x16=getvect(0x16);
        setvect(0x16,new_int0x16);
        while(getch()!=27);
        setvect(0x16,old_int0x16);
       }
```

Вариант 2(для Borland C++):

```
 #include <dos.h>
 #include <conio.h>
 void interrupt (*old_int0x16)(...);
        .     .     .
 void main(void)
      {
        old_int0x16=getvect(0x16);
        int ret;                                    //   код возврата
        union REGS regs;                           //   определение структуры регистров общего назн.
        struct SREGS sregs;                        //   определение структуры сегментных регистров
        regs.x.ax=0x2516;                         //   запись номера функции и номера прерывания
        regs.x.dx=FP_OFF(new_int0x16);        //   запись смещения адреса обработчика
        sregs.ds=FP_SEG(new_int0x16);         //   запись сегментного адреса обработчика
        ret = intdosx(&regs, &regs, &sregs);     //   вызов функции прерывания int 21H
        while(getch()!=27);
        setvect(0x16,old_int0x16);
      }
```

2.9. Предотвращение повторной загрузки резидентного обработчика прерываний в память. Передача параметров резидентам

Достаточно часто от резидентной программы требуется, чтобы она могла определить свое наличие в памяти для предотвращения повторной загрузки.

Существует множество вариантов решения этой задачи. Как правило, используются следующие методы:

1. *Сигнатуры в памяти.* При инсталляции программы в некоторое место памяти пишется специальная сигнатура, по наличию которой в дальнейшем и определяют наличие резидента. Проблема заключается в том, чтобы найти такое место, где сигнатура не мешала бы нормальной работе. В качестве кандидата на такое место можно рассматривать: неиспользуемые вектора прерываний, область данных BIOS и т.п. Однако на практике найти такое место достаточно сложно, и нет никаких гарантий, что оно не будет использоваться другими программами.

2. *Сигнатура в тексте резидента* непосредственно перед обработчиком прерывания. В этом случае достаточно проверить 1-2 байта, расположенных по адресу перед считанным из вектора, и наличие данного резидента будет установлено. К сожалению, данный способ не работает, если после установки резидента прерывание было перехвачено другой программой, даже, если сама обработка прерывания выполняется.

3. *"Магические" числа* в дополнительных обработчиках прерываний. Перед тем как оставить резидент в памяти программа инсталляции обращается к некоторому прерыванию с определенным "магическим" числом. Если существует резидентная копия обработчика, то она перехватывает это прерывание и отвечает другим заранее определенным "магическим" числом. Получив ответ, программа инсталляции завершает работу без сохранения резидента в памяти. Если получен любой другой ответ, программа инсталлируется. Поскольку обработчик дополнительного прерывания находится в одном модуле с обработчиком основного прерывания, появляется возможность изменения параметров резидентной программы (см. Пример 1 ниже). Метод не работает при полном перехвате дополнительного прерывания другой программой.

4. Использование *мультиплексного прерывания DOS* (int 2FH). Данное прерывание используется системой для связи с собственными резидентными процессами. При его вызове в регистре AH задается номер функции (номер резидентного процесса), а в регистре AL- номер подфункции. Функции с номерами от 00H до BFH зарезервированы для использования системой, а функции с C0H по FFH могут использоваться прикладными программами. При инсталляции обработчику присваивается оригинальный мультиплексный номер, по наличию которого в системе и определяется присутствие данного резидента[3,5]. Естественно, гарантии, что другой резидент не использует тот же номер, нет. Так же как и в предыдущем случае, резидент должен включать дополнительный обработчик прерывания (int 2FH), который, помимо предотвращения повторной загрузки, может использоваться для связи с резидентным процессом, например, для передачи ему параметров. Далее в разделе2.10 приводится пример использования данного метода.

5. Запись *сигнатуры в PSP резидента*. При попытке повторной загрузки инсталлятор проверяет PSP всех загруженных программ. Для этого используется функция DOS 52H, которая позволяет получать сегментные адреса управляющих блоков памяти (MCB)[2]. Метод дает полную гарантию, но относительно сложен.

Пример 1. Резидентный "замедлитель". Программа используется для замедления работы быстрых компьютеров. Она перехватывает прерывание BIOS int 08(таймер) и запускает в его обработчике цикл, состоящий из переменного числа сложений. Также она обнаруживает свое присутствие в памяти при повторной загрузке и позволяет изменять количество циклов в уже загруженной копии. Для этого она перехватывает зарезервированное DOS-прерывание 3Ah и использует технику работы с "магическими" числами.

{Copyright Boris Ivanov, 1995}

```
{$M 1024,0,0}
Program SlowDown;
  Uses DOS;
  Const    {"магические" числа}
        check=$FFC5;
        answer=$FFCF;
        ask=$FFC4;
        change=$FFC3;
        disable=$FFC2;
  Type      overlap=record    {разбиение длинного целого на слова для передачи через регистры}
                        first:word;
                        second:word
                    end;
  Var   i,N,NN,f2:longint;
```

```
        code:integer;
        stroka:string;
        Int8:procedure;    {Адрес старого обработчика прерывания}
        r:registers;
        Work:boolean;     {Признак рабочего состояния резидента}
        over:overlap absolute N;  { Наложение двух слов на длинное целое}
  {$F+}         {обработчики прерываний}
   {int 08-таймер}
    Procedure Cicle(Flags,CS,IP,AX,BX,CX,DX,SI,DI,ES,DS,BP:word);interrupt;
      Begin
        Inline($9C);
        Int8;
        if Work then for i:=1 to N do inc(f2);
      end;
  {int 3ah - неиспользуемое DOS прерывание}
    Procedure Inst(Flags,CS,IP,AX,BX,CX,DX,SI,DI,ES,DS,BP:word);interrupt;
       Begin
          case CX of
             check:CX:=answer;{подтверждение наличия в памяти}
             ask:begin       {сообщить количество циклов}
                      AX:=over.first;
                      BX:=over.second;
                  end;
             change:begin         {изменить количество циклов}
                      Work:=true;
                      over.first:=AX;
                      over.second:=BX;
                       end;
             disable:Work:=false; {отключить работу}
          end;{case}
       end;
{$F-}
 Begin    {начало программы-инсталлятора}
    Writeln(' SlowDown. Demo version 2.2');
    if  ParamCount=1 then     {прочитать данные из командной строки}
        begin
            stroka:=ParamStr(1);
           Val(stroka,NN,code);
           if code<>0 then  {неверный ввод}
                begin
                   Writeln('Number please');
                   Halt(1);
                end
          else if NN=0 then Work:=false else Work:=true;
         end
    else   {не задано число циклов}
      begin
          Writeln('Format:slow.exe number_of_circles');
          Halt(1)
      end;
    r.CX:=check; {проверить наличие обработчика в памяти}
    Intr($3a,r);
    if r.CX=answer then
        begin     {обработчик уже резидентен}
            r.CX:=ask;      {запросить параметры в памяти}
            Intr($3a,r);
            over.first:=r.AX;
```

```
            over.second:=r.BX;
            Writeln('Already installed. Number of circles-',N);
            N:=NN;      { изменить число циклов}
            r.CX:=change;
            r.AX:=over.first;
            r.BX:=over.second;
            Intr($3a,r);
            if Work then Writeln('New Number of circles-',N)
           else
              begin      {или отключить работу}
                 r.CX:=disable;
                 Intr($3a,r);
                 Writeln('Slowing is off');
              end
        end
    else    begin   {установка векторов прерываний и сохранение}
              N:=NN;
              Writeln('Installed...');
              SwapVectors;
              GetIntVec($8,@Int8);
              SetIntVec($3a,@Inst);
              SetIntVec($8,@Cicle);
              Keep(0);
            end
end.
```

На Рисунке 1.5 условно показано взаимодействие двух копий обработчика прерываний: резидентной и только что загруженной через дополнительное прерывание. При этом инсталлятор новой копии программы обращается к прерыванию int 3aH для определения загружена ли программа, для определения количества циклов в загруженной копии, для изменения количества циклов в загруженной копии и для отключения работы обработчика прерывания int 8. Обмен данными между инсталлятором и резидентным обработчиком int 3aH производится через регистры общего назначения.

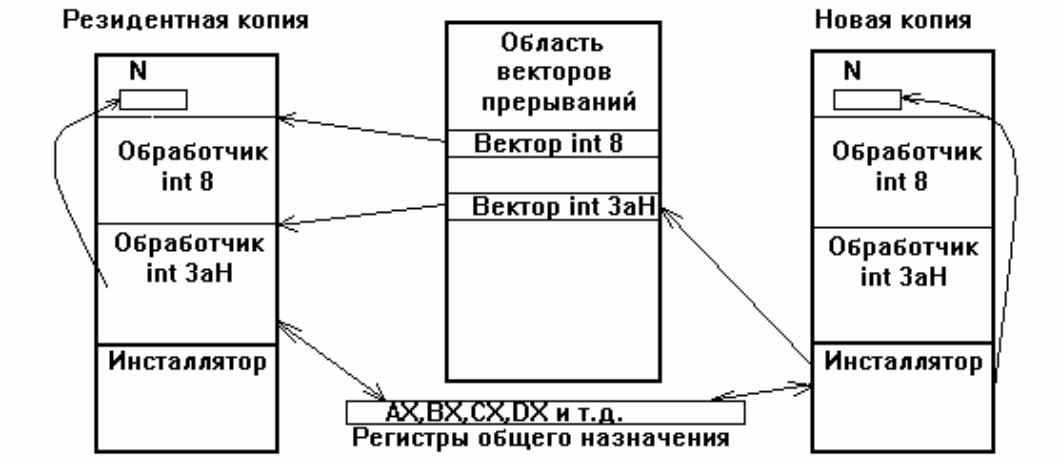

Рисунок 2.Е. Взаимодействие двух копий обработчика через дополнительное прерывание и регистры микропроцессора.

2.10. Удаление резидентного обработчика прерывания из памяти

Все рассмотренные выше резидентные программы не обеспечивают возможности выгрузки из памяти, и единственная возможность освободить занимаемую ими память - перезагрузка машины. На практике же большинство серьезных резидентных программ имеют встроенные средства выгрузки.

Выгрузка резидентного обработчика прерывания из памяти выполняется в два этапа: сначала необходимо восстановить вектор прерывания, а затем освободить память, занимаемую резидентом. Причем, прежде чем восстанавливать вектор необходимо убедиться, что никакая другая программа не захватила данное прерывание и не вызывает наш обработчик в качестве "старого". Удаление обработчика в таком случае, скорее всего, приведет к "зависанию" машины. Достаточно надежно можно удалить лишь последний загруженный резидент.

Инициатором процесса выгрузки может служить запуск специальной выгружающей программы. Удобнее всего в качестве выгружающей программы использовать саму резидентную программу, которая при повторном запуске должна получить доступ к первой копии и выгрузить ее. Так же как при передаче параметров доступ к первой копии можно осуществить через перехват дополнительных прерываний (int 2fH или любого другого свободного прерывания).

Пример 1. "Самовыгружающийся" резидентный обработчик прерывания int 1cH. При запуске программы без параметра происходит проверка повторной загрузки обработчика и инсталляции его, если он не был инсталлирован ранее. Запуск с параметром "off" приводит к выгрузке обработчика. В качестве дополнительно перехватываемого прерывания используется прерывание int 2fH. Обработчик этого прерывания использует номер 0c8H для идентификации резидентного процесса и выполняет две подфункции: 0 - проверка повторной загрузки и 1 - выгрузка резидента. Обработчик основного прерывания int 1cH каждые десять тиков таймера выводит на экран символы ASCII (см. Пример 1 раздела 2.7).

```
text        segment  'code' public byte
            assume   CS:text,DS:text
            org      256
main        proc
            jmp      init
old_2fh  dd      0              ; старое значение вектора прерывания int 2fH
old_1ch  dd      0              ; старое значение вектора прерывания int 1cH
; обработчик int 2fH
new_2fh   proc
            cmp      AH,0c8H
            jne      out_2fh
            cmp      AL,00H      ; если проверка загрузки резидента
            je       inst        ;  то выдать в AL специальный код
            cmp      AL,01h   ;  если выгрузка
            je       uninstalled   ;  то перейти к выгрузке
            jmp      short out_2fh   ; иначе - передать управление старому обработчику
 inst:      mov      AL,0ffH
            iret
 out_2fh:  jmp      CS:old_2fh
 uninstalled:
            push     DS
            push     ES
            push     DX
            mov      AX,251cH                ;  восстановить
            lds      DX,CS:old_1ch           ;   старые
            int      21H                     ;   адреса
            mov      AX,252fH                ;  обработчиков
            lds      DX,CS:old_2fh           ;  прерываний
            int      21H                     ;  в области  векторов
            mov      ES,CS:2cH     ;  освободить
            mov      AH,49H        ;  область, занимаемую
            int      21H           ;  окружением DOS (адрес из PSP)
            push     CS                      ;  освободить
```

```
            pop     ES                      ;    область,
            mov     AH,49H                  ;    занимаемую
            int     21H                     ;    программой
            pop     DX
            pop     ES
            pop     DS
            iret
 new_2fh   endp
 ; обработчик int 1cH
 new_1ch   proc      ;  основной обработчик: используя таймер пишет в видеопамять
            cli      ;  коды символов ASCII
            inc     CS:tik
            cmp     CS:tik,10
            jl      a1
            push    ES
            push    AX
            mov     AX,0b800H
            mov     ES,AX
            xor     AL,AL
            mov     CS:tik,al
            inc     CS:nch
            mov     AL,CS:nch
            mov     ES:[0],AL
            mov     AL,1eH
            mov     ES:[1],AL
            pop     AX
            pop     ES
  a1:       sti
            push    AX
            pop     AX
            iret
tik         db      ?
nch         db      0
new_1ch   endp
main      endp
end_rez=$       ;    адрес конца резидентной части
tail        db      'off'       ;    значение параметра, означающего выгрузку резидента
flag        db      0           ;    флаг  выгрузки
init        proc      ; инсталлятор
            mov     CL,ES:80H       ;    длина области параметров из PSP
            cmp     CL,0            ;    если нет параметров
            je      ahead           ;    то   проверяем  загружен ли резидент
            xor     CH,CH           ;    иначе   формируем счетчик CX
            mov     DI,81H          ;    заносим в DI смещение строки параметров
            mov     SI,offset tail  ;    заносим в SI смещение “правильного” параметра
            mov     AL,' '          ;    организует
repe        scasb                   ;    пропуск всех
            dec     DI              ;    пробелов и
            mov     CX,3            ;    проверяем значение
repe        cmpsb                   ;    параметра
            jne     ahead           ;    если не “off”, то переходим к проверке повторной загрузки
            inc     flag            ;    иначе - установим флаг выгрузки
ahead:      mov     AH,0c8H             ;    обращаемся к п/функции
            mov     AL,0                ;    определения повторной загрузки
            int     2fH                 ;    прерывания int 2fH
            cmp     AL,0ffH         ;  если ответ=ожидаемому
            je      installed       ;  то  переходим к выдаче сообщения о повторной загрузке
```

```
        mov     AX,352fH                ; иначе - сохраняем старые
        int     21H                     ; значения векторов преры-
        mov     word ptr old_2fh,BX     ; ваний int 2fH и int 1cH и
        mov     word ptr old_2fh+2,ES   ; записываем туда адреса
        mov     AX,351cH                ; своих обработчиков
        int     21H                     ;
        mov     word ptr old_1ch,BX     ;
        mov     word ptr old_1ch+2,ES   ;
        mov     AX,252fH                ;
        mov     DX,offset new_2fh       ;
        int     21H                     ;
        mov     AX,251cH                ;
        mov     DX,offset new_1ch       ;
        int     21H                     ;
        mov     AH,09H                  ; печатаем сообщение о
        mov     DX,offset mes           ; загрузке резидента
        int     21H                     ;
        mov     AX,3100H                ; выходим, сохранив в
        mov     DX,(end_rez-main+100H+0fH)/16   ; памяти резидентную
        int     21H                     ; часть программы
installed:
        cmp     flag,1      ; если необходимо выгрузить программу
        je      unins       ; то переходим на выгрузку
        mov     AH,09H      ; иначе - выдаем сообщение о том
        mov     DX,offset mes1  ; что программа уже
        int     21H             ; загружена в память
        mov     AX,4c01H        ; и выходим без
        int     21H             ; сохранения в памяти
unins:  mov     AX,0c801H   ; обращаемся п/функции удаления
        int     2fH         ; прерывания int 2fH
        mov     AH,09H      ; выдаем сообщение
        mov     DX,offset mes2  ; о выгрузке программы
        int     21H             ; из памяти
        mov     AX,4c00H        ; и выходим без
        int     21H             ; сохранения в памяти
mes     db      'Программа загружена',10,13,'$'
mes1    db      'Программа уже загружена',10,13,'$'
mes2    db      'Программа выгружена из памяти',10,13,'$'
init    endp
text    ends
        end     main
```

Программа использует параметр, задаваемый в командной строке при ее запуске. Доступ к параметрам командной строки осуществляется через область PSP, по адресу 80H в которой записывается длина строки параметров, а начиная с адреса 81H располагаются сами параметры.

2.11. Предотвращение зависания при использовании функций DOS

К сожалению, функции DOS не являются повторно входимыми, и в том случае, когда при работе обработчиков прерываний, использующих данные функции, происходит их прерывание, во время которого происходит обращение к другим функциям DOS, машина, как правило, “зависает”. Это связано с тем, что диспетчер функций DOS хранит только один адрес возврата для функций одного типа, и после завершения выполнения второй функции того же типа не может вернуться и завершить выполнение первой функции.

Тип функции DOS определяется используемым стеком. Всего функции DOS использует три варианта стека: стек ввода-вывода (функции с номерами от01H до 0cH), дисковый стек (функции с номерами 00H и от 0dH до 6cH) и стек пользовательской программы.

Таким образом, при разработке обработчиков прерываний использование функций DOS не желательно, и, по возможности, их следует заменять функциями BIOS. В тех же случаях, когда такой возможности нет, для предотвращения повторного вызова функции DOS можно использовать специальный флаг операционной системы, адрес которого сообщает функция 0x34 прерывания int 21H.

Пример 1. Часы, звонящие каждые 5 секунд. При разработке программы используется прерывание int 1cH, вызываемое при каждом “тике” таймера после int 8H. Обработчик отслеживает частоту и длительность каждого звука исполняемой мелодии. Для предотвращения нарушения работы системы при использовании функции DOS в обработчике прерывания программа получает адрес флага занятости DOS (см. функцию playback) при подключении и, прежде чем обратиться к функции DOS при отключении, определяет, не выполняется ли какая-нибудь другая функция DOS в системе (см. функцию nextTone).

```
#include <conio.h>
#include <dos.h>
#include <bios.h>
#ifdef __cplusplus      //  в зависимости от компилятора
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif
#define MAX_LENGTH 33     //  не более 33 звуков в пьесе
#define msec2ticks(msecs) (msecs*18.2/1000)   //  перевод миллисекунд в импульсы
static void interrupt new_int0x1C(__CPPARGS);   //  прототип обработчика
static void  nextTone(void);   //  прототип функции, которая играет очередную ноту
static char far *InDosPtr;    //  указатель на флаг занятости DOS
typedef struct {    //  информация об отдельном звуке
       unsigned pitch;  //     частота
       unsigned duration;   //  длительность
                }TONE;
struct  {    //    пьеса
    TONE play[MAX_LENGTH];   //     массив звуков
    int length;   //  количество звуков
        }stuff;
long volatile counter;    //  счетчик (перед использованием читать из памяти)
long longitude;             //   величина временной задержки
int volatile stillPlaying=0;   //  признак того, что одна пьеса уже проигрывается
void interrupt(*old_int0x1C)(__CPPARGS);   //  адрес старого обработчика прерывания
static void nextTone(void)      //  функция, которая играет очередную ноту
      {
       if (stuff.length-->0) {    //  если есть несыгранные ноты
            longitude=msec2ticks(stuff.play[stuff.length].duration);   // определяем продолжительность
          counter=0;   //  устанавливаем начало отсчета
          sound(stuff.play[stuff.length].pitch);   //  играем
            }
      else { nosound();    //  иначе - выключаем звук
          if (*InDosPtr) return;    //  если есть активная функция DOS, то прекращаем обработку
          setvect(0x1C,old_int0x1C);   //  иначе - возвращаем старое значение вектора
```

```
            stillPlaying=0;   //  сбрасываем  признак того, что играется пьеса
            }
 }
static void interrupt new_int0x1C(__CPPARGS)   //   обработчик прерывания int 1cH
  {
     if(++counter>=longitude)nextTONE()   // если интервал завершился, то играем след. звук
    }
int playback(TONE *play,int length)    //  проигрываем мелодию длиной length из массива play
  {
    int i;
    if (length>MAX_LENGTH) return(0);   //   если длина >максимальной, то выходим
    if (stillPlaying) return(0);    //   если музыка уже проигрывается, то выходим
    stuff.length=length;      //   копируем пьесу
    for(i=0;i<stuff.length;i++)  stuff.play[--length]=play[i];
    _AH=0x34;     //    получим адрес флага выполнения функции  DOS
    geninterrupt(0x21);
    InDosPtr=(char far*)MK_FP(_ES,_BX);
    stillPlaying=1;      //    установим признак того, что музыка уже проигрывается
    old_int0x1C=getvect(0x1C);    //    получим адрес старого обработчика прерывания int 1cH
    setvect(0x1C,new_int0x1C);     //    запишем в вектор адрес своего обработчика
    nextTone();     //   сыграем первую ноту
    return 1;
  }
 void main(void)
       {
          int played=0;
          TONE music[22];
          struct time timer;
          clrscr();
          music[0].pitch=2000;         //   музыка из двух нот
          music[0].duration=200;
          music[1].pitch=3000;
          music[1].duration=100;
          while(bioskey(1)!=0x011B)      //   до нажатия  ESC
               {
               if (bioskey(1) && bioskey(1)!=0x011B) bioskey(0);  // если не ESC, то удалим из буфера
               gettime(&timer);  //  узнаем время
               gotoxy(15,15);    //  и выведем его на экран
               cprintf("%02d:%02d:%02d",timer.ti_hour,timer.ti_min,timer.ti_sec);
                 if ((timer.ti_sec % 5)==0) {      //  если время кратно 5 сек.
                     if (!played){        //     и музыка не проигрывается,
                                 played=1;   //   то установим признак проигрывания
                                playback(music,2);   //  и начинаем играть
                                  }
                 }
               else played=0;   //  иначе устанавливаем признак, что музыка не проигрывается
           }
          while(stillPlaying);    // ожидаем восстановления старого вектора прерывания
          bioskey(0);       //    убираем ESC из буфера
  }
```

3. Связь разноязыковых модулей

3.1. Основные положения

На практике достаточно часто приходится связывать между собой программы, написанные на разных языках программирования. Как правило, это относится к подключению модулей, написанных на языке ассемблера, к программным системам, написанным на Паскале или Си. Использование модулей, написанных на языке ассемблера, позволяет сократить программу, ускорить выполнение каких-то ее частей, а иногда и позволяет выполнить действия, программирование которых с использованием языков высокого уровня невозможно или затруднительно.

При связывании разноязыковых модулей приходится решать следующие проблемы:

– как организовать передачу управления модулю и обратно;
– как передавать параметры в модуль и получить результаты его работы обратно;
– как согласовать представление данных при использовании различных языков программирования.

Каждый язык программирования высокого уровня использует свои способы представления данных и передачи управления и параметров в подпрограммы, описанные в соответствующих Руководствах. Кроме этого, при выполнение модулей, написанных на языках высокого уровня, происходят обращения к некоторым процедурам и функциям, получившим название *среды языка программирования*, что также необходимо учитывать.

3.2. Связь модулей, написанных на Турбо Паскале и Ассемблере

3.2.1. Передача управления между модулями. Размещение области параметров в стеке

Для того чтобы встроить программный модуль на ассемблере в программный продукт, написанный на Турбо Паскале, необходимо учитывать конкретную реализацию вызова подпрограмм в этом языке.

В Турбо Паскале реализованы два варианта вызова процедур и функций:

– дальний - через интерфейсную часть модуля (far);
– ближний - в секции реализации (near).

Следовательно, все библиотечные процедуры и функции на ассемблере должны программироваться как far, а все встраиваемые в сам программный продукт - как near.

Турбо Паскаль передает параметры в вызываемую подпрограмму через стек и там же размещает локальные переменные. Вызов подпрограммы реализуется по варианту:

```
push    <параметр 1>      ;   занесения параметров в стек
......
push    <параметр n>
call    <far или near> <имя подпрограммы>    ;   вызов подпрограммы
```

Вызываемые же подпрограммы имеют стандартно оформленные вход и выход:

Вход:

```
<имя>   proc    <способ вызова: near  или far>
        push    BP          ;  сохранить старое BP в стеке
        mov     BP,SP       ;  установить базу для чтения параметров из стека
        sub     SP,<объем памяти для локальных переменных>
        <сохранение регистров SP, SS, DS>
        ..........
```

Выход:

```
        mov     SP,BP   ;   освободить память для размещения локальных переменных
        pop     BP      ;   восстановить значение BP
        ret     <размер области параметров>
```

Таким образом, в момент получения управления подпрограммой в стеке находятся параметры (в виде значений или адресов) и адрес возврата в вызывающую программу (2-х или 4-х байтовый в зависимости от варианта вызова (см. рисунок 3.а а). В дальнейшем, вызываемая программа размещает в стеке старое значение BP, область локальных переменных и использует стек для своих надобностей (см.рисунок 3.а б).

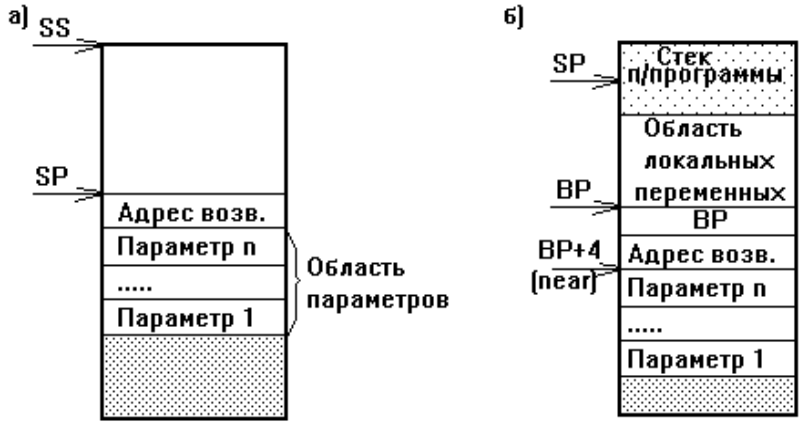

Рисунок 3.А. Содержимое стека: а) в момент передачи управления п/программе; б) во время работы п/программы.

Адрес области параметров в этом случае определяется относительно содержимого регистра BP: [BP+4] - содержит адрес последнего параметра при ближнем вызове; [BP+6] - содержит адрес последнего параметра при дальнем вызове. Адреса остальных параметров определяются аналогично с учетом длины каждого параметра в стеке (см. далее).

При выходе из подпрограммы команда **ret** должна удалить из стека всю область параметров, в противном случае произойдет нарушение работы вызывающей программы.

3.2.2. Некоторые внутренние форматы данных Турбо Паскаля

Турбо Паскаль использует следующие внутренние представления данных.

Целое -

shortint: -128..127 - байт со знаком;
byte: 0..255 - байт без знака;
integer: -32768..32767 - слово со знаком;
word: 0..65535 - слово без знака;
longint: - двойное слово со знаком.

Символ -**char**: код ASCII - байт без знака.

Булевский тип - **boolean**: 0(false) и 1(true) - байт без знака.

Указатель - **pointer**: сегментный адрес и смещение - двойное слово (сегментный адрес - в старшем слове.

Строка - **string:** символьный вектор, указанной при определении длины, содержащий текущую длину в первом байте.

Массив - **array:** последовательность элементов указанного типа, расположенных в памяти таким образом, что правый индекс возрастает быстрее левого (для матрицы - построчно).

3.2.3. Передача параметров по значению и ссылке. Возврат результатов процедур и функций

В Турбо Паскале параметры могут передаваться двумя способами: по значению и по ссылке (с указанием **var)**. В первом случае подпрограмме передаются копии значений параметров и, соответственно, она не имеет возможности менять значения передаваемых параметров в вызывающей программе. Во втором случае подпрограмма получает адреса передаваемых значений и может не только читать значения, но и менять их. И в том и в другом случае передача параметров выполняется через стек. Причем, если параметр, передаваемый по значению имеет длину больше 4-х байт (кроме вещественных чисел для которых допускается помещение в стек 6-ти байт), то в стек заносится указатель на копию данного параметра (например, для строк).

Например, задание списка параметров в следующем виде

....(a:integer; b:char; s:string;var c: byte)....

приведет к тому, что в стек последовательно будут помещены: 2 байта **a**, 2 байта **b**(так как запись в стек идет словами), 4-х байтовый указатель на копию строки **s** и 4-х байтовый указатель на байт **c**.

Процедуры Турбо Паскаля возвращают результаты через параметры, передаваемые по ссылке, а функции - как правило, через регистры:

байт - в AL;
слово - в AX;
двойное слово - в DX:AX;
вещественные числа - в DX:BX:AX;
указатели - в DX:AX.

Исключением является результат строкового типа, для размещения которого Турбо Паскаль записывает в стек до области параметров указатель на специально выделенную область. Этот указатель при выходе из подпрограммы не удаляется.

3.2.4. Особенности оформления и компоновки программного продукта, состоящего из модулей на Турбо Паскале и ассемблере

При написании конкретных программ следует учитывать следующее.

Основную программу следует писать на Турбо Паскале, что обеспечит подключение среды Турбо Паскаля. Модуль на ассемблере может включать одну или несколько подпрограмм (процедур или функций). Перед подключением он должен быть оттранслирован.

Для подключения модулей в объектном виде в Турбо Паскале используется директива компилятора **{$L <имя файла>}.** Подключаемые процедуры и функции при этом должны быть описаны в Турбо Паскале в соответствии с его правилами, причем вместо тела подпрограммы после заголовка указывается служебное слово **external** (внешний)**,** например:

Procedure ffff(y:integer; var z:char); external;

При использовании библиотечных модулей аналогичные описания вставляются в секцию **implementation**.

В ассемблере сегмент кодов должен носить имя **code**, а сегмент данных - **data**. Оба сегмента должны быть описаны **public**. При этом в процессе компоновки сегменты будут объединены, и появится возможность доступа к глобальным данным Паскаля через объявление их внешними (**extrn)** в сегменте данных ассемблерной части (см. Вариант 3 Примера 1 раздела 3.2.5). Причем, даже если таким образом осуществляется доступ к массиву, в **extrn** достаточно указать ссылку на первый элемент, например:

extrn mas:word ; mas - массив Паскаля, объявленный: **var mas:array[1:10] of integer.** Доступ к последующим элементам будет осуществляться по правилам ассемблера.

В свою очередь, данные ассемблерной части программы, даже будучи размещенными в общем сегменте данных с Паскалем, останутся для паскалевской части программы “невидимыми”. Кроме того, ассемблерные данные, размещенные в сегменте данных **data,** нельзя инициализировать.

Правила модульного программирования ассемблера также требуют, чтобы все имена программы, использующиеся отдельно транслируемыми модулями, были описаны как внутренние (**public**), а все имена, используемые ассемблером из других модулей, - как внешние**(extrn).**

3.2.5. Примеры программ

Пример 1. Программа сложения двух целых чисел.

Вариант 1. Реализация в виде процедуры и функции, подключаемых в секции реализации (вариант near). Параметры передаются через стек, а результат возвращается процедурой - через параметр, переданный по ссылке, функцией - через регистр AX.

Модуль на Турбо Паскале:

```
Program var_near;
{$l  add_near.obj}
Var a,b,c:integer;
Function fun_near(x,y:integer):integer;external;
Procedure proc_near(x,y:integer;var z:integer);external;
Begin writeln('Введите числа:');
    readln(a,b);
    proc_near(a,b,c);
    writeln('Результаты: функции - ',fun_near(a,b),
                ' процедуры - ',c);
end.
```

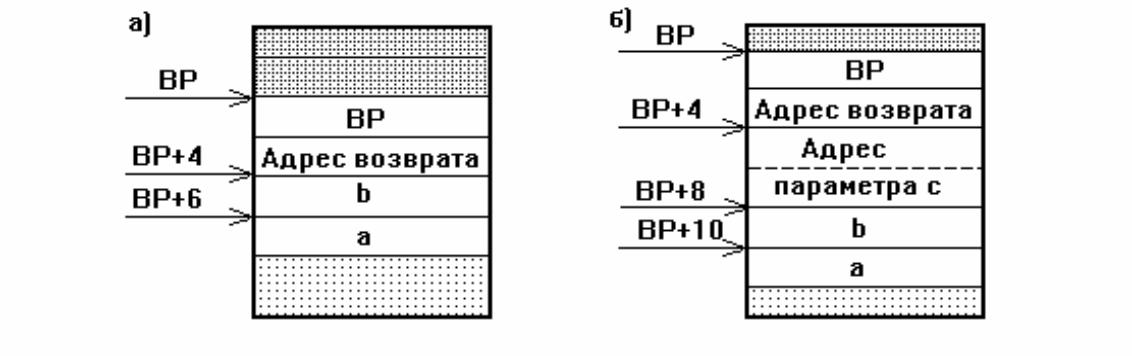

Рисунок 3.В. Структура стека во время выполнения подпрограммы на ассемблере: а - функции; б - процедуры

Модуль на ассемблере:

```
code    segment  byte public
        assume   CS:code
        public   fun_near,proc_near
; функция
fun_near  proc    near
        push    BP
        mov     BP,SP
        mov     AX,word ptr[BP+6]    ;
        add     AX,word ptr[BP+4]  ;   результат в AX
        mov     SP,BP
        pop     BP
        ret     4
fun_near  endp
; процедура
proc_near proc    near
        push    BP
        mov     BP,SP
        mov     AX,word ptr[BP+10]
        add     AX,word ptr[BP+8]
        les     DI,dword ptr[BP+4] ; загрузка адреса результата
        mov     ES:[DI],AX        ; запись результата
        mov     SP,BP
        pop     BP
        ret     8
```

```
proc_near endp
code    ends
        end
```

Вариант 2. Реализация в виде процедуры и функции, подключаемых через библиотеку (вариант far).

Библиотечный модуль на Паскале:

```
Unit bibl;        { имя библиотеки должно совпадать с именем файла}
Interface
    Function fun_far(x,y:integer):integer;
    Procedure proc_far(x,y:integer;var z:integer);
Implementation
{$l  add_far.obj}
     Function fun_far;external;
     Procedure proc_far;external;
end.
```

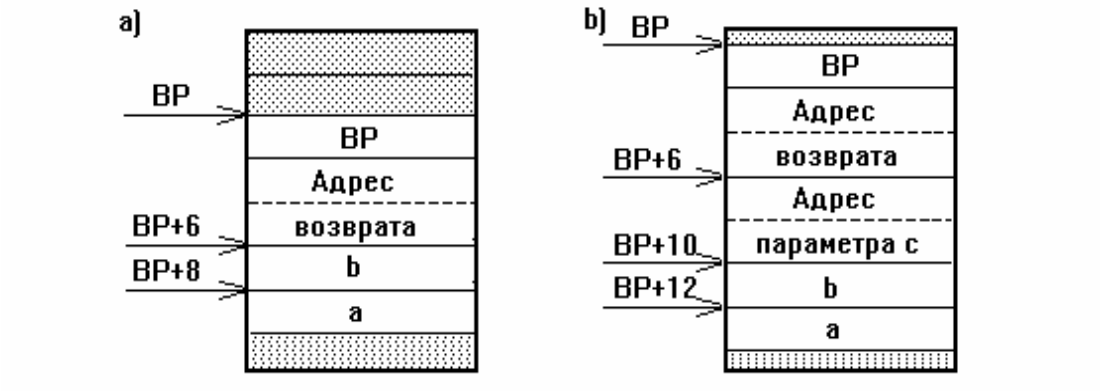

Рисунок 3.С. Структура стека во время выполнения подпрограммы на ассемблере: а - функции, б - процедуры.

Модуль на ассемблере:

```
code    segment  byte public
        assume   CS:code
        public   fun_far,proc_far
; функция
fun_far   proc    far
        push    BP
        mov     BP,SP
        mov     AX,word ptr[BP+8]
        add     AX,word ptr[BP+6]  ; результат в AX
        mov     SP,BP
        pop     BP
        ret     4
fun_far  endp
; процедура
proc_far proc    far
        push    BP
        mov     BP,SP
        mov     AX,word ptr[BP+12]
        add     AX,word ptr[BP+10]
        les     DI,dword ptr[BP+6] ; загрузка адреса результата
        mov     ES:[DI],AX        ; запись результата
        mov     SP,BP
        pop     BP
        ret     8
proc_far  endp
code    ends
        end
```

Модуль на Турбо Паскале, использующий созданную библиотеку:

```
Program ex;
Uses  bibl;
Var a,b,c:integer;
Begin writeln('Введите числа:');
      readln(a,b);
      proc_far(a,b,c);
      writeln('Результаты: функции - ',fun_far(a,b),
                  ' процедуры - ',c);
end.
```

Вариант 3. Передача параметров через “общий” сегмент данных data.

Модуль на Турбо Паскале:

```
Program pa1;
{$l  add_near.obj}
Var a,b,c:integer;
Function fun_near:integer;external;
Procedure proc_near(var z:integer);external;
Begin writeln('Введите числа:');
       readln(a,b);
      proc_near(c);
       writeln('Результаты: функции - ',fun_near,
                  ' процедуры - ',c);
end.
```

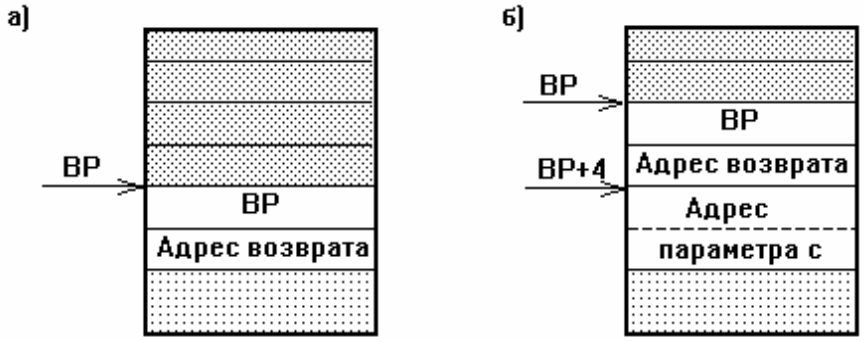

Рисунок 3.D. Структура стека во время выполнения подпрограммы на ассемблере: а - функции, б - процедуры.

Модуль на ассемблере:

```
data     segment  word public
         extrn    a:word,b:word
data     ends
code     segment  byte public
         assume   CS:code,DS:data
         public   fun_near,proc_near
; функция
fun_near  proc    near
         push    BP
         mov     BP,SP
         mov     AX,a
         add     AX,b     ; результат в AX
         mov     SP,BP
         pop     BP
         ret
fun_near  endp
; процедура
proc_near proc    near
         push    BP
         mov     BP,SP
         mov     AX,a
```

```
        add     AX,b
        les     DI,dword ptr[BP+4] ; загрузка адреса результата
        mov     ES:[DI],AX       ; запись результата
        mov     SP,BP
        pop     BP
        ret     4
proc_near endp
code    ends
        end
```

Содержимое стека в момент работы вызываемых подпрограмм для всех трех случаев см. на Рисунках 2.2, 2.3,2.4.

Пример 2. Программа удаления “лишних” пробелов в символьной строке. Пример иллюстрирует особый случай возврата значения функции типа **string** через указатель на специальную область, передаваемый функции в стеке, и обращение из подпрограммы на ассемблере к процедуре, написанной на Паскале (см. рисунок 3.е).

Модуль на Турбо Паскале:

```
Program probel;
{$l stroka.obj}
{Эта программа удаляет "лишние" пробелы.}
Var s:string;
Function sss(st:string):string;external;
Procedure print(n:byte);
Begin writeln('Длина полученной строки',n:3);end;
Begin
   readln(s);
   writeln(sss(s));
end.
```

Рисунок 3.Е. Передача параметров и управления в программе Примера 1.

Модуль на ассемблере:

```
code    segment  byte public
        assume   CS:code
        public   sss             ; объявление имени sss доступным извне
        extrn    print:near      ; объявление имени print определенным в других модулях
sss     proc     near
        push     BP        ;      состояние стека после записи BP см. на Рисунке 2.6 а
        mov      BP,SP
        push     DS     ; сохранить DS
        lds      SI,dword ptr[BP+4]    ; загрузить адрес параметра s
        les      DI,dword ptr[BP+8]    ; загрузить адрес результата
        cld      ;   установить флаг направления строковой обработки
        lodsb    ;    загрузить длину строки в AL
        inc      DI      ; пропустить место под длину результирующей строки
        mov      CL,AL     ;       занести длину строки
        xor      CH,CH    ;       в регистр CX
        jcxz     pusto       ;   если длина = 0, то перейти на завершение обработки
        mov      BX,1        ;    установить признак “гашение” пробелов
        mov      DL,0       ;   обнулить счетчик длины строки
```

```
cycl1:   lodsb          ;  загрузить символ в AL
         cmp    AL,' '  ;  сравнить с пробелом
         je     prod1   ; если пробел, перейти на обработку пробелов
         mov    BX,0    ; иначе  сбросить признак "гашения" пробелов
         inc    DL      ;  увеличить длину на 1
         stosb          ;  записать символ в строку результата
         loop   cycl1   ;  вернуться в цикл
prod1:   cmp    BX,1    ;  если признак "гашения" пробелов установлен
         je     prod2   ;  то вернуться в цикл
         mov    BX,1    ;  иначе установить признак "гашения" пробелов
         inc    DL      ; увеличить длину на 1
         stosb          ; записать разделительный пробел в строку результата
prod2:   loop   cycl1   ;  вернуться в цикл
         cmp    DL,0    ;  если все символы - пробелы
         je     prod3   ;  то  перейти к завершению обработки
         cmp    BX,1    ;  если последний символ не пробел
         jne    prod3   ;  то перейти к завершению обработки
         dec    DL      ;  иначе  - уменьшить длину на единицу
prod3:   mov    AL,DL   ;  запись в AL длины строки
pusto:   les    DI,dword ptr[BP+8]  ;  повторная загрузка адреса результата
         stosb          ;  занесение длины строки
         pop    DS   ; восстановить DS иначе процедура на Паскале не будет работать!
         xor    AH,AH
         push   AX      ;  записать в стек параметр
         call   print   ;  вызвать процедуру на Паскале
         mov    SP,BP
         pop    BP
         ret    4       ;  вернуться и удалить область параметров из стека
sss      endp
code     ends
         end
```

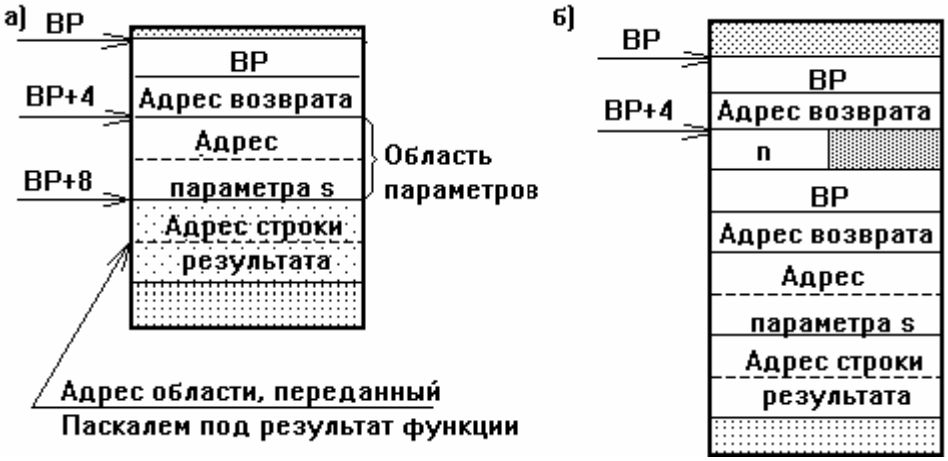

Рисунок 3.F. Структура стека: а - в начале выполнения ассемблеровского модуля, б - в начале выполнения процедуры на Паскале.

На Рисунке 2.6 а показано состояние стека после сохранения в нем содержимого регистра BP в начале подпрограммы на ассемблере. Область параметров, передаваемых подпрограмме на ассемблере, включает только объявленный в описании функции **sss** в программе на Паскале адрес исходной строки **s**. Адрес области, переданной Паскалем под результат функции типа **string**, пишется в стек до параметров и удалению функцией не подлежит. В свою очередь, передавая управление процедуре на Паскале, ассемблеровская программа помещает в стек параметр **n** (несмотря на то, что длина параметра 1 байт, в стек все равно помещается два байта, но второй байт не определен). При выполнении команды **call** туда же будет помещен адрес возврата, после чего уже процедура на Паскале поместит в стек значение регистра BP(см. рисунок 3.f б).

3.3. Связь модулей, написанных на Турбо C++(Borland C++) и Ассемблере

3.3.1. Особенности передачи управления между модулями, написанными на Си

В настоящее время реализованы два варианта передачи управления и параметров между основной программой и подпрограммой. Один, описанный выше, используется в программах на Турбо Паскале. Другой, использующий обратный порядок записи параметров в стек, реализован в Си. Кроме этого, вызываемая программа на Си не освобождает область параметров. Это делает вызывающая программа после возврата управления.

Так при вызове функции с прототипом:

void a(int p1, int p2, long int p3);

в стек сначала будет занесен параметр **p3** (длиной 4 байта), затем **p2** и **p1** (по два байта каждый), а затем уже адрес возврата (ближний или дальний в зависимости от используемой модели памяти) (см. рисунок 3.g а).

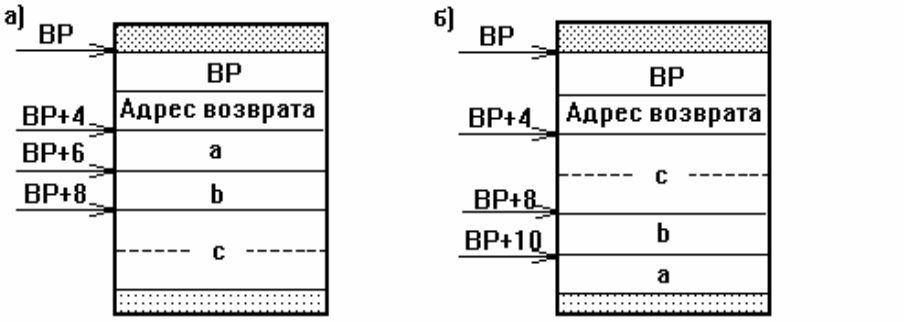

Рисунок 3.G. Структура стека при передаче параметров: а - по варианту, принятому в Си, б - по варианту, принятому в Паскале.

После вызова функции стек восстановит вызывающая программа, что происходит приблизительно следующим образом. (Точный текст программ не приводится, чтобы не усложнять картину.)

Вызывающая программа (модель Small):

```
 _TEXT segment byte public 'CODE'
    assume cs:_TEXT
    extrn   @a$qiil:near
_main       proc    near
    push    BP
    mov     BP,SP
    ....
    push    <параметр 3>
    push    <параметр 2>
    push      <параметр 1>
    push      <параметр 1>
    call      near ptr @a$qiil
    add       SP,8
    ....
    mov     SP,BP
    pop     BP
    ret
_main       endp
_TEXT  ends
    end
```

Вызываемая программа:

```
_TEXT   segment byte public 'CODE'
            assume cs:_TEXT
            public   @a$qiil
@a$qiil  proc    near
     push    BP
     mov     BP,SP
```

```
    ....
    pop     BP
    ret
@a$qiil  endp
_TEXT    ends
    end
```

Как видно из приведенных выше фрагментов существует еще одна особенность внутреннего представления программ на Си: компилятор языка изменяет используемые имена. Так, перед глобальными именами ставится символ подчеркивания, а к именам функций при использовании компиляторов Турбо С++ или Borland C++ в начало добавляется символ **@**, а в конец дописываются знаки **$q** и символы, кодирующие типы параметров функции:

void - v char - zc int - i float - f double - d
short - s long - l *, [ ] - p ... - e

Например, **fa(int *s[], char c, short t) => @fa$qppizcs.**

Примечание. Турбо С++ также предусматривает возможность использовать паскалевский способ передачи параметров. Для этого соответствующая функция должна быть объявлена как **pascal,** например: **void pascal f(int a,int b,long int c);** соответствующее размещение параметров в стеке см.рисунок 3.g б. Для отмены дополнения имен символом подчеркивания в этом случае рекомендуется указать опцию **Generate undebars... Off**.

3.3.2. Передача параметров. Возврат результатов

Как уже говорилось выше, передача параметров в Си осуществляется через стек. Причем что именно помещается в стек (значение или адрес) определяется явно средствами языка. При передаче параметров Си руководствуется следующими правилами:

Целое -

int, short int: -32768..32767 - слово со знаком;
unsigned int: 0..65535 - слово без знака;
long int: - двойное слово со знаком;
unsigned long int: - двойное слово без знака;
char: - -128..127 байт со знаком (передается слово);
unsigned char: 0..255 - байт без знака (передается слово).

Указатель, массив - **far**: сегментный адрес и смещение - двойное слово;
near : только смещение - слово.

Определенную сложность представляет лишь определение длины передаваемого указателя, так как последняя зависит от используемой модели памяти, так же как и тип вызова (near или far) подпрограммы (см.Таблицу 2.1).

Таблица 3.А. Типы указателей к функциям и к данным в зависимости от моделей памяти.

Модель памяти	Указатель к функции	Указатель к данным
Крохотная	near, (по умолчанию _cs)	near, (по умолчанию _ds)
Малая	near, (по умолчанию _cs)	near, (по умолчанию _ds)
Компактная	far	near, (по умолчанию _ds)
Средняя	near, (по умолчанию _cs)	far
Большая	far	far
Огромная	far	far

Если используется функция с переменным числом параметров, то это отразится только на размере области параметров, так как каждый параметр будет помещен в стек, а удаление параметров будет выполнять вызывающая программа.

Возвращаемые значения должны быть записаны в регистры:

char, short, int, enum, указатели **near -** в регистр AX;

указатели **far, huge** и прочие 4-х байтовые величины - в регистры DX:AX;

float, double - в регистры TOS и ST(0) сопроцессора;

struct - записывается в память, а в регистр записывается указатель (структуры длиной в 1 и 2 байта возвращаются в AX, а 4 байта - в DX:AX).

3.3.3. Особенности компоновки программного продукта, состоящего из модулей на Турбо С++ и ассемблере

Для того чтобы скомпоновать модули на ассемблере с программой, написанной на Си, необходимо следовать определенным соглашениям.

При компиляции исходной программы на Си создаются следующие сегменты:

– сегмент кода;

– сегмент данных;

– сегмент неинициализированных данных.

Причем во всех моделях памяти кроме Huge, два последних сегмента объединяются в группу DGROUP и адресуются регистром DS (см. рисунок 3.h). При использовании модели Huge сегмент неинициализированных данных вообще не создается и объединение в группу отсутствует, но появляется возможность определить столько инициируемых сегментов, столько модулей включает программа (таким образом, снимается ограничение на количество статических данных программы).

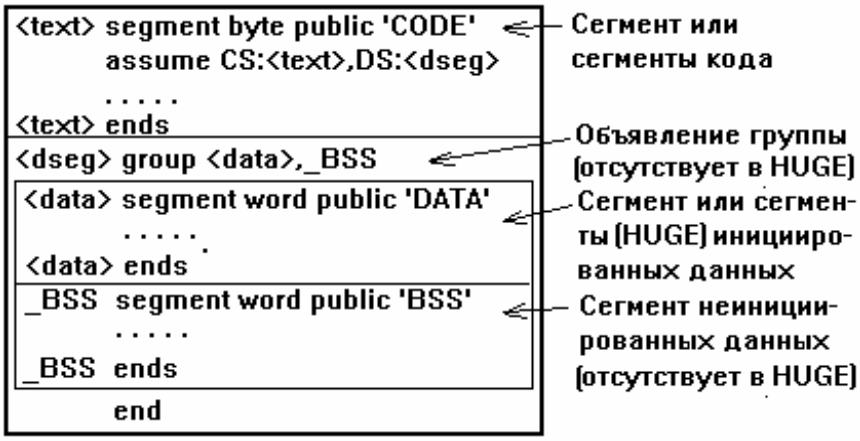

Рисунок 3.H. Структура сегментов модуля на Турбо С++.

На этапе компоновки сегменты, принадлежащие различным модулям, но имеющие одинаковые имена, объединяются.

Используемая модель памяти влияет не только на тип вызываемой функции и указателей на данные, но и на то, какие сегменты будет использоваться программой. В Таблице 2.2 приведены имена сегментов, используемые Си для различных моделей памяти.

Таблица 3.B. Имена сегментов, используемые различными моделями памяти.

Модель памяти	Сегмент кодов	Сегмент инициированных данных	Группа сегментов данных, адресуемых DS
Крохотная	_TEXT	_DATA	DGROUP
Малая	_TEXT	_DATA	DGROUP
Компактная	_TEXT	_DATA	DGROUP
Средняя	<имя файла>_TEXT	_DATA	DGROUP
Большая	<имя файла>_TEXT	_DATA	DGROUP
Огромная	<имя файла>_TEXT	<имя файла>_DATA	<имя файла>_DATA

Примечание. Компактная и большая модели памяти могут включать несколько дополнительных сегментов данных, но эти сегменты будут доступны только внутри модулей.

3.3.4. Определение глобальных и внешних имен

В отличие от Паскаля Си позволяет ассемблеру увеличивать список глобальных переменных, доступных для всех модулей. Это достигается за счет размещения переменных в сегменте данных, отведенном для глобальных переменных, и описания его внутренним **public.** Имя такой переменной по правилам Си должно начинаться со знака подчеркивания. Прочие модули, использующие данное имя, должны включать его описание как **extrn** (на ассемблере) или **extern** (на Си).

Аналогичным образом в ассемблере и Си должны описываться и функции, определяемые в одном месте и используемые в другом (см. рисунок 3.i) .

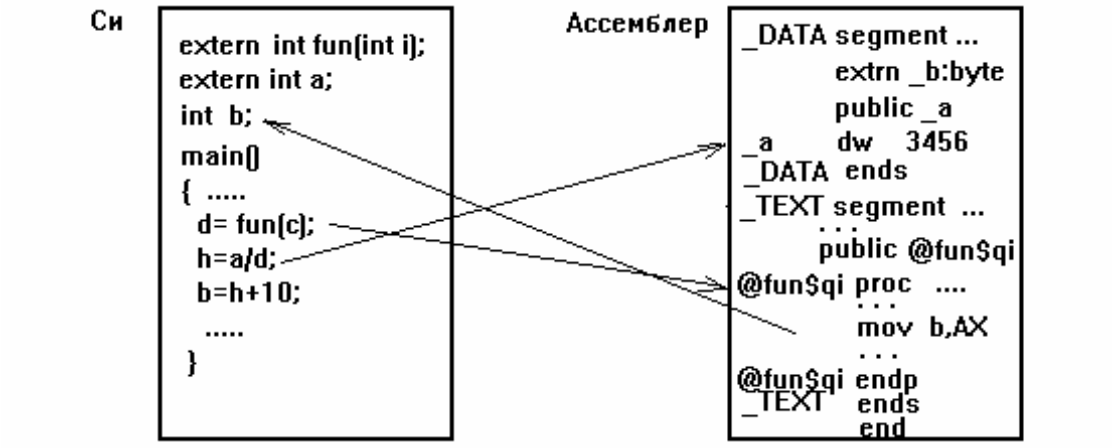

Рисунок 3.I. Взаимодействие модулей на Си и ассемблере по вызовам функций и доступу к данным.

3.3.5. Примеры программ

Пример 1. Определение минимального из двух чисел.
Вариант 1. Модель памяти Small. Функция типа near, адрес возврата в стеке длиной 2 байта.
Модуль на Турбо С++:

```
#include <stdio.h>
extern int amin(int x,int y);     //   Определение внешней функции
void main()
  { int a=3,b=5,c;
    c=amin(a,b);
    printf("c=%d",c);}
```

Модуль на ассемблере:

```
_TEXT segment byte public 'CODE'
          assume  CS:_TEXT
          public   @amin$qii
@amin$qii   proc    near      ;   функция определена с двумя параметрами int
          push   BP
           mov    BP,SP
          mov    AX,[BP+4]    ;   загрузка первого параметра
          cmp    AX,[BP+6]    ;   сравнение со вторым параметром
          jle      exit
          mov    AX,[BP+6]
exit:      pop    BP
          ret
@amin$qii   endp
_TEXT  ends
          end
```

Вариант 2. Модель памяти Medium. Функция типа far, адрес возврата в стеке длиной 4 байта.
Модуль на Турбо С++:

```
#include <stdio.h>
extern int amin(int a,int b);
void main()
  { int a=3,b=5,c;
    c=amin(a,b);
    printf("c=%d",c);}
```

Модуль на ассемблере:

```
EEE_TEXT segment byte public 'CODE'
         assume  CS:EEE_TEXT
         public   @amin$qii
@amin$qii   proc    far
          push    BP
          mov     BP,SP
          mov     AX,[BP+6]       ; загрузка первого параметра
          cmp     AX,[BP+8]       ;  сравнение со вторым параметром
          jle       exit
          mov    AX,[BP+8]
exit:      pop    BP
          ret          ;   адрес возврата - дальний
@amin$qii   endp
EEE_TEXT ends
          end
```

Вариант 3. Модель Small. Второй параметр передается своим адресом, занимающим в стеке 2 байта.
Модуль на Турбо С++:

```
#include <stdio.h>
extern int amin(int x,int *y);
void main()
```

```
  { int a=3,b=5,c;
    c=amin(a,&b);
    printf("c=%d",c);}
```

Модуль на ассемблере:

```
_TEXT segment byte public 'CODE'
        assume  CS:_TEXT
        public  @amin$qipi
@amin$qipi  proc   near
        push   BP
        mov    BP,SP
        mov    AX,[BP+4]    ;   загружаем первый параметр
        mov    BX,[BP+6]    ;   загружаем в BX адрес второго параметра
        cmp    AX,[BX]
        jle    exit
        mov    AX,[BX]
exit:   pop    BP
        ret
@amin$qipi   endp
_TEXT   ends
        end
```

Вариант 4. Модель Huge. Обращение из ассемблера к глобальной переменной a.

Модуль на Турбо С++:

```
#include <stdio.h>
extern int amin(int x);
int a;      //   a  размещается в сегменте статических данных GGG_DATA
void main()
  { int b=5,c;
    a=3;
    c=amin(b);
printf("c=%d",c);}
```

Модуль на ассемблере:

```
GGG_DATA segment byte public 'CODE'
        extrn _a:word      ;   описание _a внешней переменной
GGG_DATA ends
GGG_TEXT segment word public 'FAR_DATA'
        assume  CS:GGG_TEXT,DS:GGG_DATA
        public  @amin$qi
@amin$qi   proc   far
        push   BP
        mov    BP,SP
        mov    AX,_a      //   обращение к глобальной переменной  a
        cmp    AX,[BP+6]
        jle    exit
        mov    AX,[BP+6]
exit:   pop    BP
        ret
@amin$qi   endp
GGG_TEXT   ends
        end
```

Вариант 5. Модель Compact. Создание ассемблером глобального параметра.

Модуль на Турбо С++:

```
#include <stdio.h>
extern void amin(int x,int y);
extern int c;
void main()
   { int a=3,b=5;
     amin(a,b);
```

```
        printf("c=%d",c);}    //   в ячейку с результат пишется  в ассемблере
```

Модуль на ассемблере:

```
DGROUP group  _DATA
_DATA segment word public 'DATA'
          public _c
_c        dw      0       ;   создание глобального параметра с
_DATA ends
_TEXT  segment byte public 'CODE'
          assume  CS:_TEXT,DS:DGROUP
          public   @amin$qii
@amin$qii   proc   near
          push   BP
          mov    BP,SP
          mov    AX,[BP+4]
          cmp    AX,[BP+6]
          jle      exit
          mov    AX,[BP+6]
exit:     mov    _c,AX
          pop    BP
          ret
@amin$qii   endp
_TEXT  ends
            end
```

Пример 2. Определение минимального значения из заданных. Реализация с переменным количеством параметров функции. Модель Small.

Модуль на Турбо С++:

```
#include <stdio.h>
extern int amin(int count,int v1,int v2,...);   //   первый параметр - счетчик
void main()
  { int a=3,b=5,c;
    c=amin(5,a,b,1,10,0);
    printf("c=%d",c);}
```

Модуль на ассемблере:

```
_TEXT segment byte public 'CODE'
          assume  CS:_TEXT
          public    @amin$qiiie
@amin$qiiie   proc   near
          push    BP
          mov     BP,SP
          mov     AX,0
          mov     CX,[BP+4]    ;  в CX заносится количество значений
          cmp     CX,AX
          jle      exit
          mov    AX,[BP+6]     ;   в AX заносится первое значение из списка
          jmp     short ltest
compare: cmp    AX,[BP+6]
          jle      ltest
          mov    AX,[BP+6]
ltest:     add    BP,2
          loop   compare
exit:     pop     BP
          ret
@amin$qiiie  endp
_TEXT    ends
          end
```

Пример 3. Определение среднего арифметического последовательности из 10 чисел. Си вызывает функцию на ассемблере для суммирования чисел, а ассемблер вызывает функцию на Си для выполнения операции деления в вещественной арифметике(см. рисунок 3.j) .

Модуль на Турбо С++:

```
#include <stdio.h>
extern float Average(int far * ValuePtr, int NumberOfValues);
#define NUMBER_OF_TEST_VALUES 10
int TestValues[NUMBER_OF_TEST_VALUES] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
main()
{      printf("The average value is: %f\n",
        Average(TestValues, NUMBER_OF_TEST_VALUES));
}
float IntDivide(int Dividend, int Divisor)
{      return( (float) Dividend / (float) Divisor );
}
```

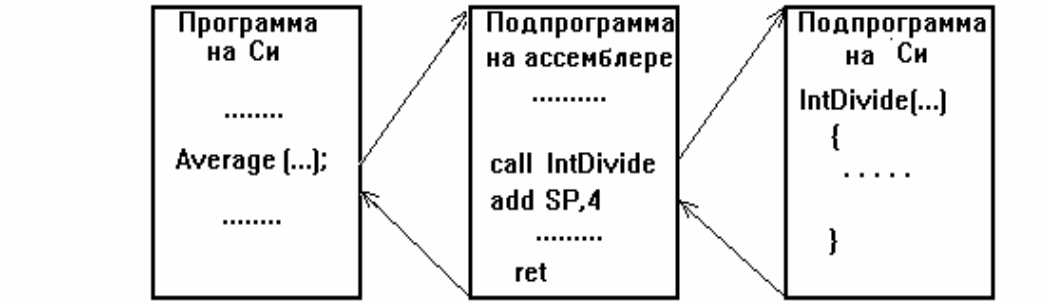

Рисунок 3.J. Структура программы определения среднего арифметического нескольких чисел.

Модуль на ассемблере:

```
        DOSSEG
        .MODEL  SMALL
        EXTRN   @IntDivide$qii:PROC
        .CODE
        PUBLIC  @Average$qnii
@Average$qnii   PROC
        push    BP
        mov     BP,SP
        les     BX,[BP+4]       ; загрузка в ES:BX адреса массива значений
        mov     CX,[BP+8]       ;   загрузка количества чисел
        mov     AX,0            ;     обнуление суммы
AverageLoop:
        add     AX,ES:[BX]      ;  добавление очередного значения
        add     BX,2            ;       переход к следующему значению
        loop    AverageLoop
        push    WORD PTR [BP+8]    ;  запись в стек количества чисел (второй параметр)
        push    AX              ;   запись в стек суммы чисел (первый параметр)
        call    @IntDivide$qii     ;     вызов функции на Си
        add     SP,4            ;     удаление параметров
        pop     BP
        ret                     ;   среднее значение находится в регистре TOS  8087
@Average$qnii   ENDP
        end
```

4. ЛИТЕРАТУРА

1. Д. Валдин. Резидентные программы на языке С.Часть 1.//Монитор. - 1993 - N5.
2. Д. Валдин. Резидентные программы на языке С.Часть 2.//Монитор. - 1993 - N6.
3. О. Шарапов. Загружен - незагружен, загружен - незагружен, загружен...?//Монитор. - 1994 - N1.
4. П. Дубнер. Прерывания и их обработчики.//Монитор. - 1994 - N6.
5. П.И.Рудаков, К.Г.Финогенов. Программируем на языке ассемблера IBM PC: В 4-х частях. Ч.2. Прикладное программирование - М.:”Энтроп”,1995.