МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра вычислительные системы и технологии

Лабораторная работа № 3

Разработка программы ввода-вывода и обработки числовой информации Вариант № 8

ОТЧЕТ

по лабораторной работе

по дисциплине

Принципы и методы  
организации системных программных средств

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Викулова Е.Н.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Игнаков К. М.

19-В-2

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2021

**Цель работы**

Разработать программу для ввода, вывода и обработки данных в массиве с целочисленными данными различных знаков. Научиться проверять корректность работы программы в ходе её выполнения с помощью проверок на переполнение, деления на нуль и прочие. Следует научиться в ходе работы использовать макросы и процедуры для увеличения скорости разработки.

**Задание**

Написать программу, выполняющую:

* ввод массива целых многозначных десятичных чисел со знаком;
* арифметическую и логическую обработку массива чисел (по вариантам);
* преобразование числового результата в ASCII-строку и отображение результата на экране.

В программе должен быть предусмотрен контроль ситуаций переполнения и деления на 0 с выводом на экран соответствующих сообщений.

**Вариант:** вычислить сумму, произведение и среднее введенных чисел, вычислить сколько процентов из введенных чисел – положительные.

**Структура программы**

Разработана программа является многосегментной и имеет расширение exe. В программе имеет 3 сегмента:

1. Сегмент данных:
   1. in\_str label byte ; String (only for 6 chars)
   2. razmer db 7 ; Buffer (for 6 chars)
   3. kol db (?) ; Count of input chars
   4. stroka db 7 dup (?) ; Buffer input value
   5. number dw 5 dup (0) ; Array of values
   6. sumres dw 0 ; Sum of values in the array
   7. mulres dw 0 ; Mul of values in the array
   8. sredres dw 0 ; Avarage value of values in the array
   9. prochres dw 0 ; Percent of positive value of values in the array
   10. siz dw 5 ; Lenght of the array
   11. Остальные данные для вывода сообщений в консоль
2. Сегмент кода:
   1. DIAPAZON – проверка диапазона вводимых чисел
   2. DOPUST – проверка допустимости вводимых символов
   3. AscToBin – перевод ASCII кодов чисел в двоичное число
   4. BinToAsc – обратная AscToBin
3. Стек: dw 100 dup (?)

Для решения задачи также были использованы некоторые макросы:

* Print – выводит строку в консоль
* Input – “читает” данные из консоли

**Алгоритм обработки введенных значений.**

Для упрощения обработки ситуации выхода за пределы допустимого диапазона было решено ограничить диапазон от -29999 до 29999. При такой обратке нам необходимо контролировать чтобы было введено 5 символом и первым числовым символом является цифра 2

; Data in (-29999, +29999)

; Args:

; Buffer input - stroka

; Res:

; bh - error flag

DIAPAZON PROC

xor bh, bh

xor si, si

cmp kol, 05h ; Count is 5

jb dop

cmp stroka, 2dh ; If enter 5 and more, check for `-`

jne plus ; If first is not `-` - check count of numbers

cmp kol, 06h ; If first `-` and values less 6

jb dop

inc si ; Chech first value

jmp first

plus:

cmp kol, 6 ; If count of data is 6 and first is not `-`

je error1 ; ERROR

first:

cmp stroka[si], 32h ; Check first simbol with `2`

jna dop ; If first simbol <= '2' - check the validity of the characters

error1:

mov bh, flag\_err ; Else bh = flag\_err

dop:

ret

DIAPAZON ENDP

**Проверка на вхождение в диапазон.**

Если строка имеет 5 символов и первый числовой символ это 2, то необходимо проверить все оставшиеся символы на вхождение в числовой диапазон. Для этого мы убеждаемся, что ASCII коды введенных символов принадлежат промежутку от 30h до 39h.

; Chech data is value or `-`

; Args:

; Buffer input - stroka

; si - number of simbol

; Res:

; bh - error flag

DOPUST PROC

xor bh, bh

xor si, si

xor ah, ah

xor ch, ch

mov cl, kol ; Count of input chars

m11:

mov al, [stroka + si] ; In `al` - first

cmp al, 2dh ; If first char is `-`

jne testdop ; If no

cmp si, 00h ; If yes

jne error2 ; If another - ERROR

jmp m13

testdop:

cmp al, 30h ; Data in [30h, 39h]

jb error2

cmp al, 39h

ja error2

m13:

inc si

loop m11

jmp m14

error2:

mov bh, flag\_err ; Another error

m14:

ret

DOPUST ENDP

**Перевод строки в число и наоборот**

Для этого из кода каждой цифры вычитаем 30h, умножаем результат на вес разряда и складываем полученные произведения. Отрицательные числа представляем в дополнительном коде при помощи команды neg. Для вывода полученных результатов переводим числа в их ASCII коды, т.е. выполняем обратную операцию.

; ASCII to number

; Args:

; cx - Count of input simbols

; bx - number of simbol from last

; Res:

; number - buffer numbers

; di - number of value in the array

AscToBin PROC

xor ch, ch

mov cl, kol

xor bh, bh

mov bl, cl

dec bl

mov si, 01h ; In si, the weight of the discharge

n1:

mov al, [stroka + bx]

xor ah, ah

cmp al, 2dh ; Check positive or negative

je otr ; if negative

sub al, 30h

mul si

add [number + di], ax

mov ax, si

mov si, 10

mul si

mov si, ax

dec bx

loop n1

jmp n2

otr:

neg [number + di] ; Show negative in `adding` code

n2:

ret

AscToBin ENDP

; Number to ASCII

; Args

; ax - number

; Res

; out\_str - numbers buffer

BinToAsc PROC

xor si, si

add si, 05h

mov bx, 0Ah

push ax

cmp ax, 00h

jnl mm1

neg ax

mm1:

cwd

idiv bx

add dl,30h

mov [out\_str + si], dl

dec si

cmp ax, 00h

jne mm1

pop ax

cmp ax, 00h

jge mm2

mov [out\_str + si], 2dh

mm2:

ret

BinToAsc ENDP

**Алгоритмы:**

1. – сумма элементов

Начало:

Сумма = 0

Цикл по элементам массива:

Сумма += элемент

Конец.

Результат при суммировании записывается в один регистр, значит переполнение будет происходить, если результатом будет двойное слово.

; Sum of elements

searchSum:

mov cx, siz ; cx - array size

mov si, offset number

searchLoop:

mov ax, [si]

addToRes:

add sumres, ax

jo overFlowErrSum ; Переход, если есть переполнение

nextVal:

inc si

inc si

loop searchLoop

1. – произведение элементов

Начало:

Произведение = 1

Цикл по элементам массива:

Произведение \*= элемент

Конец.

Переполнение при умножении будет возникать при цикличном умножении чисел, переполнение при умножении возникает, если одно из чисел это двойное слово. Если перемножаются два максимальных числа и их результат двойное слово, то результат убирается в регистровую пару.

; Multiply of elements

searchNegMul:

mov cx, siz

mov si, offset number

mov ax, 1 ; ax используется как первое частное, если поставим 1, то просто получим само число

minusEl:

mov bx, [si]

imul bx ; multiply

jo overFlowErr ; OF устанавливаются, когда значащие биты переносятся в верхнюю половину результата, и очищаются, когда результат точно соответствует нижней половине результата

plusEl:

inc si

inc si

loop minusEl

mov mulres, ax ; mow answer to variable

1. – среднее значение

Начало:

Среднее = Сумма / размер массива

Конец.

; Average value of elements

searchSrednee:

mov ax, sumres

cwd

idiv siz

mov sredres, ax

1. – процент положительных

Начало:

Процент = 0

Цикл по элементам массива:

Если элемент > 0:

Процент += 100

Процент = Процент / размер массива

Конец.

Формула для нахождения:

; Percent of positive elements

; Formula (count \* 100) / SIZE %

searchProchent:

mov cx, siz

mov si, offset number

mov ax, 0h

findPosElem:

mov bx, [si]

cmp bx, 0h

jl foundNegElem ; If value is negative - skip

add ax, 100d ; 1 peace \* 100 - check formula

jo overFlowErrSum

foundNegElem:

inc si

inc si

loop findPosElem

cwd

idiv siz

mov prochres, ax

jmp resOutput

**Листинг**

; Вычислить сумму, произведение и среднее введенных чисел, вычислить сколько процентов из введенных чисел - положительные

; Macros for output data

print macro srting

push ax

push dx

mov dx, offset srting

mov ah, 09h

int 21h

pop dx

pop ax

endm

; Macros for input data

input macro srting

push ax

push dx

mov dx, offset srting

mov ah, 0Ah

int 21h

pop dx

pop ax

endm

d1 SEGMENT para public 'data'

in\_str label byte ; String (only for 6 chars)

razmer db 7 ; Buffer (for 6 chars)

kol db (?) ; Count of input chars

stroka db 7 dup (?) ; Buffer input value

number dw 5 dup (0) ; Array of values

sumres dw 0 ; Sum of values in the array

mulres dw 0 ; Mul of values in the array

sredres dw 0 ; Avarage value of values in the array

prochres dw 0 ; Percent of positive value of values in the array

siz dw 5 ; Lenght of the array

lab3 db 'Laboratory work #3','$'

author db 13,10, 'Ignakov Konstantin 19-V-2'

perevod db 10,13,'$'

inputError db 'Input Error!', 10,10,'$'

divByZeroError db 'Divition by zero!', 10,10,'$'

overflow db 'Overflow!', 10,10,'$'

overflowSum db 'Overflow Sum!', 10,10,'$'

sumText db 13,10, 'Sum of elements: ','$'

mulText db 13,10, 'Mul of elements: ','$'

sredText db 13,10, 'Srednee: ','$'

prochText db 13,10, 'Prochent: ','$'

maxElText db 13,10, 'Max element: ','$'

minElText db 13,10, 'Min element: ','$'

out\_str db 6 dup (' '),'$'

enter\_please db 'Input number from -29999 to 29999: $'

flag\_err equ 1

d1 ENDS

st1 SEGMENT para stack 'stack'

dw 100 dup (?)

st1 ENDS

; Entry point

c1 SEGMENT para public 'code'

ASSUME cs:c1, ds:d1, ss:st1

start:

mov ax, d1

mov ds, ax

mov ax, 03h ; Installing text video mode, clearing the screen

int 10h ; ah=0 (number of function), al=3 (number of mode)

print lab3 ; Print work info

print author ; Print author

print perevod ; Print `\n`

; Input data with Error Flag

xor di, di ; di - Number of an element in the array

mov cx, siz ; cx - Lenght of the array

inputVal:

push cx

check:

print enter\_please ; Print info for input data

input in\_str ; Input data like string

print perevod

call diapazon ; Chech that value in (-29999, 29999)

cmp bh, flag\_err ; Compare `BH` and `FLAG\_ERROR`

je inErr ; If `BH` == 1: smth went wrong

call dopust ; Проверка допустимости вводимых символов

cmp bh, flag\_err ; Сравним bh и flag\_err

je inErr ; Если равен 1 сообщение об ошибке ввода

call AscToBin ; Преобразование строки в число

inc di

inc di

pop cx

loop inputVal

jmp searchSum

inErr:

print inputError

jmp check

; Sun of elements

searchSum:

mov cx, siz ; cx - array size

mov si, offset number

searchLoop:

mov ax, [si]

addToRes:

add sumres, ax

jo overFlowErrSum

nextVal:

inc si

inc si

loop searchLoop

; Multiply of elements

searchNegMul:

mov cx, siz

mov si, offset number

mov ax, 1 ; mul always use ax (smth \* 1 = smth)

minusEl:

mov bx, [si]

imul bx ; multiply

jo overFlowErr ; chech overflow

plusEl:

inc si

inc si

loop minusEl

mov mulres, ax ; mow answer to variable

; Average value of elements

searchSrednee:

mov ax, sumres

cwd

idiv siz

mov sredres, ax

; Percent of positive elements

; Formula (count \* 100) / SIZE %

searchProchent:

mov cx, siz

mov si, offset number

mov ax, 0h

findPosElem:

mov bx, [si]

cmp bx, 0h

jl foundNegElem ; If value is negative - skip

add ax, 100d ; 1 peace \* 100 - check formula

jo overFlowErrSum

foundNegElem:

inc si

inc si

loop findPosElem

cwd

idiv siz

mov prochres, ax

jmp resOutput

; Print errors

overFlowErr:

print overflow ; Print overflow

jmp progend

zero:

print divByZeroError ; Print DivisionByZero

jmp progend

overFlowErrSum:

print overflowSum ; Print owerflow from sum

jmp progend

; Print results

resOutput:

print sumText

mov ax, sumres

call BinToAsc

print out\_str

mov cx,6 ; clear buffer

xor si,si

clear1:

mov [out\_str+si],' '

inc si

loop clear1

print mulText

mov ax,mulres

call BinToAsc

print out\_str

mov cx,6 ; clear buffer

xor si, si

clear2:

mov [out\_str+si],' '

inc si

loop clear2

print sredText

mov ax,sredres

call BinToAsc

print out\_str

mov cx,6 ; clear buffer

xor si,si

clear3:

mov [out\_str+si],' '

inc si

loop clear3

print prochText

mov ax, prochres

call BinToAsc

print out\_str

jmp progend

progend:

mov ax,4c00h

int 21h

; Data in (-29999, +29999)

; Args:

; Buffer input - stroka

; Res:

; bh - error flag

DIAPAZON PROC

xor bh, bh

xor si, si

cmp kol, 05h ; Count is 5

jb dop

cmp stroka, 2dh ; If enter 5 and more, check for `-`

jne plus ; If first is not `-` - check count of numbers

cmp kol, 06h ; If first `-` and values less 6

jb dop

inc si ; Chech first value

jmp first

plus:

cmp kol, 6 ; If count of data is 6 and first is not `-`

je error1 ; ERROR

first:

cmp stroka[si], 32h ; Check first simbol with `2`

jna dop ; If first simbol <= '2' - check the validity of the characters

error1:

mov bh, flag\_err ; Else bh = flag\_err

dop:

ret

DIAPAZON ENDP

; Chech data is value or `-`

; Args:

; Buffer input - stroka

; si - number of simbol

; Res:

; bh - error flag

DOPUST PROC

xor bh, bh

xor si, si

xor ah, ah

xor ch, ch

mov cl, kol ; Count of input chars

m11:

mov al, [stroka + si] ; In `al` - first

cmp al, 2dh ; If first char is `-`

jne testdop ; If no

cmp si, 00h ; If yes

jne error2 ; If another - ERROR

jmp m13

testdop:

cmp al, 30h ; Data in [30h, 39h]

jb error2

cmp al, 39h

ja error2

m13:

inc si

loop m11

jmp m14

error2:

mov bh, flag\_err ; Another error

m14:

ret

DOPUST ENDP

; ASCII to number

; Args:

; cx - Count of input simbols

; bx - number of simbol from last

; Res:

; number - buffer numbers

; di - number of value in the array

AscToBin PROC

xor ch, ch

mov cl, kol

xor bh, bh

mov bl, cl

dec bl

mov si, 01h ; In si, the weight of the discharge

n1:

mov al, [stroka + bx]

xor ah, ah

cmp al, 2dh ; Check positive or negative

je otr ; if negative

sub al, 30h

mul si

add [number + di], ax

mov ax, si

mov si, 10

mul si

mov si, ax

dec bx

loop n1

jmp n2

otr:

neg [number + di] ; Show negative in `adding` code

n2:

ret

AscToBin ENDP

; Number to ASCII

; Args

; ax - number

; Res

; out\_str - numbers buffer

BinToAsc PROC

xor si, si

add si, 05h

mov bx, 0Ah

push ax

cmp ax, 00h

jnl mm1

neg ax

mm1:

cwd

idiv bx

add dl,30h

mov [out\_str + si], dl

dec si

cmp ax, 00h

jne mm1

pop ax

cmp ax, 00h

jge mm2

mov [out\_str + si], 2dh

mm2:

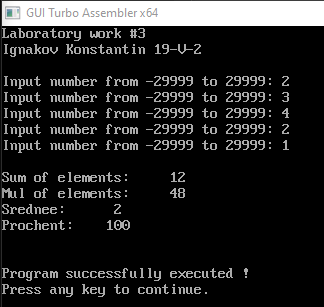
ret

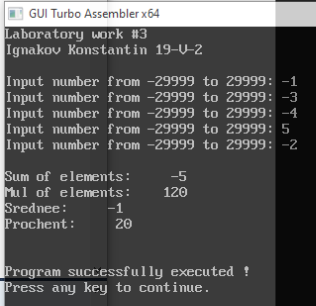
BinToAsc ENDP

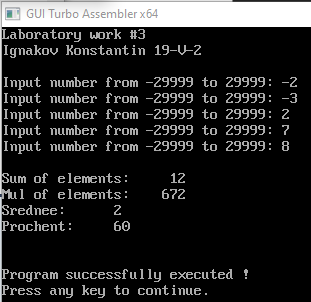
c1 ENDS

end start

Работа программы.







**Трансляция**

Транслятор обрабатывает текст программы, формируя локальный счетчик команд.

Каждый сегмент программы транслируется отдельно (начиная с 0). Значение счетчика в текущей точке программы возвращает ***оператор*** $. При обработке очередной команды (или данных) локальный счетчик увеличивается на число соответствующее байтов. Таким образом, значение счетчика определяет смещение команды или данных в соответствующем сегменте.

Транслятор при обработке текста

* выполняет директивы,
* формирует макрорасширения,
* вычисляет выражения,
* выполняет операторы ассемблера,
* формирует:
  + таблицу описания сегментов и групп:

Имя сегмента (группы) *Bit Size Align Combine Class*  
(таблица помещается в объектный модуль для обработки компоновщиком)таблицу локальных символических имен:

* + для констант: Имя, тип, значения
  + для переменных: имя, тип, сегмент: смещение.

Встретив в тексте программы ссылку на имя, транслятор ищет это имя в таблице символических имен. При построении кода имя переменной заменяется адресом переменной, имя константы – значением.

* таблицу глобальных (внешних) имен, определенных директивой *EXTRN* (таблица помещается в объектный модуль для обработки компоновщиком)

**Компоновка**

Компоновщик может создать выполнимый файл с расширениями: .com; .exe.

.com – содержит только код без какой-либо дополнительной информации (весь код в одном сегменте 64К, необходим ключ /t)

.exe – файл содержит заголовок и загрузочный модуль. В заголовке находится размер загрузочного модуля, относительные адреса: сегмента стека и *sp*, сегмента кода и *ip*.

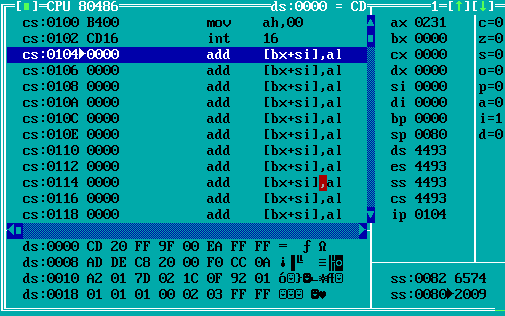
**Функция 00h: прочитать данные с клавиатуры**

Перед вызовом прерывания требуется записать в регистр AH значение функции: 00h

После выполнения функции в регистр будет помещена следующая информация:

* AH – скан-код символа
* AL – ASCII-код символа

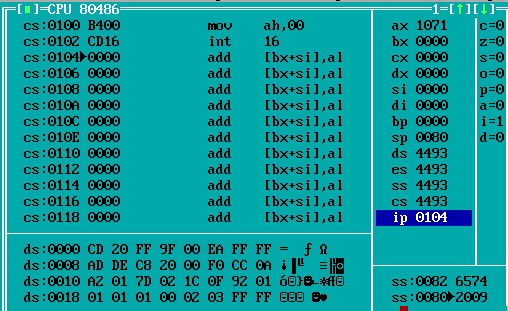
Нажатие клавиши 1 (основной клавиатуры, не numpad)



AH: 02h – скан код символа 1

AL: 31h - ASCII код символа 1

Нажатие клавиши q

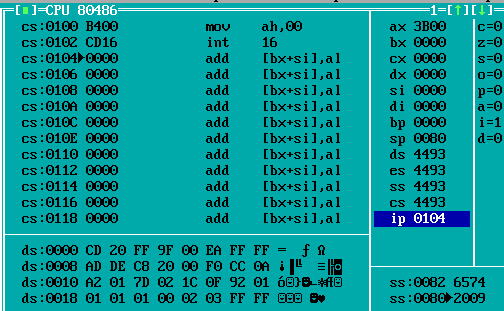


AH: 10h – скан код символа q

AL: 71h - ASCII код символа q

При нажатии управляющих и функциональных клавиш функция выдает ASCII-код со значением 0, благодаря чему их можно отличить от алфавитно-цифровых. Однако таким способом функциональные клавиши можно отличать лишь по скан-кодам.

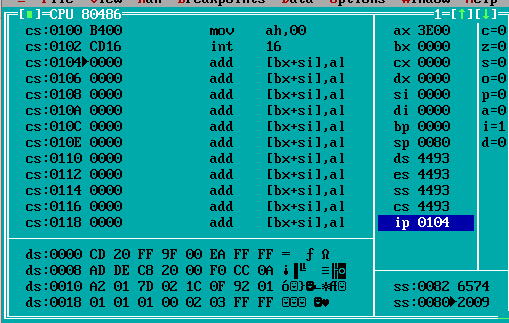
Нажатие клавиши F1



AH: 3Bh – скан код символа F1

AL: 00 - ASCII код для функциональных клавиш

Нажатие клавиши F4



AH: 3Eh – скан код символа F1

AL: 00 - ASCII код для функциональных клавиш

**Функция 00h: прочитать данные с клавиатуры**

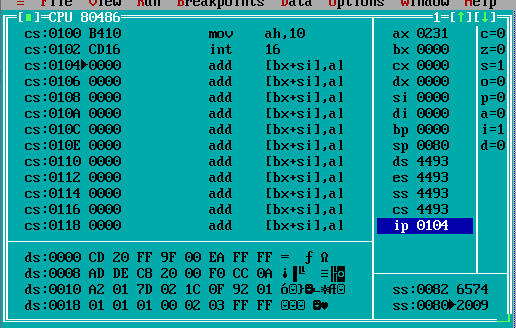
Функция 10h является усовершенствованным вариантом функции 00h. Она позволяет получить скан-коды функциональных клавиш F1-F12, а также клавиш управления курсором.

Перед вызовом прерывания требуется записать в регистр AH значение функции: 10h

После выполнения функции в регистр будет помещена следующая информация:

* AH – расширенный скан-код символа
* AL – ASCII-код символа

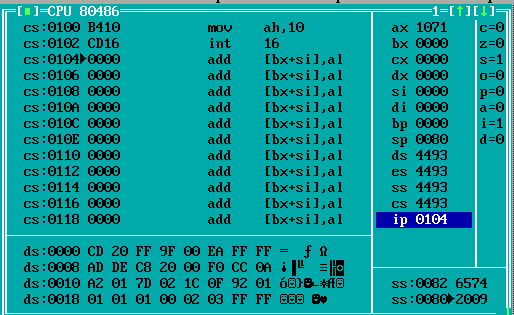
Нажатие клавиши 1 (основной клавиатуры, не numpad)



AH: 02h – скан код символа 1

AL: 31h - ASCII код символа 1

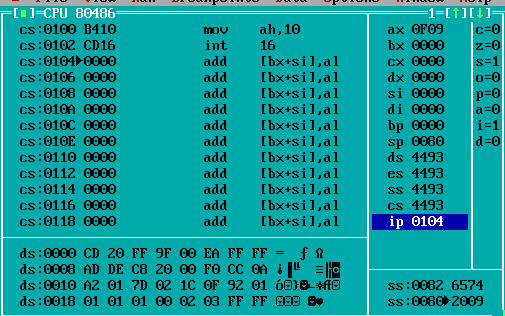
Нажатие клавиши q



AH: 10h – скан код символа q

AL: 71h – ASCII код символа q

Нажатие управляющей клавиши tab



AH: 0Fh – скан код клавиши enter

AL: 09h – ASCII код клавиши tab

Прерывание Int 16h имеет еще несколько функций:

* 01h – получить состояние клавиатуры
* 02h – получить состояние флагов клавиатуры
* 03h – управление режимом автоповтора
* 04h – вкл/выкл звуковой сигнал клавиш
* 05h – поместить символ в буфер клавиатуры
* 11h – получить состояние расширенной клавиатуры
* 12h – получить состояние флагов расширенной клавиатуры

Однако их рассмотрение выходит за рамки данного теоретического вопроса.

**Вывод**

При выполнении работы я научился работать с массивами, выполнять арифметическую и логическую обработку данных, использовать макросы и процедуры, преобразовывать данные “из байтов в строки”. Работать с условными переходами.