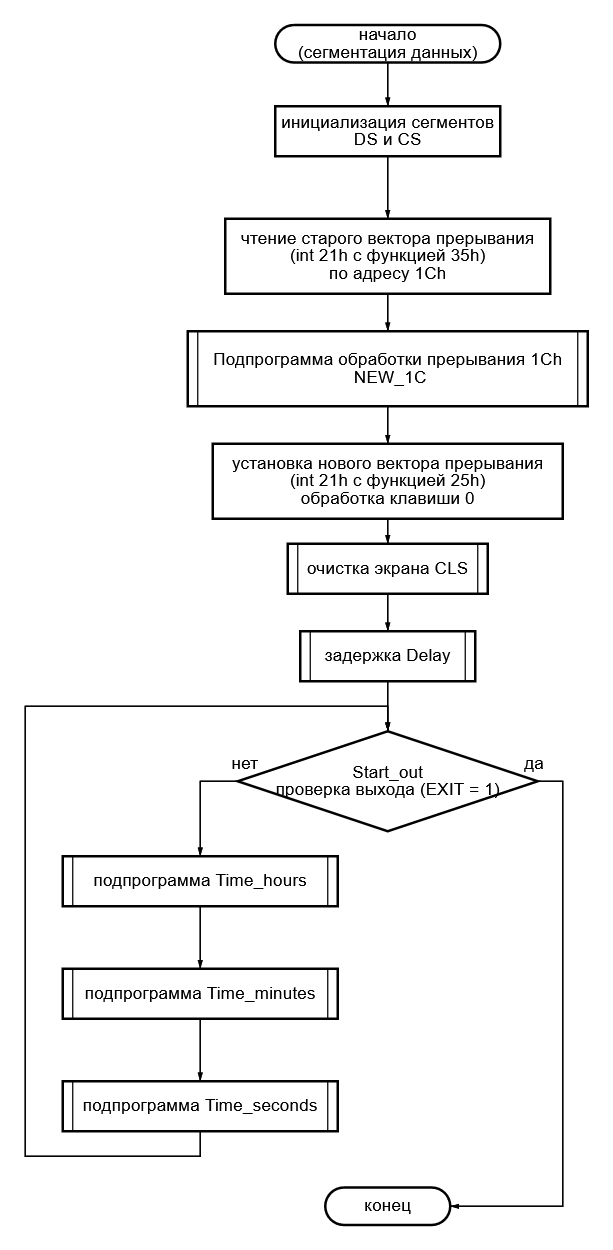
**Лабораторная работа №7**

**Тема**: «Обработка прерываний»  
**Цель работы**: Целью работы является изучить обработку прерываний в MS-DOS, изучить создание и применение обработчиков прерываний на языке ассемблера.

**Примечание:** Вариант 25  
Задание: Выводить ежесекундно в правом верхнем углу экрана системное время “часы:минуты:секунды”

**Ход работы. Блок-схема:**



**Код:**

data segment ; директива начала сегмента данных

EXIT db 0 ; признак завершения программы (не 0)

POS\_H1 dw 144 ; позиция начального вывода часов

POS\_H2 dw 146

POS\_M1 dw 150 ; позиция начального вывода минут

POS\_M2 dw 152

POS\_S1 dw 156 ; позиция начального вывода секунд

POS\_S2 dw 158

OLD\_CS dw ? ; адрес сегмента старого вектора 1Сh

OLD\_IP dw ? ; адрес смещения старого вектора 1Сh

data ends ; директива конца сегмента данных

code segment ; директива начала сегмента кодов

assume cs:code, ds:data ; установка соответствий сегментов и их адресов

; cs - указывает на код программы, ds - на данные

; Основная программа

START:

mov ax, data ; сегментация данных

mov ds, ax

; чтение вектора прерывания

mov ah, 35h

mov al, 1Ch

int 21h

mov OLD\_IP, bx

mov OLD\_CS, es

; установка вектора прерывания

push ds

mov dx, offset NEW\_1C

mov ax, seg NEW\_1C

mov ds, ax

mov ah, 25h

mov al, 1Ch

int 21h

pop ds ; сохраняем наш сегмент данных в стек

mov ax, 0B800h ; адрес сегмента буфера дисплея

mov es, ax

call CLS ; очистка экрана

call DELAY ; стартовая задержка

; цикл вывода времени на экране с определенной задержкой

Start\_out:

cmp EXIT, 1 ; если 1, то выходим

je quit

call Time\_hours ; вызов подпрограммы отображения часов

call Time\_minutes ; вызов подпрограммы отображения минут

call Time\_seconds ; вызов подпрограммы отображения секунд

jmp Start\_out ; зацикливание

; очистка экрана и завершение программы

quit:

call CLS ; очистка экрана

mov dx, OLD\_IP ; восстановление адреса

mov ax, OLD\_CS ; восстановление адреса

mov ds, ax

mov ah, 25h ; восстанавливаем первоначальный адрес

mov al, 1Ch

int 21h

mov ax, 4c00h ; Код завершения 0

int 21h ; Выход в DOS

; Подпрограмма обработки прерывания 1Сh

NEW\_1C proc far

push ax ; сохранить все регистры

push bx

push cx

push dx

push ds

push es

mov ax, data ; установить ds на сегмент данных

mov ds, ax ; основной программы

mov ax, 40h ; установить es на

mov es, ax ; сегмент данных bios

mov ax, es:[1ch] ; пишем в регистр голову буфера клавиатуры

mov bx, es:[1ah] ; в другой пишем хвост буфера клавиатуры

cmp bx , ax ; если они равны, то буфер пуст

jne Interrupt\_start ; если не равны, то переходим к метке

jmp back ; в ином случае переходим к восстановлению регистров

Interrupt\_start:

mov al, es:[bx] ; заносим в al символ находящийся в голове

mov es:[1ch], bx ; заносим в хвост символ, находящийся в голове

cmp al, 30h ; проверка на нажатие клавиши 0

jnz back

mov EXIT, 1 ; если клавиша равна 0, то меняем значение на 1

back:

pop es ; восстановление всех регистров

pop ds

pop dx

pop cx

pop bx

pop ax

iret ; возвращаемся в основную программу с восстановлением CS,IP и регистра флагов

NEW\_1C endp

; Подпрограмма очистки экрана

CLS proc near

push cx ; сохранить все регистры

push ax

push si

xor si, si

mov ah, 7

mov dl, ' '

mov cx, 2000 ;80\*25

;это столбцы и строки цветного видеобуффера,которые будут заполнены через цикл...

CL1:

mov es:[si], ax ;...нулями,собственно вот этой инструкцией

inc si

inc si

loop CL1 ; повторяем все это дело 2000 раз,ибо это число байт видеобуфера

pop si ; восстановление регистров

pop ax

pop cx

ret ; возвращаемся в основную программу с восстановлением CS,IP

CLS endp

; Подпрограмма задержки

DELAY proc near

push cx ; сохраняем регистр,так как его старое значение еще понадобится

mov cx, 18 ; в секунде 18,2 тика => задержка на 18 тиков

pop cx ; восстанавливаем регистр

ret ; возвращаемся в основную программу с восстановлением CS,IP

DELAY endp

; Подпрограммы вывода времени

Time\_hours proc near

push ax ; сохраняем регистры

push bx

push cx

mov ah,02h ; читать время из "постоянных" (CMOS) часов реального времени

int 1Ah

mov al,ch ; час в формате BCD

ror ax,1 ; циклический сдвиг вправо на 4 бита

ror ax,1

ror ax,1

ror ax,1

shr ah,1 ; логический сдвиг вправо на 4 бита

shr ah,1

shr ah,1

shr ah,1

or ax,'00' ; в общем эта конструкция(156-164) выполняет корректировку полученного ASCII-кода часов

mov bx, POS\_H1 ; указываем позицию

call DELAY ; производим задержку

mov es:[bx], al ; производим отображение на позиции символа

mov bx, POS\_H2 ; указываем позицию

call DELAY ; производим задержку

mov es:[bx], ah ; производим отображение на позиции символа

pop cx ; восстанавливаем регистры

pop bx

pop ax

ret ; возвращаемся в основную программу с восстановлением CS,IP

Time\_hours endp

Time\_minutes proc near

push ax ; сохраняем регистры

push bx

push cx

mov ah,02h ; читать время из "постоянных" (CMOS) часов реального времени

int 1Ah

mov al,cl ; минуты в формате BCD

ror ax,1 ; циклический сдвиг вправо на 4 бита

ror ax,1

ror ax,1

ror ax,1

shr ah,1 ; логический сдвиг вправо на 4 бита

shr ah,1

shr ah,1

shr ah,1

or ax,'00' ; в общем эта конструкция(188-196) выполняет корректировку полученного ASCII-кода минут

mov bx, POS\_M1 ; указываем позицию

call DELAY ; производим задержку

mov es:[bx], al ; производим отображение на позиции символа

mov bx, POS\_M2 ; указываем позицию

call DELAY ; производим задержку

mov es:[bx], ah ; производим отображение на позиции символа

pop cx ; восстанавливаем регистры

pop bx

pop ax

ret ; возвращаемся в основную программу с восстановлением CS,IP

Time\_minutes endp

Time\_seconds proc near

push ax ; сохраняем регистры

push bx

push dx

mov ah,02h ; читать время из "постоянных" (CMOS) часов реального времени

int 1Ah

mov al,dh ; секунды в формате BCD

ror ax,1 ; циклический сдвиг вправо на 4 бита

ror ax,1

ror ax,1

ror ax,1

shr ah,1 ; логический сдвиг вправо на 4 бита

shr ah,1

shr ah,1

shr ah,1

or ax,'00' ; в общем эта конструкция(220-228) выполняет корректировку полученного ASCII-кода секунд

mov bx, POS\_S1 ; указываем позицию

call DELAY ; производим задержку

mov es:[bx], al ; производим отображение на позиции символа

mov bx, POS\_S2 ; указываем позицию

call DELAY ; производим задержку

mov es:[bx], ah ; производим отображение на позиции символа

pop dx ; восстанавливаем регистры

pop bx

pop ax

ret ; возвращаемся в основную программу с восстановлением CS,IP

Time\_seconds endp

CODE ends

end START

**Вывод:** Изучил обработку прерываний в MS-DOS, создание и применение обработчиков прерываний на языке ассемблера.

**Контрольные вопросы**

**1. В чём суть концепции прерывания?**

Прерывание — одна из базовых концепций вычислительной техники, которая заключается в том, что при наступлении какого-либо события происходит передача управления специальной процедуре, называемой обработчиком прерываний (ISR, англ. Interrupt Service Routine). В отличие от условных и безусловных переходов, прерывание может быть вызвано в любом месте программы, в том числе если выполнение программы приостановлено, и обусловлено обычно внешними по отношению к программе событиями.

В MS-DOS различают аппаратные и программные прерывания. Первые возникают по запросу периферийных устройств. Вторые позволяют использовать предоставляемые системой MS-DOS и BIOS большой набор подпрограмм, выполняющих различные полезные действия и оформленные как программные прерывания.

**2. Как работает система прерывания по вектору?**

При возникновении прерывания микроконтроллер завершает текущую команду, сохраняет в стеке содержимое счетчика команд и совершает переход на адрес соответствующего вектора прерывания. По этому адресу, как правило, находится команда безусловного перехода к подпрограмме обработки прерывания.

**3. В чём отличие команд ret и iret?**

Каждая подпрограмма обработки прерывания, в отличие от обычной подпрограммы, завершается командой iret, которая похожа на команду ret, но восстанавливает из стека кроме CS и IP ещё и регистр флагов.

**4. Какие способы получения/изменения вектора прерывания Вы знаете?**

MS-DOS предоставляет две функции 35h и 25h прерывания 21h для чтения и установки вектора прерывания

Функция 35h  
Выполняет чтение адреса подпрограммы обработки прерывания.

• Вызов:

ah = 35h  
al = номер прерывания

• Возвращаемое значение:

es:bx – указатель на подпрограмму прерывания

Примечание. Функция получает адрес, указанного AL прерывания из таблицы векторов прерываний. BX содержит смещение, а ES сегмент адреса подпрограммы.

Функция 25h  
Выполняет занесение нового вектора прерывания.

• Вызов:

ah = 25h  
al = номер прерывания  
ds:dx – указатель на подпрограмму обработки прерывания

• Возвращаемое значение:

Нет

Примечание. Функция устанавливает адрес прерывания, указанного в al в таблицу векторов прерываний. dx содержит смещение, а ds сегмент устанавливаемой подпрограммы.

**5. Как вызвать программное прерывание?**

mov ah,[функция прерывания]  
int [номер прерывания]

**6. Какие существуют способы передачи параметров в подпрограмму обработки прерываний и возврата параметров из неё?**

 Простейший способ – передача параметров через регистры: основная программа записывает параметры в какие-либо регистры, а процедура по мере необходимости извлекает их из этих регистров и использует в своей работе.  
Удобнее передавать параметры через стек. В этом случае основная программа записывает параметры в стек и вызывает подпрограмму, подпрограмма работает с параметрами и, возвращая управление, очищает стек.  
Для работы с подпрограммами в систему команд процессора включены специальные команды, это вызов подпрограммы CALL и возврат управления RET.

**7. Какие действия производит процессор при получении запроса на прерывание?**Действия при обработке прерываний выполняются независимо от того вызвано ли оно аппаратно или программно. При получении сигнала на прерывание (при аппаратном прерывании от программируемого контроллера прерываний, при программном командой процессора **int**) процессор содержимое CS, IP и регистра флагов сохраняет в стеке. В CS и IP помещается адрес подпрограммы обработки прерывания, которая и выполняется, после чего восстанавливаются из стека содержимое CS, IP и регистра флагов, и процессор продолжает выполнение программы.

**8. В чём отличие команд call и int?**Call вызывает подпрограмму, int вызывает прерывание.