

Санкт-Петербургский государственный политехнический университет  
Институт компьютерных наук и технологий  
**Кафедра «Информационные и управляющие системы»**

## **КУРСОВАЯ РАБОТА**

Программирование на ассемблере

По дисциплине «Архитектура ЭВМ. Часть 1»

Выполнил студент гр. 23534/1

Стойкоски. Н.С.

Руководитель

проф. д.т.н.

С.А. Молодяков

Санкт-Петербург  
2017

## Оглавление

Введение.....	3
Программа 1.....	4
Блок схема .....	4
Список использованных прерываний BIOS.....	6
Текст программы .....	7
Скриншоты .....	12
Программа 2.....	13
Блок схема .....	13
Используемые устройства.....	14
Список использованных прерываний BIOS.....	15
Текст программы .....	16

# Введение

Ассемблер - низкоуровневый машинно-ориентированный язык программирования. Реализация языка зависит от типа процессора и определяется архитектурой вычислительной системы. Ассемблер позволяет напрямую работать с аппаратурой компьютера. Программа на языке ассемблера включает в себя набор команд, которые после трансляции преобразуются в машинные команды.

## Программа 1

Организовать печатание произвольного текстового файла на экране. Вертикальный размер окна (количество строк) может меняться. Для печатания желательно использовать PgUP и PgDOWN.

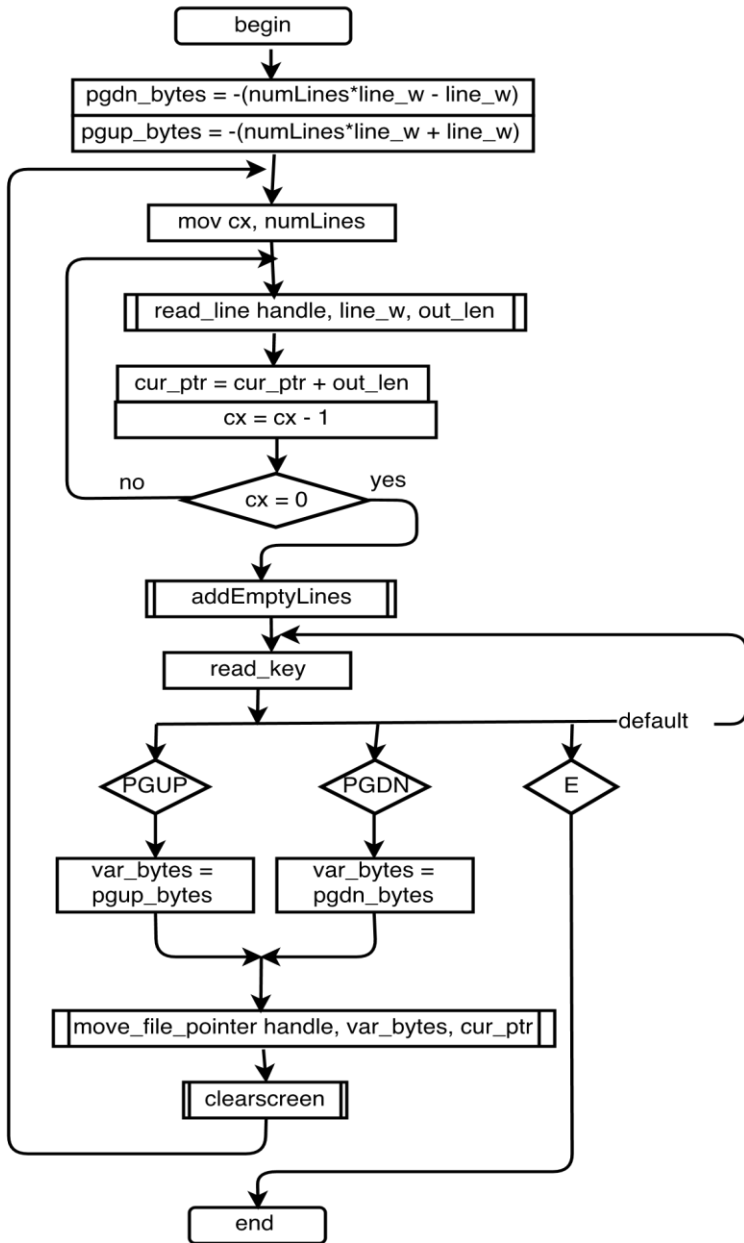
## Программа 2

Музыкальный инструмент. Напишите программу в которой имитируется музыкальный инструмент. Запоминается последовательность нажатия на кнопки клавиатуры и затем проигрывается звуковой фрагмент. Мелодия зависит от последовательности нажатых кнопок.

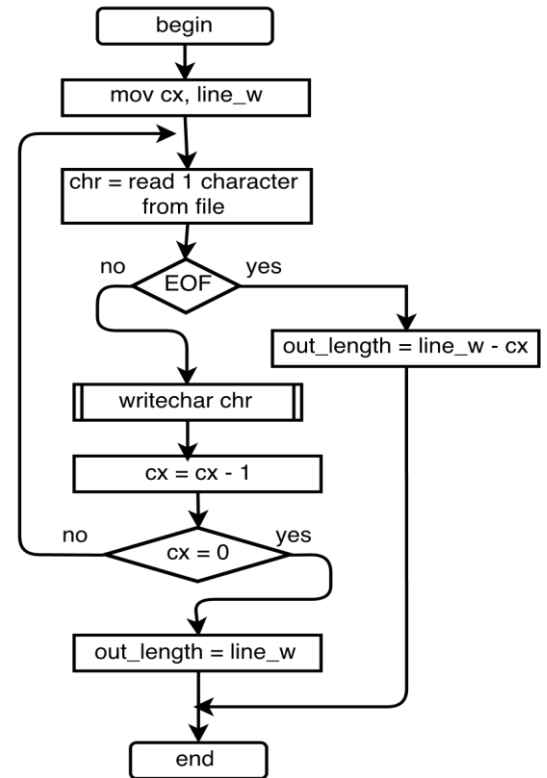
# Программа 1

## Блок схема

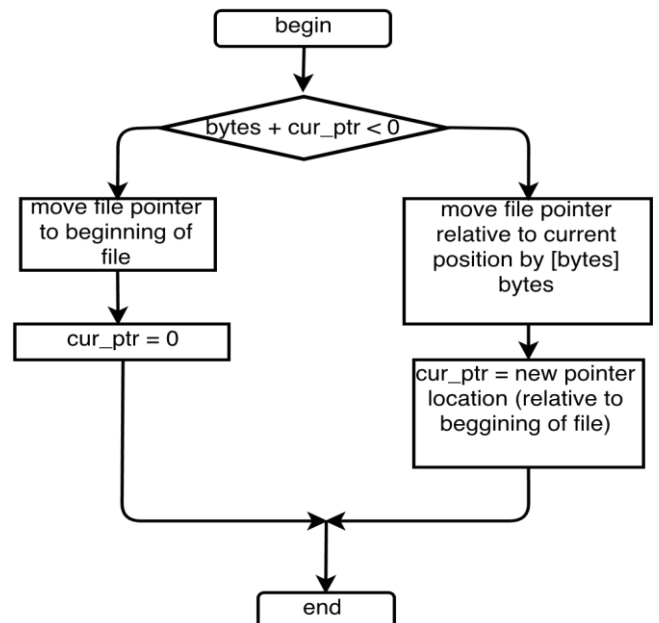
**read\_file( handle, line\_w, numLines)**



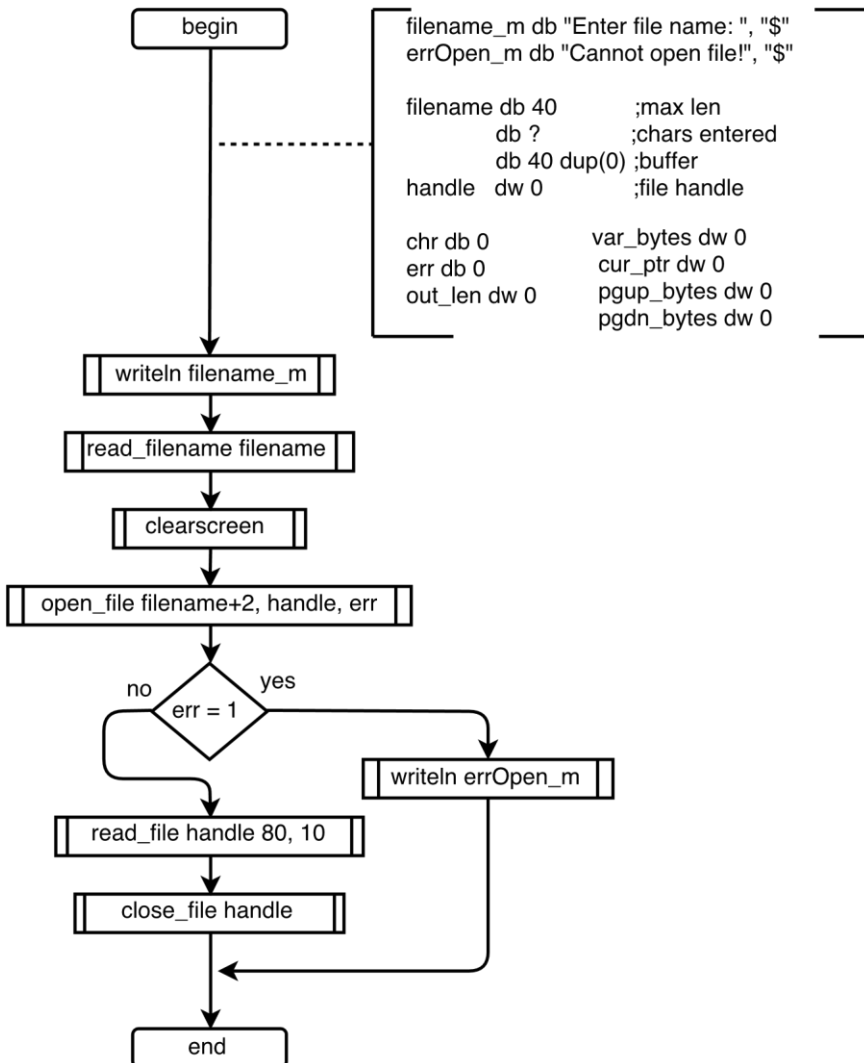
**read\_line ( handle, line\_w, out\_length)**



**move\_file\_pointer ( handle, bytes, cur\_ptr )**



## main program



## additional macros

**writeChar MACRO chr**  
;output character chr

**newline MACRO**  
;output newline

**addEmptyLines MACRO**  
;output 10 empty lines

**clearscreen MACRO**  
;clear the console window

**write MACRO msg**  
;output the string msg

**writeln MACRO msg**  
;output the string msg  
;and make a newline

**read\_filename MACRO filename**  
;read a filename from input

**open\_file MACRO filename, handle, err**  
;open the file with full filepath filename, and  
;get the file handle. If the file cannot be opened  
;then make err = 1

**close\_file MACRO handle**  
;close the file

## Список использованных прерываний BIOS

### INT 21h – Main DOS API

AH	Description
02h	Character output
09h	Display string
0Ah	Buffered keyboard input
3Dh	Open file
3Eh	Close file
3Fh	Read file or device
42h	Move file pointer
4C00h	Terminate with return code 0

### INT 16h – BIOS keyboard services

AH	Description
00h	Read character

## Текст программы

```
DOSSEG
.model small;

.stack 100H

writeChar MACRO chr ;печатает символ chr на экране
    mov dl, chr
    mov ah, 02h
    int 21h
ENDM
newline MACRO ; переход на новая строка
    writeChar 10 ;newline char
    writeChar 13 ;return char
ENDM
addEmptyLines MACRO ;печатает 10 пустых строк
    LOCAL L1 ;используется после печатание
    mov cx, 10 ;вывода текста, чтобы поставить
    L1: ;текст в середину экране
        newline
    loop L1
ENDM
clearscreen MACRO ;очищает экран путем
    LOCAL L1 ;вывода много пустых строк
    mov cx, 50
    L1:
        newline
    loop L1
ENDM
write MACRO msg ;печатает строка msg на экране
    mov dx, offset msg
    mov ah, 09h
    int 21h
ENDM
writeln MACRO msg ;печатает строка msg на экране и
    write msg ;переходит на следующая строка
    newline
ENDM
read_filename MACRO filename ;читает имя файла из
    ;стандартного ввода
    mov dx, offset filename ;и записывает результат
    mov ah, 0Ah ;в filename
    int 21h

;прочитанная строка заканчивается с символом перевод строки
;его надо заменить с chr(0) для правильная работа с прерывание "open file"
mov si, offset filename + 1 ;берем поле которое хранит длину строки
mov cl, [si] ;записываем это значение в cx
mov ch, 0
inc cx ;получаем позицию требуемый символ
add si, cx ;сейчас si указывает на этот символ в строку
mov al, 0
mov [si], al ;заменяем на chr(0)
newline;
```

ENDM

**open\_file MACRO filename, handle, err** ;макро для открытие файла  
;filename – входной параметр, в результате получаем "file handle" в handle,  
;а если не удалось открыть файл, тогда записывается '1' в err

```
mov ah, 3Dh          ;открываем файл через прерывание "Open file"
mov dx, offset filename ;в dx ожидает имя файла,
mov al, 0            ;al – флаги, в нашем случае al=0 – открываем
int 21h              ;файл только для чтения
jc open_error        ;если возникла ошибка тогда переходим на open_error
```

```
mov handle, ax        ;в результате мы получили "file handle" в ax,
                      ;перезаписываем его в выходного параметра handle
jmp openfile_end
```

```
open_error: ;ошибка при открытие файла, ставим '1' в err
    mov err, 1
openfile_end:
```

ENDM

**close\_file MACRO handle** ;макро для закрытия файла

```
mov bx, handle
mov ah, 3Eh
int 21h
```

ENDM

**move\_file\_pointer MACRO handle, bytes, cur\_ptr** ;макро для сдвига файлового  
;указателя на bytes байты (bytes может быть как и положительно так и  
;отрицательное число), принимается дополнительный параметр cur\_ptr –  
;текущий указатель файла для проверки диапазона, является входной и выходной  
;параметр

```
mov ax, cur_ptr      ;проверка диапазона
mov bx, bytes         ;если (bytes + cur_ptr < 0 ) тогда надо сдвинуть файловой
add ax, bx            ;указатель на самого начало файла, а не перед того
cmp ax, 0
jnl move_to_begin
```

```
mov cx, 0             ;[cx:dx] bytes to move (signed)
mov dx, bytes
cmp dx, 0
jnl negative_dir
jmp move_ptr
```

```
negative_dir:
    mov cx, -1 ; так как прерывание "move file pointer" ожидает количество
;байтов в [cx:dx], тогда если у нас dx – отрицательное число, тогда надо
;присвоить 1 к cx, чтобы [cx:dx] соответствовало нашему dx
```

```
move_ptr:
    mov ah, 42h
    mov al, 1 ; сдвиг указателя относительно текущей позиции
    mov bx, handle
    int 21h
    mov cur_ptr, ax ; новая позиция файлового указателя (в байтах от
                    ; самого начала файла)
    jmp move_file_pointer_end ;переход на конец макроса
```



```

move_to_begin:
    mov ah, 42h
    mov al, 0 ;сдвиг относительно начало файла
    mov bx, handle
    mov cx, 0 ;делаем [cx:dx]=0 так чтобы остался на самого начало файла
    mov dx, 0
    int 21h;
    mov cur_ptr, ax

    move_file_pointer_end:
ENDM

read_line MACRO handle, line_w, out_length ;макро для чтение 1 строка и
;ее вывод на экране, line_w - это константный входной параметр который дает
;максимальное число символов которые вмещаются в одна строка на экране
;out_length - это выходной параметр, где записываем сколько символов мы
;успели прочитать, обычно оно равно line_w, но при обнаружение EOF будет
;меньше
mov cx, line_w
    read_line_loop: ; цикл в котором посимвольно читаем строка
        push cx

        mov ah, 3Fh
        mov bx, handle
        mov cx, 1
        lea dx, chr ;чтение одного символа с файла через прерывание "Read file
        int 21h ; or device" и его запись в буфер chr

        cmp ax, cx ; если ax=cx тогда мы достигли EOF
        j1 end_of_file

        writeChar chr ; печатаем полученный символ на экране

        pop cx
    loop read_line_loop

    mov out_length, line_w ; цикл в полностью выполнен, записываем line_w в
;out_length, и переходим на конца макроса
    jmp read_line_end

end_of_file: ; обнаружен EOF
    pop cx
    mov ax, line_w
    sub ax, cx
    mov out_length, ax ;в Out_length записываем (line_w - cx) т.е. сколько
; всего символы мы успели прочитать

    read_line_end:
ENDM

read_file MACRO handle, line_w, numLines ;головное макро которое с помощью
;остальник макросов, выводит содержимое файла на экране, переходит на строка
;выше или ниже при нажатия кнопки PGUP/PGDN, завершается при нажатия 'e'
; line_w - это константный входной параметр который дает ;максимальное число
символов которые вмещаются в одна строка на экране
;numLines - вертикальный размер окна (количество строк)

```

```

;handle - файловой handle

mov al, numLines ;вычисляем сколько символов содержит 1 блок
mov ah, line_w   ; (numLines * line_w)
mul ah
push ax          ; записываем результат в стеке для дальнейшего вычисления
                ; количество байтов при сдвиг вверх/вниз (pgup/pgdn)

mov bx, line_w
sub ax, bx
mov bx, -1       ;вычисляем на сколько байтов надо передвинуть файловой
imul bx          ;указать при ситуации когда нажата кнопка PGDN
mov pgdn_bytes, ax ; pgdn_bytes = -(numLines*line_w - line_w)

pop ax
mov bx, line_w
add ax, bx
mov bx, -1       ; вычисляем на сколько байтов надо передвинуть файловой
imul bx          ; указать при ситуации когда нажата кнопка PGUP
mov pgup_bytes, ax ;pgup_bytes = -(numLines * line_w + line_w)

read_loop:

mov cx, numLines ; нам надо вывести numLines строк
readlines_loop: ; цикл который печатает numLines строк из файла
push cx         ; и отслеживает текущую позицию в файла
read_line handle, line_w, out_len

mov ax, cur_ptr
mov bx, out_len
add ax, bx
mov cur_ptr, ax ; cur_ptr = cur_ptr + out_len
                ; к текущая позиция прибавляем количество
                ; прочитанные символы

pop cx
loop readlines_loop

addEmptyLines    ; печатаем пустые строки, чтобы текст оказался в
                ; середине экрана

readKeyLoop:     ; ждет пока не нажата одна из принятых клавиш, и переходит
                ; на соответствующее место для выполнение операции

mov ah, 00h
int 16h ;get keystroke

cmp ah, 81 ; pgdn-81
je move_page_dn

cmp ah, 73 ; pgup-73
je move_page_up

cmp al, 'e'
je readfile_end

jmp readKeyLoop

move_page_up:    ;если нажата клавиша PGUP тогда var_bytes = pgup_bytes
mov ax, pgup_bytes ; и переходим на move_fptr

```

```

    mov var_bytes, ax
    jmp move_fptr
move_page_dn:    ;если нажата клавиша PGUP тогда var_bytes = pgdn_bytes
    mov ax, pgdn_bytes    ; и переходим на move_fptr
    mov var_bytes, ax
    jmp move_fptr

move_fptr:
    move_file_pointer handle, var_bytes, cur_ptr ; выполняем макро сдвига
                                                ; файлового указателя на var_bytes байтов
    clearscreen    ; очищаем экран
    jmp read_loop ; идем назад в головного цикла

readfile_end:
ENDM
.data
;сообщения для пользователя
filename_m db "Enter file name: ", "$"
errOpen_m db "Cannot open file!", "$"

;файловые переменные
filename db 40          ;максимальное число символов
          db ?          ;количество введенных символов
          db 40 dup(0) ;буфер
handle dw 0             ;"file handle"

;переменные
chr db 0
err db 0
out_len dw 0
var_bytes dw 0
cur_ptr dw 0
pgup_bytes dw 0
pgdn_bytes dw 0

.code
    mov ax, @data
    mov ds, ax
    mov es, ax

    writeln filename_m    ; печатаем сообщение "Enter file name:"
    read_filename filename ; читаем имя файла

    clearscreen           ; очищаем экран

    open_file filename + 2, handle, err ; открываем файл filename

    mov al, err
    cmp al, 1
    jne main_program_readfile
    writeln errOpen_m     ; если была ошибка при открытие файла тогда
    jmp main_program_end  ; печатаем сообщение об ошибки и выходим из
                          ; программу

```

```
main_program_readfile:
```

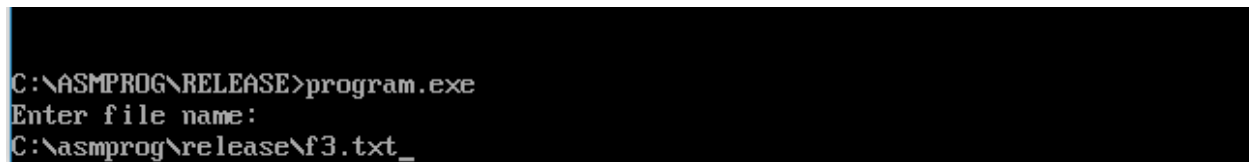
```
    read_file handle, 80, 10 ;вызываем макро для чтение файла, задаем  
;параметры 80 - максимальное число символов которые вмещаются в одна строка  
;на экране, 10 - вертикальный размер окна
```

```
    close_file handle ; закрываем файл
```

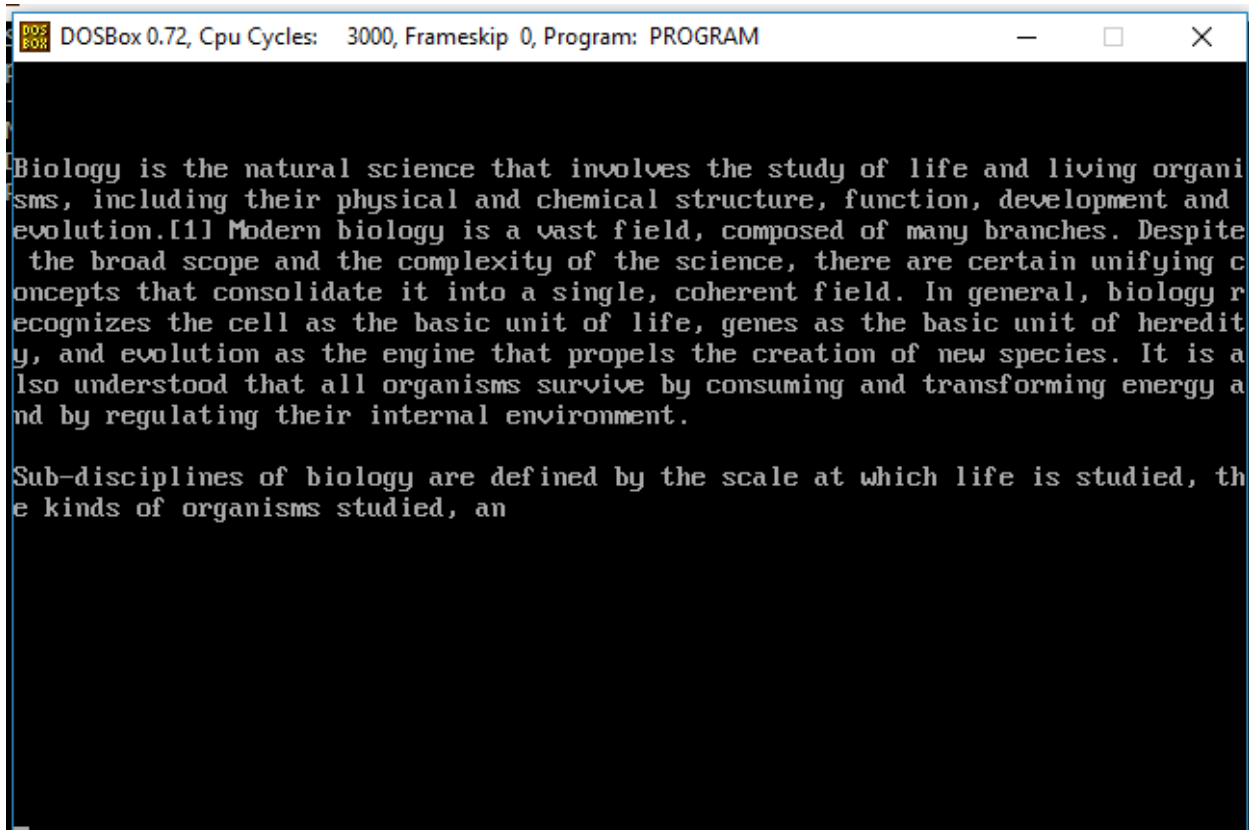
```
main_program_end:
```

```
    mov     ax, 4c00h  
    int     21h          ; завершение программы  
end
```

## Скриншоты



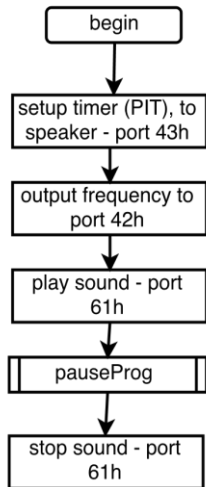
```
C:\ASMPROG\RELEASE>program.exe  
Enter file name:  
C:\asmprog\release\3.txt_
```



# Программа 2

## Блок схема

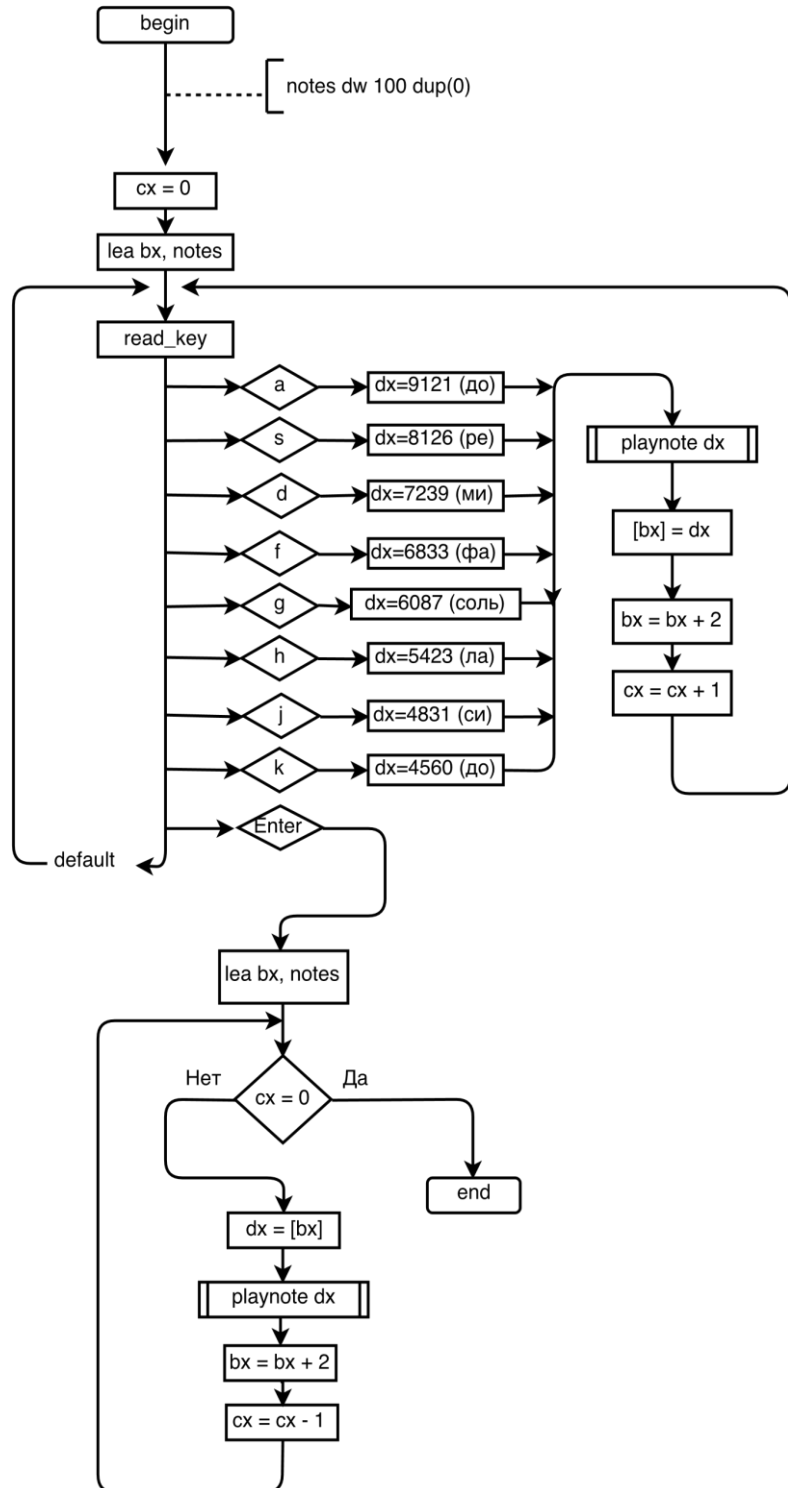
### playnote (frequency)



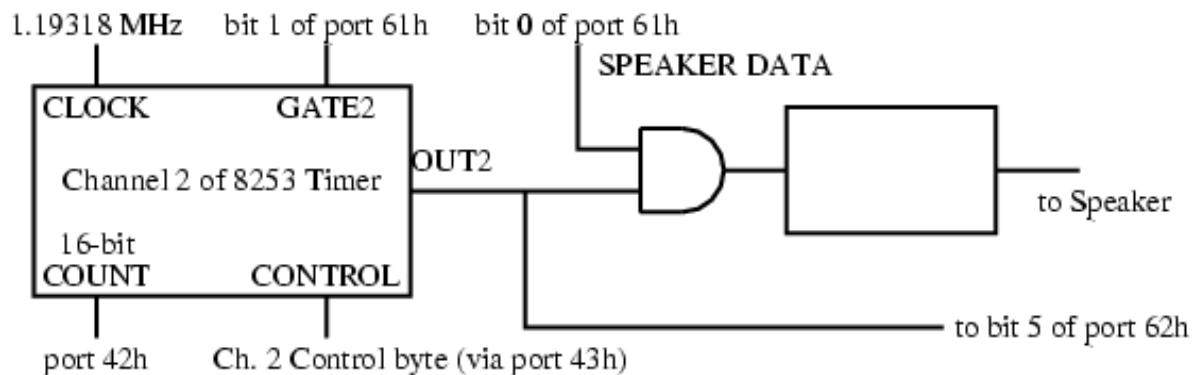
### additional macros

**pauseProg** MACRO length  
 ; pauses the program for  
 ; bx\*length decrement operations

### main program



## Используемые устройства



- Port 43h – Programmable Interval Timer (PIT)

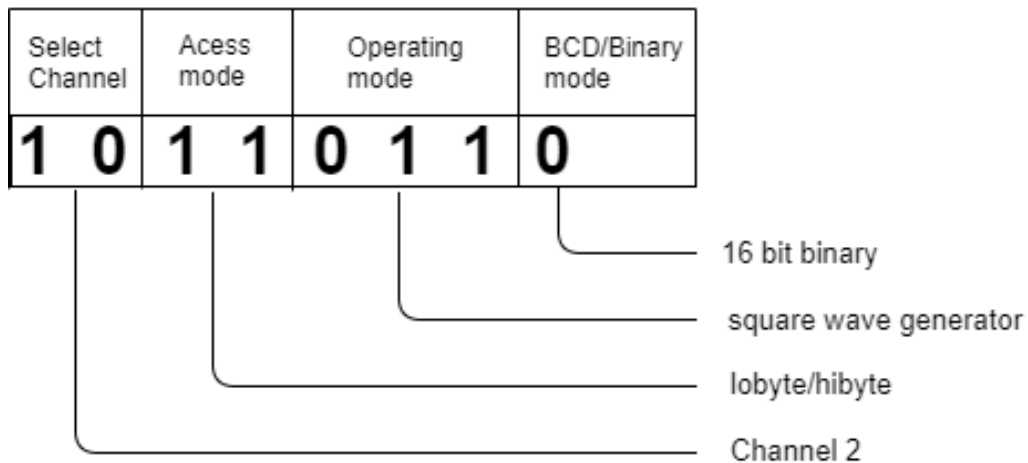
The Mode/Command register at I/O address 0x43 contains the following:

Bits	Usage
6 and 7	Select channel : 0 0 = Channel 0 0 1 = Channel 1 1 0 = Channel 2 1 1 = Read-back command (8254 only)
4 and 5	Access mode : 0 0 = Latch count value command 0 1 = Access mode: lobyte only 1 0 = Access mode: hibyte only 1 1 = Access mode: lobyte/hibyte
1 to 3	Operating mode : 0 0 0 = Mode 0 (interrupt on terminal count) 0 0 1 = Mode 1 (hardware re-triggerable one-shot) 0 1 0 = Mode 2 (rate generator) 0 1 1 = Mode 3 (square wave generator) 1 0 0 = Mode 4 (software triggered strobe) 1 0 1 = Mode 5 (hardware triggered strobe) 1 1 0 = Mode 2 (rate generator, same as 010b) 1 1 1 = Mode 3 (square wave generator, same as 011b)
0	BCD/Binary mode: 0 = 16-bit binary, 1 = four-digit BCD

Исходя из схема, для генерации звука нам надо подключить PIT к динамику, для этого нам надо установить биты 6, 7 соответственно на 1, 0 (канал 2). Это делается через байт управления который отправим на порт 43h

Полное описание выбранного байта управления:

Control byte: **0B6h**



- Port 61h – System Speaker

Бит 0 порта 61h выбирает источник для динамика (0 = manual, 1 = PIT).

Бит 1 порта 61h – это enable бит (выключатель) динамика.

Соответственно будем использовать эти 2 бита для включения и выключения звука.

## Список использованных прерываний BIOS

### INT 21h – Main DOS API

AH	Description
4c00h	Terminate with return code 0

### INT 16h – BIOS keyboard services

AH	Description
00h	Read character

## Текст программы

```
DOSSEG
.model small;

.stack 500H

pauseProg MACRO length ;макро для задержки/ остановки программы
    LOCAL pause1          ;входной параметр length - сколько раз будет
                           ;выполняться внутренний цикл, bx - внешний цикл

    LOCAL pause2

    pause1:                ; внешний цикл
    mov cx, length
    pause2:                ; внутренний цикл
    dec cx
    jnz pause2             ; пока cx не станет равным нулю
    dec bx
    jnz pause1             ; пока bx не станет равным нулю

ENDM

playnote MACRO frequency ; макро которое воспроизводит тон на динамике
                           ; на фреквенции frequency

    push ax
    push bx
    push cx

    mov al, 0B6h
    out 43h, al           ; задаем контрольный байт

    ;задаем данная фреквенция
    mov ax, frequency
    out 42h, al           ; устанавливаем lobyte
    mov al, ah
    out 42h, al           ; устанавливаем hibyte

    in al, 61h
    or al, 00000011b
    out 61h, al           ; включаем биты 0, 1 (включаем динамик и получаем звук)

    mov bx, 25
    pauseProg 10000       ; задержка

    in al, 61h
    and al, 11111100b
    out 61h, al           ; выключаем биты 0, 1 (выключаем звук)

    pop cx
    pop bx
    pop ax

ENDM
```



```

.data
notes dw 100 dup(0) ; последовательность нажатых кнопок (фреквенции)

.code
mov ax, @data
mov ds, ax

lea bx, notes
mov cx, 0 ; количество сыгранных нот
jmp afterAddNote

read_key_loop:

addNote:
    playnote dx ; играем ноту dx
    mov [bx], dx ; записываем значение в массив
    add bx, 2 ; переход на следующая ячейка
    inc cx ; добавляем 1 к количеству сыгранных нот

afterAddNote:

mov ah, 00h ;
int 16h ; читаем нажатый клавиш

cmp al, 'a'
je note1 ; до
cmp al, 's'
je note2 ; ре
cmp al, 'd'
je note3 ; ми
cmp al, 'f'
je note4 ; фа
cmp al, 'g'
je note5 ; соль
cmp al, 'h'
je note6 ; ла
cmp al, 'j'
je note7 ; си
cmp al, 'k'
je note8 ; до
cmp ah, 28 ; если нажата кнопка Enter
je play_music ; выходим из цикл

jmp afterAddNote

; к dx присваиваем соответствующая фреквенция
note1:
    mov dx, 9121
    jmp read_key_loop
note2:
    mov dx, 8126
    jmp read_key_loop
note3:
    mov dx, 7239
    jmp read_key_loop
note4:
    mov dx, 6833

```

```

        jmp read_key_loop
note5:
    mov dx, 6087
        jmp read_key_loop
note6:
    mov dx, 5423
        jmp read_key_loop
note7:
    mov dx, 4831
        jmp read_key_loop
note8:
    mov dx, 4560
        jmp read_key_loop

```

Note	Frequency	Frequency #
C	130.81	9121
C#	138.59	8609
D	146.83	8126
D#	155.56	7670
E	164.81	7239
F	174.61	6833
F#	185.00	6449
G	196.00	6087
G#	207.65	5746
A	220.00	5423
A#	233.08	5119
B	246.94	4831
Middle C	261.63	4560

```

play_music:

```

```

    lea bx, notes

```

```

playSounds:    ; воспроизводим последовательность нажатых кнопок
    mov dx, [bx] ; читаем значение ячейки в dx
    add bx, 2    ; переход на следующая ячейка
    playnote dx  ; воспроизводим тон
loop playSounds ; пока cx не станет равным нулю

```

```

    mov ax, 4c00h

```

```

    int 21h      ; завершение программы

```

```

end

```