**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И. РАЗЗАКОВА**

**КАФЕДРА «ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА»**

# МИКРОПРОЦЕССОРЫ И МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ

**Методические указания к выполнению лабораторных работ. Вводная часть. Для студентов специальности 552801.01 «Вычислительные машины, комплексы, системы и сети»**

**Бишкек – 2011**

“Рассморено”
на заседании кафедры
“ИВТ”
Протокол № 2 от 23.09.2010 г.

“Одобрено”
Методическим советом
ФИТ
Протокол № 2 от 25.10.2010 г.

УДК 681.3.

Составители: АЛЫМКУЛОВ С.А., ТЕНТИЕВА С.М., ТУЛЬТЕМИРОВА Г.У.

Микропроцессоры и микропроцессорные системы. Методические указания к выполнению лабораторных работ. Вводная часть. Для студентов специальности 552801.01 «Вычислительные машины, комплексы, системы и сети» /КГТУ им. И.Раззакова; сост.: С.А.Алымкулов, С.М.Тентиева, Г.У.Тультемирова. – Б.: ИЦ «Текник», 2011. – 25 с.

Излагаются теоретические сведения для изучения микропроцессорного стенда Pesona 16SDK и выполнения на его базе лабораторных работ по курсу «Микропроцессоры и микропроцессорные системы».

Предназначены для студентов дневной формы обучения.

Иллюстр. 9. Библиогр.: 8 назв.

Рецензент к.т.н., доцент каф. «ИВТ» Шабданов М.А.

## Теоретическое введение к выполнению лабораторных работ

### 1.1 Описание микропроцессорного лабораторного стенда Pesona 16 SDK

SDKP16 представляет собой компактную блочную плату, что обеспечивает широкую область применения для проведения экспериментов, а также для разработки интегрированных систем.

Система предназначена для разработки приложений на базе микроконтроллерного уровня. В системе используется 16-битный RISC микропроцессор.

SDKP16 предлагает множество практических применений, которые делают его полезным инструментом для использования в электротехнике, для исследования в университетах и колледжах. Система поможет пользователям понять, как микропроцессор Pesona 16 взаимодействует с интерфейсами, с памятью и подсистемами.

Кроме того, она помогает программисту понять операции в микрокомпьютерной системе с программным обеспечением.

SDKP16 могут быть использованы в качестве основы для разработок систем контроля и управления событиями. Пользователи будут иметь возможность полностью сосредоточиться на разработке программного обеспечения, не беспокоясь о разработке микроконтроллера.

SDKP16 может работать в двух режимах: обычном режиме и в режиме эмулятора. Обычный режим подходит для пользователей, которые требуют постоянного хранения программ и данных в EPROM (который прошиваетя системой, и не затирается, когда подача питания прервана). С другой стороны, режим эмулятора полезен для пользователей, которым необходимо изменить программы без стирания или перепрограммирования в EPROM.

Микропроцессорный стенд состоит из следующих компонентов:

1. Интерфейс ввода/вывода;
2. Последовательный порт;
3. Параллельный порт;
4. Часы реального времени;
5. Система шифрования (DES-SECReT IC);
6. ЖК дисплей (24 x 2 точечный дисплей);
7. Клавиатура;
8. Внешние коннекторы.

## Описание комплекса SDKP16

**Особенности в SDKP16**. Аппаратные средства вычислительной системы:

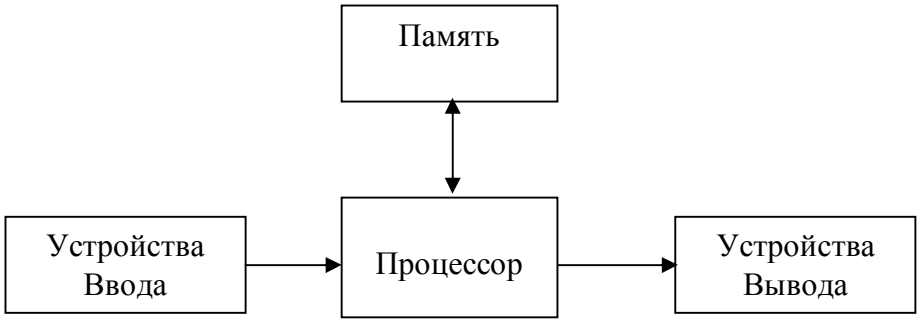


Рис. 1. Структурная схема SDKP 16

Связи между различными функциональными блоками в системе иллюстрирует рис.2:

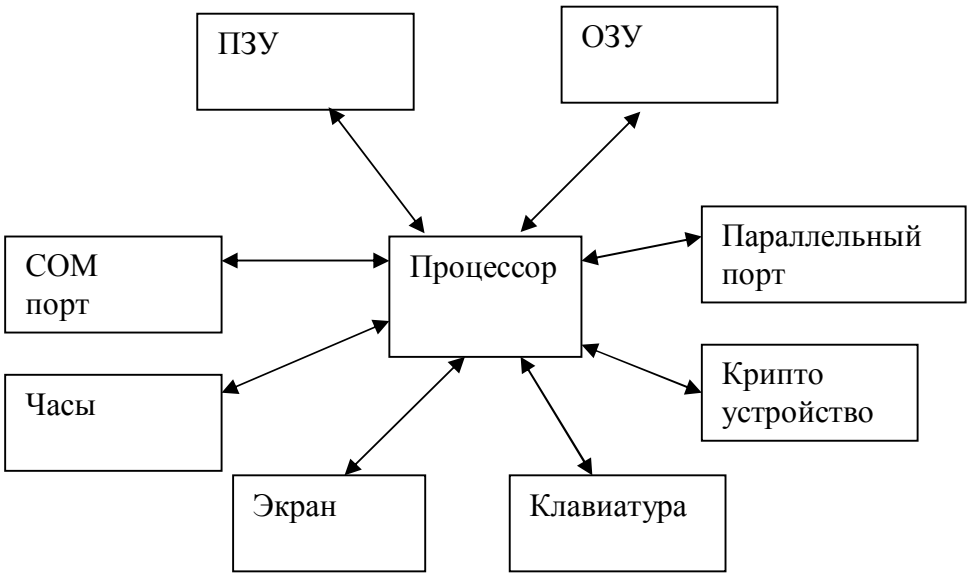


Рис. 2. Схема взаимодействия процессора с устройствами МПС

**PESONA 16 (MAP100) - процессор RISC Micro.** В основе SDK лежит процессор RISC PesonaTM 16 (MAP100) (Рис.3). Он отвечает за все арифметические операции и логические решения. Помимо арифметических и логических функций, PesonaTM 16 имеет систему управления операциями.

А PesonaTM 16 Макро Ассемблер (ASMP16) используется для поддержки PesonaTM 16 ассемблерных языков. Особенности: 16 бит архитектура RISC; 16 x 16-битных рабочих регистров общего назначения; 4 x 16-битных контроллера состояния регистров; 26 инструкций - в основном однотактных; Внешние и внутренние источники прерываний; Источник питания: 5В; Частота: 0-20МГц; PLCC 68-Pin микросхема.

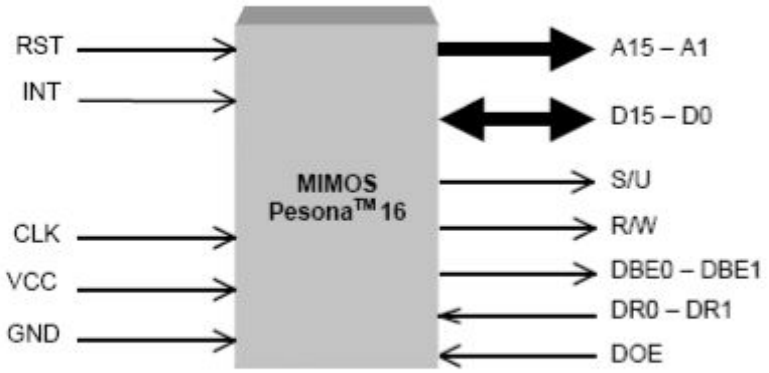


Рис. 3.Функциональное назначение процессора PESONA 16.

**Ассемблер для процессора P16 (ASMP16).** ASMP16 является макроассемблером, который работает на любой MS-DOS микроЭВМ, в том числе с IBM PC и PS / 2 семействах. При использовании файла с подходящим содержимым на Pesona 16 ассемблере, ASMP16 создает HEX файл представление объектного кода, собранного из исходного файла с инструкциями. Шестнадцатеричный файл содержит только символы ASCII и следовательно, может быть переданы в устройство посредством серийного порта.

**Интерфейс памяти.** В SDKP16 содержится 4Kbytes X 2 EPROM (стираемое программируемое ПЗУ) (8Kbyte x 2 для эмулятора режиме), которые постоянно хранят программы и данные, которые являются резидентами в системе. Эти программы и данные не затираются, когда питания отсоединяется. В том случае, программы и данные должны быть изменены, надо подвергнуть EPROM ультрафиолетовому излучению в течение примерно 30 минут. В SDKP16 также содержится 24Kbyte x 2 SRAM (статическая память случайного доступа) (16Kbyte x 2 для эмулятора режиме), который сохраняются данные до тех пор, как DC питание включено.

Определенный артикль SRAM хранит временные данные и используется, когда размер чтения / записи памяти сравнительно небольшой. ПЗУ полезна для постоянных данных и хранения программ, но не может предоставить для системы всю память. Данные и программы, которые изменяются в системе должны быть сохранены в памяти. Pesona 16 микропроцессор способен записывать данные на любой 16-битной или 8-битной архитектуре памяти. В последнем, 16-разрядные данные должны быть разделены на две отдельные секции, которые имеют размер 8 бит , так что Pesona 16 может писать либо половину или обе половинки. Рис. 5 иллюстрирует два раздела памяти.

Один раздел (нижний байт D0-D7) проводит все нечетные ячейки памяти, в то время как другой раздел (верхний байт D8-D15) занимает четные номера ячеек памяти.

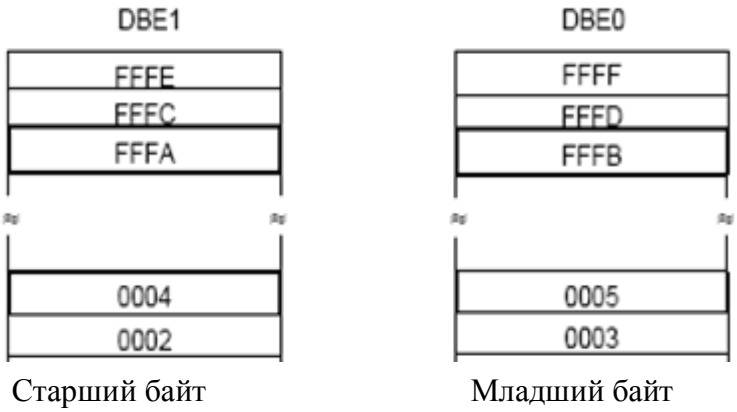


Рис. 4. Нечетные и четные разделы памяти.

**Интерфейс Ввода/Вывода.** В SDKP16 I/O интерфейс обеспечивает эффективное взаимодействие между компонентами. I/O интерфейс позволяет SDKP16 получать информацию из внешней системой (вход) и представление данных и команд с внешней системой (результат). I/O интерфейс схемы преобразует сообщения протокола, и напряжений и токов от внешнего мира в цифровой форме - находится в микропроцессорной системе, или он может преобразовать цифровые сигналы в микропроцессоре в напряжения и токи, что требуют внешние устройства. Два основных интерфейса - последовательные и параллельные - используются для выполнения ввода / вывода, функции между двунаправленной внутренней шиной из SDKP16 и линий, связанных с внешним устройством. Клавиатура, LCD, часы реального времени и DES I/O драйвер также входят в SDKP16.

**Последовательный интерфейс.** Последовательный интерфейс обеспечивает последовательную передачу данных.

Motorola в 6850 ACIA обеспечивает схему для подключения последовательных асинхронных устройств, как, например, ЭЛТ терминал, телетайп или модем. Шина интерфейса включает в себя селектор, позволяющий читать / писать и прервать сигнал, в дополнение к 8-битной двунаправленной шине данных. Параллельно данные в SDKP16 является серийно передаваемые и получаемые (одновременно) в ASCII надлежащего

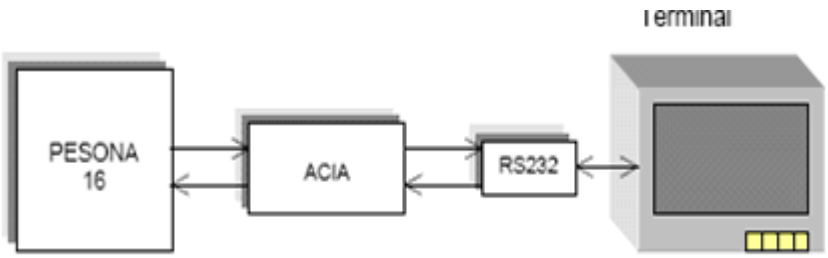


Рис. 5. Схема взаимодействия PESONA 16 и терминала

формата и проверки ошибок. Интерфейс RS232 используется для подключения коммуникационных устройств с ACIA. Интерфейс RS232 должен соответствовать ACIA стандартам (Рис.5-6).

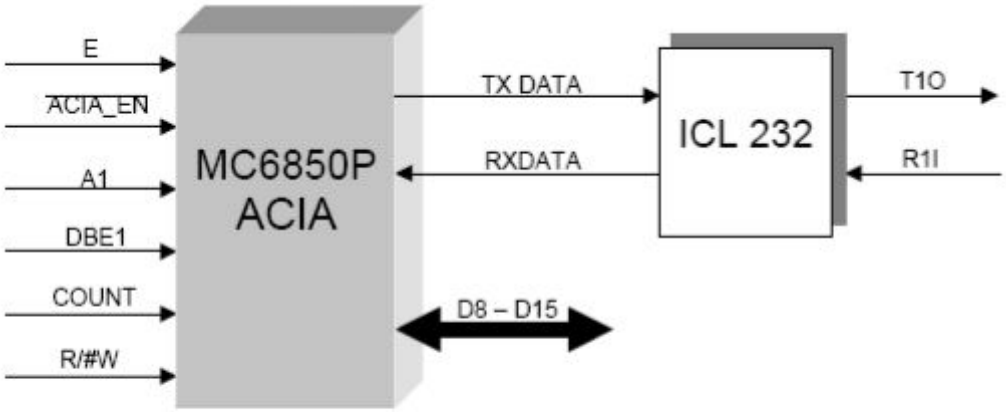


Рис. 6. Подключения коммуникационных устройств с ACIA

**Параллельный интерфейс.** Параллельный интерфейс является самым быстрым и самым популярным механизмом ввода/вывода. Много линий может быть использована для передачи данных в одном шаге. Периферийные устройства взаимодействуют с SDK P16 через интерфейс Intel 8255A.B нем содержится 24 I/O pin, которые могут быть индивидуально запрограммированы в 2 группы 12 pin, и используется в 3 основных режима работы (Рис. 7). 24 I / O pin можно разделить на 3 порта: 1). Порт A (8-pin) 2). Порт B (8-pin) 3). Порт C (8 -pin) - 4 верхних бита и 4 нижних бита.

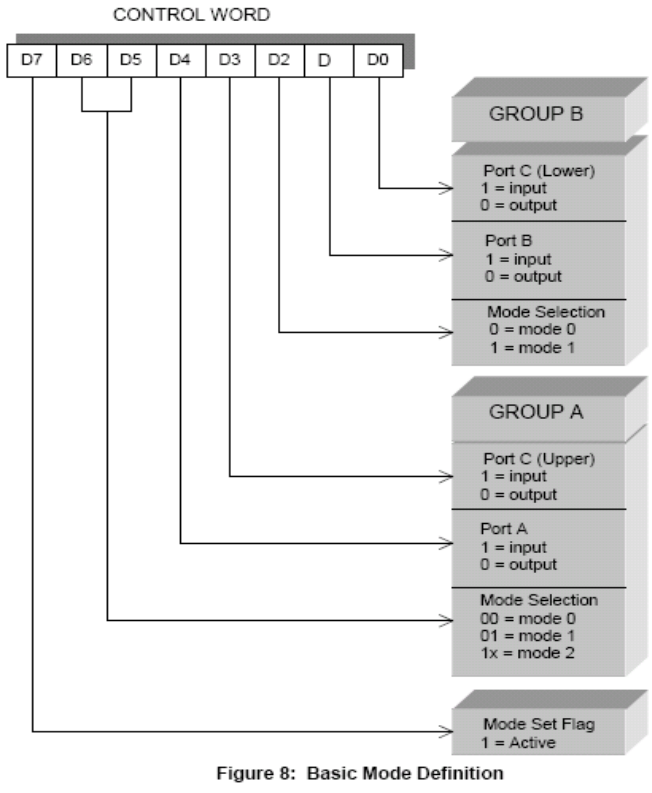


Figure 8: Basic Mode Definition

Рис. 7. Режимы работы ввода/вывода

**Часы реального времени.** Часы реального времени реализованы на базе микросхемы ICM7170 (Рис.8). Часы, как и все устройства находятся на процессорной шине.

Часы нужны для приложений, которые требуют использования данных о времени, таких, как: промышленные системы управления, системы управления бытовыми приборами и прочее. Для доступа к часам используется 8-битная двунаправленная шина.

Импульсы подаются из кварцевого генератора схемы. Особенности микросхемы ICM7170: 8-бит; Совместимый микропроцессорная шина; Измерение времени 1/100s доли секунды до 99 лет; Программное обеспечение по выбору 12/24 часового формата; Полный календарь с автоматической коррекцией "високосный год"; Доступ со скоростью меньше 300ns (наносекунд); On-Chip сигнал компаратора и RAM.

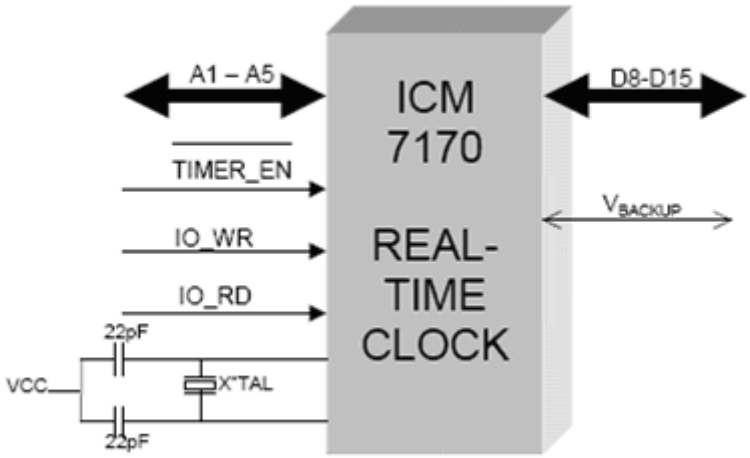


Рис. 8. Часы реального времени

**Жидкокристаллический дисплей (LCD).** ЖК используется для отображения символов или сообщений (Рис.9). ЖК используемый в SDKP16 является алфавитно-цифровым, 24X2 матричным дисплеем. Модуль оснащен интегрированной CMOS, микропроцессором и ЖК-дисплеем. Этот модуль использует один 5x7 матричный формат - в комплекте с курсором, и способен отображать полный набор символов ASCII, плюс до 8 дополнительных программируемых пользователем пользовательских символов. В SDKP16 общается с LCD используя шину данных, и только 3 контрольных линий , RS, R / W и E. Передача данных требуется только, когда у SDKP16 возникает потребность обновить или прочитать данные дисплея. Особенности ЖК модуля: 5В; 2мА; единый источник питания; Высокая Контрастность; Матричные символы для хорошей читаемости; Широкий регулируемый угол обзора; Компактный и легкий TTL и 5В CMOS совместимый; Интерфейсы для 8-битных шин данных; Мощные инструкции ASCII совместимый 160 различных букв, цифр и символов.

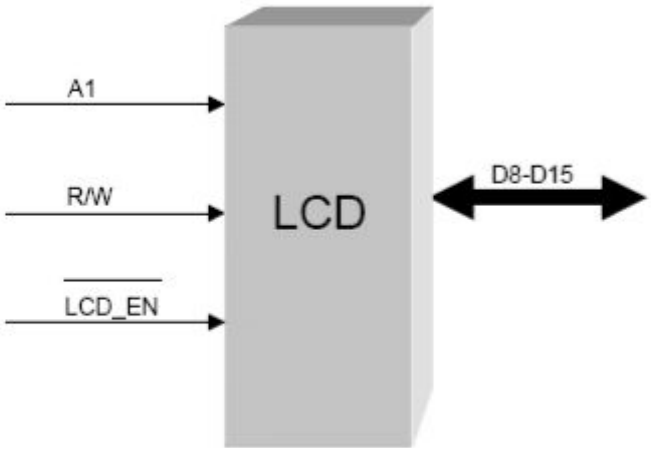


Рис. 9. Экран

**Клавиатура**. Клавиатура работает в качестве устройства ввода данных с клавиш. Каждый ключ, когда нажат, может вызвать уникальные события, от которых зависят выполняемые программы. Функции определяются системами управления. 4X4 миниатюрная клавиатура не требует внешнего питания и интерфейса непосредственно IC 74C922.

Кодировщик преобразует матрицу данных с клавиатуры в ASCII код и интерфейс через 4-разрядную шину данных передает данные процессору.

## Написание программы

Программа для микропроцессора разрабатывается в любом текстовом редакторе на специальном диалекте ассемблера ASMP16. Оформление файла программы происходит в соответствии со стандартными синтаксическими правилами оформления файла ассемблера. Расширение файла исходного кода программы обязательно должно быть .p16.

## Компиляция

Компиляция программ для микропроцессорного стенда производится при помощи специального компилятора ASMP16. В связи с тем, что микропроцессор работает в виртуальном режиме, после компиляции необходимо над байт кодом выполнять сценарий сдвига на стартовый адрес 4000(hex) и сценарий трансляции в формат данных motorolla S19 byte code. Для того, чтобы выполнить компиляцию файла программы необходимо в консоли (command.com) перейти в консоли в директорию с компилятором, вызвать сценарий компиляции L с параметром имя файла. После чего начнется процесс компиляции. На выходе будет получен файл с расширением .s19, который представляет из себя бинарный код в формате motorolla S19. Этот файл и записывается в устройство для исполнения. В случае возникновения ошибок в файл трансляции будут вставлены коды и названия ошибок в те места, где допущены ошибки. Необходимо прогонять сценарий компиляции до тех пор, пока не будет получен файл .s19 без ошибок.

## Подключение системы к ПК

Система подключается к ПК через последовательный порт при помощи кабеля типа МП RS 232. После подключения необходимо запустить программу hyper terminal, создать новое подключение на порту COM1, выбрать скорость 9600 б/с, отключить аппаратное управление потоком и нажать на устройстве кнопку «RESET». На экране отобразится приглашение устройства со списком возможных действий.

## Запись в ОЗУ системы

Для того чтобы записать программу в устройство, необходимо нажать клавишу “L” и в hyper terminal отправить текстовый файл с программой. Программа размещается в виртуальной памяти микропроцессорного стенда начиная с адреса 4000(аппаратное требование виртуального режима работы процессора). После того, как программы загрузится в память стенда, ее можно запускать.

## Запуск и исполнение программы

Для запуска программы необходимо дать команду “g 4000” в hyper terminal. После этого программа запустится. 4000 – это стартовый базовый адрес, с которого и начинается исполнение байткода формата .s19. Остановить выполнение программы в микропроцессорном стенде можно при помощи кнопки reset, расположенной на передней панели микропроцессорного стенда.

Ниже приведен пример программы на ASMP16 с комментариями. Данная программа дается для того, чтобы понять, как написать программы для микропроцессорного стенда. Это своего рода шаблон, в котором присутствуют макроопределения, функции инициализации, которые пригодятся для выполнения последующих лабораторных работ

```
; r15 = стэк поинтер
; r14, r13 = прерывания
; r12, r11 = макропроцессы

;-------- Макроопределения -------
ldi                 macrodata, reg      ; загрузка в
                        регистр
                    mlo   data, reg
                    mhi   data, reg
                    endm


push                macro reg
```

```
                sw      reg,r15
                mlo     2, r12
                sub     r15,r12,r15
                endm

pop             macro reg
                mlo     2, r12
                add     r15,r12,r15
                lw      r15,reg
                endm

jmpi            macroloc                ; переход к
                        локации
                mlo     loc, r12
                mhi     loc, r12
                jmp     r12
                endm

sbtm    macro data, mem         ; сбросить          байт в память
                mlo     mem, r12
                mhi     mem, r12
                mlo     data, r11
                sb      r11, r12
                endm

lbfm    macro mem                                   ; взять байт
                        из памяти
                mlo     mem, r12
                mhi     mem, r12
                lb      r12, r11
                endm

incr            macro reg
                mlo     1, r12
                add     reg, r12, reg
                endm
decr            macro reg
                mlo     1, r12
                sub     reg, r12, reg
                endm

;--------------------------------------
;установка битов слова состояния PSR
psr_fz          equ     8000h           ;
```

```
psr_ufz         equ    7fffh                    ;
psr_pm          equ    4000h                    ;
psr_cm          equ    2000h                    ;
psr_ei          equ    1000h                    ;
psr_di          equ    efffh                    ;
psr_ex          equ    0007h                    ;
;---------------------------------------

;---------------------------------------
;регистры LCD
lcd_stat  equ   3101h           ; регистр статуса
lcd_data        equ   3103h           ; регистр данных
;LCD control data
lcd_busy        equ   80h              ; флаг занятости
lcd_mode              equ    38h                        ; 8 бит + 2 линии
lcd_curs        equ    06h                       ; инкрементация
                         курсора
lcd_disp        equ    0eh                       ; включаем
                         дисплей и курсор
lcd_clr         equ    01h                       ; очистка дисплея
lcd_ddl1        equ    80h                       ; указатель на 1
                         строку в RAM
lcd_ddl2        equ    a8h                       ; указатель на 2
                         строку в RAM
lcd_cg          equ    40h                       ;
;---------------------------------------

;---------------------------------------
ascii_$         equ   '$'           ; символ конца
                          строки
ascii_a         equ    'A'
;---------------------------------------
; стэк
stack      equ   fffeh           ; начало стэка

;--------------------------------

                       org    4000h

;--------------------------------
start:   ldi       stack,r15              ; R15 указатель
                          стэка
                 bsr     initlcd
```

```
fire:   mlo     lcd_clr,r9
                bsr     lcdctrl
                ldi   firemsg,r10               ; вывести 1
                        фразу
                bsr     lcdtmsg
                bsr     delay                     ; задержка

help:           mlo   lcd_clr,r9
                bsr     lcdctrl
                ldi   helpmsg,r10         ; вывести
                                        вторую  фразу
                bsr     lcdtmsg
                bsr     delay                     ; задержка
                jmpi   fire
;*************************************************************

;-------------------------------
; R9 = регистр данных
lcdctrl:        push  r2
                        push  r4
          push    r6
                        ldi     lcd_stat, r4        ; указываем
                           на регистр
                           статуса
                        mlo    lcd_busy, r5           ; ставим флаг
                            занятости
lcdwaitb:   lb   r4, r6            ; читаем статус
                            занятрости
                        and    r6, r5, r6         ; проверка на
                           занятость
                        bne0  r6, lcdwaitb   ; цикл если
                            занят
                        sb      r9, r4            ; установка
                            функций

                        pop    r6
            pop  r4
                        pop    r2
                        rts
;---------------------------------
;r10 = адрес сообщения
lcdtmsg:        push  r2
                        push  r4
                        push  r5
```

```
                    push  r6
                    clr   r9
lcdtnxt:      lb    r10, r9          ; получаем
                      строку

                    mlo   ascii_$, r4
                    xor   r4, r9, r4
                    beq0  r4, lcdtend   ; если конец
                      сообщения
                    bsr   lcdcout       ; посылка
                      на дисплей
                    mlo   1,r4          ;
                    add   r4,r10,r10    ; указатель на
                      следующий
                      символ
                    jmpi  lcdtnxt

lcdtend:      pop   r6
                    pop   r5
                    pop   r4
                    pop   r2
                    rts
;--------------------------------
; R9 = данные для передачи
lcdcout:      push  r4
                    push  r5
                    push  r6

                    ldi   lcd_stat, r4      ; указатель
                      на статус
                      регистр
                    mlo   lcd_busy, r5
lcdwta:  lb   r4, r6            ; читаем
                      статус LCD
                    and   r6, r5, r6      ; проверка на
                      занятость
                    bne0  r6, lcdwta    ; если не
                      свободен,
                      то цикл
                    ldi   lcd_data, r4    ; указатель на
                      регистр данных
                    sb    r9, r4          ; посылка
                      символа
                    pop   r6
```

```
                    pop    r5
                    pop    r4
                    rts
;--------------------------------
initlcd:  push r2
                    ldi    lcd_stat, r4
                    mlo    lcd_busy, r7          ; Установка
                       флага занятости
inilcd1:  lb     r4, r6              ; чтение статус
                       LCD
                    and    r6, r7, r8        ; проверка
                       статуса
                       флага BISY
                    bne0   r8, inilcd1    ; цикл если
                        занято
                    mlo    lcd_mode, r5      ;
                    sb     r5, r4           ; установка
                       функции

inilcd2:  lb     r4, r6              ; чтение
                      статуса LCD
                    and    r6, r7, r8        ; проверка
                      статуса
                      флага BISY
                    bne0   r8, inilcd2   ; цикл если
                       занято
                    mlo    lcd_curs, r9      ; set reg r9=$06(increment mode)
                    sb     r9, r4           ;

inilcd3:  lb     r4, r6              ; чтение
                      статуса LCD
                    and    r6, r7, r8        ; проверка
                       статуса
                       флага BISY
                    bne0   r8, inilcd3    ; цикл если
                       занято
                    mlo    lcd_disp, r9
                    sb     r9, r4           ;

inilcd4:  lb     r4,r6               ; чтение
                       статуса LCD
                    and    r6,r7,r8          ; проверка
                       статуса
                       флага BISY
```

```
                        bne0  r8,inilcd4       ; цикл если
                           занято
                        mlo   lcd_clr, r9       ;
                        sb    r9,r4             ;
                        pop   r2
                        rts
;-----------------------------------
delay: push     r2
            push  r4
            ;push r5
       ;push        r6
            mlo   10, r4
     mhi ffh, r5
        mlo   ffh, r5
        mlo   1, r6
delay1:
            sub   r5, r6, r5
        bne0  r5, delay1            ;
        decr  r4
        bne0  r4, delay1            ;
            ;pop  r6
            ;pop  r5
            pop   r4
       pop r2
        rts
;--------message-------------------------
firemsg         db          "   STROKA1$$"
helpmsg         db          "           STROKA2$$"

end
```

## 1.2. Команды на языке Assembler, используемые в микропроцессоре P16 (ASMP 16)

### Команды пересылки данных

**LDA -** Загрузить аккумулятор содержимым ячейки памяти**.**

Команда пересылает содержимое ячейки памяти в аккумулятор. В соответствии с записываемыми в аккумулятор данными устанавливаются биты в регистре признаков.

**LDX -** Загрузить индексный регистр содержимым ячейки памяти**.**

Команда пересылает содержимое ячейки памяти в индексный регистр. В соответствии с записываемыми данными устанавливаются разряды регистра признаков.

**STA -** Записать содержимое аккумулятора в ячейку памяти.

Переместить содержимое аккумулятора в ячейку памяти. Содержимое аккумулятора остается неизменным.

**STX -** Записать содержимое индексного регистра в ячейку памяти.

Переместить содержимое индексного регистра в ячейку памяти. Содержимое индексного регистра не изменяется.

**TAX -** Записать содержимое аккумулятора в индексный регистр.

Записать в индексный регистр содержимое аккумулятора. Содержимое аккумулятора не изменяется.

**TXA -** Записать содержимое индексного регистра в аккумулятор

Записать в аккумулятор содержимое индексного регистра. Содержимое индексного регистра не изменяется.

**CLR -** Обнулить.

В аккумулятор, индексный регистр или ячейку памяти записывается ноль.

**RSP -** Инициализация указателя стека.

Установить указатель стека на вершину стека.

### Команды передачи управления

Команды передачи управления включают в себя команды безусловного перехода, команды вызова подпрограммы и возврата из нее, а также команды условных переходов (ветвления) по различным условиям (состояниям флагов) и по состоянию битов. Команды ветвления вызывают переход по состоянию любого бита из первых 256 ячеек памяти. Эти команды имеют длину три байта и являются комбинацией прямой и относительной адресации. Прямо адресуется бит для тестирования, адрес которого содержится во втором байте команды. Третий байт представляет собой знаковое смещение для перехода по состоянию бита. Процессор вычисляет адрес перехода, складывая содержимое программного счетчика и третьего байта команды, если условие перехода выполняется. Условие тестирования битов содержится в коде операции. Диапазон перехода – от - 128 до +127 байт от адреса следующей за командой

ячейки памяти. Процессор, кроме того, записывает тестируемый бит в бит переноса регистра условий (CC).

**BRA -** Безусловный переход.

Команда осуществляет безусловный переход по адресу, вычисляемому по приведенной ниже формуле, где Rel – относительное смещение, которое содержится в последнем байте кода команды.

PC ←(PC)+$0002+Rel.

**BRN -** Нет перехода.

Команда используется как двухбайтовый вариант команды NOP (нет операции) при отладке программ, когда нужно отменить действие команды перехода, и выполняется за 3 цикла. Действие этой команды противоположно действию команды BRA.

**BRCLR -** Переход, если бит n ячейки памяти равен 0.

PC ← (PC)+$0003+Rel, если бит n ячейки M равен нулю.

Команда проверяет бит n ячейки памяти M на равенство нулю. Переход осуществляется, если проверяемый бит равен нулю. Бит C регистра признаков устанавливается в состояние тестируемого бита.

**BRSET -** Переход, если бит n ячейки памяти равен 1.

PC ←(PC)+$0003+Rel, если бит n ячейки M равен единице.

Команда проверяет бит n ячейки памяти M на равенство единице. Переход осуществляется, если проверяемый бит равен единице. Бит C регистра признаков устанавливается в состояние тестируемого бита.

**BHI -** Переход, если больше.

PC←(PC)+$0002+Rel, если (C)+(Z)=0 т.е. если (AC)>(M)

Возникает переход, если оба бита C и Z регистра признаков сброшены в ноль. Когда команда BHI используется сразу после выполнения команд CMP или SUB, возникает переход, если беззнаковое число, содержащееся в аккумуляторе, больше, чем беззнаковое число в ячейке памяти M.

**BLS -** Переход , если меньше или равно.

PC¬"(PC)+$0002+Rel, если [(C)+(Z)]=1 т.е. если (AC)≤(M)

Возникает переход, если бит C или Z регистра признаков установлен в единицу. Когда команда BLS используется сразу после выполнения команд CMP или SUB, возникает переход, если беззнаковое число, содержащееся в аккумуляторе, меньше или равно беззнакового числа в ячейке памяти M.

**BCC -** Переход, если сброшен флаг переноса.

PC←(PC)+$0002+Rel, если (C)=0

Команда BCC является полным аналогом команды BHS. Проверяется состояние бита C регистра признаков. Переход возникает, если бит C сброшен.

**BHS -** Переход, если больше или равно.

Команда BHS является полным аналогом команды BCC. Мнемонику BCC принято использовать после команд CMP и SUB, так как переход в данном случае возникает, если беззнаковое число в аккумуляторе больше или равно беззнакового числа, содержащегося в ячейке памяти M.

**BCS -** Переход, если установлен флаг переноса.

PC←(PC)+$0002+Rel, если (C)=-1

Команда BCS является полным аналогом команды BLO. Тестируется состояние бита C регистра признаков. Если бит C установлен в единицу, возникает переход.

**BLO -** Переход, если меньше.

PC←(PC)+@0002+Rel, если (C)=1 т.е. если (ACCX)<(M)

Команда BLO является полным аналогом команды BCS. Если команда BLO выполняется сразу после выполнения команд CMP или SUB, переход возникает, если беззнаковое число, содержащееся в аккумуляторе, меньше беззнакового числа, содержащегося в ячейке памяти M.

**BNE -** Переход, если не равно.

PC←(PC)+$0002+Rel, если (Z)=0

Тестируется состояние бита Z регистра условий. Возникает переход, если бит Z сброшен. Следуя после сравнения или вычитания, BNE вызовет переход, если аргументы не равны.

**BEQ -** Переход, если равно.

PC←(PC)+$0002+Rel, если (Z)=1

Тестируется состояние бита Z регистра условий. Возникает переход, если бит Z установлен. Следуя после сравнения или вычитания, BEQ вызовет переход, если аргументы равны.

**BHCC - ** Переход, если сброшен флаг переноса из младшей тетрады.

PC←(PC)+$0002+Rel, если (H)=0

Тестируется состояние бита H регистра условий. Возникает переход, если бит H сброшен. Команда используется при работе с числами в BCD формате.

**BHCS - ** Переход, если установлен флаг переноса из младшей тетрады.

PC←(PC)+$0002+Rel, если (H)=1

Тестируется состояние бита H регистра условий. Возникает переход, если бит H установлен. Команды используется при работе с числами в BCD формате.

**BPL -** Переход, если плюс.

PC←(PC)+$0002+Rel, если (N)=0

Тестируется состояние бита N регистра условий. Возникает переход, если бит N сброшен.

**BMI -** Переход, если минус.

PC←(PC)+$0002+Rel, если (N)=1

Тестируется состояние бита N регистра условий. Возникает переход, если бит N установлен.

**BMC -** Переход, если бит маскирования прерываний сброшен.

PC←(PC)+$0002+Rel, если (I)=0

Тестируется состояние бита I регистра условий. Возникает переход, если бит I сброшен (то есть прерывания разрешения).

**BMS -** Переход, если бит маскирования прерываний установлен.

PC←(PC)+$0002+Rel, если (I)=1

Тестируется состояние бита I регистра условий. Возникает переход, если бит I установлен (то есть прерывания запрещены).

**BIL -** Переход, если вход запроса прерывания в 0.

PC←(PC)+$0002+Rel, если IRQ\=0

Тестируется состояние выводы микропроцессора запроса внешнего прерывания. Возникает переход, если на входе нулевой логический уровень.

**BIH -** Переход, если вход запроса прерывания в 1.

PC←(PC)+$0002+Rel, если IRQ\=1

Тестируется состояние выводы микропроцессора запроса внешнего прерывания. Возникает переход, если на входе логическая единица.

**BSR -** Переход на подпрограмму.

Программный счетчик увеличивается на 2 от адреса кода операции BSR, таким образом в PC оказывается адрес следующей за BSR команды, который является адресом возврата из подпрограммы. Младший байт программного счетчика помещается в стек. Указатель стека уменьшается на 1. Затем старший байт программного счетчика помешается в стек, указатель стека уменьшается на 1. Производится переход по адресу, определяемому смещением.

**JMP -** Длинный безусловный переход.

PC ← Эффективный адрес

Осуществляется переход по эффективному адресу. Эффективный адрес вычисляется в соответствии с правилами для расширенного, прямого и индексного режимов адресации.

**JSR -** Длинный вызов подпрограммы.

Программный счетчик увеличивается на n, то есть до адреса следующей за JSR команды. Затем PC сохраняется в стеке. Не используемые биты программного счетчика сохраняются как единицы. Указатель стека указывает на следующую свободную ячейку памяти. Осуществляется переход по эффективному адресу. Эффективный адрес вычисляется в соответствии с правилами для расширенного, прямого и индексного режимов адресации.

**SWI -** Программное прерывание

Содержимое программного счетчика увеличивается на единицу. Программный счетчик, индексный регистр, аккумулятор и регистр признаков помещаются в стек. Содержимое указателя стека уменьшается на единицу каждый раз после того, как в стек помещен байт. Бит маскирования прерываний устанавливается в единицу. В программный счетчик записывается адрес, хранящийся в векторе прерывания SWI (адрес расположен по адресу n-0002 и n-0003, где n-адрес, соответствующий единичному состоянию всех адресных выходов микропроцессора.). Команда SWI не маскируется установкой бита I регистра признаков.

**RTI -** Возврат из подпрограммы обработки прерывания

Регистр признаков, аккумулятор, индексный регистр и программный счетчик восстанавливаются из стека. Бит I сбрасывается, если соответствующий бит регистра признаков в стеке был нулевым.

Биты регистра признаков принимают значения в соответствии с байтом, взятым из стека.

**RTS -** Возврат из подпрограммы

Указатель стека увеличивается на единицу. Содержимое ячейки памяти, на которую указывает указатель стека, записывается в старший байт программного счетчика. Указатель стека еще раз увеличивается на единицу. Содержимое ячейки памяти, на которую указывает указатель стека, записывается в младший байт программного счетчика.

**Арифметические и логические команды, битовые операции**

**ADD** - Сложить содержимое ячейки памяти с аккумулятором

AC←(AC)+(M)

Сложить содержимое M и содержимое AC и разместить в AC.

**ADC** - Сложить содержимое ячейки памяти с аккумулятором и флагом переноса.

AC←(AC)+(M)+(C)

Сложить содержимое бита C и сумму содержимого M и содержимого AC и разместить в AC.

**SUB** - Вычесть.

AC←(AC)-(M)

Вычесть содержимое M из AC и разместить результат в AC.

**SBC** - Вычесть содержимое ячейки памяти из аккумуляторе с флагом заема.

AC←(AC)-(M)-(C)

Вычесть содержимое M из AC и разместить результат в AC.

**AND** - Операция И содержимого ячейки памяти и аккумулятора.

AC←(AC)*(M)

Произвести логическое умножение между содержимым аккумулятора и содержимым M и разместить результата в AC.

**ORA** - Операция ИЛИ содержимого ячейки памяти и аккумулятора.

AC←(AC)+(M)

Произвести логическое сложение между содержимым аккумулятора и содержимым M и разместить результата в AC.

**CMP** - Арифметическое сравнение аккумулятора и ячейки памяти.

(AC)-(M)

Сравнить содержимое M и AC и установить биты в регистре признаков. Содержимое AC и M не изменяется.

**CPX** - Арифметическое сравнение индексного регистра и ячейки памяти.

(X)-(M)

Сравнить содержимое M и X и установить биты в регистре признаков. Содержимое X и M не изменяется.

**BIT** - Поразрядное сравнение ячейки памяти с аккумулятором.

(ACCX)*(M)

Производит сравнение содержимого AC и M посредством операции логическое И и устанавливает соответственно результату биты в регистре признаков. Ни содержимое AC, ни содержимое M не изменяются

**MUL** - Умножение.

X : A←=X x A

Умножается 8-битное число в индексном регистре X на 8-битное число в аккумуляторе. 16-битный результат помещается в индексный регистр и аккумулятор. В индексном регистре старшие 8 бит результата, в аккумуляторе младшие 8 бит.

**INC** - Инкремент на 1

AC←(AC)+$01 или M←(M)+$01 или X←(X)+$01

Добавляет единицу к содержимому аккумулятора, индексного регистра или ячейки памяти. Биты N и Z регистра признаков устанавливаются или сбрасываются в соответствии с результатом. Бит C регистра признаков не изменяются, поэтому после команды INC имеют смысл только следующие команды перехода BEQ, BNE, BPL и BMI.

**DEC** - Декремент на 1

AC←(AC)-$01 или M←(M)-$01 или X←(X)-$01

Вычитает единицу из содержимого аккумулятора, индексного регистра или ячейки памяти. Биты N и Z регистра признаков устанавливаются или сбрасываются в соответствии с результатом. Бит C регистра признаков не изменяются, поэтому после команды DEC имеют смысл только следующие команды перехода:

**COM** - Дополнение.

AC←(AC\)=$FF-(AC) или M←(M\)=$FF-(M) или X←X\=$FF-(X)

Содержимое аккумулятора, индексного регистра или ячейки памяти заменяется на его дополнение до 1. Каждый бит заменяется на его дополнение, то есть инвертируется.

**NEG** - Дополнение до двух (инверсия знака).

AC←(AC) или M←(M) или X←X

Содержимое аккумулятора, индексного регистра или ячейки памяти заменяется на его дополнение до 2. Заметим, что величина $80 не изменяется.

**ROL** - Циклический сдвиг влево через флаг переноса

C←b7-...-b0←C

Все биты аккумулятора, индексного регистра или ячейки памяти сдвигаются влево. Бит 0 загружается содержимым бита C. Бит C загружается содержимым старшего бита аккумулятора, индексного регистра или ячейки памяти.

**ROR** - Циклический сдвиг вправо через флаг переноса

C→b7-...-b0→C

Все биты аккумулятора, индексного регистра или ячейки памяти сдвигаются вправо. Бит 7 загружается содержимым бита C. Бит C загружается содержимым младшего бита аккумулятора, индексного регистра или ячейки памяти.

**LSL** - Логический сдвиг влево

C←b7-...-b0←0

Все биты аккумулятора, индексного регистра или ячейки памяти сдвигаются влево. Бит 0 загружается нулем. Бит C загружается содержимым старшего бита аккумулятора, индексного регистра или ячейки памяти.

**LSR** - Логический сдвиг вправо.

0→b7-...-b0→C

Все биты аккумулятора, индексного регистра или ячейки памяти сдвигаются вправо. Бит 7 загружается нулем. Бит C загружается содержимым младшего бита аккумулятора, индексного регистра или ячейки памяти.

**ASR** - Арифметический сдвиг вправо

←→b7-...-b0→C

Все биты аккумулятора, индексного регистра или ячейки памяти сдвигаются влево. Бит 7 остается неизменным. Бит C загружается содержимым младшего бита аккумулятора, индексного регистра или ячейки памяти.

**TST** - Проверка на отрицательность и равенство нулю

(AC)-$00 или (X)-$00 или (M)-$00

Установить биты N и Z в регистре признаков в соответствии с содержимым аккумулятора, индексного регистра или ячейки памяти. Содержимое аккумулятора, индексного регистра или ячейки памяти не изменяется.

**BSET** n (n=0...7) - Установить бит n

Mn=1

Устанавливается в 1 бит n (n=7,6,5,4,3,2,1,0) ячейки памяти M, другие биты не изменяются. Ячейкой памяти может быть ОЗУ или I/O регистр от адреса $0000 до $00FF (то есть используется прямая адресация).

**BCLR** n (n=0...7) - Сбросить бит n

Mn=0

Сбрасывается бит n (n=7,6,5,4,3,2,1,0) ячейки памяти M, другие биты не изменяются. Ячейкой памяти может быть ОЗУ или I/O регистр от адреса $0000 до $00FF (то есть используется прямая адресация).

**SEC** - Установить флаг переноса

C бит ←1

Установить бит C регистра признаков. Команда SEC может применятся для предустановки бита C перед использованием команд сдвига.

**CLC** - Сбросить флаг переноса

C бит ←0

Сбрасывается бит C регистра признаков. Команда SEC может применятся для предустановки бита C перед использованием команд сдвига.

**SEI** - Установить флаг маскирования прерываний.

I бит←1

Устанавливается бит маскирования прерываний в регистре признаков. Когда бит C установлен, прерывания запрещены.

**CLI** - Сбросить флаг маскирования прерываний

I бит←0

Сбрасывается бит маскирования прерываний в регистре признаков. Когда бит C сброшен, прерывания разрешены. На очистку бита требуется задержка на один цикл, таким образом, если прерывания до выполнения команды CLI были запрещены, то следующая за CLI команда всегда выполняется.

**Прочие команды**

**NOP -** Нет операции

Эта однобайтная команда вызывает только увеличение программного счетчика на единицу. Другие регистры не изменяются.

**STOP** - Останов

Выполнение команды приводит к уменьшению энергопотребления.

1) Сбрасывается определитель таймера;
2) Запрещаются прерывания от таймера;
3) Сбрасывается флаг прерывания от таймера;
4) Разрешаются внешние прерывания;
5) Останавливается тактовый генератор.

Когда поступает внешний сигнал RESET\ или IRQ\, включается тактовый генератор, после задержки на инициализацию, которая длится 1920 циклов процессора, защелкиваются вектор прерывания и вектор начальной установки, и запускается на выполнение процедуры обработки RESET или IRQ, в зависимости от поданного сигнала.

Флаг I сбрасывается.

**WAIT** - Ожидание

Выполнение команды приводит к уменьшению энергопотребления. Таймер, делитель таймера и другая встроенная периферия продолжают работать, так как они являются потенциальными источниками прерываний. WAIT разрешает прерывания сбросом бита I регистра признаков и останавливает цепи синхронизации процессора.

Когда поступает внешний сигнал RESET\, IRQ\ или прерывание от встроенной периферии, включаются цепи синхронизации процессора, защелкиваются вектор прерывания и вектор начальной установки, и запускается на выполнение процедура, в зависимости от поданного сигнала.

## Литература


1. Чжен Ю., Гибсон Т. Микропроцессоры семейство 8086/8088. – М.: Радио и связь, 1987.
2. Микропроцессорный комплект К1810/ Под ред. Ю.М.Казаринова – М.: Высшая школа, 1990.
3. Григорьев В.Л. Программирование однокристальных микропроцессоров – М.: Энергоатомиздат, 1987.
4. Микро- и миниЭВМ. Балашов Е.П., Григорьев В.П., Петров Г.А. – П.: Энергоатомиздат. 1989.
5. Проектирование радиоэлектронной аппаратуры на микропроцессорах, Алексеенко А.Т., Галицин А.А., Иванников А.Д.
6. Калабеков Б.А. Микропроцессоры и их применение в системах передачи и обработки сигналов. – М.: Радио и связь, 1988.
7. Основы проектирования микропроцессорных устройств. – М.: Энергоатомиздат, 1987.
8. Хвощ С.Т., Варианский Н.Н., Попов Е.А. Микропроцессоры и микроЭВМ в системах автоматического управления, Справочник. – Л.: Машиностроение, 1987.

Микропроцессоры и микропроцессорные системы
Методические указания к выполнению лабораторных работ. Вводная часть. Для студентов специальности 552801.01 «Вычислительные машины, комплексы, системы и сети»
Составители: ***С.А.Алымкулов, С.М.Тентиева, Г.У.Тультемирова***

Тех. редактор ***Субанбердиева Н.Е.***

Подписано к печати 23.05.2011 г. Формат бумаги 60x84$^{1}/_{16}$.
Бумага офс. Печать офс. Объем 1,5 п.л. Тираж 50 экз. Заказ 205.

Бишкек, ул. Сухомлинова, 20. ИЦ “Текник” КГТУ им. И.Раззакова, т.: 54-29-43
e-mail: beknur@mail.ru