Коми Республикаса велӧдан, наука да том йӧз политика министерство
Министерство образования, науки и молодежной политики Республики Коми
Государственное профессиональное образовательное учреждение
«Сыктывкарский целлюлозно – бумажный техникум»

РАССМОТРЕНО
на заседании ПЦК информационных
дисциплин ГПОУ «СЦБТ»
Протокол № __
«__» _____ г.
Председатель ПЦК _____ В.А. Шулепов
Преподаватель _____ Д.Д. Расов

УТВЕРЖДАЮ
Зам.директора по УПР
ГПОУ «СЦБТ»
_____ Е.В. Соколова
«____» _____ 20__ г.

Методические указания по выполнению лабораторных работ

по учебной дисциплине МДК02.01 «Микропроцессорные системы»
для специальности
09.02.01 Компьютерные системы и комплексы

Программирование микропроцессора i8086 на языке Ассемблер

Сыктывкар 2018

Лабораторная работа № 10, 11

КОМАНДЫ, ОБСЛУЖИВАЮЩИЕ РАБОТУ С КЛАВИАТУРОЙ

Цель работы: Освоить команды **считывания данных и управления клавиатурой**. Изучить способы работы с процедурами.

1. Средства BIOS

Так же как и для вывода на экран, **BIOS** предоставляет больше возможностей по сравнению с **DOS** для считывания данных и управления клавиатурой. Например, функциями DOS нельзя определить нажатие комбинаций клавиш типа Ctrl-Alt-Enter или нажатие двух клавиш Shift одновременно, DOS не может определить момент отпускания нажатой клавиши, и наконец, в DOS нет аналога функции C **ungetch ()**, помещающей символ в буфер клавиатуры, как если бы его ввел пользователь. Все это можно осуществить, используя различные функции прерывания **16h** и операции с байтами состояния клавиатуры.

1.1. INT 16h, AH = 0, 10h, 20h - Чтение символа с ожиданием

Ввод:	**AH = 00h (83/84-key), 10h (101/102-key), 20h (122-key)**
Вывод:	**AL = ASCII-код символа, 0 или префикс скан-кода** **AH = скан-код нажатой клавиши или расширенный ASCII-код**

Каждой клавише на клавиатуре соответствует так называемый скан-код (см. приложение 1), соответствующий только этой клавише. Этот код посылается клавиатурой при каждом нажатии и отпускании клавиши и обрабатывается BIOS (обработчиком прерывания **INT 9**). Прерывание **16h** дает возможность получить код нажатия, не перехватывая, этот обработчик.

Если нажатой клавише соответствует ASCII-символ, то в **AH** возвращается код этого символа, а в **AL** - скан-код клавиши. Если нажатой клавише соответствует расширенный ASCII-код, в **AL** возвращается префикс скан-кода (например, E0 для серых клавиш) или 0, если префикса нет, а в **AH** - расширенный ASCII-код. Функция **00H** обрабатывает только комбинации, использующие клавиши 84-клавишной клавиатуры, **l0h** обрабатывает все 101 - 105-клавишные комбинации, **20h** - 122-клавишные. Тип клавиатуры можно определить с помощью функции **09h** прерывания **16h**, если она поддерживается BIOS (поддерживается ли эта функция, можно узнать с помощью функции **C0h** прерывания **15h**).

1.2. INT 16h, AH = 1, 11h, 21h - Проверка символа

Ввод:	**AH = 01h (83/84-key), 11h (101/102-key), 21h (122-key)**
Вывод:	**ZF = 1, если буфер пуст** **ZF = 0, если в буфере присутствует символ, в этом случае** **AL = ASCII-код символа, 0 или префикс скан-кода** **AH = скан-код нажатой клавиши или расширенный ASCII-код**

Символ остается в буфере клавиатуры, хотя некоторые BIOS удаляют символ из буфера при обработке функции **01h**, если он соответствует расширенному ASCII-коду, отсутствующему на 84-клавишных клавиатурах.

1.3. INT 16h, AH = 05h - Поместить символ в буфер клавиатуры

Ввод:	**AH = 05h** **CH = скан-код** **CL = ASCII-код**
Вывод:	**AL = 00, если операция выполнена успешно** **AL = 01h, если буфер клавиатуры переполнен** **AH модифицируется многими BIOS**

Обычно можно поместить **0** вместо скан-кода в **CH**, если функция, которая будет выполнять чтение из буфера, будет использовать именно ASCII-код.

Например, следующая программа при запуске из DOS вызывает команду **DIR** (но при запуске из некоторых оболочек, например **FAR**, этого не произойдет).

Пример №1.1

.model tiny	**; модель памяти, в которой сегменты кода, данных и стека объединены.**
.code	**; сегмент кода, который содержит данные.**
org 100h	**; начало COM-файла**
begin:	**; метка начала кода программы**
mov cl,'d'	**; заносим в регистр cl - ASCII-код буквы "d"**
call ungetch	**; переходим на метку ungetch: (вызов подпрограммы)**
mov cl,'i'	**; заносим в регистр cl - ASCII-код буквы "i"**
call ungetch	**; вызываем подпрограмму**
mov cl,'r'	**; заносим в регистр cl - ASCII-код буквы "r"**
call ungetch	**; вызываем подпрограмму**
mov cl,0Dh	**; перевод строки**
Ungetch proc	**; метка начала подпрограммы**
mov ah,05h	**; AH = номер функции**
mov ch,0	**; CH = 0 (скан-код неважен)**
int 16h	**; вызов DOS (прерывание)**
ret	**; функция DOS "завершить работу процедуры"**
Ungetch endp	**; окончание подпрограммы**
end begin	**; метка окончания кода программы**

1.4. Оформление процедур (подпрограмм).

Call ungetch

Ungetch proc

Ungetch endp

Ungetch - название процедуры

Call - вызов подпрограммы

Proc - procedure - процедура

endp - end procedure - конец процедуры

Стоит еще отметить, что из одной подпрограммы можно вызывать другие. Из других - третьи. Чаще всего с целью упрощения программы и, тем самым, уменьшения ее размера программисты используют простую связку команд:

Call метка

метка:

Любой вызов подпрограмм заканчивается возращение в блок кода командой **ret** (**ret** - return - возврат).

1.5. INT 16h, AH = 02h, 12h, 22h - Считать состояние клавиатуры

Ввод:	**AH = 02h (83/84-key), 12h (101/102-key), 22h (122-key)**
Вывод:	**AL = байт состояния клавиатуры 1** **AH = байт состояния клавиатуры 2 (только для функций 12h и 22h)**

Байт состояния клавиатуры 1 (этот байт всегда расположен в памяти по адресу **0000h: 0417h** или **0040h: 0017h**):

Бит 7: Ins включена

Бит 6: CapsLock включена

Бит 5: NumLock включена

Бит 4: ScrollLock включена

Бит 3: Alt нажата (любая Alt для функции 02h, часто только левая Alt для 12h/22h)

Бит 2: Ctrl нажата (любая Ctrl)

Бит 1: Левая Shift нажата

Бит 0: Правая Shift нажата

Байт состояния клавиатуры 2 (этот байт всегда расположен в памяти по адресу **0000h: 0418h** или **0040h: 0018h**):

Бит 7: SysRq нажата

Бит 6: CapsLock нажата

Бит 5: NumLock нажата

Бит 4: ScrollLock нажата

Бит 3: Правая Alt нажата

Бит 2: Правая Ctrl нажата

Бит 1: Левая Alt нажата

Бит 0: Левая Ctrl нажата

Оба этих байта постоянно располагаются в памяти, так что вместо вызова прерывания часто удобнее просто считывать значения напрямую. Более того, в эти байты можно записывать новые значения, и BIOS изменит состояние клавиатуры соответственно.

Помимо этих двух байт BIOS хранит в своей области данных и весь клавиатурный буфер, к которому также можно обращаться напрямую. Буфер занимает **16 слов с 0h**: **041Eh** по **0h: 043Dh** включительно, причем по адресу **0h: 041Ah** лежит адрес (ближний) начала буфера, то есть адрес, по которому располагается следующий введенный символ, а по адресу **0h: 041Ch** лежит адрес конца буфера, так что если эти два адреса равны, буфер пуст. Буфер действует как кольцо: если начало буфера - **043Ch**, а конец - **0420h**, то в буфере находятся три символа по адресам **043Ch**, **041Eh** и **0420h**. Каждый символ хранится в виде слова - того же самого, которое возвращает функция **10h**

прерывания **INT 16h**. В некоторых случаях (если) буфер размещается по другим адресам, тогда адрес его начала хранится в области данных BIOS по адресу **0480h**, а конца - по адресу **0482h**. Прямой доступ к буферу клавиатуры лишь немногим быстрее, чем вызов соответствующих функций BIOS, и для приложений, требующих максимальной скорости, таких как игры или демо-программы, используют управление клавиатурой на уровне портов ввода-вывода.

2. Средства DOS

Как и в случае вывода на экран, DOS предоставляет набор функций для чтения данных с клавиатуры, которые используют стандартное устройство ввода **STDIN**, так что можно использовать в качестве источника данных файл или стандартный вывод другой программы.

2.1. Функция DOS 0Ah - Считать строку символов из STDIN в буфер

Ввод:	**AH = 0Ah** **DS: DX = адрес буфера**
Вывод:	**Буфер содержит введенную строку**

Для вызова этой функции надо подготовить буфер, первый байт которого содержит максимальное число символов для ввода (1-254), а содержимое, если оно задано, может использоваться как подсказка для ввода. При наборе строки обрабатываются клавиши **Esc**, **F3**, **F5**, **BS**, **Ctrl-C/Ctrl-Break** и т.д., как при наборе команд DOS (то есть **Esc** начинает ввод сначала, **F3** восстанавливает подсказку для ввода, **F5** запоминает текущую строку как подсказку, **Backspace** стирает предыдущий символ). После нажатия клавиши **Enter** строка (включая последний символ **CR** (**0Dh**)) записывается в буфер,

начиная с третьего байта. Во второй байт записывается длина реально введенной строки без учета последнего **CR**.

Рассмотрим пример программы, выполняющей преобразование десятичного числа в шестнадцатеричное.

Пример №2.1

.model tiny	**; модель памяти, в которой сегменты кода, данных и стека объединены.**
.code	**; сегмент кода, который содержит данные.**
org 100h	**; начало COM-файла**
start:	**; метка начала кода программы**
mov dx,offset messagel	**; DS: DX - адрес строки**
mov ah,9	**; номер функции DOS в AH**
int 21h	**; вывести приглашение ко вводу message1**
mov dx,offset buffer	**; DS: DX - адрес строки**
mov ah,0Ah	**; номер функции DOS в AH**
int 21h	**; считать строку символов в буфер**
mov dx,offset crlf	**; DS: DX - адрес строки**
mov ah,9	**; номер функции DOS в AH**
int 21h	**; перевод строки**
xor di,di	**; DI = 0 - номер байта в буфере**
xor ax,ax	**; AX = 0 - текущее значение результата**
mov cl,blength	
xor ch,ch	**; обнуляем регистр ch**
xor bx,bx	**; обнуляем регистр bx**
mov si,cx	**; SI - длина буфера**
mov cl,10	**; CL = 10, множитель для MUL**
asc2hex:	**; метка начала блока asc2hex:**
mov bl,byte ptr bcontents [di]	
sub bl,'0'	**; цифра = код цифры - код символа "0",**
jb asc_error	**; если код символа был меньше, чем код "0",**
cmp bl,9	**; или больше, чем "9",**
ja asc_error	**; выйти из программы с сообщением об ошибке,**
mul cx	**; иначе: умножить текущий результат на 10,**
add ax,bx	**; добавить к нему новую цифру,**
inc di	**; увеличить счетчик**
cmp di,si	**; если счетчик+1 меньше числа символов -**
jb asc2hex	**; продолжить (счетчик считается от 0)**

push ax	; сохранить результат преобразования
mov ah,9	; номер функции DOS в AH
mov dx,offset message2	; DS: DX - адрес строки
int 21h	; вывести приглашение ко вводу message2
pop ax	; считать из стека
push ax	; записать в стек
xchg ah,al	; поместить в AL старший байт
call print_al	; вывести его на экран
pop ax	; восстановить в AL младший байт
call print_al	; вывести его на экран
ret	; завершение COM-файла
asc_error:	; начало блока asc_error:
mov dx,offset err_msg	; DS: DX - адрес строки
mov ah,9	; номер функции DOS в AH
int 21h	; вывести сообщение об ошибке
ret	; завершить программу
print_al:	; метка начала блока print_al:
mov dh,al	; заносим в dh значение регистра al
and dh,0Fh	; DH - младшие 4 бита
shr al,4	; AL - старшие
call print_nibble	; вывести старшую цифру
mov al,dh	; теперь AL содержит младшие 4 бита
print_nibble:	; процедура вывода 4 бит (шестнадцатеричной цифры)
cmp al,10	; три команды, переводящие цифру в AL
sbb al,69h	; в соответствующий ASCII-код
das	; (см. описание команды DAS)
mov dl,al	; код символа в DL
mov ah,2	; номер функции DOS в AH
int 21h	; вывод символа
ret	; этот RET работает два раза - один раз для возврата из процедуры print_nibble, вызванной для старшей цифры и второй раз - для возврата из print_al
messagel db "Десятичное число: $"	; строка с содержащая выводимые данные.
message2 db "Шестнадцатеричное число: $"	; строка с содержащая выводимые данные.
err_msg db "Ошибка ввода"	; строка с содержащая выводимые данные.
crlf db 0Dh,0Ah, '$'	; строка с содержащая выводимые данные.
Buffer db 6	; максимальный размер буфера ввода
blength db?	; размер буфера после считывания
bcontents:	; содержимое буфера располагается за концом COM-файла

end start	**; метка окончания кода программы**

Функция **0Ah** предоставляет удобный, но ограниченный способ ввода данных. Чаще всего используют функции посимвольного ввода, позволяющие контролировать отображение символов на экране, реакцию программы на функциональные и управляющие клавиши и т.д.

Функция DOS **01h** - Считать символ из **STDIN** с эхом, ожиданием и проверкой на **Ctrl-Break**.

Ввод:	**AH = 01h**
Вывод:	**AL = ASCII-код символа или 0. Если AL = 0, второй вызов этой функции возвратит в AL расширенный ASCII-код символа**

При чтении с помощью этой функции введенный символ автоматически немедленно отображается на экране (посылается в устройство **STDOUT** - так что его можно перенаправить в файл). При нажатии **Ctrl-C** или **Ctrl-Break** выполняется команда **INT 23h**. Если нажата клавиша, не соответствующая какому-нибудь символу (стрелки, функциональные клавиши **Ins**, **Del** и т.д.), то в **AL** возвращается **0** и функцию надо вызвать еще один раз, чтобы получить расширенный ASCII-код (см. приложение 1).

В трех следующих вариантах этой функции код символа возвращается в **AL** по такому же принципу.

2.2. Функция DOS 08h - Считать символ из STDIN без эха, с ожиданием и проверкой на Ctrl-Break

Ввод:	**AH = 08h**
Вывод:	**AL = код символа**

2.3. Функция DOS 07h - Считать символ из STDIN без эха, с ожиданием и без проверки на Ctrl-Break.

Ввод:	**AH = 07h**
Вывод:	**AL = код символа**

2.4. Функция DOS 06h - Считать символ из STDIN без эха, без ожидания и без проверки на Ctrl-Break.

Ввод:	**AH = 07h** **DL = 0FFh**
Вывод:	**ZF = 1, если не была нажата клавиша, и AL = 00** **ZF = 0, если клавиша была нажата. В этом случае AL = код символа**

2.5. Служебные функции DOS для работы с клавиатурой.

Кроме перечисленных функций используются некоторые служебные функции DOS для работы с клавиатурой.

2.5.1. Функция DOS 0Bh - Проверить состояние клавиатуры

Ввод:	**AH = 0Bh**
Вывод:	**AL = 0, если не была нажата клавиша** **AL = 0FFh, если была нажата клавиша**

Эту функцию удобно использовать перед функциями **01**, **07** и **08**, чтобы не ждать нажатия клавиши. Кроме того, вызов этой функции позволяет проверить, не считывая символ с клавиатуры, была ли нажата комбинация клавиш **Ctrl-Break**; если это произошло, выполнится прерывание **23h**.

2.5.2. Функция DOS 0Ch - Очистить буфер и считать символ

Ввод:	**AH = 0Ch** **AL = Номер функции DOS (01, 06, 07, 08, 0Ah)**
Вывод:	**Зависит от вызванной функции**

Функция **0Ch** очищает буфер клавиатуры, так что следующая функция чтения символа будет ждать ввода с клавиатуры, а не использовать нажатый ранее и еще не обработанный символ. Например, именно эта функция используется для считывания ответа на вопрос **"Уверен ли пользователь в том, что он хочет отформатировать диск?".**

Функции посимвольного ввода без эха можно использовать для интерактивного управления программой.

.model tiny	**; модель памяти, в которой сегменты кода, данных и стека объединены.**
.code	**; сегмент кода, который содержит данные.**
org 100h	**; началю COM-файла**
Begin:	**; метка начала кода программы**
call Wait_key	**; вызываем подпрограмму**
cmp al,27	**; сравниваем значение в al с кодом 27 (ESC)**
je Quit_prog	**; если да - то на метку Quit_prog**
cmp al,0	**; код клавиши расширенный? (F1-F12 и т.п.)**
je Begin	**; да - повторим запрос...**
call Out_char	**; вызываем процедуру вывода нажатой клавиши на экран**
jmp Begin	**; ждем дальше...**
Quit_prog:	**; метка, на которую придет программа в случае нажатия ESC**
mov al,32	**; помещаем в AL <пробел>**
call Out_char	**; вызываем процедуру вывода символа в AL**
int 20h	**; выходим...**
Wait_key proc	**; процедура ожидания клавиши от пользователя**
mov ah,10h	**; окончание подпрограммы**
int 16h	**; прерывание DOS**
ret	**; функция DOS "завершить работу процедуры"**
Wait_key endp	**; окончание процедуры Wait_key**
Out_char proc	**; началю процедуры out_char**
push cx	**; сохраним все регистры, которые будут изменены подпрограммой...**
push ax	**;... сделаем это для того, чтобы в последствии не было путаницы**
push es	**; сохраним сегментный регистр**
push ax	**; сохраним AX, т.к в нем код нажатой клавиши...**

mov ax,0B800h	**; установим ES на сегмент видеобуфера**
mov es,ax	
mov di,0	**; DI - первый символ первой строки**
mov cx, 2000	**; выводим 2000 символов (80 символов в строке 25 строк)**
pop ax	**; восстановим код клавиши**
mov ah,31	**; цвет символа**
Next_sym:	**; метка для цикла**
mov es: [di],ax	**; заносим код клавиши и ее цвет (цвет всегда 31)**
inc di	**; увеличиваем указатель на 2 (первый байт - символ, второй байт - цвет)**
inc di	
loop Next_sym	**; обработка следующего символа**
pop es	**; восстановим сохраненные регистры и выровним стек**
pop ax	
pop cx	
ret	**; вернемся из процедуры**
Out_char endp	**; окончание процедуры Out_char**
end Begin	**; метка окончания кода программы**

Программа делает следующее:

Ждет от пользователя клавиши;

- если это расширенный ASCII (F1-F12, стрелки), то игнорирует ее;
- если это не расширенный ASCII (A-Z, 0-9 и т.п.) - заполнить экран данным символом;
- если нажимаем ESC (27 или 1Bh), то заполнить экран пробелами (mov al,32) и выйти.

3. Задание для выполнения.

3.1. С помощью редактора эмулятора **EMU 8086** напишите программы, исходный текст которых приводится в примерах данной лабораторной работы.

3.2. Создайте исполняемые файлы типа ***.com**.

3.3. Изучите работу полученных программ.

3.4. Напишите программу для вывода на экран содержимого регистра **DS** (на основе примера **№2.1**). Сравните результат работы своей программы и того, что показывает отладчик.

3.5. Опишите работу команд **DIV**, **PUSH**, **POP**, **SHL**, **TEST**.

3.6. Установите (найдите адреса и запишите), где находятся числа, помещенные в стек.

3.7. Напишите программу для вывода на экран содержимого регистра **CS** (на основе **примера №3.1**).

3.8. Предложите другие способы решения поставленных задач.

4. Контрольные вопросы

4.1. Преимущества использования команды SHL вместо TEST (пример №1.1)?

4.2. Чем отличаются команды **SHL dx,1** *и* **SHL dx, cl**

4.3. Как переслать содержимое X в стек и получить обратно?

4.4. Опишите методику вывода значения байта в десятеричной системе счисления?

4.5. Опишите методику вывода значения байта в шестнадцатеричной системе счисления?

4.6. Опишите методику вывода двоичного кода числа, записанного в регистр X

4.7. Стек. Принцип работы. Команды работы со стеком.

4.8. Укажите отличия в работе тандема команд:

push DS

pop ES

от

push DS

pop ES

Приложение №1

Основные Скан-Коды клавиш клавиатуры.

Клавиша	Код	Клавиша	Код	Клавиша	Код	Клавиша	Код
Esc	01h	Enter	1Ch	K*	37h	Ins	52h
1!	02h	Ctrl	1Dh	Alt	38h	Del	53h
2 @	03h	A	1Eh	SP	39h	SysRq	54h
3 #	04h	S	1Fh	Caps	3Ah	Macro	56h
4 $	05h	D	20h	F1	3Bh	F11	57h
5%	06h	F	21h	F2	3Ch	F12	58h
6 ^	07h	G	22h	F3	3Dh	PA1	5Ah
7 &	08h	H	23h	F4	3Eh	F13/LWin	5Bh
8 *	09h	J	24h	F5	3Fh	F14/RWin	5Ch
9 (	0Ah	K	25h	F6	40h	F15/Menu	5Dh
0)	0Bh	L	26h	F7	41h	F16	63h
- _	0Ch	;:	27h	F8	42h	F17	64h
= +	0Dh	' "	28h	F9	43h	F18	65h
BS	0Eh	` ~	29h	F10	44h	F19	66h
Tab	0Fh	LShift	2Ah	Num	45h	F20	67h
Q	10h	\ |	2Bh	Scroll	46h	F21	68h
W	11h	Z	2Ch	Home	47h	F22	69h
E	12h	X	2Dh	-	48h	F23	6Ah
R	13h	C	3Eh	PgUp	49h	F24	6Bh
T	14h	V	2Fh	K-	4Ah	EraseEOF	6Dh
Y	15h	B	30h		4Bh	Copy/Play	6Fh
U	16h	N	31h	K5	4Ch	CrSel	72h
I	17h	M	32h	®	4Dh	Delta	73h
O	18h	, <	33h	K+	4Eh	ExSel	74h
P	19h	. >	34h	End	4Fh	Clear	76h
[{	1Ah	/?	35h	I	50h		
] }	1Bh	RShift	36h	PgDn	51h		