Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра радиотехнических систем

В.Н. Левкович, А.А. Кащеев

ЦИФРОВЫЕ И МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА

Лабораторный практикум

для студентов специальностей
I-39 01 02 «Радиоэлектронные системы»,
I-39 01 03 «Радиоинформатика»,
I-39 01 04 «Радиоэлектронная защита информации»
дневной формы обучения

В 2-х частях

Часть 2

Минск 2006

УДК 681.325.5-181.48(075.8)
ББК 32.973.26-04 я 73
Л 37

Р е ц е н з е н т:
доцент кафедры сетей и устройств телекоммуникаций БГУИР,
канд. техн. наук И.И. Астровский

Левкович В.Н.

Л 37 Цифровые и микропроцессорные устройства: Лаб. практикум для студ. спец. I-39 01 02 «Радиоэлектронные системы», I-39 01 03 «Радиоинформатика», I-39 01 04 «Радиоэлектронная защита информации» дневной формы обуч.: В 2 ч. Ч. 2 / В.Н. Левкович, А.А. Кащеев. – Мн.: БГУИР, 2006. – 36 с.: ил.

ISBN 985-444-945-9 (ч. 2)

Во вторую часть практикума вошли теоретические сведения, задания и контрольные вопросы к лабораторным работам по трем темам: «Алгоритмы и программирование процедур формирования импульсных сигналов с заданными временными параметрами», «Алгоритмы и программирование процедур отображения цифровой информации» и «Алгоритмы и программирование процедур ввода данных с клавиатуры».

УДК 681.325.5-181.48(075.8)
ББК 32.973.26-04 я 73

Часть 1 издана в БГУИР в 2005 г.

ISBN 985-444-945-9 (ч. 2)
ISBN 985-444-818-5

СОДЕРЖАНИЕ

Лабораторная работа № 2

АЛГОРИТМЫ И ПРОГРАММИРОВАНИЕ ПРОЦЕДУР ФОРМИРОВАНИЯ ИМПУЛЬСНЫХ СИГНАЛОВ С ЗАДАННЫМИ ВРЕМЕННЫМИ ПАРАМЕТРАМИ

Цель работы:

1) изучение принципов разработки программ для микропроцессорных систем на Ассемблере;

2) приобретение навыков программирования процедур формирования импульсных сигналов с заданными временными параметрами.

2.1. Составление схем алгоритмов

Современные вычислительные машины способны выполнять широкий круг задач по получению, передаче, хранению, переработке информации, принятию решений и т. д. Но все эти действия должны быть заранее подготовлены, запрограммированы человеком.

Создание программы основывается на алгоритме решения задачи, в соответствии с ним создается последовательность команд, которая и составляет программу.

Под алгоритмом понимается конечный набор действий для выполнения некоторой процедуры, удовлетворяющий трем основным требованиям: массовости, детерминированности и результативности.

Требование массовости предполагает, что предписание должно обеспечивать выполнение не одной конкретной процедуры, а быть пригодным для реализации класса однородных процедур. Бессмысленно писать программу для получения суммы двух констант, но имеет смысл программа, определяющая сумму двух переменных, которые могут принимать множество значений в некотором диапазоне.

Детерминированность означает, что действия, образующие алгоритм, должны быть однозначно понимаемы, т.е. при одинаковых исходных данных независимо от исполнителя должна обеспечиваться одинаковость результатов.

Результативность обеспечивает конечность применения указаний. Результат должен быть получен за конечное число шагов либо за конечное число шагов должно быть получено указание о неприменимости данного алгоритма для решения поставленной задачи.

Алгоритм может быть описан в виде набора предложений, отражающих последовательность действий исполнителя, или в виде схемы, состоящей из ряда блоков, обозначающих действия исполнителя. Последний вариант хотя и имеет большую трудоемкость, но отличается большей наглядностью.

Для успешного решения задачи на вычислительной машине (микроконтроллере) разработчик должен пройти следующих семь этапов: 1) постановка задачи; 2) выбор приемлемого алгоритма; 3) определение типов входных и выходных данных; 4) распределение аппаратных ресурсов микроконтроллера, т.е. портов ввода/вывода и периферийных устройств на кристалле; 5) проектирование и анализ решения, в том числе составление схем, описаний и пр.; 6) кодирование алгоритма на языке программирования; 7) проверка и отладка программы.

Процесс решения задачи носит, как правило, итерационный характер. Это означает, что, получив решение, разработчик часто бывает вынужден вернуться вновь к третьему, второму и даже первому этапу.

По-видимому, первые рассмотренные выше четыре этапа относятся к тому, что принято называть «искусством разработки». Здесь успех в основном определяется опытом разработчика, его знанием объекта разработки, поэтому дать конкретные рекомендации не представляется возможным. Однако можно детальнее остановиться на составлении схем алгоритмов.

Изображение алгоритма решения задачи в виде схемы – важный этап подготовки задачи к решению на вычислительной машине. Схема позволяет разработчику адекватно представить работу программы. Кроме этого, схема алгоритма является одной из важных частей документации на разрабатываемую систему или устройство.

Схема алгоритма составляется из отдельных операторов. Различают шесть типов операторов, каждый из которых имеет один или несколько входов или один или несколько выходов (рис. 2.1). Стрелками обозначают направление хода действий.

Оператор в форме прямоугольника (рис. 2.1, *а*) символизирует выполнение каких-либо операций по обработке данных; текст внутри прямоугольника является кратким описанием этого процесса обработки. Например, если в схеме алгоритма содержится оператор «Очистка аккумулятора», то это означает, что на данном этапе работы машины аккумулятор должен быть обнулен. В таком случае для выполнения этой операции микроконтроллеру требуется одна машинная команда. Однако могут быть заданы сложные действия, для которых требуется целый набор команд.

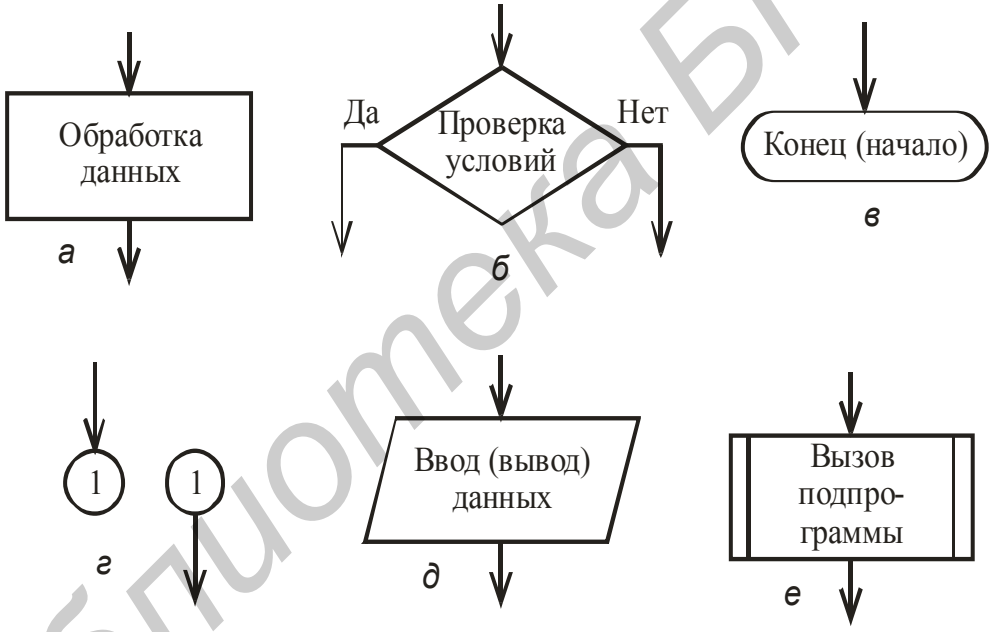

Рис. 2.1. Операторы обработки данных (*а*), проверки условия (*б*), начала или конца алгоритма (*в*), соединения (*г*), вода или вывода данных (*д*), вызова подпрограммы (*е*)

Оператор, имеющий форму ромба (рис. 2.1, *б*), используется для символического обозначения проверки выполнения какого-либо условия с целью принятия решения о направлении последующего хода вычислений. Внутри ромба описывается условие, подлежащее проверке в той точке схемы алгорит-

ма, где размещается данный оператор. Возможные результаты проверки указываются на линиях, выходящих из ромба. Для проведения подобной проверки требуется использование одной или нескольких команд микроконтроллера.

Оператор овальной формы используется для символического обозначения начала или конца алгоритма (рис. 2.1, *в*). Текст внутри овала, как правило, состоит из одного слова — «Начало», «Конец», «Выход», «Возврат» или имени подпрограммы.

В тех случаях, когда необходимо «разорвать» линию потока вычислений, идущую от одного оператора к другому, применяются так называемые соединители в виде окружности с указанной внутри нее цифрой или буквой (рис. 2.1, *г*). Наличие другого идентичного соединителя (с той же цифрой или буквой) означает, что прерванная в месте расположения первого соединителя линия продолжается с того места, где находится второй подобный соединитель. Использование соединителей упрощает внешний вид схемы алгоритма, что позволяет избежать пересечения линий и даёт возможность размещать схему алгоритма на нескольких страницах.

Для обозначения процедур ввода или вывода применяется оператор, имеющий форму параллелограмма (рис. 2.1, *д*). Внутри параллелограмма указывают обычно переменные, подлежащие вводу или выводу.

Оператор, изображенный на рис. 2.1, *е*, используется для обозначения вызова подпрограммы. Внутри него обычно помещается имя вызываемой подпрограммы.

Алгоритмы в зависимости от порядка следования операций бывают трех типов: линейные, ветвящиеся и циклические.

Схемы линейных алгоритмов не содержат операторов проверки условий, в них операторы располагаются строго друг за другом.

Ветвящиеся алгоритмы состоят из двух и более параллельных линейных ветвей, при этом ветвления реализуются через операторы проверки условий.

Циклический алгоритм предписывает многократные циклические проходы группы операторов.

Реальные схемы алгоритмов, как правило, представляют собой комбинации из трех перечисленных выше типов схем алгоритмов.

2.2. Общие правила Ассемблера

Обычно программа начинается со строки комментариев, в которых указывается, чья это программа и зачем написана. Строки комментариев могут быть в любом месте программы.

Далее после комментариев (если они есть) и перед первыми командами программы размещаются строки директив. Директивы – это указания Ассемблеру, как именно работать. Список директив Ассемблера довольно обширный, но мы на данном этапе будем использовать только самые необходимые. Комментарии и директивы не включаются в исполняемый код программы и, следовательно, не попадают в память программ. Строки директив также могут быть в любом месте программы, но некоторые директивы обязательно должны быть заданы до первой строки команд.

Рассмотрим обязательные элементы программы.

Директива LIST с опцией P указывает, с каким микроконтроллером идет работа.

Директива EQU присваивает символическому имени определенное выражение или значение. Присвоенное значение впоследствии в программе переопределить нельзя. Эта директива позволяет программисту оперировать в программе не физическими адресами регистров (ячеек памяти), а их условными именами, которые придумывает сам программист. Если при этом в условное имя закладывается физический смысл переменной, размещающейся в этом регистре, то его легче помнить.

Директива END означает, что текст программы закончился. Эта директива должна быть в последней строке программы.

Каждая строка программы может иметь до четырех разделов (полей) и содержать до 255 символов.

С первой позиции в строке начинается поле метки. Метка – это условное символическое имя, присваиваемое конкретной строке, а также переменной или константе. Метка должна начинаться с буквы или символа подчеркивания (_) и содержать до 32 символов букв или цифр. В поле метки прописные и строчные буквы различаются. Поле метки должно заканчиваться символом пробела, табуляции или конца строки.

Далее следует второе поле – мнемонический код команды. Оно начинается со второй позиции в строке, а если перед ним стоит метка, то от метки мнемоника должна быть отделена двоеточием, одним или более пробелом или символом табуляции.

В третьем поле задаются операнды. От мнемоники они отделяются одним или более пробелом или символом табуляции. Операндов должно быть столько, сколько требует формат команды. Между собой операнды разделяются запятой. Если команда предусматривает переменное количество операндов, то они считаются до конца строки или до четвертого поля (начало комментария).

Четвертое поле – комментарии. Начинаются с символа точки с запятой. От остальных полей отделяются одним или более пробелом или символом табуляции. Могут занимать всю строку с первой позиции.

Программа обязательно заканчивается директивой END.

2.3. Пример простейшей программы управления портами

Пусть требуется спроектировать устройство на микроконтроллере (МК) PIC16F628, которое обеспечивает мигание светодиода на время нажатия и удержания кнопки. Другими словами: пока нажата кнопка, светодиод мигает, при отпущенной кнопке светодиод погашен.

Решение. В этой задаче один источник информации и один объект управления. Источник информации представляет собой два нормально разомкнутых

контакта SB. При нажатии кнопки контакты замыкаются, при отпускании – размыкаются. Микроконтроллер должен определять состояния кнопки. Таких состояний два: «не нажата» и «нажата». Эти состояния можно условно описать одной булевой переменной x, которая может принимать два значения: 0 и 1. Таким образом, кнопка как источник даёт один бит информации. Для приёма этой информации достаточно задействовать лишь какой-либо один из входов МК. Сама по себе кнопка является электрически пассивным источником, и для определения ее состояния с помощью порта МК необходимо преобразовать состояние контактов в электрический сигнал напряжения. Наиболее просто это можно выполнить по схеме, изображенной на рис. 2.2.

В состоянии «не нажата» по порту МК будет читаться логическая единица, а в состоянии «нажата» — логический нуль. Учитывая, что в PIC16F628 все разряды порта B имеют подтягивающие внутренние резисторы, внешний резистор R можно не использовать, а кнопку подключить к выводу порта и к общему проводу. При этом в программе необходимо позаботиться о включении подтягивающих резисторов. Подключим в проектируемом устройстве кнопку SB к порту RB0.

Объектом управления в устройстве является светодиод. Его состояние «включен» обеспечивается пропусканием тока $I_Д$ силой в несколько миллиампер (обычно 5–10 мА). Поскольку нагрузочная способность портов МК 20 мА, то светодиод можно подключать без дополнительного усилителя непосредственно к порту, предусмотрев лишь токоограничивающий резистор R, как показано на рис. 2.3. Сопротивление резистора можно рассчитать по формуле $R = (E^1 – U_{VD}) / I_Д$, где $E^1 = 5$ В — уровень логической единицы, $U_{VD} = 2{,}4$ В — падение напряжения на открытом светодиоде. Зададимся $I_Д = 5$ мА, тогда $R = (5–2{,}4) / (5 \cdot 10^{-3}) = 2{,}6 / (5 \cdot 10^{-3}) = 2600 / 5 = 520$ Ом.

Выберем для управления светодиодом порт RA0.

Для обеспечения требований к временным параметрам устройства необходимо выбрать частоту и вариант тактирования работы МК. Выберем вариант синхронизации с помощью кварцевого резонатора, имеющего резонансную

частоту 4 МГц. Таким образом, длительность машинного цикла (время выполнения большинства команд) составит 1 мкс.

Составим схему алгоритма управляющей программы (рис. 2.4).

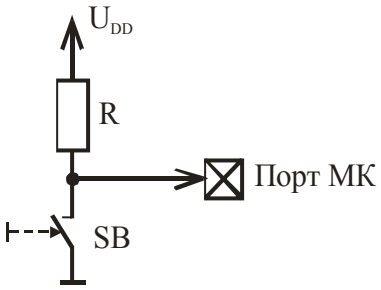

Рис. 2.2. Схема подключения кнопки к порту МК

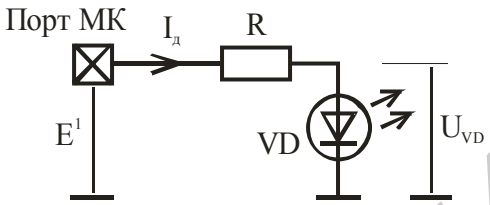

Рис. 2.3. Схема подключения светодиода к МК

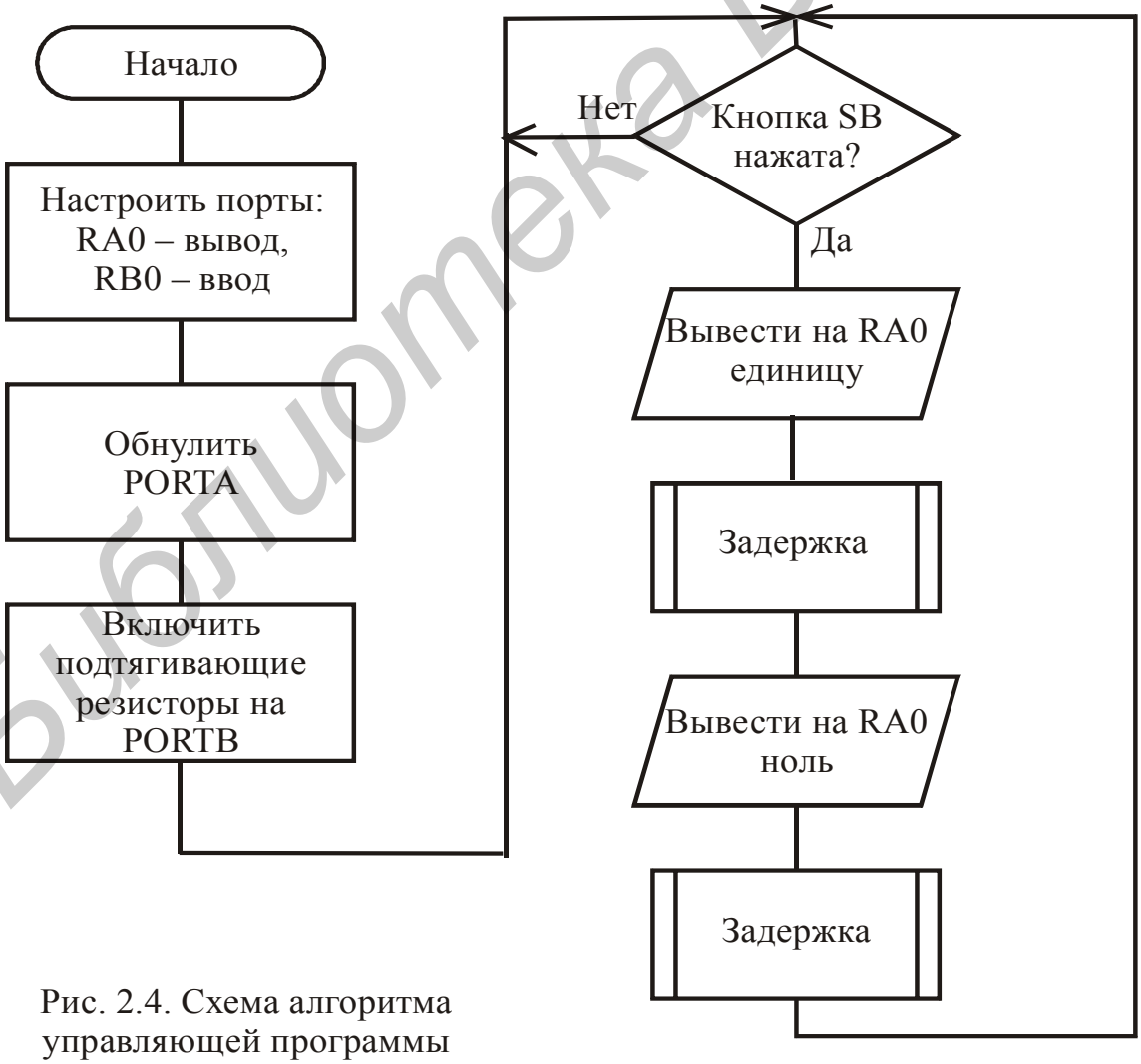

Рис. 2.4. Схема алгоритма управляющей программы

От схемы алгоритма легко перейти к программе.

```
; Учебная программа №1.
        LIST p=16f628     ; Директива Ассемблеру с указанием типа МК PIC16F628.
  ; Определяем имена констант, используемых в этой программе.
  ; После этого определения можно вместо шестнадцатеричного адреса регистра
  ; или номера разряда в регистре указывать его имя, которое сами придумаем.
  ; Это облегчает чтение и понимание работы программы.
STATUS      EQU     03h   ; Символ  h  в конце записи числа является
RP0         EQU     05h   ; указателем шестнадцатеричной
TRISA       EQU     05h   ; системы счисления.
TRISB       EQU     06h   ;
PORTA       EQU     05h   ;
PORTB       EQU     06h   ;
RA0         EQU     0h    ;
RB0         EQU     0h    ;
OPTION_REG  EQU     01h   ;
RBPU        EQU     07h   ;
        ; Настраиваем порты
        BSF   STATUS,RP0      ; Включаем банк памяти 1 для доступа к регистрам TRIS.
        BCF   TRISA,RA0       ; Линия RA0 – на вывод  данных.
        BSF   TRISB,RB0       ; Линия RB0 – на ввод данных.
        BCF   OPTION_REG,RBPU ; Включаем подтягивающие резисторы на входах
                              ; порта B, запрограммированных как входы.
        BCF   STATUS,RP0      ; Включаем банк памяти 0 для доступа к портам.
        CLRFPORTA             ; Обнуляем порт A (гасим светодиод).
                              ; Ожидаем нажатия кнопки SB.
Wait    BTFSC PORTB,RB0       ; В цикле проверяем линию RB0.
        GOTO    Wait          ; При нажатии кнопки на линии будет читаться ноль,
                              ; что вызовет пропуск команды GOTO и выход из цикла.
        BSF    PORTA,RA0      ; Зажигаем светодиод.
        CALL     Delay        ; Вызываем подпрограмму задержки.
        BCF   PORTA,RA0       ; Гасим светодиод.
        CALL     Delay        ; Снова задержка.
        GOTO     Wait         ; Переход в точку контроля нажатия кнопки.
Delay       ;Подпрограмма задержки.
            ; Методы формирования задержек рассматриваются  ниже.
            ; В простейшем варианте это цепочка команд NOP.
        NOP
        NOP
        NOP
        RETURN    ; Возврат из подпрограммы задержки.
        END       ; Конец программы.
```

2.4. Программирование задержек

Процедура задержки является одной из самых часто используемых, особенно в алгоритмах управления, а также формирования и анализа сигналов. В рассматриваемом микроконтроллере она может быть реализована тремя способами:

- линейной цепочкой пустых команд NOP;
- многократным циклическим повторением цепочки команд, в том числе NOP;
- с применением счетчика-таймера.

Первый способ самый простой, но он пригоден только для реализации коротких задержек. Величина задержки рассчитывается по формуле $\Delta t = t_{кц} N$, где $t_{кц}$ – длительность командного цикла (при тактовой частоте 4МГц она равна 1 мкс), N – количество команд NOP в цепочке.

По второму способу организуется конечное число проходов цепочки команд. Для этого отводится одна из ячеек памяти данных, в которой организуется счетчик циклов. В ячейку заносится константа, которая определяет количество повторений. После каждого прохода из счетчика циклов вычитается единица. Выход из цикла обеспечивается по нулевому значению счетчика. Максимальное значение счетчика циклов ограничивается разрядностью ячейки. Оно не может быть больше 255. Задержка, обеспечиваемая этим способом, рассчитывается по формуле $\Delta t = t_{цк} M$, где $t_{цк}$ – время выполнения цепочки команд, M – количество повторений.

Рассмотрим пример циклической программы задержки.

```
N       EQU     20h     ; Резервируем ячейку памяти по адресу 20h под счетчик циклов.
        MOVLW   07h     ; Заносим число 7 в рабочий регистр,
        MOVWF   N       ; а затем переписываем его в счетчик циклов.
Cycl                    ; Начало циклического участка программы.
        NOP             ; Команды NOP для удлинения времени выполнения участка,
        NOP             ; в реальной программе могут отсутствовать.
        NOP             ;
        DECFSZ  N,1     ; Вычитание из счетчика циклов единицы и проверка условия
        GOTO    Cycl    ; выхода из цикла.
```

Для организации больших задержек могут быть использованы вложенные циклы. Рассмотрим фрагмент программы, реализующий задержку с двукратным вложенным циклом.

```
N0      EQU     20h   ; Ячейка памяти по адресу 20h под счетчик внутренних циклов.
N1      EQU     21h   ; Ячейка памяти по адресу 21h под счетчик внешних циклов.
        MOVLW   0Eh   ; Задаем количество внешних циклов
        MOVWF   N1    ;       N1=14.
Cycl_1                ; Начало внешнего цикла.
        MOVLW   08h   ; Задаем количество внутренних циклов
        MOVWF   N0    ;       N0=8.
Cycl_0                ; Начало внутреннего цикла.
        DECFSZ  N0,1  ; Вычитание из счетчика внутренних циклов единицы
        GOTO    Cycl_0      ; и проверка условия выхода из цикла.
        DECFSZ  N1,1  ; Вычитание из счетчика внешних циклов единицы
        GOTO    Cycl_1      ; и проверка условия выхода из цикла.
```

Недостатком рассмотренных способов организации задержек является то, что микроконтроллер занят только задержками и не может выполнять других полезных действий.

Наиболее совершенный способ реализации задержек — с помощью счетчика-таймера TMR0. Как и в предыдущем случае, задержка отмеряется по обнулению счетчика, в который предварительно записывается начальная константа К. Но в отличие от предыдущего случая счетчик переключается не программно, а аппаратно и обнуление счетчика фиксируется через систему прерываний. Так что в промежутках времени от запуска задержки до ее окончания микроконтроллер может выполнять любые другие полезные действия. TMR0 в процессе работы инкрементируется с частотой командных циклов. Его обнуление происходит через переполнение. Таким образом, реализуемая задержка $\Delta t = (255–K)\, t_{кц}$, где $t_{кц}$ – длительность командного цикла.

Увеличить время задержки можно, подключив к входу TMR0 предварительный делитель и задав с помощью регистра OPTION_REG его коэффициент деления N. Тогда реализуемая задержка $\Delta t = N(255–K)\, t_{кц}$.

Рассмотрим реализацию генератора прямоугольных импульсов (меандра) на базе таймера TMR0. Пусть выходной сигнал формируется на выводе RA0, а

частота синхронизации микроконтроллера равна 4 МГц. Тогда длительность командного цикла $t_{кц}$ = 1 мкс.

```
; Пример программной  реализации генератора прямоугольных импульсов.

        LIST  p=16f628
        #INCLUDE        p16f628.inc         ; Включаем в программу файл p16f628.inc,
                        ; в котором описаны имена регистров МК и их физические адреса.

; Определяем константы и адреса регистров, флагов и переменных.
K       EQU  0AAh       ; Начальное значение таймера TMR0=170.
W_TEMP  EQU  20h        ; Ячейка памяти для сохранения регистра W
                        ; на время обработки прерывания.
STATUS_TEMP  EQU 21h    ; Ячейка памяти для сохранения регистра
                        ; STATUS на время обработки прерывания.
        ; Начало программы.
        ORG 0           ; Директива Ассемблеру установить программный счетчик в нуль.
        GOTO    Begin   ; Перейти к началу основной программы.
        ORG 4           ; Установить программный счетчик на адрес вектора прерывания.
        GOTO    Int     ; Перейти на программу обработки прерывания.
Begin   ; Начало основной программы.

; Выполняем настройки режимов работы функциональных узлов микроконтроллера.

        BSF     STATUS,RP0      ; Включаем банк регистров 1.
        BCF     TRISA,RA0       ; Порт RA0 включаем на выдачу сигналов.
        BCF     OPTION_REG,T0CS ;Включаем внутреннее тактирование TMR0.
        BSF     INTCON,T0IE     ; Разрешаем прерывания по переполнению TMR0.
        BCF     STATUS,RP0      ; Включаем банк регистров 0.
        CLRF    PORTA           ; Обнуление выходного сигнала.
        MOVLW   K               ; Загрузка TMR0
        MOVWF   TMR0            ; начальным значением.
        BSF     INTCON,GIE      ; Разрешаем прерывания от переполнения TMR0.
        ; Здесь могут располагаться команды,
        ; выполняющие некоторые полезные действия, например,
        ; обслуживание индикатора, ввод информации с клавиатуры и т.д.
Wait    GOTO Wait       ; Зацикливаем участок программы в ожидании прерывания.
        ; Выход отсюда возможен только по прерыванию.
Int     ; Начало подпрограммы обработки прерывания. Здесь на каждое
        ; прерывание производится инверсия порта А. Поскольку на выход
        ; настроен только вывод RA0, то инвертировать будет только он.
        MOVWF   W_TEMP  ; Сохранить W в регистре TEMP.
```

```
SWAPF     STATUS,0
MOVWF     STATUS_TEMP        ; Сохранить STATUS в регистре TEMP.
BCF       STATUS,RP0         ; Включаем банк регистров 0.
COMF      PORTA,1            ; Инверсия выходной линии.
MOVLW     K                  ; Запись начальной константы K
MOVWF     TMR0               ; в таймер TMR0.
BCF       INTCON,T0IF        ; Сброс флага прерывания по переполнению TMR0.
                  ; Восстанавливаем STATUS и W.
SWAPF     STATUS_TEMP,0      ;
MOVWF     STATUS             ;
SWAPF     W_TEMP,1           ;
SWAPF     W_TEMP,0           ;
RETFIE            ; Вернуться из прерывания, разрешить следующее прерывание.
END               ; Конец программы.
```

2.5. Лабораторное задание

1. Изучить теоретический материал, изложенный в подразд. 2.1. …2.4.

2. Получить задание у преподавателя.

3. Составить схему алгоритма, написать и отладить в среде MPLAB программу генерирования на заданном выводе микроконтроллера периодический импульсный сигнал с заданной длительностью импульсов и заданной паузой между импульсами.

Для формирования длительности импульса использовать метод многократного циклического повторения цепочки команд, а для формирования паузы – метод с применением счетчика-таймера.

Длительности импульсов рекомендуется задавать в пределах от 30 до 1000 мкс, длительности пауз – от 1 до 64 мс.

Частоту тактирования микроконтроллера принять равной 4 МГц.

4. Оформить отчет, который должен содержать: титульный лист, цель работы, вариант лабораторного задания, разработанную схему алгоритма и листинг программы генерирования импульсов.

2.6. Контрольные вопросы

1. Поясните определение термина «алгоритм».

2. Назовите и поясните суть этапов решения задачи на вычислительной машине.

3. Поясните правила составления схем алгоритмов.

4. Поясните основные правила записи команд на Ассемблере.

5. Назовите и поясните назначение основных директив Ассемблера.

6. Поясните принцип формирования задержек с помощью циклического повторения участка программы.

7. Поясните принцип формирования задержек с помощью счетчика-таймера.

8. С помощью интегрированной среды MPLAB докажите, что разработанная вами программа функционирует в соответствии с выданным вам заданием.

Лабораторная работа № 3
АЛГОРИТМЫ И ПРОГРАММИРОВАНИЕ ПРОЦЕДУР ОТОБРАЖЕНИЯ ЦИФРОВОЙ ИНФОРМАЦИИ

Цель работы:

1) изучение принципов организации отображения данных в микропроцессорных системах;

2) приобретение навыков программирования процедур управления индикаторными устройствами.

3.1. Схема подключения индикатора к микроконтроллеру

В микропроцессорных системах отображение информации осуществляется с помощью светодиодных или жидкокристаллических индикаторов. В зависимости от конфигурации отдельных сегментов и их взаимного размещения различают семисегментные и матричные индикаторы.

В настоящей работе ограничимся рассмотрением наиболее простого варианта отображения цифровых данных – на светодиодных семисегментных индикаторах.

На рис. 3.1 показано размещение и условное обозначение светодиодных сегментов в цифровых разрядах индикаторов. Для отображения какой-либо цифры соответствующую комбинацию сегментов активизируют, пропуская через них ток. Величина рабочего тока сегмента составляет несколько миллиампер.

На рис. 3.2 показана принципиальная электрическая схема одноразрядного семисегментного светодиодного индикатора, приведен вариант схемы с общим катодом. Промышленность выпускает также варианты схем с общим анодом. (В дальнейшем все пояснения будем проводить для варианта индикатора с общим катодом без дополнительных оговорок). В процессе работы ток общего вывода r индикатора в 7 раз больше, чем ток одного сегмента. В схеме под-

ключения индикатора к МК при недостаточной нагрузочной способности портов в цепях управления общими выводами разрядов индикатора необходимо предусматривать усилители.

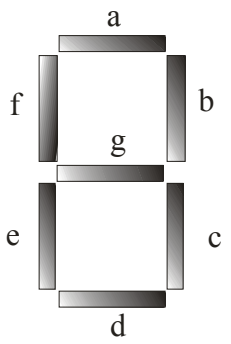

Рис. 3.1. Размещение сегментов в разрядах индикатора

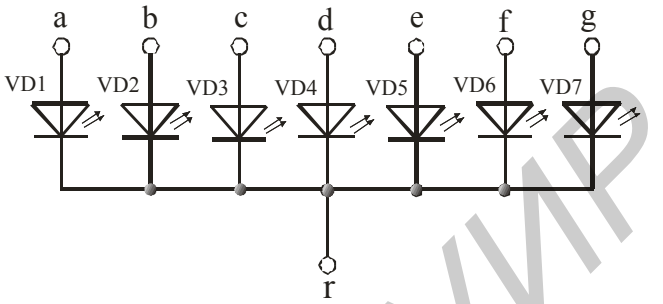

Рис. 3.2. Принципиальная электрическая схема разряда индикатора

Используются два способа управления сегментными индикаторами: статический и динамический.

При статическом способе на каждый цифровой разряд индикатора предусматривается свой регистр, в регистры записываются и хранятся на все время отображения семисегментные коды соответствующих цифр. Выходы регистров соединяются через токоограничивающие резисторы с входами соответствующих сегментов разрядов индикатора, общие выводы разрядов индикатора подключаются к общему проводу. Способ отличается простотой программного обслуживания, но требует повышенных аппаратных затрат (регистров и резисторов), применяется для малоразрядных индикаторов.

При большом количестве разрядов применяют динамический способ управления индикатором. Способ характеризуется минимальными аппаратными затратами, но сравнительно сложной программной поддержкой.

Схема подключения 4-разрядного индикатора в динамическом режиме к микроконтроллеру показана на рис. 3.3. В схеме одноименные сегменты всех разрядов объединяются и обслуживаются семью выводами одного из портов микроконтроллера. Общие выводы разрядов индикатора обслуживаются инди-

видуально четырьмя выводами другого порта микроконтроллера. Первый порт обеспечивает возбуждение определенной для каждой цифры комбинации сегментов, а второй порт активизирует тот или иной разряд индикатора. Суть динамического управления индикатором сводится к поочередному циклическому возбуждению разрядов индикатора. Если разряды индикатора будут подсвечиваться с частотой выше 25 Гц, то в силу инерционности человеческого зрения мерцания изображения заметны не будут. Временные диаграммы работы схемы динамической индикации приведены на рис. 3.4.

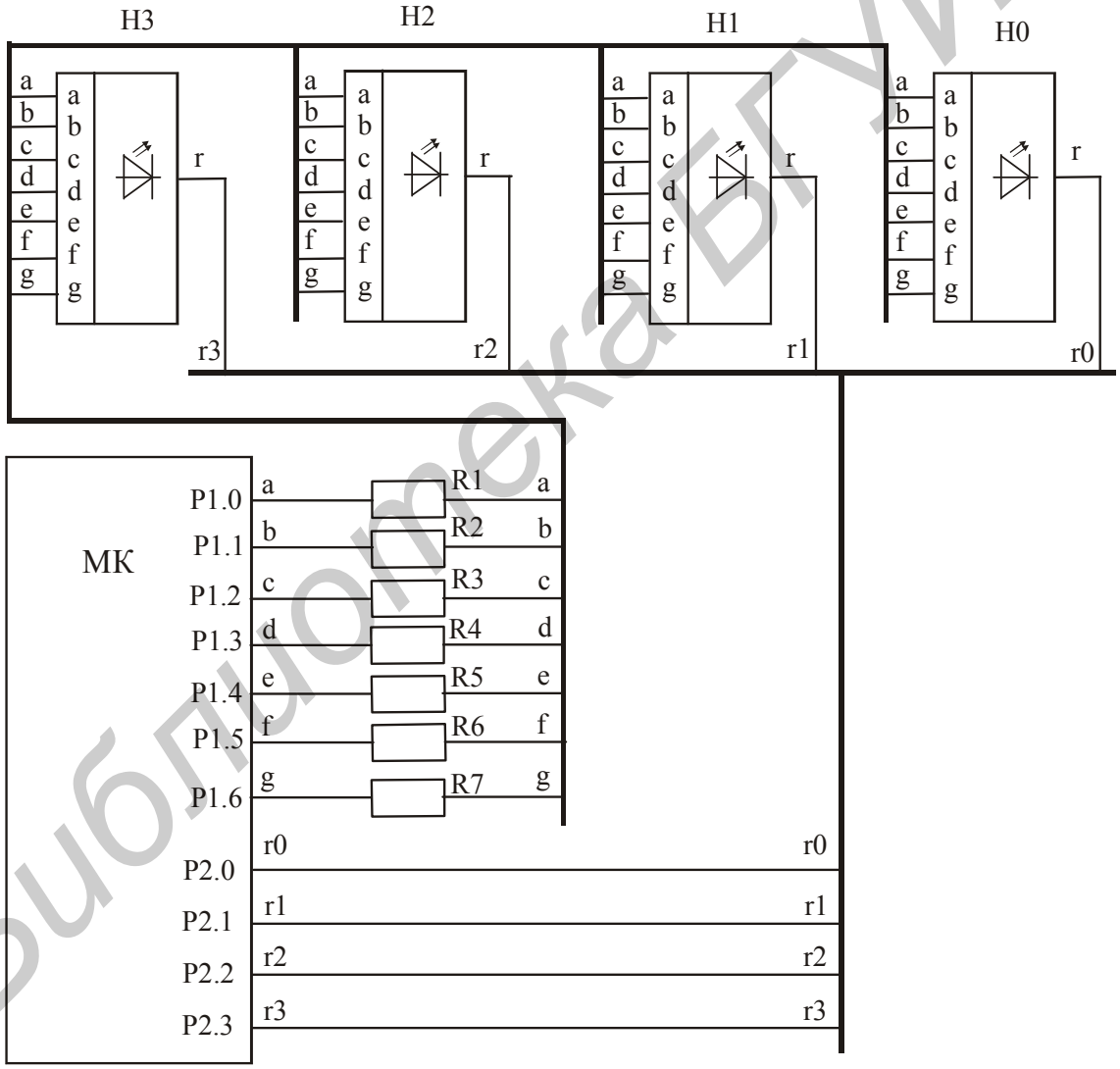

Рис. 3.3. Схема подключения индикатора к микроконтроллеру в динамическом режиме

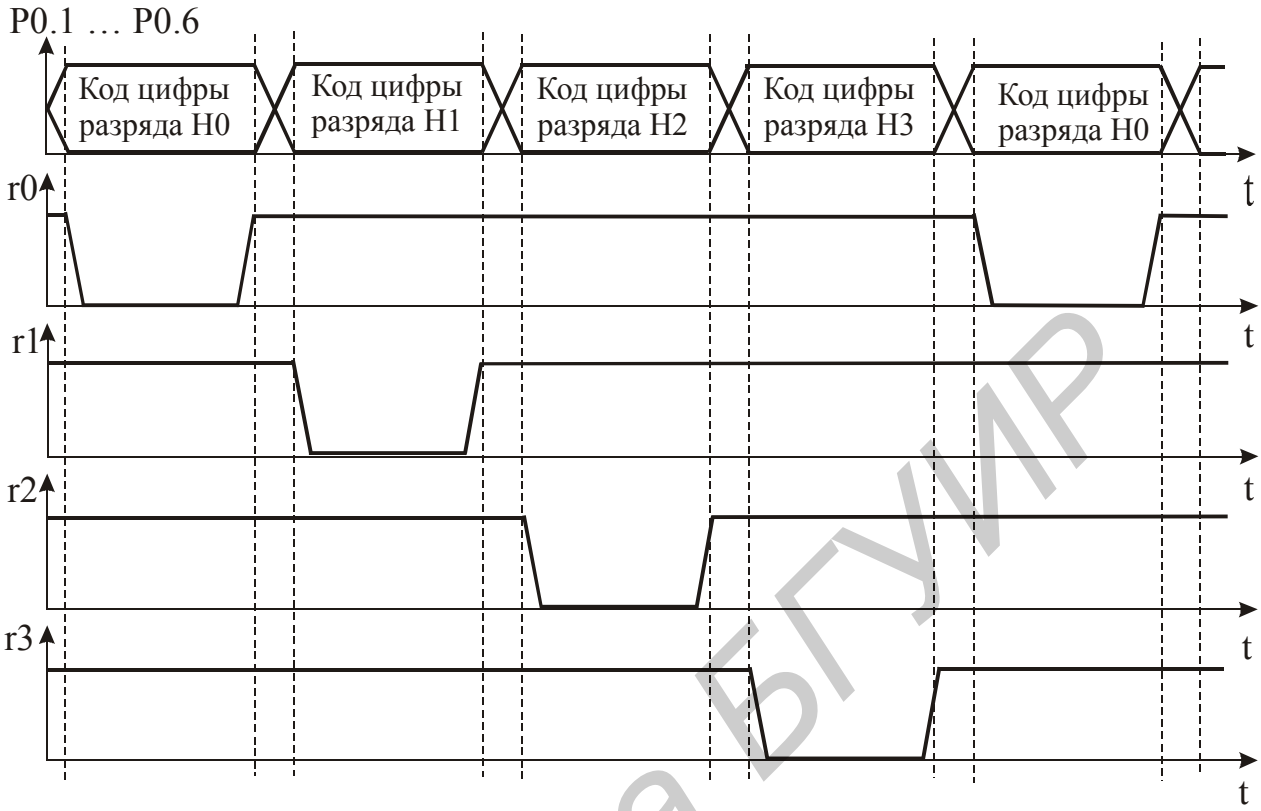

Рис. 3.4. Временные диаграммы работы схемы динамической индикации

3.2. Программа управления индикатором

```
; Программа обслуживания 4-разрядного индикатора в динамическом режиме.

; Младший правый разряд считается нулевым, старший левый считается третьим.
; Светодиоды сегментов индикатора соединены по схеме с общим катодом.
; Общие выводы разрядов индикатора обслуживаются выводами порта A:
; разряд 0 – RA0, разряд 1 – RA1, разряд 2 – RA2, разряд 3 – RA3.
; Сегменты разрядов обслуживаются выводами порта B:  a – RB0, b – RB1, c – RB2, d – RB3,
; e – RB4, f – RB5, g – RB6.
; Для зажигания сегментов индикатора на порт B надо выводить единицы, а на порт A – нули.
; Подпрограмма индикации вызывается по прерываниям от переполнения счетчика-таймера.

      LIST p=16f628, f=inhx8m      ; Задание типа контроллера и формата исполнимого
                                   ; файла программы с расширением .hex.
      #INCLUDE p16f628.inc ; Включить файл описания адресов регистров контроллера.
; Задание адресов переменных и буферов индикатора.
INR   EQU      20h   ; Номер текущего разряда, принимает значения 0, 1, 2, 3.
IANB      EQU  21h   ; Адрес начала буфера индикатора.
IR0   EQU      21h   ; Ячейка хранит двоичный код отображаемой в разряде 0 цифры.
IR1   EQU      22h   ; Ячейка хранит двоичный код отображаемой в разряде 1 цифры.
IR2   EQU      23h   ; Ячейка хранит двоичный код отображаемой в разряде 2 цифры.
```

```
IR3     EQU     24h     ; Ячейка хранит двоичный код отображаемой в разряде 3 цифры.
STATUS_TEMP EQU 70h     ; Ячейка для сохранения состояния регистра STATUS.
IR3     EQU     24h     ; Ячейка хранит двоичный код отображаемой в разряде 3 цифры.
STATUS_TEMP EQU 70h     ; Ячейка для сохранения состояния регистра STATUS.
PCLATH_TEMP EQU 71h     ; Ячейка для сохранения состояния регистра PCLATH.
W_TEMP      EQU 72h     ; Ячейка для сохранения состояния регистра W.
        ORG     0       ; Вектор сброса.
        GOTO    Begin   ; Переход на основную программу.
        ORG     4       ; Вектор прерывания.
        GOTO    Int     ; Переход на подпрограмму обработки прерываний.
Begin   ; Основная программа выполняет начальные установки режимов работы
        ; контроллера и регистров, ждет прерывания от таймера.
        BCF     STATUS,RP0      ; Банк 0.
; Установка портов для состояния "индикатор погашен".
        CLRF    PORTB
        MOVLW   0xFF
        MOVWF   PORTA
; Установки режимов работы.
        BSF     STATUS,RP0      ; Банк 1.
        MOVLW   0xF0
        MOVWF   TRISA   ; Выводы RA0…RA3 настроить как выходы.
        MOVLW   0x01
        MOVWF   TRISB   ; Выводы RB1…RB7 настроить как выходы.
        BCF  OPTION_REG,PS2     ; Коэффициент предварительного делителя 1:16.
        BCF  OPTION_REG,PSA     ; Предварительный делитель включить перед TMR0.
        BCF  OPTION_REG,T0CS    ; Задать внутреннее тактирование TMR0.
        BCF  STATUS,RP0         ; Банк 0.
; Обнуление буфера индикации.
        CLRF    INR
        CLRF    IR0     ; Задание состояния буфера индикации.
        CLRF    IR1     ; Задание состояния буфера индикации.
        CLRF    IR2     ; Задание состояния буфера индикации.
        CLRF    IR3     ; Задание состояния буфера индикации.
        ; Настройка системы прерывания.
        BCF     INTCON, T0IF    ; Сброс флага прерывания по таймеру.
        BSF     INTCON, T0IE    ; Разрешить прерывания от таймера.
        BSF     INTCON, GIE     ; Разрешить все незамаскированные прерывания.
Wait    GOTO    Wait    ; Ожидание прерывания.
;----------Подпрограммы----------
Int     ; Подпрограмма обработки прерываний. В этом примере прерывание возможно
        ; только от таймера, поэтому без анализа причины прерывания производится вызов
        ; подпрограммы обслуживания индикатора.
        MOVWF   W_TEMP  ; Сохранить W в регистре текущего банка.
```

```
        SWAPF   STATUS,W        ; Поменять местами полубайты и сохранить в W.
        CLRF    STATUS          ; Выбрать банк 0.
        MOVWF   STATUS_TEMP     ; Сохранить регистр STATUS.
        MOVF        PCLATH,W            ;
        MOVWF   PCLATH_TEMP     ; Сохранить регистр PCLATH.

        CALL    Indic           ; Вызов подпрограммы обслуживания индикатора.

        BCF  INTCON,T0IF        ; Сброс флага прерывания по переполнению таймера.

        MOVF    PCLATH_TEMP,W       ;
        MOVWF   PCLATH          ; Восстановить регистр PCLATH.
        SWAPF   STATUS_TEMP,W       ; Прочитать регистр STATUS_TEMP в W,
                                    ; восстанавливая банк памяти программ.
        MOVWF   STATUS          ; Переписать W в регистр STATUS.
        SWAPF   W_TEMP,F        ; Поменять местами полубайты в W_TEMP.
        SWAPF   W_TEMP,W        ; Поменять местами полубайты в W_TEMP и
                                    ; записать в W.
        RETFIE          ; Возврат из прерывания.

Indic   ; Подпрограмма обслуживания индикатора. Выполняет следующие действия:
        ; настройку TMR0 для очередного цикла задержки;
        ; преобразование числа текущего разряда в семисегментный код и вывод его на порт B;
        ; преобразование двоичного кода номера текущего разряда
        ; в позиционный код (нуль активный) и вывод этого кода на порт A;
        ; смещение указателя текущего разряда на 1 в пределах 0...3.
        CLRF    STATUS          ; Выбрать банк 0.
        MOVLW   0x6D
        MOVWF   TMR0        ; Коррекция TMR0 для получения требуемой задержки.
        BCF     STATUS,RP0      ; Банк 0.
        MOVLW   0xFF        ;
        MOVWF   PORTA       ; Погасить индикатор.
        MOVLW   IANB
        ADDWF   INR,W           ; Вычисление косвенного адреса текущего разряда.
        MOVWF   FSR   ; Пересылка косвенного адреса текущего разряда в регистр FSR.
        MOVF    INDF,W      ; Двоичный код десятичной цифры разряда – в W.
        CALL    Tab1
        MOVWF   PORTB
; Семисегментный код десятичной цифры текущего разряда выведен на порт B.
        MOVF    INR,W       ; Двоичный код номера текущего разряда – в W.
        CALL    Tab2  ; Преобразование двоичного кода номера текущего разряда
                      ; в позиционный код.
        MOVWF   PORTA       ; Подсветка текущего разряда.
        INCF    INR,F       ;
        BTFSC   INR,2       ; Смена номера текущего разряда: INR=INR+1.
        CLRF    INR         ;
        RETURN
```

```
Tab1  ; Перекодировка двоичного кода десятичной цифры в семисегментный код.
      ; Перед вызовом подпрограммы в W загружается преобразуемое число.
      ; Результат выполнения программы также помещается в W.
      ADDWF    PCL
      RETLW    b'00111111'      ; Код цифры 0.
      RETLW    b'00000110'      ; Код цифры 1.
      RETLW    b'01011011'      ; Код цифры 2.
      RETLW    b'01001111'      ; Код цифры 3.
      RETLW    b'01100110'      ; Код цифры 4.
      RETLW    b'01101101'      ; Код цифры 5.
      RETLW    b'01111101'      ; Код цифры 6.
      RETLW    b'00000111'      ; Код цифры 7.
      RETLW    b'01111111'      ; Код цифры 8.
      RETLW    b'01101111'      ; Код цифры 9.
      RETLW    b'00000000'      ; Код для погашенного состояния разряда.

Tab2        ; Преобразование двоичного кода номера текущего разряда в позиционный код
      ; с активным уровнем 0. Перед вызовом подпрограммы в W загрузить текущий номер
      ; разряда. Результат после выполнения подпрограммы – в W.
      ADDWF     PCL
      RETLW    b'00001110'      ; 0 – разряд.
      RETLW    b'00001101'      ; 1 – разряд.
      RETLW    b'00001011'      ; 2 – разряд.
      RETLW    b'00000111'      ; 3 – разряд.

      END
```

3.3. Лабораторное задание

1. Изучить теоретический материал, изложенный в подразд. 3.1 …3.2.

2. Получить задание у преподавателя. В задании должны быть указаны:

а) разрядность (два или три) обслуживаемого индикатора в динамическом режиме;

б) выводы порта B для управления сегментами индикатора;

в) выводы порта A для управления общими выводами разрядов индикатора;

г) десятичное число для отображения на индикаторе.

3. Составить схему подключения индикатора к микроконтроллеру, схему алгоритма программы, а также написать и отладить в среде MPLAB программу управление индикатором в динамическом режиме в соответствии с индивидуальным заданием.

4. Оформить отчет, который должен содержать: титульный лист, цель работы, вариант лабораторного задания, схему подключения индикатора, схему алгоритма программы и листинг программы управления индикатором.

3.4. Контрольные вопросы

1. Поясните принцип статического режима работы индикатора.

2. Поясните принцип динамического режима работы индикатора.

3. Из каких соображений выбирается частота переключения разрядов индикатора в динамическом режиме?

4. Поясните принцип построения программы табличного преобразования для микроконтроллера PIC16F628.

5. Поясните схему алгоритма работы программы управления индикатором для вашего варианта задания.

6. С помощью интегрированной среды MPLAB докажите, что разработанная вами программа функционирует в соответствии с выданным вам заданием.

Лабораторная работа № 4
АЛГОРИТМЫ И ПРОГРАММИРОВАНИЕ ПРОЦЕДУР ВВОДА ДАННЫХ С КЛАВИАТУРЫ

Цель работы:

1) изучение принципов организации ввода данных в микропроцессорных системах с клавиатуры;

2) приобретение навыков программирования процедур ввода данных с клавиатуры.

4.1. Подключение клавиатуры к микроконтроллеру

Функцию ввода данных с клавиатуры в микропроцессорных системах реализуют как отдельный самостоятельный аппаратно-программный модуль. На него отводятся определенные аппаратные и программные ресурсы микроконтроллера.

Для небольшого количества клавиш для обслуживания каждой из них выделяется отдельный вывод микроконтроллера, как показано на рис. 2.2.

Для большого количества клавиш используют матричную схему, которая требует значительно меньшего количества выводов микроконтроллера. Схема подключения к микроконтроллеру (МК) 16-кнопочной клавиатуры показана на рис. 4.1. Как видно, для обслуживания 16 кнопок требуется всего лишь 8 выводов МК. Выводы P1.0 … P1.3 микроконтроллера настраиваются как выходы и являются линиями сканирования состояния клавиатуры. Выводы P2.0 … P2.3 настраиваются как входы и являются линиями опроса состояний кнопок клавиатуры. Линии сканирования образуют строки, а линии опроса – столбцы матрицы клавиатуры. Потенциалы линий опроса с помощью резисторов R1 … R4 подтягиваются до уровня напряжения питания. При опросе состояния клавиатуры при незамкнутых клавишах на этих линиях читаются логические единицы. В случае нажатия какой-либо клавиши происходит электрическое замыкание линий строки и столбца, на пересечении которых находится нажатая кнопка. Если при этом на эту линию строки выведен логический ноль, то на линии

столбца с нажатой кнопкой будет читаться также логический ноль. Таким образом, при опросе состояния клавиатуры микроконтроллер по линиям опроса будет читать для каждой нажатой кнопки свой уникальный код возврата KW, который может быть впоследствии преобразован в двоичный код нажатой кнопки KNK. Код возврата не чувствителен к строке, в которой находится нажатая кнопка, поэтому необходима корректировка KNK. Она может быть выполнена по формуле KNK = KNK + 4 i, где i – номер строки с нажатой кнопкой, принимает значения от 0 до 3.

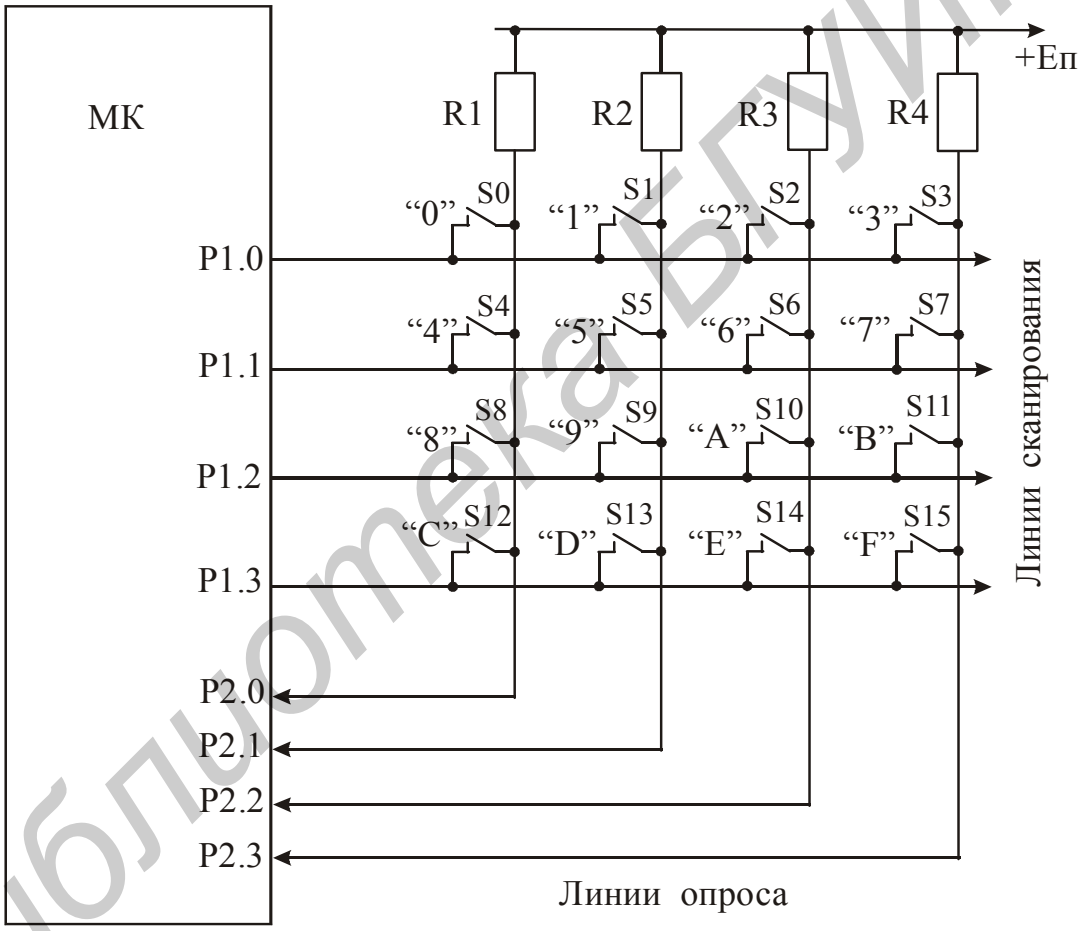

Рис. 4.1. Схема подключения клавиатуры к микроконтроллеру

4.2. Алгоритмы программы управления клавиатурой

Взаимодействие процедуры обслуживания клавиатуры, назовем ее KLAW, с другими процедурами программы лучше всего обеспечивать через буфер клавиатуры BK, куда процедура KLAW заносит двоичный код нажатой кнопки.

При программном обслуживании клавиатуры необходимо решать так или иначе следующий набор задач:

1) контроль и ожидание освобождения буфера клавиатуры,

2) определение факта нажатия кнопки,

3) устранение дребезга контактов,

4) определение кода нажатой кнопки,

5) занесение в буфер кода нажатой кнопки,

6) возведение флага буфера клавиатуры.

Рассмотрим подробнее способы решения названных задач.

Алгоритм контроля и ожидания освобождения буфера клавиатуры может базироваться на контроле значения флага буфера клавиатуры FBK. Если FBK = 0, то буфер пуст, если FBK = 1, то буфер занят. Процедура KLAW заносит в буфер код нажатой клавиши, возводит в единицу FBK, а другие процедуры забирают оттуда код и сбрасывают флаг в ноль. При этом пока флаг остается возведенным в единицу, процедура KLAW игнорирует нажатие кнопок.

Определение факта нажатия кнопки удобно сделать через дополнительную переменную – флаг нажатия кнопки – FNK, которая отражает текущее состояние кнопок: нажата – FNK = 1 или не нажата – FNK = 0, как показано на рис. 4.2. Производя логический анализ двух соседних значений FNK (текущего FNKI и предыдущего FNKJ), легко фиксировать момент нажатия кнопок. Нажатие будет иметь место, если выполняется условие $FNKi \wedge \overline{FNKj} = 1$.

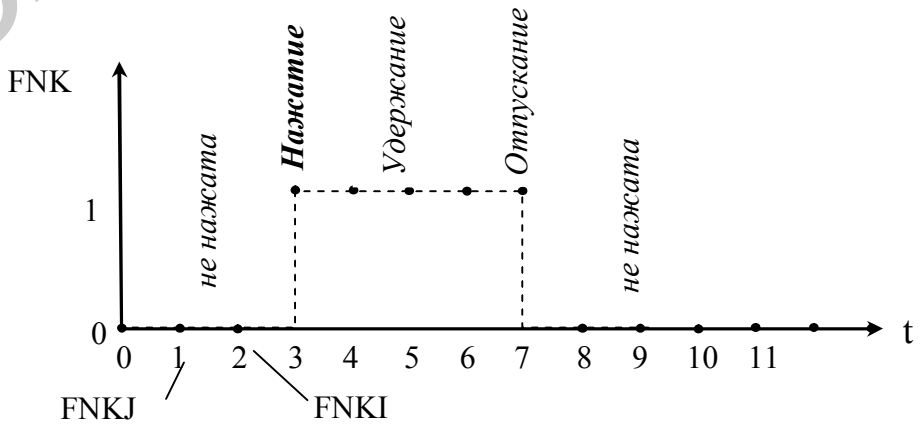

Рис. 4.2. Состояния кнопок

Значения FNK можно определить по следующему алгоритму:

- по линиям сканирования выставить логические нули;
- прочитать код возврата KW на линиях опроса;
- если KW = F, то FNK = 0, кнопки не нажаты;
- если KW ≠ F, то FNK =1, одна или более кнопок нажаты.

Устранение дребезга контактов лучше всего делать путем опроса состояния клавиатуры с частотой (5 … 20 Гц). При этом все переходные процессы обычно завершаются до следующего опроса состояния кнопок. Периодический опрос клавиатуры лучше всего организовать по прерываниям от переполнения счетчика-таймера. В реальных микропроцессорных системах опрос клавиатуры обычно совмещают в цикле с обслуживанием индикатора в динамическом режиме.

Определение кода нажатой кнопки KNK проводят методом последовательной проверки гипотез: нажатая кнопка находится на линии 0-й строки, 1-й строки, 2-й строки, 3-й строки.

Для проверки, находится ли нажатая кнопка в i-й строке (где i = 0 … 3), на ней выставляется логический нуль, а на всех других – логические единицы.

Читают код возврата KW на линиях опроса. Если KW = F, то нажатая кнопка не в этой строке. Если KW ≠ F, то нажатая кнопка в этой строке.

Далее преобразуют позиционный код нажатой кнопки в двоичный по правилам: если KW = E, то KNK = 0; если KW = D, то KNK = 1; если KW = B, то KNK = 2; если KW = 7, то KNK = 3.

И последний шаг этой процедуры – корректировка KNK с учетом веса строки с нажатой кнопкой по формуле KNK = KNK + 4 i, где i – номер строки с нажатой кнопкой.

Занесение в буфер кода нажатой кнопки обеспечивается операцией BK = KNK.

Возведение флага буфера клавиатуры реализуется операцией FBK = 1.

4.3. Программа управления клавиатурой

```
; Программа обслуживания 16-кнопочной клавиатуры.

; Порты RB0 … RB3 используются как линии сканирования состояния клавиатуры.
; Порты RB4 … RB7 используются как линии опроса состояния клавиатуры.
; Контакты клавиатуры подключаются в узлах пересечений линий сканирования и опроса.
; Контакты клавиатуры в нормальном состоянии разомкнуты,
; при нажатии клавиш – замыкаются.
; Кнопки, включенные в узлах пересечений линии RB0 с линиями RB4, RB5, RB6 и RB7,
; соответствуют цифрам 0, 1, 2 и 3 соответственно.
; Кнопки, включенные в узлах пересечений линии RB1 с линиями RB4, RB5, RB6 и RB7,
; соответствуют цифрам 4, 5, 6 и 7 соответственно.
; Кнопки, включенные в узлах пересечений линии RB2 с линиями RB4, RB5, RB6 и RB7,
; соответствуют цифрам 8, 9, A и B соответственно.
; Кнопки, включенные в узлах пересечений линии RB3 с линиями RB4, RB5, RB6 и RB7,
; соответствуют цифрам C, D, E и F соответственно.

; Собственно подпрограмма обслуживания клавиатуры вызывается по прерываниям
; от переполнений счетчика-таймера.

      LIST p=16f628, f=inhx8m      ; Задание типа контроллера и формата исполнимого
                                   ; файла программы с расширением .hex.
      #INCLUDE p16f628.inc ; Включить файл описания адресов регистров контроллера.
; Задание адресов переменных и буферов индикатора.
BK    EQU  20h   ;Буфер клавиатуры, сюда помещается код нажатой клавиши для
                 ; передачи в другие подпрограммы.
FBK   EQU  21h   ; Флаг буфера клавиатуры, сигнализирует о наличии необработанной
                 ; нажатой клавиши.
FNK   EQU  22h   ; Флаг нажатия клавиши.
KW    EQU  23h   ; Код возврата при опросе состояния клавиш.
KNK   EQU  24h   ; Код нажатой клавиши.
FNKI  EQU  25h   ; Текущее значение FNK.
FNKJ  EQU  26h   ; Предыдущее значение FNK.
STATUS_TEMP  EQU  70h   ; Ячейка для сохранения состояния регистра STATUS.
PCLATH_TEMP  EQU  71h   ; Ячейка для сохранения состояния регистра PCLATH.
W_TEMP       EQU  72h   ; Ячейка для сохранения состояния регистра W.
                 ; Ячейки для сохранения общих регистров размещены в области,
                 ; доступной из любого из четырех банков.

      ORG    0          ; Вектор сброса.
      GOTO   Begin      ; Переход на основную программу.
      ORG    4          ; Вектор прерывания.
      GOTO   Int        ; Переход на подпрограмму обработки прерываний.

Begin ; Основная программа,
      ; выполняет начальные установки режимов работы контроллера и регистров,
      ; ждет прерывания от таймера.
      BCF    STATUS,RP0 ; Банк 0.
```

```
; Установка портов в состояние "опрос клавиш выключен".
        MOVLW   0xFF
        MOVWF   PORTB
; Обнуление переменных.
        CLRF    BK
        CLRF    FBK
        CLRF    FNK
        CLRF    KW
        CLRF    KNK
        CLRF    FNKI
        CLRF    FNKJ
; Установки режимов работы.
        BSF     STATUS,RP0      ; Банк 1.
        MOVLW   0xF0
        MOVWF   TRISB           ; Выводы RB0…RB3 настроить как выходы.
        BCF  OPTION_REG,PS2     ; Коэффициент предварительного делителя 1:16.
        BCF  OPTION_REG,PSA     ;Предварительный делитель включить перед TMR0.
        BCF  OPTION_REG,T0CS    ; Внутреннее тактирование TMR0.
        BCF  STATUS, RP0        ; Банк 0.
; Настройка системы прерывания.
        BCF     INTCON, T0IF    ; Сброс флага прерывания по таймеру.
        BSF     INTCON, T0IE    ; Разрешить прерывания от таймера.
        BSF     INTCON, GIE     ; Разрешить все незамаскированные прерывания.

Wait  GOTO      Wait            ; Ожидание прерывания.


                        ; Подпрограммы.

Int     ; Подпрограмма обработки прерываний.
        ; В этом примере прерывание возможно только от таймера, поэтому без анализа
        ; причины прерывания производится вызов подпрограммы обслуживания клавиатуры.
; Сохранение состояний общих регистров.
        MOVWF   W_TEMP          ;
        SWAPF   STATUS,W        ;
        MOVWF   STATUS_TEMP     ;
        MOVF    PCLATH,W        ;
        MOVWF   PCLATH_TEMP     ;
; Настройка TMR0 для очередного цикла задержки.
        CLRF    STATUS          ; Банк 0.
        MOVLW   0x6D
        MOVWF   TMR0

        CALL    Klaw            ; Вызов подпрограммы обслуживания клавиатуры.
        BCF     INTCON,T0IF     ; Сброс флага по переполнению таймера.
; Восстановление состояний общих регистров.
        MOVF    PCLATH_TEMP,W   ;
```

```
        MOVWF   PCLATH          ;
        SWAPF   STATUS_TEMP,W   ;
        MOVWF   STATUS          ;
        SWAPF   W_TEMP,F        ;
        SWAPF   W_TEMP,W        ;
        RETFIE  ;Возврат из прерывания (разрешаются незамаскированные прерывания).


Klaw    ; Подпрограмма обслуживания клавиатуры.
; Контроль и ожидание освобождения буфера клавиатуры.
        CLRF    STATUS          ; Банк 0.
        CLRF    FNK     ; Обнуление флага нажатия клавиши.
        BTFSC   FBK,0   ; Буфер клавиатуры свободен?
        RETURN          ; Если нет, то выход из подпрограммы.
; Определение факта нажатия кнопки.
        CLRF    PORTB   ; Установить нули на линиях сканирования.
        NOP     ;
        NOP     ; Задержка для установления уровней сигналов на выводах портов.
        NOP     ;
        SWAPF   PORTB,W ; Прочитать в W,
        ANDLW   0x0F    ; выделить код на линиях возврата
        MOVWF   KW      ; и сохранить его в ячейке KW.
        SUBLW   0x0F    ;  Проверка: KW=F?
        BTFSS   STATUS,Z ; Если нет,
        BSF     FNK,0   ; то FNK=1
        MOVF    FNKI,W  ;
        MOVWF   FNKJ    ; FNKJ= FNKI.
        MOVF    FNK,W   ;
        MOVWF   FNKI    ; FNKI= FNK.
        COMF    FNKJ,W  ;
        ANDLW   0x01    ;
        ANDWF   FNKI,W  ; NOT (FNKJ) ^ FNKI.
        BTFSC   STATUS,Z ;
        RETURN
; Определение кода нажатой кнопки.
                        ; Проверка: нажатая кнопка в строке 0?
        MOVLW   0xFE
        MOVWF   PORTB
        NOP     ;
        NOP     ; Задержка для установления уровней сигналов на выводах портов.
        NOP     ;
        SWAPF   PORTB,W ; Прочитать в W,
        ANDLW   0x0F    ; выделить код на линиях возврата
        MOVWF   KW      ; и сохранить его в ячейке KW.
```

```
        SUBLW   0x0F        ;  Проверка: KW=F?
        BTFSC   STATUS,Z    ; Если да,
        GOTO    Met1        ; то переход к проверке следующей строки.
        CALL    Kod         ; Вызов подпрограммы определения кода нажатой кнопки.
        MOVF    KNK,W
        MOVWF   BK
        INCF    FBK,F
        RETURN
Met1                ; Проверка: нажатая кнопка в строке 1?
        MOVLW   0xFD
        MOVWF   PORTB
        NOP     ;
        NOP     ; Задержка для установления уровней сигналов на выводах портов.
        NOP     ;
        SWAPF   PORTB,W ; Прочитать в W,
        ANDLW   0x0F    ; выделить код на линиях возврата
        MOVWF   KW      ; и сохранить его в ячейке KW.
        SUBLW   0x0F    ;  Проверка: KW=F?
        BTFSC   STATUS,Z ; Если да,
        GOTO    Met2    ; то переход к проверке следующей строки.
        CALL    Kod     ; Вызов подпрограммы определения кода нажатой кнопки.
        MOVF    KNK,W
        ADDLW   4       ; Корректировка KNK с учетом строки.
        MOVWF   BK
        INCF    FBK,F
        RETURN
Met2                ; Проверка: нажатая кнопка в строке 2?
        MOVLW   0xFB
        MOVWF   PORTB
        NOP     ;
        NOP     ; Задержка для установления уровней сигналов на выводах портов.
        NOP     ;
        SWAPF   PORTB,W ; Прочитать в W,
        ANDLW   0x0F    ; выделить код на линиях возврата
        MOVWF   KW      ; и сохранить его в ячейке KW.
        SUBLW   0x0F    ;  Проверка: KW=F?
        BTFSC   STATUS,Z ; Если да,
        GOTO    Met3    ; то переход к проверке следующей строки.
        CALL    Kod     ; Вызов подпрограммы определения кода нажатой кнопки.
        MOVF    KNK,W
        ADDLW   8       ; Корректировка KNK с учетом строки.
        MOVWF   BK
        INCF    FBK,F
```

```
        RETURN
Met3                    ; Проверка: нажатая кнопка в строке 3?
        MOVLW   0xF7
        MOVWF   PORTB
        NOP     ;
        NOP     ; Задержка для установления уровней сигналов на выводах портов.
        NOP     ;
        SWAPF   PORTB,W ; Прочитать в W,
        ANDLW   0x0F    ; выделить код на линиях возврата
        MOVWF   KW      ; и сохранить его в ячейке KW.
        SUBLW   0x0F    ; Проверка: KW=F?
        BTFSC   STATUS,Z ; Если да,
        RETURN          ; то выход из подпрограммы (поскольку последняя строка).
        CALL    Kod     ; Вызов подпрограммы определения кода нажатой кнопки.
        MOVF    KNK,W
        ADDLW   12      ; Корректировка KNK с учетом строки.
        MOVWF   BK
        INCF    FBK,F
        RETURN

Kod     ; Преобразование кода возврата KW в двоичный код нажатой кнопки.
        MOVF    KW,W    ;
        SUBLW   0x0E    ; Проверка: KW=E?
        BTFSS   STATUS,Z ; Если нет,
        GOTO    Met10   ; то переход к следующей проверке.
        MOVLW   0
        MOVWF   KNK     ; KNK=0.
        RETURN
Met10
        MOVF    KW,W    ;
        SUBLW   0x0D    ; Проверка: KW=D?
        BTFSS   STATUS,Z ; Если нет,
        GOTO    Met11   ; то переход к следующей проверке.
        MOVLW   1
        MOVWF   KNK     ; KNK=1.
        RETURN
Met11
        MOVF    KW,W    ;
        SUBLW   0x0B    ; Проверка: KW=B?
        BTFSS   STATUS,Z ; Если нет,
        GOTO    Met12   ; то переход к следующей проверке.
        MOVLW   2
        MOVWF   KNK     ; KNK=2.
```

```
        RETURN
Met12
        MOVLW   3
        MOVWF   KNK         ; KNK=3
        RETURN
        END
```

4.4. Лабораторное задание

1. Изучить теоретический материал, изложенный в подразд. 4.1. …4.3.

2. Получить у преподавателя индивидуальное задание.

В вариантах заданий использовать:

- количество клавиш меньше 16;

- вариант подключения клавиатуры к микроконтроллеру матричный;

- варианты конфигурации клавиатуры, заданной в виде (количество линий сканирования)х(количество линий опроса) 2 х 2, 2 х 3, 2 х 4, 3 х 2, 3 х 3, 3 х 4;

- в качестве линий сканирования и линий опроса использовать порты А и В микроконтроллера PIC16F628;

- линии сканирования и линии опроса должны быть соседними и в пределах одного порта.

3. В соответствии с индивидуальным заданием, выданным преподавателем, составить схему подключения клавиатуры к микроконтроллеру, разработать схему алгоритма, составить и отладить в среде MPLAB программу управления клавиатурой.

4. Оформить отчет, который должен содержать: титульный лист, цель работы, вариант лабораторного задания, схему подключения клавиатуры к микроконтроллеру, схему алгоритма управления заданным вариантом клавиатуры и программу на Ассемблере управления заданным вариантом клавиатуры.

4.5. Контрольные вопросы

1. Поясните принцип матричного подключения клавиатуры к микроконтроллеру.

2. Назовите задачи, решаемые управляющей программой по обслуживанию клавиатуры.

3. Поясните алгоритмы решения задач по обслуживанию клавиатуры.

4. Поясните схему алгоритма работы программы управления клавиатурой, которая разработана вами в соответствии с полученным заданием.

5. С помощью интегрированной среды MPLAB докажите, что разработанная вами программа функционирует в соответствии с выданным вам заданием.

ЛИТЕРАТУРА

1. PIC16F62X. Однокристальные 8-разрядные FLASH CMOS микроконтроллеры компании Microchip technology incorporated: Пер. с англ. – М.: ООО «Микрочип», 2001. – 148 с. www.microchip.ru

2. Бурак А.И., Левкович В.Н. Интегрированная среда MPLab IDE разработки программ для микроконтроллеров PICmicro фирмы Microchip: Метод. пособие к лабораторным работам по курсу «Цифровые и микропроцессорные устройства». – Мн: БГУИР, 2003. – 31 с.

3. Левкович В.Н. и др. Конструирование программ на Ассемблере для микроконтроллеров семейства PICmicro: Учеб. пособие по курсу «Цифровые и микропроцессорные устройства». – Мн: БГУИР, 2004. – 80 с.

4. Левкович В.Н. Цифровые и микропроцессорные устройства: Лабораторный практикум: В 2 ч. Ч. 1. – Мн: БГУИР, 2005. – 38 с.

Св. план 2006, поз. 71

Учебное издание

Левкович Василий Николаевич,
Кащеев Александр Анатольевич

ЦИФРОВЫЕ И МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА

Лабораторный практикум
для студентов специальностей
I-39 01 02 «Радиоэлектронные системы»,
I-39 01 03 «Радиоинформатика»,
I-39 01 04 «Радиоэлектронная защита информации»
дневной формы обучения

В 2-х частях

Часть 2

Редактор Т.Н. Крюкова
Корректор Н.В. Гриневич

Подписано в печать 25.05.06. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 2,33.
Уч.-изд. л. 1,6. Тираж 250 экз. Заказ 675.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131518 от 30.04.2004.
220013, Минск, П. Бровки, 6