**Программирование и перепрограммирование микроконтроллеров PIC, Ассемблер, мигание светодиодов**

1. **Общие определения**

Цель кафедры «Системы управления» - устройства автоматики самого разнообразного назначения: электронные, пневматические, гидравлические, механические, оптические и другие. В свою очередь они подразделяются на разомкнутые и замкнутые. В производственных условиях или быту встречаются и те и другие, но замкнутые, то есть с обратной связью, более точны, поэтому исследуются углубленно. В начале развития автоматики микроконтроллеров не было. Системы управления были аналоговыми, то есть непрерывными, все элементы информационной части делались из «железа», изготавливались блоки усилителей, интеграторов, сумматоров, дифференциаторов и т.д. Затем, со временем, появились электронные вычислительные машины, компьютеры, промышленные контролеры, микроконтроллеры. Автоматика перешла от аналоговой к цифровой, так как оказалось, что почти все «железные» блоки можно описать цифровыми уравнениями, не делать их отдельно, и ввести в виде программы в систему управления, т.е. сделать её малогабаритной, недорогой. Имеется ввиду, что исполнительные устройства, например, манипуляторы, остались почти прежними, а вот информационная часть стала цифровой, в виде программы. Но компьютер нет смысла ставить везде, возможны более простые устройства автоматики, например в стиральных машинах, микроволновках и т.д., поэтому разработаны несложные цифровые устройства — микроконтроллеры, объёма технических способностей которых вполне достаточно в данной ситуации. Для самых простых устройств автоматики без обратных связей достаточно микроконтроллеров с разрядностью 4-8. Однако в системах с обратными связями необходима разрядность 16-32.

Использование микроконтроллеров предполагает три достаточно объемных рода знаний и работы:

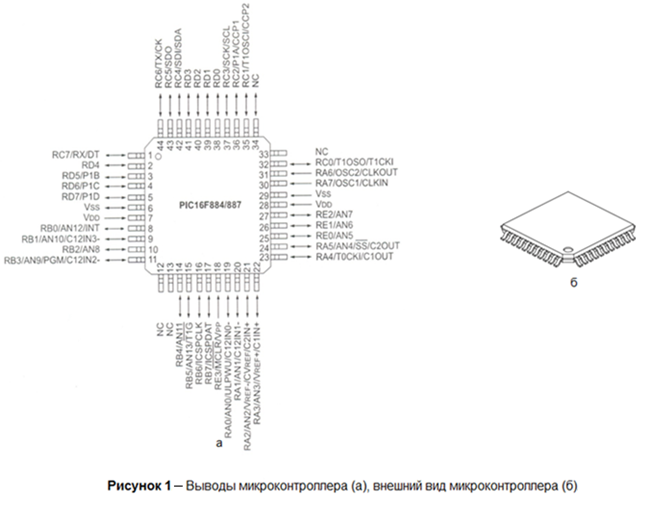
**1.**Знание устройства микроконтроллеров, принципов их функционирования, имеются довольно массивные источники литературы;

**2.**Знание способов программирования и умение написать программы;

**3.**Работа по транслированию программ, отладка, запись программ в микроконтроллер, при необходимости - стирание программ и перезапись.

Первый пункт пользователь может и не знать также как и последующие. Но если необходимо приобрести хотя бы простейшие навыки в программировании микроконтроллеров, то помощь-настоящее пособие, лабораторная работа, в которой выбраны PIC- контролеры, наиболее простые в обращении в сравнении с другими.

Целью лабораторной работы является составление краткой, простейшей программы на ассемблере (одном из языков программирования) для устройства автоматики, действующего по разомкнутому контуру (без обратной связи), трансляция этой программы, запись в микроконтроллер, стирание, перезапись и исследование схемы, управляемой программой микроконтроллера.



Название PIC расшифровывается следующим образом: Peripheral - Interface Controller (периферийный контроллер интерфейса). Это малогабаритное интегральное электронное устройство, в которое можно записать (ввести) программу. Действует эта программа автоматически, выдает на выводы микроконтроллера (выводы регистра данных порта) сигналы «нуль» или «единица» («низкий» или «высокий» уровни напряжения, примерно 0,1 или 4,9 Вольта).

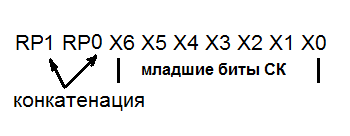
Представленные обычно в двоичном коде, эти сигналы управляют различного типа устройствами автоматики, начиная с простейших электронных часов, заканчивая робототехникой. То же самое, что делают программы в микроконтроллерах, можно реализовать на дискретных (отдельных) электронных схемах, но габариты при этом увеличиваются (от нескольких раз до десятков тысяч раз). Тем не менее и для микроконтроллера, интегрального и малогабаритного, требуется компьютер, с помощью которого разрабатывается программа; программатор-отдельное устройство(вносит программу в микроконтроллер)

1. **Организация памяти**

В микроконтроллерах PIC существует два блока памяти - память программ и память данных(гарвардская структура микроконтроллера) в отличие от структуры ЭВМ (структура Неймана- данные хранятся наравне с командами). Каждый блок имеет собственную шину, таким образом, доступ к блокам может происходить одновременно (минимизация времени). Память данных, в свою очередь, разделена на специальные регистры и регистры общего применения (ОЗУ пользователя). Специальные регистры применяются для хранения битов состояния, определяющих работу портов вода/вывода, таймеров и других периферийных модулей контроллера. Подробно каждый специальный регистр описан при рассмотрении соответствующего модуля. Кроме специальных регистров и ОЗУ пространство памяти данных может содержать ячейки постоянной памяти EEPROM(в некоторых микроконтроллерах отсутствует), запись и перезапись которых осуществляется электрическим способом. Эта область памяти не может быть адресована непосредственно, и доступ к ней получают через специальный регистр косвенной адресации EEADR, в который записывают порядковый номер ячейки. Обычно EEPROM используется для хранения констант, значения которых не должны пропадать при отключении питания, например кодов управления, индивидуальных номеров и т.п. Важным достоинством EEPROM является то, что данные в ней могут быть изменены даже после занесения программы в однократно программируемый кристалл. Число циклов перезаписей перезапуска 10 000.

Примечание: *Для хранения программных переменных следует использовать только ОЗУ пользователя.*

Микроконтроллеры группы PIC имеют 13-битный счётчик команд (СК) (содержит адрес памяти, с которого считывается команда на выполнение, а его значение автоматически увеличивается на единицу - автоинкремент).13 разрядов позволяют адресовать до 8К( К=2^10=1024) ячеек памяти. Семь младших бит(разрядов)счётчика (так решил производитель) работают в соответствии с автоинкриментом и позволяют адресовать =128 адресов. Так как по каждому адресу хранится информация в 8-ми битах (микроконтроллер байтовый), то пишут о 128 адресуемых битах. Когда в счётчике все биты (семь) равны единице (111111), то следующее прибавление единицы сбрасывает значение счётчика в 0 (000000) с потерей единицы переноса со старшего разряда, то есть 128 байт адресует счётчик самостоятельно. Такое адресуемое самим счётчиком пространство называется банком памяти. Разработчики отвели дополнительно два старших разряда 7 и 8 биты счётчика для адресации уже банков памяти. Эти разряды принудительно берутся из специального регистра STATUS (5-ый, 6-ой разряды). Для упрощения написания программ им присвоены специальные имена (RP0, RP1). Это позволяет адресовать 4 банка памяти. Информация разрядов RP0 и RP1 путём конкатенации (соединение, сочленение) объединяется с информацией, содержащейся в младших битах счётчика команд, что позволяет адресовать 4\*128=512 байт.



|  |  |
| --- | --- |
| **RP1: RP0** | Банк |
| 00 | 0 |
| 01 | 1 |
| 10 | 2 |
| 11 | 3 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Карта памяти для микроконтроллеров P16F887** | | | | | | | |
| **Регистр косвенной**  **адресаций** | **Адрес**  **00h** | **Регистр косвенной**  **адресации** | **Адрес**  **80h** | **Регистр косвенной**  **адресаций** | **Адрес**  **100h** | **Регистр косвенной**  **адресации** | **Адрес**  **180h** |
| TMR0 | 01h | OPTION\_REG | 81h | TMR0 | 101h | OPTION\_REG | 181h |
| PCL | 02h | PCL | 82h | PCL | 102h | PCL | 182h |
| STATUS | 03h | STATUS | 83h | STATUS | 103h | STATUS | 183h |
| FSR | 04h | FSR | 84h | FSR | 104h | FSR | 184h |
| PORTA | 05h | TRISA | 85h |  | 105h |  | 185h |
| PORTB | 06h | TRISB | 86h | PORTB | 106h | TRISB | 186h |
| PORTC | 07h | TRISC | 87h |  | 107h |  | 187h |
| PORTD(1) | 08h | TRISD(1) | 88h |  | 108h |  | 188h |
| PORTE(1) | 09h | TRISE(1) | 89h |  | 109h |  | 189h |
| PCLATCH | 0Ah | PCLATH | 8Ah | PCLATH | 10Ah | PCLATH | 18Ah |
| INTCON | 0Bh | INTCON | 8Bh | INTCON | 10Bh | INTCON | 18Bh |
| PIR1 | 0Ch | PIE1 | 8Ch | EEDATA | 10Ch | EECON1 | 18Ch |
| PIR2 | 0Dh | PIE2 | 8Dh | EEADR | 10Dh | EECON2 | 18Dh |
| TMR1L | 0Eh | PCON | 8Eh | EEDATH | 10Eh | Резерв | 18Eh |
| TMR1H | 0Fh |  | 8Fh | EEADRH | 10Fh | Резерв | 18Fh |
| T1CON | 10h |  | 90h | Регистры  общего  назначения  16 байт | 110h | Регистры  общего  назначения  16 байт | 190h |
| TMR2 | 11h | SSPCON2 | 91h | 111h | 191h |
| T2CON | 12h | PR2 | 92h | 112h | 192h |
| SSPBUF | 13h | SSPADD | 93h | 113h | 193h |
| SSPCON | 14h | SSPSTAT | 94h | 114h | 194h |
| CCPR1L | 15h |  | 95h | 115h | 195h |
| CCPR1H | 16h |  | 96h | 116h | 196h |
| CCP1CON | 17h |  | 97h | 117h | 197h |
| RCSTA | 18h | TXSTA | 98h | 118h | 198h |
| TXREG | 19h | SPBRG | 99h | 119h | 199h |
| RCREG | 1Ah |  | 9Ah | 11Ah | 19Ah |
| CCPR2L | 1Bh |  | 9Bh | 11Bh | 19Bh |
| CCPR2H | 1Ch |  | 9Ch | 11Ch | 19Ch |
| CCP2CON | 1Dh |  | 9Dh | 11Dh | 19Dh |
| ADRESH | 1Eh | ADRESL | 9Eh | 11Eh | 19Eh |
| ADCONO | 1Fh | ADCON1 | 9Fh | 11Fh | 19Fh |
| Регистры  общего  назначения  96 байт | 20h  7Fh | Регистры  общего  назначения  80байт | A0h  EFh | Регистры  общего  назначения  80 байт | 120h  16Fh | Регистры  общего  назначения  80 байт | 1A0h  1EFh |
| Доступ к 70h-7Fh  (16 байт) | F0h  FFh | Доступ к 70h-7Fh  (16 байт) | 170h  17Fh | Доступ к 70h-7Fh  (16 байт) | 1F0h  1FFh |
| **Банк 0** | | **Банк 1** | | **Банк 2** | | **Банк 3** | |

**Таблица 1.**

Примечание: закрашенные участки памяти не реализованы, значение при чтении 00h

1. **Система команд**

Полный список команд приведён в **таблице 2**.

Система команд аккумуляторного типа (одноадресная), оригинальна и разделена на три основных группы:

1.Байт ориентированные команды;

2.Бит ориентированные команды;

3. Команды управления и операций с константами.

Для байт ориентированных команд ‘f’ является указателем регистра, а 'd' указателем адресата результата. Указатель регистра определяет, какой регистр должен использоваться в команде. Указатель адресата определяет, где будет сохранен результат. Если 'd'=0, результат сохраняется в регистре W. Если 'd'=1, результат сохраняется в регистре, который используется в команде. В бит ориентированных командах 'b' определяет номер бита участвующего в операции, а ‘f’ - указатель регистра, который содержит этот бит. Мнемоника команд, поддерживаемая ассемблером MPASM, показана в **таблице 2**. Во всех примерах используется следующий формат шестнадцатеричных чисел: 0xhh, где h - шестнадцатеричная цифра.

**Таблица 2. Описание команд**

|  |  |  |
| --- | --- | --- |
| Мнемоника  команды | Описание | Изм.  Флаги |
|
| ADDWF f,d | Сложение W и f | C,DC,Z |
| ANDWF f,d | Побитное 'И' W и f | Z |
| CLRF f | Очистить f | Z |
| CLRW - | Очистить W | Z |
| COMF f,d | Инвертировать f | Z |
| DECF f,d | Вычесть 1 из f | Z |
| DECFSZ f,d | Вычесть 1 из f и пропустить если 0 |  |
| INCF f,d | Прибавить 1 к f | Z |
| INCFSZ f,d | Прибавить 1 к f и пропустить если 0 |  |
| IORWF f,d | Побитное 'ИЛИ' W и f | Z |
| MOVF f,d | Переслать f | Z |
| MOVWF f | Переслать W в f |  |
| NOP - | Нет операции |  |
| RLF f,d | Циклический сдвиг f влево через перенос | C |
| RRF f,d | Циклический сдвиг f вправо через перенос | C |
| SUBWF f,d | Вычесть W из f | C,DC,Z |
| SWAPF f,d | Поменять местами полубайты в регистре f |  |
| XORWF f,d | Побитное 'исключающее ИЛИ' W и f | Z |
| BCF f,b | Очистить бит b в регистре f |  |
| BSF f,b | Установить бит b в регистре f |  |
| BTFSC f,b | Проверить бит b в регистре f, пропустить если 0 |  |
| BTFSS f,b | Проверить бит b в регистре f, пропустить если 1 |  |
| ADDLW k | Сложить константу с W | C,DC,Z |
| ANDLW k | Побитное 'И' константы и W | Z |
| CALL k | Вызов подпрограммы |  |
| CLRWDT - | Очистить WDT | -TO,-PD |
| GOTO k | Безусловный переход |  |
| IORLW k | Побитное 'ИЛИ' константы и W | Z |
| MOVLW k | Переслать константу в W |  |
| RETFIE - | Возврат из подпрограммы с разрешением прерываний |  |
| RETLW k | Возврат из подпрограммы с загрузкой константы в W |  |
| SLEEP - | Перейти в режим SLEEP | -TO,-PD |
| SUBLW k | Вычесть W из константы | C,DC,Z |
| XORLW k | Побитное 'исключающее ИЛИ' константы и W | Z |

***Примечание:***

При выполнении операции "чтение - модификация - запись" с портом ввода/вывода исходные значения считываются с выводов порта, а не из выходных защелок. Например, если в выходной защелке было записана '1 ', а на соответствующем выходе низкий уровень сигнала, то обратно будет записано значение '0'.

**Подробное описание команд**

**ADDLW** Сложить константу с W

Синтаксис: [*label*] ADDLW k

Операнды: 0 ≤ k ≤ 255

Операция: (W) + k → (W)

Измен. флаги: C, DC, Z

Описание: Содержимое регистра W складывается с 8-разрядной

константой 'k'. Результат сохраняется в регистре W.

**BCF** Очистить бит b в регистре f

Синтаксис: [*label*] BCF f,b

Операнды: 0 ≤ f ≤ 127

0 ≤ b ≤ 7

Операция: 0 → (f<b>)

Измен. флаги: Нет

Описание: Очистить бит 'b' в регистре 'f'.

**BSF** Установить бит b в регистре f

Синтаксис: [*label*] BSF f,b

Операнды: 0 ≤ f ≤ 127

0 ≤ b ≤ 7

Операция: 1 → (f<b>)

Измен. флаги: Нет

Описание: Установить бит 'b' в регистре 'f'.

**CLRF** Очистить f

Синтаксис: [*label*] CLRF f

Операнды: 0 ≤ f ≤ 127

Операция: 00h → (f)

1 → Z

Измен. флаги: Z

Описание: Очистить содержимое регистра 'f' и установить флаг Z

**CLRW** Очистить W

Синтаксис: [*label*] CLRW

Операнды: Нет

Операция: 00h → (W)

1 → Z

Измен. флаги: Z

Описание: Очистить содержимое регистра W и установить флаг Z

**COMF** Инвертировать f

Синтаксис: [*label*] COMF f,d

Операнды: 0 ≤ f ≤ 127

d ∈ [0,1]

Операция: (-f) → (dest)

Измен. флаги: Z

Описание: Инвертировать все биты кроме 'f'.

Если d=0, значение сохраняется в регистре W.

Если d=1, результат сохраняется в регистре 'f'.

**DECFSZ** Вычесть 1 из f и пропустить если 0

Синтаксис: [*label*] DECFSZ f,d

Операнды: 0 ≤ f ≤ 127

d ∈ [0,1]

Операция: (f) - 1 → (dest); пропустить если результат равен 0

Измен. флаги: Нет

Описание: Декрементировать содержимое регистра 'f'. Если d=0,

результат сохраняется в регистре W. Если d=1, результат

сохраняется в регистре 'f'.

Если результат не равен '0' , то исполняется следующая

инструкция. Если результат равен '0' , то следующая

инструкция не выполняется, команда выполняется за два

цикла. Во втором цикле выполняется NOP.

**GOTO** Безусловный переход

Синтаксис: [*label*] GOTO k

Операнды: 0 ≤ k ≤ 2047

Операция: k → PC<10:0>,

(PCLATH<4:3>) → PC<12:11>

Измен. флаги: Нет

Описание: Выполнить безусловный переход. Одиннадцать бит

адреса загружаются из кода команды в счетчик команд

PC<10:0>. Два старших бита загружаются в счетчик

команд PC<12:11> из регистра PCLATH. Команда GOTO

выполняется за два цикла.

**MOVF** Переслать f

Синтаксис: [*label*] MOVF f,d

Операнды: 0 ≤ f ≤ 127

d ∈ [0,1]

Операция: (f) → (dest)

Измен. флаги: Z

Описание: Содержимое регистра 'f' пересылается в регистр

адресата. Если d=0, значение сохраняется в регистре W.

Если d=1, значение сохраняется в регистре 'f'. d=1

используется для проверки содержимого регистра 'f' на ноль.

**MOVLW** Переслать константу в W

Синтаксис: [*label*] MOVLW k

Операнды: 0 ≤ k ≤ 255

Операция: k → (W)

Измен. флаги: Нет

Описание: Переслать константу 'k' в регистр W. В неиспользуемых

битах ассемблер устанавливает '0'.

**MOVWF** Переслать W в f

Синтаксис: [*label*] MOVWF f

Операнды: 0 ≤ f ≤ 127

Операция: (W) → (f)

Измен. флаги: Нет

Описание: Переслать содержимое регистра W в регистр 'f'.

**NOP** Нет операции

Синтаксис: [*label*] NOP

Операнды: Нет

Операция: Нет операции

Измен. флаги: Нет

Описание: Нет операции.

**XORWF** Побитное 'исключающее ИЛИ' W и f

Синтаксис: [*label*] XORWF f,d

Операнды: 0 ≤ f ≤ 127

d [0,1]

Операция: (W) .XOR. (f) → (dest)

Измен. флаги: Z

Описание: Выполняется побитное 'исключающее ИЛИ' содержимого

регистров W и 'f'. Если d=0, результат сохраняется в

регистре W. Если d=1, результат сохраняется в регистре 'f'.

1. **Перевод чисел в разные системы счисления.**

1. В повседневной жизни мы используем десятичную систему счисления, процессоры (микроконтроллеры) работают только с двоичной системой. Программистам более удобно использовать 16-ую систему счисления (иногда восьмеричную, более удобная запись). Все перечисленные системы счисления являются позиционными - одна и та же цифра в разной позиции написания имеет свой вес, связанный однозначно с основанием системы счисления. Есть и пример непозиционной системы счисления - римская, где цифра всегда имеет одно и тоже значение в не зависимости от позиции написания. Ниже приведены примеры записи чисел в различных системах счисления.

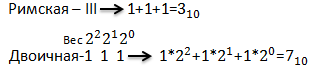


Таблица степени числа 2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n (степень) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| http://inf.e-alekseev.ru/extra/ris10.gif | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |

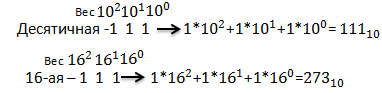


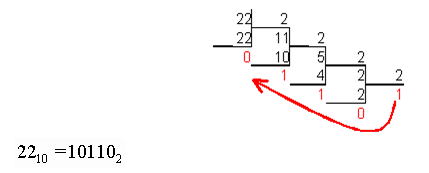
Таблица степени числа 16

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| n (степень) | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| http://inf.e-alekseev.ru/extra/ris9.gif | 1 | 16 | 256 | 4096 | 65536 | 1048576 | 16777216 |

Эти записи показывают алгоритм перевода различных систем счисления в десятичную систему счисления.

2.Перевод чисел из 10-ой системы счисления в другие осуществляется путём деления этого числа на основание новой системы.

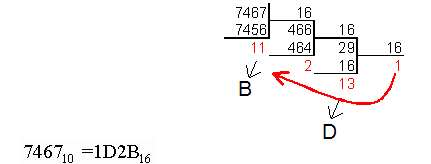
**Пример.** Число http://inf.e-alekseev.ru/extra/ris14.gif перевести в двоичную систему счисления.



Число в двоичной системе записывается как последовательность цифр последнего результата деления и остатков от деления в обратном порядке.

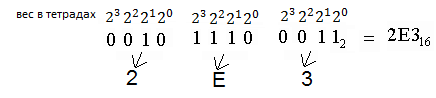
3. Для перевода десятичного числа в шестнадцатеричную систему его необходимо последовательно делить на 16 до тех пор, пока не останется остаток, меньший или равный 15. Число в шестнадцатеричной системе записывается как последовательность цифр последнего результата деления и остатков от деления в обратном порядке.

**Пример.** Число http://inf.e-alekseev.ru/extra/ris19.gifперевести в шестнадцатеричную систему счисления.



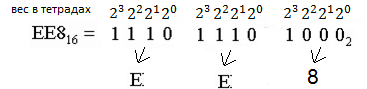
4. Чтобы перевести число из двоичной системы в шестнадцатеричную можно путём перевода числа в десятичную (вспомогательную), но есть и более удобный способ: число нужно разбить на тетрады (четверки цифр), начиная с младшего разряда, в случае необходимости дополнив старшую тетраду нулями, и каждую тетраду заменить соответствующей 16-теричной цифрой. Для этого в каждой тетраде учитывается отдельно вес цифры, значение всех тетрад записывается в виде 16-го эквивалента.

**Пример.** Число http://inf.e-alekseev.ru/extra/ris24.gif перевести в шестнадцатеричную систему счисления.



5. Для перевода шестнадцатеричного числа в двоичное необходимо каждую цифру заменить эквивалентной ей двоичной тетрадой, и записать её в тетраде с учётом веса каждой цифры.

**Пример.** Число http://inf.e-alekseev.ru/extra/ris33.gif перевести в двоичную систему счисления.



1. **Программа**

**Примечание: (готовая программа находится на жестком диске компьютера, или взять её у преподавателя).**

;Мордань Вадим,Ширма Евгений,студенты гр.722405,722403,3курс ;(1)

list p=16F887;(2)

#include <p16F887.inc>;(3)

\_\_CONFIG \_CONFIG1, \_LVP\_OFF & \_FCMEN\_OFF & \_IESO\_OFF & \_BOR\_OFF & \_CPD\_OFF & \_CP\_OFF & \_MCLRE\_OFF & \_PWRTE\_ON & \_WDT\_OFF & \_INTRC\_OSC\_NOCLKOUT

\_\_CONFIG \_CONFIG2, \_WRT\_OFF & \_BOR21V ;(4-5)

count1 EQU 0x20;(6)

count2 EQU 0x21;(7)

ledset EQU 0x22;(8)

ORG 0x000;(9)

clrf PORTD;(10)

bsf STATUS, RP0 ;(11)

movlw b'11111110';(12)

movwf PORTD ;(13)

bcf OPTION\_REG,7 ;(14)

bcf STATUS,RP0;(15)

clrf ledset;(16)

loop:;(17)

movlw b'00000001' ;(18)

xorwf ledset,f; (19)

movf ledset,w ;(20)

movwf PORTD;(21)

movlw 0xFF;(22)

movwf count1;(23)

cycle1:;(24)

movlw 0xFF ;(25)

movwf count2 ;(26)

cycle2:;(27)

decfsz count2,f ;(28)

goto cycle2 ;(29)

decfsz count1,f ;(30)

goto cycle1 ;(31)

goto loop ;(32)

end ;(33)

1. **Краткий комментарий к программе**

В 1 строке обычно приводят фамилию автора, название программы, дату и другие данные которые могут потребоваться через неопределённое время, данная строка не обязательна.

Cтрока 2. list p=16F887 определяет тип микроконтроллера. Обычно в программе, написанной на языке ассемблер, применяется два типа команд: инструкции микроконтроллера, которые транслятор преобразует в исполняемый код микроконтроллера, и директивы, которые руководят работой самого транслятора (включение каких-либо библиотек, запись конфигураций и т.д.).

Строка 3. #include <p16F887.inc> - записана директива она означает, что в общую программу надо включить из библиотечных файлов все то, что относиться к выбранному типу микроконтроллера.

Строки 4 -5 \_\_CONFIG \_CONFIG1, \_LVP\_OFF & \_FCMEN\_OFF & \_IESO\_OFF & \_BOR\_OFF & \_CPD\_OFF & \_CP\_OFF & \_MCLRE\_OFF & \_PWRTE\_ON & \_WDT\_OFF & \_INTRC\_OSC\_NOCLKOUT\_\_CONFIG \_CONFIG2, \_WRT\_OFF & \_BOR21V здесь набор директив ассемблера под руководством которой заносится информация в некоторые частные директивы состояния процессора.

\_CP\_OFF - бит защиты кода после программирования не устанавливать.

\_WDT\_OFF - сторожевой таймер отключен.

\_PWRTE\_ON - таймер задержки сброса при подаче питания включен.

\_XT\_OSC - тип резонатора: керамический или кварцевый (кроме XT также могут быть значения LP, HS или RC, в соответствии с резонатором в схеме).

Строки 6-8. Определены необходимые для программы три пользовательских регистра (count1, count2, ledset). Для упрощения программирования им присвоены соответствующие имена (может быть произвольная запись, но имеется ряд некоторых ограничений: длинна записи, алфавит и др.) Запись EQU означает, что каждому пользовательскому регистру присваивается адрес (в общем случае произвольный номер, обязательно из диапазона пользовательских адресов в области данных, см. таблицу 1).

Строка 9. ORG 0x000 директива определяющая номер ячейки в которой должна быть записана команда по переходу к первой инструкции программы (вектор сброса процессора). При включении питания этот вектор (номер первой команды программы) загружается в СК и начинается выполнение программы. Запись вектора возлагается на компоновщик, иногда на транслятор, если последний генерирует абсолютный код (управление транслятором).

Строка 10. (clrf PORT D) служит для настройки порта D (обнуление регистра данных порта D). В приведённой программе этого можно и не делать на уровне «чайника», но правильный стиль программирования PIC микроконтроллеров требует, чтобы значения в выходных защёлках регистров данных портов были явно определены перед тем, как некоторые линии будут настроены на вывод. Поэтому в программе недостает команд на обнуление всех остальных портов (PORT A, PORT B, PORT C). Правильный стиль базируется на том, что к любым выводам регистра данных порта может быть подключено ведомое внешние устройство и в процессе написания программы уже придётся следить за тем, чтобы при нулевой информации на выходе порта (или шине) внешние оборудование было в состояние покоя и его запуск производится только путём выдачи ненулевой информации на органы управления. Обычно под понятием порт подразумевают два регистра: регистр состояния порта (TRIS), который определяет направление передачи значений каждого бита (ввод или вывод) в регистр данных порта (PORT). Если, например, младший(нулевой) бит регистра TRIS равен нулю, то по этому же номеру (биту) регистра данных порта будет осуществляться вывод информации, если в регистре TRIS младший бит равен единице, то запланирован ввод информации по этой линии в регистр данных.

Строка 11. (BSF STATUS, RP0) – это инструкция, смысл которой в том, что устанавливается в «1» бит RP0, чтобы дополнить этой единицей информацию в счётчике команд путём конкатенации для намерения работать в адресном пространстве 1-го банка памяти. По умолчанию при включении питания он сбрасывается в «0» (как и бит RP1). Но в микроконтроллере PIC6F887 память разбита на четыре банка. Поэтому желательно было бы ввести ещё информацию по обнулению бита RP1.

Строка 12. (MOVLW b’11111110’) означает, что эта двоичная информация помещается в аккумулятор W. Это необходимо делать потому, что система команд одноадресная (в командах можно указать только один адрес, регистр W не адресуется, имеет определённый и постоянный адрес). Поэтому при записи каких-либо констант в любой регистр(кроме аккумулятора) или перемещения данных из регистра в регистр приходится использовать промежуточный пункт (аккумулятор W).

Строка 13. (MOVWF PORT D) означает, что информация из аккумулятора (в данном случае константа «11111110») помещается в регистр TRIS D (расположен в первом банке памяти). Поэтому более квалифицированно (более понятна цель перехода в область адресов 1-го банка памяти) написать команду в виде MOVWF TRIS D. Но для транслятора запись PORT D или TRIS D- равноценна (транслятор генерирует одну и туже информацию при любой записи).

Строка 14. bcf OPTION\_REG,7 обязывает включить встроенную нагрузку только порта В (1-нагрузка отключена, 0-нагрузка включена). Это отключаемая нагрузка в виде резисторов подключает к шине питания все разряды (шины) порта данных. Но если линия настроена на вывод, эти резисторы автоматически отключаются. Таким образом, подтягивающие резисторы функционируют при вводе информации в порт, и служат для отстройки от помех (более чёткая фиксация логического нуля или единицы). На уровне «чайника» команда бесполезна, но правильный стиль программирования требует более чёткой настройки портов.

Строка 15. bcf STATUS,RP0;(15) – это инструкция, смысл которой в том, что сбрасывается в «0» бит RP0, и в дальнейшим предполагается работа в пространстве 0-го банка памяти. Все ниже перечисленные команды могут работать и в 1-ом и в 0-ом банках (адреса пользовательских регистров не зависят от номера этих двух банков) за исключением одного регистра (регистра данных порта D). В отличие от регистра состояния (см. строку 10), он расположен в адресном пространстве нулевого банка.

Строк 16. clrf ledset обязывает микроконтроллер очистить пользовательский регистр ledset (всем битам присваиваются нулевые значения). Эта операция уже положена в основу алгоритма по миганию выбранных светодиодов. Чтобы лучше понимать применение тех или других команд, необходимо разработать простейший алгоритм решения лабораторной задачи. В общем виде его можно привести в виде следующей схемы.

1. Ввод нулевой константы в вспомогательный регистр ledset, в дальнейшем для запоминания необходимой информации.

2. Определить 8 битную константу (К) по числу светодиодов. «1» в бите определит подачу напряжения 5 вольт на выход порта, «0» в бите определит подачу 0 вольт(светодиод не будет гореть).

3. Ledset  (К). Это булева операция (исключающее ИЛИ, выполняется в аккумуляторе W). На нечётном шаге установит в ledset необходимый бит в «1». Светодиод будет гореть. На чётном шаге в необходимые биты в ledset установит в «0» .Светодиоды не горят.

4. Результат операции отправить в порт на вывод (на управление светодиодами).

5. Установить временную задержку в работе программы (установится мерцание светодиодов). Обычно это зацикливание (несколько циклов)программы, на выполнение каждого цикла затрачивается время.

7. Возврат в программу для многократного повторения алгоритма.

8.СТОП

**Рисунок 2 -** Алгоритм решения лабораторной задачи

Это не единственный алгоритм решения задачи. Например, на уровне «чайника» действия (3 и 4 алгоритма) можно заменить операцией «не» с данными регистра ledset. На нечётном шагу всё будет в соответствии с приведённым выше алгоритмом, а на чётном, где были единицы, будут нули и наоборот. Но разряды с единицей настроены не на вывод, а на ввод и не окажут существенного влияния на свечение светодиодов. Существенный недостаток рассмотренного решения задачи - это зацикливание программы (микроконтроллер зависает) и остановка выполняется только выключением питания лабораторной установки.

Строка 17. loop(петля, контур) определяет для транслятора метку возврата (обычно используется для организации циклов). При написании меток имя должно заканчиваться двоеточием (:) (это синтаксис ассемблера).

Строка 18. movlw b ‘0000001’ – необходимая константа (К) (см. алгоритм) записывается в аккумулятор W.

Строка 19 xorwf ledset,f – производится операция «исключающее ИЛИ» и результат сохраняется в пользовательском регистре ledset.

Строки 20 и 21 программы через вспомогательный регистр W заносят результат «исключающего ИЛИ» в регистр данных порта D для непосредственного зажигания светодиода на нечётном шаге прохождения 0-го (главного) цикла.

Строки 22 - 23. В счётчик циклов count1 (пользовательский регистр) через аккумулятор (одноадресная система команд) заносится максимальное 8-битное число (=15\*16+15=255).

Строка 24. cycle1 указывает транслятору метку возврата для организации цикла 1. Если метка loop служит для организации бесконечного цикла, то здесь цикл 1 будет выполнятся определённое число раз.

Строки 25-27. Служит тем же целям что и строки 22,23,24, но только для организации 2-го цикла. Эти 2 цикла являются примером компактного написания программы для временной задержки. Совместная работа циклов будет выполняться N=255\*255 раз, теоретически можно было время задержки организовать и одним циклом, если в его тело вписать N выполняемых команд (большой объём затрачиваемой памяти и огромное число строк).

Строки 28 и 29. decfsz count2,f –декремент счётчика ; 29 goto cycle2 если содержимое count2 не ноль то переход на метку cycle2, если ноль, то пропуск команды goto cycle2, т.е. переход к команде 30. Эти команды являются частью цикла 2.

Строки 27–29 – организуют тело цикла 2. Когда цикл 2 выполняет задачу (=15\*16+15=255) раз, то будет выполнен переход к команде decfsz count1,f.

Строки 30 - 31 decfsz count1,f ; goto cycle1 служат тем же целям, что и строки 28,29, рассмотренные выше. Если последние строки служат частью цикла 2, то строки 30,31 являются частью (последними командами цикла 1), таким образом тело цикла 1 образуют строки (команды) 24-31, туда входят все команды 2-го цикла (программисты говорят, что цикл 2 вложен в цикл 1).

Строка 32 goto loop. Если учитывать взаимодействие команды 31 и 32 то этот переход к метке loop будет выполняться при условии, когда в счётчике цикла 1 (count1) будет нулевая информация. Цикл будет функционировать до выключения питания лаб. установки. С точки зрения «прогонки» программы это явный недостаток. Но программисту, работающему с PIC, следует учитывать, что микроконтроллер – это неотъемлемая часть периферийного оборудования. И если питание оборудования не выключено, то микроконтроллер тоже должен функционировать (например, если печь включена, а программа управления отключена, то последствия непредсказуемые вплоть до её возгорания). Поэтому здесь, как и в приведённых программах в [1], имеется бесконечный цикл для непосредственного функционирования вплоть до выключения питания (это цикл можно назвать главным, «нулевым»). В системе команд (таблица 2) есть команда остановки функционирования СК (счётчик команд) (SLEEP), тогда PIC будет «спать» с минимальным энергопотреблением. Повторный запуск осуществляется кратковременной подачей низкого потенциала на вывод MCLR (см. рисунок 1, клемма 18).

Строка 33. end последняя директива, должна всегда присутствовать в программе. Она обозначает для транслятора конец текста программы (завершение процесса трансляции).

1. **Создание проекта, Mplab IDE V8.46**

**Примечание по Mplab IDE.**

**Среда Mplab IDE V8.46. предварительно установлена на компьютер, если её нету, то взять программу у преподавателя.**

Интегрированная среда проектирования MPLAB-IDE

Программное обеспечение MPLAB-IDE предназначено для разработки программного обеспечения 8-разрядных микроконтроллеров PICmicro, работающее под управлением операционной системы Windows.

Основные характеристики MPLAB-IDE:

* Многофункциональные возможности:
* Редактор;
* Симулятор;
* Программатор (приобретается отдельно);
* Эмулятор (приобретается отдельно).
* Полнофункциональный редактор.
* Организатор проекта.
* Настройка панелей инструментов и параметров отображения.
* Строка состояния.
* Интерактивная помощь.

MPLAB-IDE позволяет Вам:

* Редактировать исходные файлы написанные на языке ассемблера или C.
* Быстро выполнять трансляцию и компиляцию проекта автоматически загружая параметры используемого микроконтроллера PICmicro.
* Выполнять отладку программы с использованием:
* Исходных файлов;
* Листинга программы;
* Объектного кода.

Однотипная работа инструментальных модулей интегрированной среды проектирования MPLAB-IDE позволяет легко перейти от программного симулятора MPLAB-SIM к использованию полнофункционального эмулятора.

***Программный симулятор MPLAB-SIM***

Симулятор MPLAB-SIM позволяет проследить выполнение программы микроконтроллеров PICmicro на уровне команд по шагам или в режиме анимации. Под этим термином подразумевается, что в библиотечных файлах есть программа имитирующая работу микроконтроллера на выполнение написанной программы с некоторыми сервисными функциями (в основе лежит «виртуальный» микроконтроллер). На любой команде выполнение программы может быть остановлено для проверки и изменения памяти. MPLAB-SIM полностью поддерживает символьную отладку, используя MPLAB-C17, MPLAB-C18 и MPASM. MPLAB-SIM является доступным и удобным средством отладки программ микроконтроллеров PICmicro.

***Универсальный эмулятор MPLAB-ICE***

Универсальный эмулятор MPLAB-CE обеспечивает разработчиков полным набором инструментальных средств для проектирования устройств с применением микроконтроллеров PICmicro. Здесь понимается, что «виртуальный» микроконтроллер способен обеспечивать внешние воздействия и обрабатывать их в соответствии с алгоритмом программ (работа в реальном времени)поэтому без имитации внешних воздействий (заложено в программе) отладка такой программы не будет полноценной ( с помощью только симулятора). Эмулятор позволяет менять внешние уровни на выводах и записывать заданные значения напрямую в регистры таким образом, что в эмуляторе кроме «виртуального» микроконтроллера есть программа симуляции внешних воздействий. Управление работой эмулятора выполняется из интегрированной среды проектирования MPLAB-IDE с возможностью редактирования, компиляции, загрузки и выполнения программы.

Универсальная архитектура MPLAB-ICE дает возможность поддерживать новые типы микроконтроллеров PICmicro.Эмулятор MPLAB-ICE был разработан как система эмуляции (анимации) в реальном масштабе времени с дополнительными возможностями, присутствующих в дорогих инструментальных средствах. Эмулятор работает под управлением распространенной операционной системы Microsoft Windows 3.x/95/98.

MPLAB-ICE 2000 - полнофункциональная система эмуляции с усовершенствованными функциями трассировки, триггеров и управляющих особенностей. Оба эмулятора используют одинаковые поды и работают во всех допустимых режимах микроконтроллеров PICmicro.В настоящей лабораторной работе с целью экономии времени среда **Mplab IDE V8.46** предварительно установлена.

**1)** Предварительно на любом диске создать папку для хранения проекта.

**2)** Запустить программу Mplab IDE V8.46.

**3)** Выбор микроконтроллера. Курсором выбрать в строке кнопок меню «**Configure**», выбрать «**Select Device**», щелкнуть. Появляется окно «**Select Device**». В правой части белой строки находится список, щелкнуть по нему, появляется список микроконтроллеров, найти нужный (PIC16F887), щелкнуть по нему. Щелкнуть по кнопке «Ок».

**4)** В окне **MPLAB IDE V8.46**, в верхней части меню щёлкнуть по «**Project**», появляется вертикальный список, выбрать «**Project Wizard**», щелкнуть, появляется окно «**Project Wizard**». В нижней части щелкнуть по кнопке «**Далее**», появляется другое окно, в центре этого окна убедиться в правильности ранее выбранного микроконтроллера. В нижней части этого окна щелкнуть по кнопке «**Далее**», повторить это действие в новом окне, не меняя конфигурацию окон. В следующем окне располагается кнопка «**Browse**», щелкнуть, указать путь к папке созданной ранее в пункте 1.Возвратиться к окну «**Project Wizard**», у которого вверху, в белой строке, прописан путь, где хранится разрабатываемый проект. Щелкнуть по кнопке «**Далее**». Повторить то же самое в следующем окне, появляется новое окно, в нем щелкнуть по кнопке «Готово». Теперь имя проекта создано, вместо окна «**Untitled Workspace**» пявится окно <имя проекта>.

**5)** Создание файла с расширением **.ASM**. В окне MPLAB IDE V8.46 в строке меню выбрать и щелкнуть кнопку «**File**», щелкнуть по строке «**New**», открывается окно «**Untitled**», необходимо его сохранить и добавить в разрабатываемый проект. Для этого в строке меню щелкнуть по кнопке «**File**», выбрать из появившегося списка кнопку «**Save As**», щелкнуть по ней, появляется окно «**Сохранить как**», в белой строке «**Имя файла**» записать с клавиатуры имя, такое же как имя проекта, но только обязательно с расширением .**ASM** (**например 111. ASM**). Далее щелкнуть по кнопке «Сохранить», вместо окна «Untitled», появится окно к примеру D:/Microchip/lll. ASM. (D:/ 111. ASM). Файл 111. ASM необходимо добавить в разрабатываемый проект. Для этого в строке меню щелкнуть по кнопке «**Project**», появляется список, щелкнуть по строке «**Add Files to Project**», появляется окно «**Add Files to Project**», в верхней части окна щелкнуть по имени сохраненного файла (например, 111. ASM). Далее щелкнуть по кнопке «Открыть. Рядом с кнопкой «**Source Files**» ниже должна появиться закладка с разрабатываемым проектом и расширением. ASM (например, 111. ASM). В окне D: (Microchip) 111. ASM напечатать программу, приведенную в данном пособии. **Готовая программа находится на диске**.

**6)** Щелкнуть по кнопке «**Programmer**», чуть ниже появляется строка «**Select Programmer**», щелкнуть по ней. Появляется список программаторов, поднести курсор и щелкнуть по PICKIT 2. Если программатор не подключен, в окне «**Output**» появляется строка красного цвета «**Picket not found**». При включении программатора будет строка черного цвета «**PIC16F887 found (Rev.ox2)**», Красных строк не должно быть.

**7)** Чтобы записать программу в микроконтроллер надо вначале очистить его память от предыдущих записей. Для этого подвести курсор на кнопку «**Programmer**» и щелкнуть, появится список, щелкнуть по строке «**Erase**». В окне «**Output**» в вкладке PICKIT 2 должна появиться запись «**Erasing Target**», подтверждающая успешную очистку.

**8)** Откомпилировать (перевести в машинный код). Для этого в меню «**MPLAB**» щелкнуть по кнопке «**Project**», в появившемся окне щелкнуть по «**QuickBuild**», появится новое окно, щелкнуть по «**Absolute**», проверить отсутствие ошибок. Строка «**Errors**» должна быть равна нулю в окне «**Output**». При наличии ошибок в окне «**Output**» появится строка красного цвета: «**Build Failed**». Единственный способ устранения ошибок – отладка программы.

**9)** Запись программы в микроконтроллер. Щелкнуть по значку «**Programmer**», появится список, в нем щелкнуть по строке «**Program**». Теперь новая программа введена и записана в микроконтроллер, мигает тот или те светодиоды, которые заданы преподавателем.

**8.Описание лабораторной установки**

Промышленность выпускает контроллеры, имеющие различные конструктивные исполнения. Их форма может быть квадратной, прямоугольной, выводы - штыревые, плоские и т.д. При штыревых выводах микроконтроллер вставляют в разъем, в случае порчи вынимают из разъема и при отключенном напряжении питания вставляют новый, запрограммированный. В принятом в лабораторной работе варианте выводы плоские, должны припаиваться, следовательно, в случае порчи, необходимо его выпаивать. Микроконтроллеры могут поставляться как в виде отдельных микросхем, так и уже припаянные на печатной плате с дополнительными функциональными элементами. В данной лабораторной работе используется последний вариант. На рисунке 1, а) эти выводы пронумерованы от 1 до 44, там же приведены названия портов, например вывод 1 - порт RC7. На корпусе микроконтроллера также есть метка, в данном случае точка (.). Отсчёт производится от метки, первый вывод начинается с точки против часовой стрелки. Но это потому, что наблюдатель смотрит на выводы сверху. Если смотреть на выводы снизу, со стороны припаиваемых площадок, то отсчёт номеров выводов производить по часовой стрелке, как принято и для всех других микросхем, в том числе и микроконтроллеров. В корпусе макета из оргстекла, в котором размещён микроконтроллер, дополнительно расположен программатор (Pickit 2)и блок питания +5В, предназначенный для снабжения электроэнергией микроконтроллера для включения в работу.

**9.Методика выполнения лабораторной работы**

1. Ознакомиться с текстом описания лабораторной работы.
2. Проверить наличие среды MPLAB и начальной программы, приведённой на винчестере (в случае отсутствия взять диск у преподавателя и загрузить необходимые компоненты).
3. Произвести очистку «памяти» микроконтроллера, ввести начальную программу в память микроконтроллера, запустить PIC - программатор в работу. Наблюдать свечение нулевого светодиода.
4. Проверить некоторые варианты программы, отмеченные в разделе «краткий комментарий к программе»

Задание. 1**.**

строка 12) movlw b'11111110'

строка 18) movlw b'00000001'

Задание. 2.

строка 12) movlw b'11111110'

строка 18) movlw b'11111111'

Задание. 3.

строка 12) movlw b'00000000'

строка 18) movlw b'00000001'

Строку 19 заменить на COMF ledset,f

Сделать выводы о проделанной работе.

1. Видоизменить начальную программу с учётом номеров зажигания светодиодов (номера уточнить у преподавателя).Нумерация светодиодов справа-налево, нумерация начинается с 0. Прогнать программу.
2. Изменить интервал мигания светодиодов в соответствии с заданием преподавателя. Увеличить интервал свечения по отношению к исходной программе. Зафиксировать время свечения t1.
3. Указание: необходимо выполнить изменения в программе путём организации 3-го и 4-го циклов. Для этого организовать дополнительные пользовательские регистры count3, count4 и дополнительный 3-ий и 4-ый цикл, в тела этих циклов вводить (если необходимо) дополнительные команды NOP. Изменением значения записи в счётчики циклов (count1, count2, count3, count4) и числа команд NOP, добиться необходимого результата. Время зафиксировать секундомером. Для удобства числа можно вводить в десятичном эквиваленте (диапазон от 1 до 255), запись в виде «.255» или «D’255’».
4. Изменить четвёртый (последний вложенный) цикл вводом одной дополнительной команды NOP. Зафиксировать время горения t2.
5. Рассчитать время выполнения команды NOP (быстродействие PIC).

**10. Контрольные вопросы**

1. Рассказать об основных этапах процесса программирования микроконтроллеров.
2. Рассказать о логике программирования «Ассемблер»(директивы, инструкции команды).
3. Назначение программатора.
4. Рассказать о среде программирования, возможных источниках ее получения.
5. Рассказать о процедуре установки (ввода) среды в компьютер.
6. Рассказать о программе на языке «Ассемблер», основных ее частях, последовательности составления, циклах, способе организации временных интервалов для мерцания светодиодов.
7. Какие сигналы выводятся из портов в микроконтроллере PIC16F887, их уровни?
8. Директивы и инструкции (команды) в программе.
9. Типовой трафарет при составлении программы на языке «Ассемблер».
10. Банки памяти. Их объём, адреса.
11. Двоичные, десятичные и шестнадцатеричные числа. Алгоритм перевода.
12. Рассказать об алгоритме данной программы, возможных других алгоритмах.
13. Циклы с ограничением и без ограничений.
14. Как удалить введенную среду из памяти компьютера?
15. Как удалить из памяти компьютера папки и другие знаки, ошибочно введенные или после окончания лабораторной работы?
16. Ввод программы с диска в память компьютера.
17. Отладка программы. Эмуляция и симуляция.

**11.Литература**

Яценков В.С. Микроконтроллеры Microchip. Практическое руководство. – 2-е изд. испр. и дополн. – М.: Горячая линия – Телеком. 2008. – 280с. ил.

Мортон Дж. Микроконтроллеры AVR. Вводный курс. /Пер. с англ. – М. Издательский дом «Додека – ХХI» 2006 – 272с. ил.