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

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

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

Учебно-методическое пособие

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

Лабораторный практикум по микроконтроллеру PIC16F84A

Сыктывкар 2018

СОДЕРЖАНИЕ

Введение

Цифровые методы обработки информации в настоящее время составляют одно из основных направлений развития радиоэлектроники, компьютерной и микропроцессорной техники, телекоммуникационных и информационных систем. Микропроцессорные устройства и системы широко применяются практически во всех видах современной профессиональной радиоэлектронной аппаратуры и компьютерной техники, в телекоммуникациях и системах связи.

Дисциплина «Микропроцессорные системы» (МПС) посвящена рассмотрению принципов построения, функционирования и проектирования микропроцессорных устройств и систем, различных подсистем МПС, а также вопросов практической реализации названных устройств и систем.

Освоение материала требует знания логических операций и теории переключательных функций, двоичной системы счисления и действий над двоичными числами, а также схемотехники цифровых логических элементов и устройств.

Цель преподавания дисциплины МПС – научить студентов базовым знаниям, выработать навыки анализа, проектирования, моделирования и экспериментального исследования микропроцессорных устройств и систем.

Предметом изучения являются микропроцессоры (микроконтроллеры), применяемые главным образом в управляющих системах, а также основы построения микропроцессорных систем.

Лабораторные занятия по дисциплине нацелены на освоение студентами основ проектирования встроенных микропроцессорных систем на базе микроконтроллеров PIC и разработке управляющих ассемблерных программ для них.

Задачи изучения дисциплины «Микропроцессорные системы» определяются требованиями, установленными в Федеральном образовательном стандарте высшего профессионального образования к подготовке бакалавров по всем техническим направлениям кафедры «Информационный и электронный сервис».

Перед проведением лабораторной работы студент должен изучить теоретические сведения по соответствующей учебной и справочной литературе.

В отчете по лабораторной работе студент обязан сделать выводы о соответствии теоретических и модельных результатов, а также объяснить возможные расхождения.

Отчеты по лабораторным работам выполняются на листах формата А4. По окончании лабораторного практикума студентом формируется журнал отчетов и сдается преподавателю.

При работе в лаборатории студент обязан строго выполнять правила техники безопасности, соблюдать порядок и тишину при проведении лабораторных работ, заботиться о сохранности лабораторного оборудования и мебели.

Распределение тем лабораторных занятий по времени

№	Наименование темы лабораторного занятия	Тема дисциплины	Используемые приборы и лабораторное оборудование
1	*Лабораторная работа № 1.* Интегрированная среда разработки MPLAB для PIC-микроконтроллеров. Изучение системы команд микроконтроллеров PIC16F8x /1-4/, /7/, /9-10/	2	ПК
2	Лабораторная работа № 2. Программирование микроконтроллеров PIC16F8xx на языке ассемблера MPASM /1/, /3/, /7/, /13/	5,7,8,9	ПК
3	*Лабораторная работа №3.* Программное управление портами ввода-вывода микроконтроллера PIC16F84A /1-4/, /6/, /8-12/	3	ПК
4	Лабораторная работа №4. Система прерываний МК PIC16F84A. Использование прерываний для программного управления таймером TMR0 /1-4/, /6/, /8-12/	4	ПК

Лабораторная работа № 1. Интегрированная среда разработки MPLAB для PIC-микроконтроллеров. Изучение системы команд микроконтроллеров PIC16F8xx

Цель работы: Изучение интегрированной среды разработки MPLAB IDE и системы команд МК PIC16F84A, получение начальных навыков написания управляющих программ для этого микроконтроллера.

ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ MPLAB IDE

MPLAB IDE – бесплатная интегрированная среда разработки для микроконтроллеров PICmicro фирмы Microchip Technology Inc. MPLAB IDE позволяет писать, отлаживать и оптимизировать текст управляющей программы для МК этой серии. MPLAB IDE включает в себя редактор текста, симулятор и менеджер проектов, поддерживает работу эмуляторов (MPLAB-ICE, PICMASTER) и программаторов (PICSTART plus, PRO MATE) фирмы Microchip и других отладочных средств фирмы Microchip и третьих производителей.

Легко настраиваемые инструментальные средства, тематическая помощь, выпадающие меню и назначение «горячих» клавиш в MPLAB IDE позволяют:

- получить код программы;
- производить отладку программы;

- наблюдать выполнение программы с помощью симулятора, или в реальном времени, используя эмулятор (требуется аппаратная часть);
- определять время выполнения программы;
- просматривать текущие значения переменных и специальных регистров;
- наблюдать значения отдельных переменных в окне *Watch*;
- работать с программаторами PICSTAR и PRO MATE II;
- использовать систему помощи по MPLAB IDE.

MPLAB IDE позволяет создавать исходный текст программы в полнофункциональном текстовом редакторе, легко исправлять ошибки при помощи окна результатов компиляции, в котором указываются возникшие ошибки и предупреждения.

Используя менеджер проектов, можно указать исходные файлы программы, объектные файлы, библиотеки и файлы сценария.

MPLAB IDE обеспечивает разнообразные средства симуляции и эмуляции исполняемого кода для выявления логических ошибок. Их основные особенности:

- большое количество сервисных окон, чтобы контролировать значения регистров памяти данных и выполнение инструкций микроконтроллера;
- окна исходного кода программы, листинга программы, кода программы - позволяют оценить качество компиляции;
- пошаговое выполнение программы, система точек остановки, трассировки, сложных условий предназначена для быстрой и удобной отладки программы.

MPLAB IDE состоит из нескольких модулей, обеспечивающих единую среду разработки:

1. *Менеджер проекта MPLAB*. Используется для создания и работы с файлами, относящимися к проекту. Позволяет одним щелчком мыши выполнить компиляцию исходного текста, включить симулятор или внутрисхемный эмулятор и т.д.
2. *Редактор MPLAB*. Предназначен для написания и редактирования исходного текста программы, шаблонов и файлов сценария линкера (компоновщика).
3. *Отладчик MPLAB ICD*. Недорогой внутрисхемный отладчик для микроконтроллеров семейства PIC16Fxx (в лабораторных работах не используется).
4. *MPLAB-SIM симулятор*. Программный симулятор моделирует выполнение программы в микроконтроллере с учетом состояния портов ввода/вывода.
5. *MPLAB ICE эмулятор*. Эмулирует работу микроконтроллера в масштабе реального времени непосредственно в устройстве пользователя (в лабораторных работах не используется).
6. *MPASM ассемблер / MPLINK линкер / MPLIB редактор библиотек*. MPASM компилирует исходный текст программы. MPLINK создает заключительный код программы, связывая различные модули полученные из MPASM, MPLAB-C17, MPLAB-C18. MPLIB управляет библиотеками.

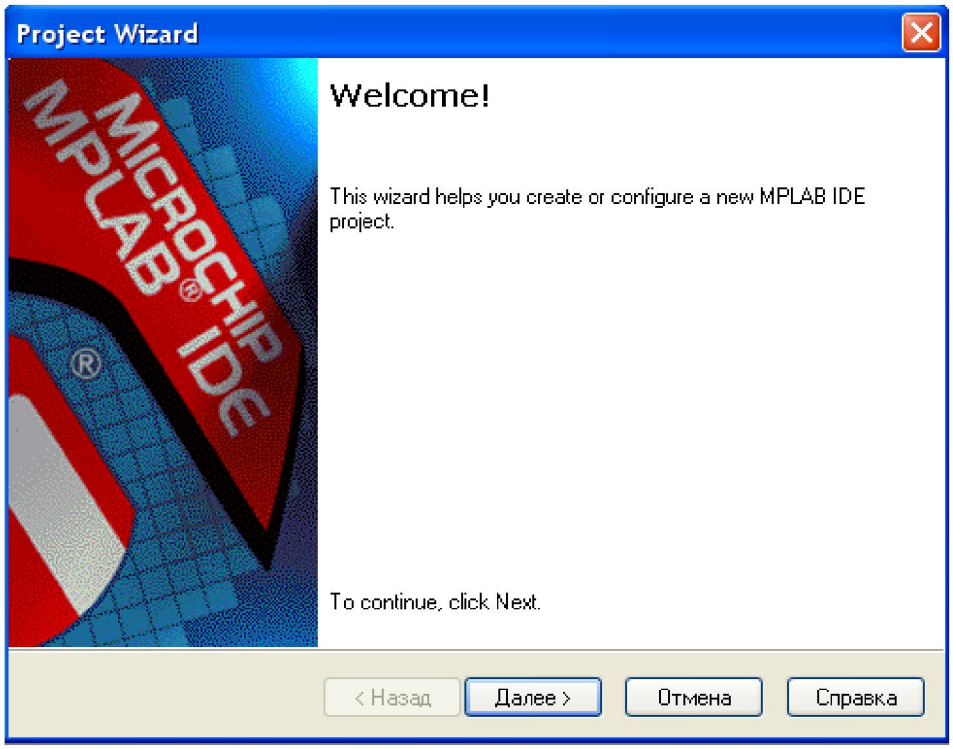
7. *Программаторы PRO MATE и PICSTART plus*. Работают под управлением MPLAB IDE и предназначены для программирования микроконтроллеров кодом программы, полученной в результате компиляции исходных файлов. Программатор PRO MATE может работать самостоятельно, без использования MPLAB IDE (в лабораторных работах не используется).
8. *Эмуляторы MPLAB-ICE, PICMASTER-CE и PICMASTER.* Применяются для моделирования работы микроконтроллера в устройстве пользователя в масштабе реального времени (в лабораторных работах не используется).

Все проекты, создаваемые в MPLAB IDE, включают несколько шагов:

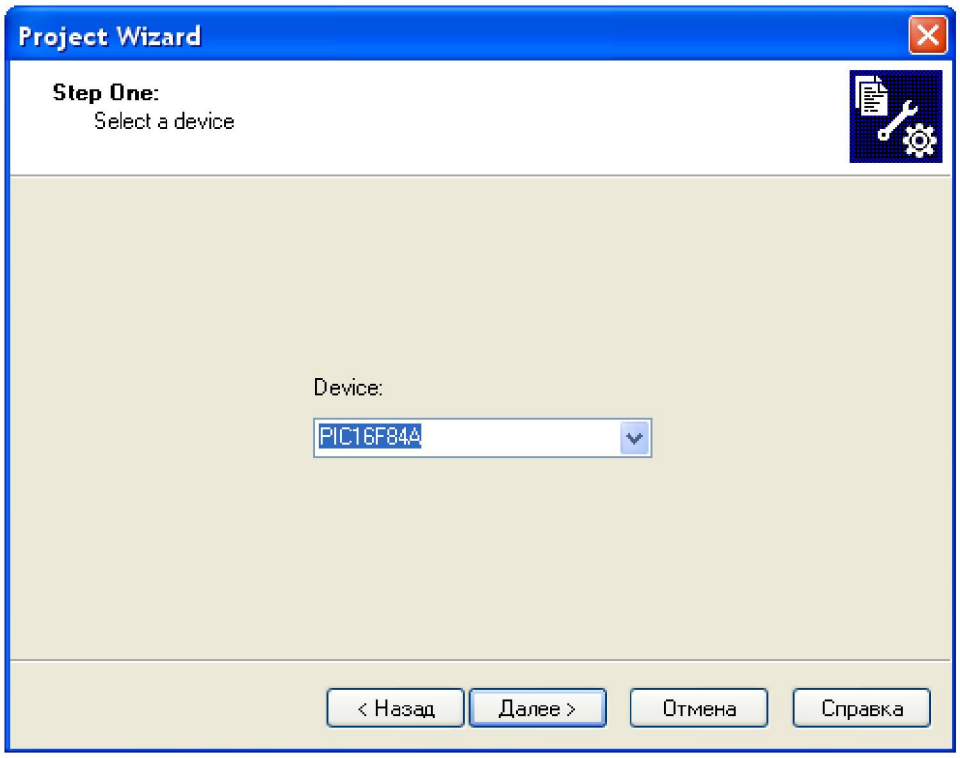
- выбрать тип микроконтроллера;
- создать проект;
- выбрать языковые средства (ассемблер и компоновщик);
- добавить файлы в проект;
- создать программный код;
- скомпилировать проект (build);
- протестировать и отладить код с помощью симулятора.

СОЗДАНИЕ ПРОЕКТА В MPLAB IDE

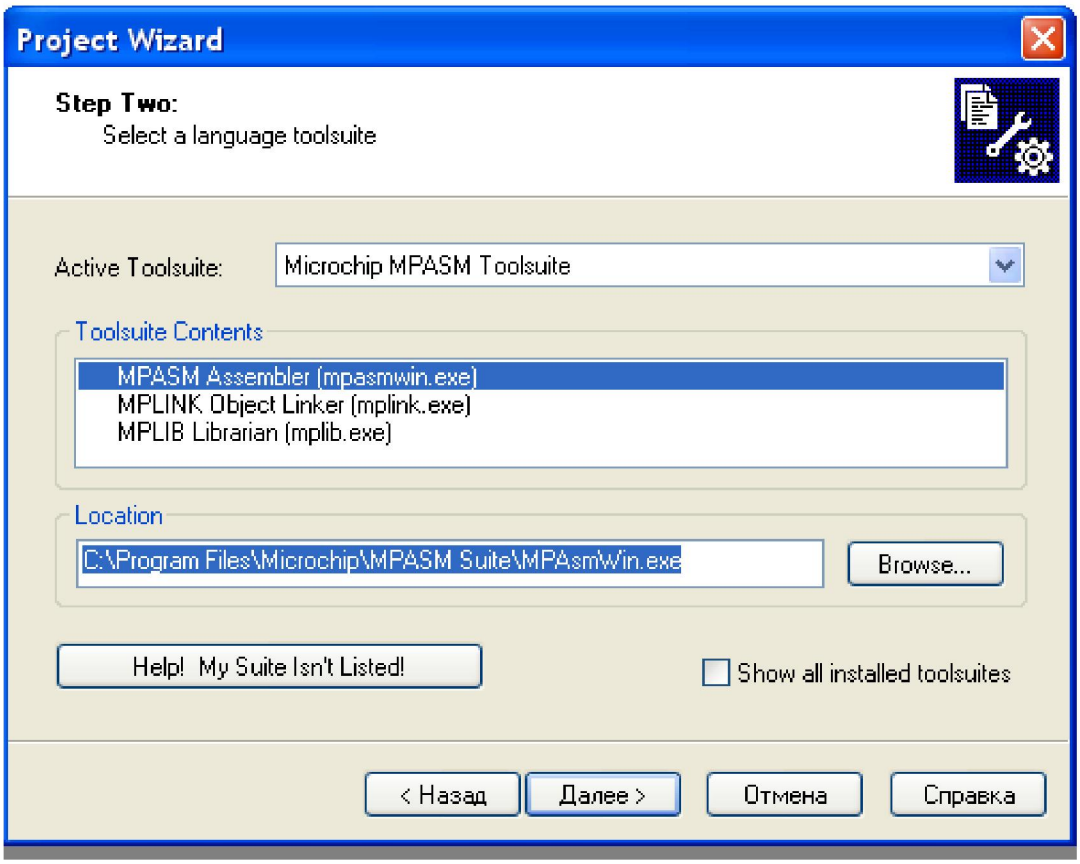
1. При создании проекта в MPLAB IDE первый шаг заключается в выборе используемого микроконтроллера по его типу. Это делается щелчком мыши на пункте меню **Configure** (Конфигурировать). Далее нужно щелкнуть на команде подменю **Select Device** (Выбор устройства). В появившемся диалоговом окне выберите из списка устройств МК, в нашем случае выберите PIC16F84A. Эти действия сконфигурируют IDE на тип МК, который будет использоваться в проекте, - *они должны быть выполнены в первую очередь, чтобы обеспечить правильную работу IDE*.
2. Щелкните на пункте меню **Project** и выберите команду **Project Wizard…**, затем следуйте указаниям мастера проектов, как показано на следующих рисунках.
3. Выполните команду меню **View/Project**, чтобы открыть окно диспетчера проектов.
4. Выполните команду меню **File/New** и скопируйте в открывшееся окно текстового редактора приведенный ниже шаблон исходного файла.
5. Щелкните правой кнопкой мыши в окне редактора, в контекстном меню выберите команду *Properties…* В открывшемся диалоговом окне *Editor Options* перейдите на вкладку *File Type*, установите флажок *Line Numbers* и нажмите кнопку *OK*.
6. Сохраните созданный текстовый файл с расширением .asm в каталоге проекта, выполнив команду меню **File/Save As…** и присвоив файлу имя, например, prj_1.asm.
7. Щелкните правой кнопкой мыши на папке *Source File* диспетчера проектов, в контекстном меню выберите команду *Add Files…* и добавьте к проекту созданный файл prj_1.asm.



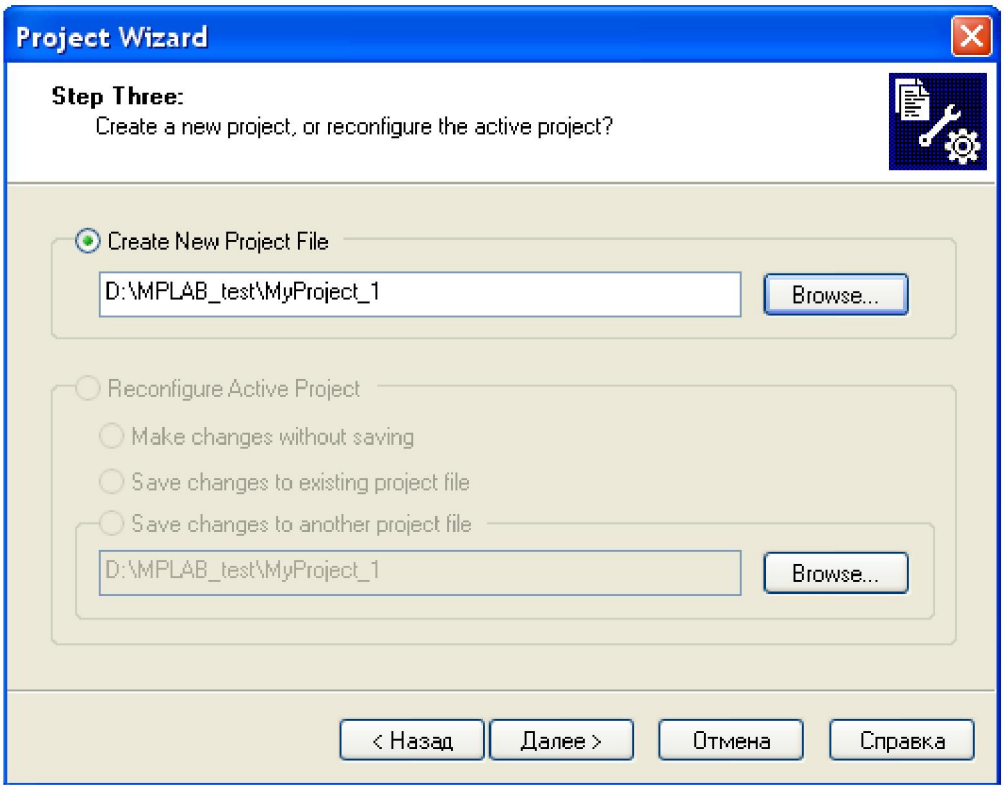
Приветствие мастера проектов.



Шаг 1 мастера проектов.

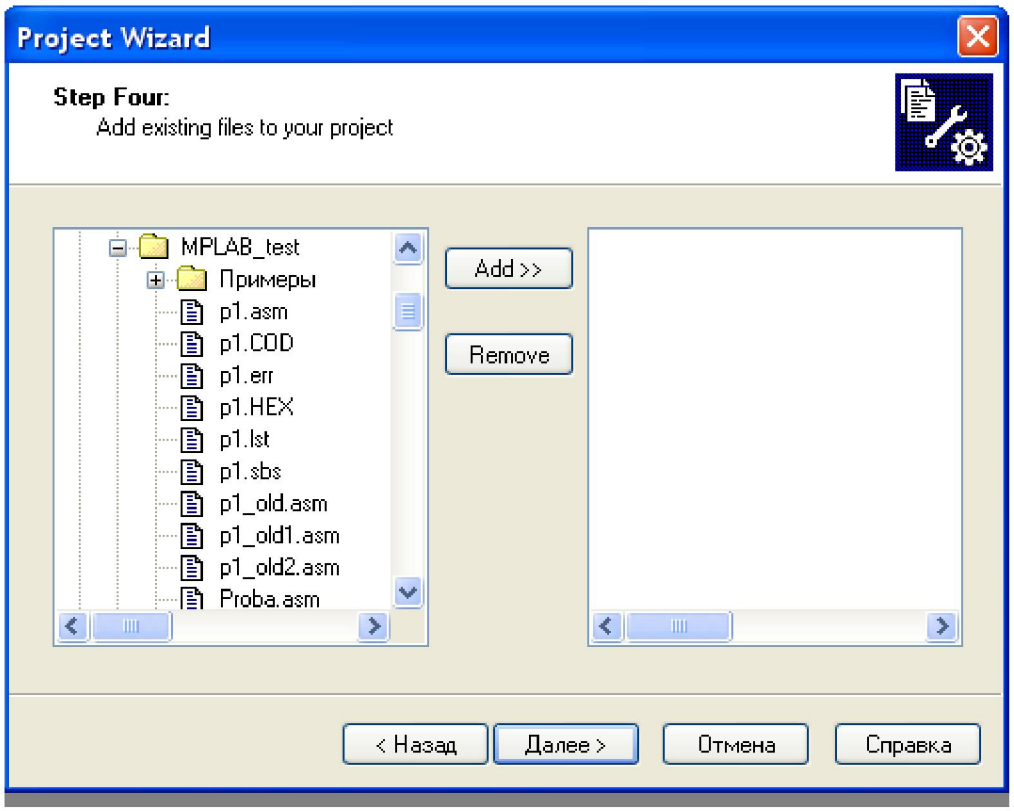


Шаг 2 мастера проектов.

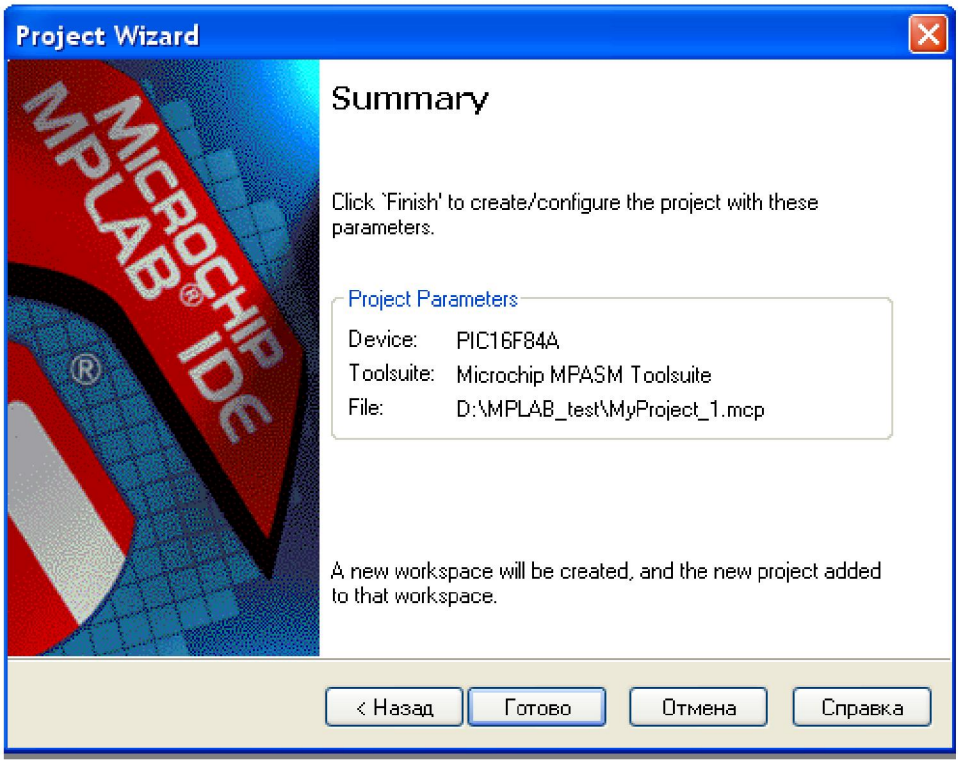


Шаг 3 мастера проектов.

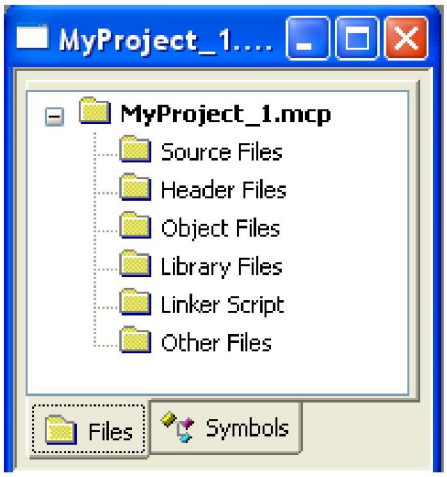
(***Внимание***. Папка *MPLAB_test*, в которой будут храниться файлы проекта, должна быть предварительно создана).



Шаг 4 мастера проектов.

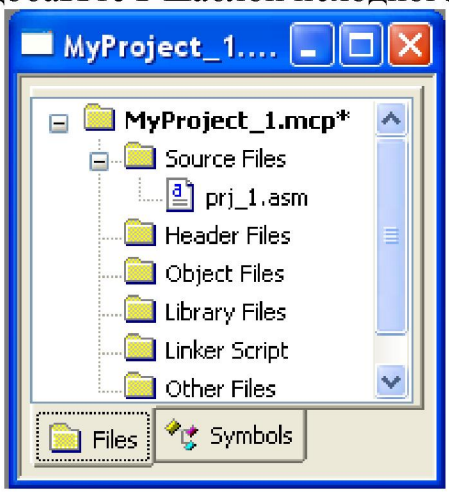


Заключительный шаг мастера проектов.



Окно диспетчера проектов.

8. В текстовом редакторе добавьте в шаблон исходного кода свои команды.



9. Проект готов к использованию.

КОМПИЛИРОВАНИЕ, ВЫПОЛНЕНИЕ И ОТЛАДКА ПРОЕКТА

1. Чтобы выбрать средство отладки, выполните команду меню **Debugger/Select Tool/MPLAB SIM**.
2. Чтобы скомпилировать проект, выполните команду меню **Project/Make** (горячая клавиша F10).
3. Запустите проект на выполнение, выполнив команду меню **Debugger/Run** (клавиша F9).
4. Для остановки зацикленной управляющей программы выполните команду меню **Debugger/Halt** (клавиша F5).
5. Для выполнения проекта в пошаговом режиме вместо клавиши F9 последовательно нажимайте клавишу F7. Чтобы при этом наблюдать за состоянием специальных регистров, памяти данных и программ, предварительно выберите пункт меню **View** и выберите инструменты просмотра, такие как *File Registers*, *Special Function Registers*, *Program Memory* и др.
6. Для сброса процессора МК и перехода к первой команде управляющей программы нажмите клавишу F6.

Для удобства изучения все команды МК PIC16F84A разделены на 6 категорий:

- *2 команды очистки регистров*: CLRF, CLRW;
- *3 команды загрузки и пересылки*: MOVLW, MOVWF, MOVF;
- *6 команд арифметических операций*: INC F, DEC F, ADDLW, ADDWF, SUBLW, SUBWF;
- *10 команд логических операций*: COMF, ANDLW, ANDWF, IORLW, IORWF, XORLW, XORWF, RLF, RRF, SWAPF;
- *4 условные команды*: BTFSC, BTFSS, DECFSZ, INCFSZ;
- *10 команд управления*: CLRWDT, BCF, BSF, CALL, GOTO, NOP, RETFIE, RETLW, RETURN, SLEEP.

СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРА PIC16F84

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1 (2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1 (2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

Подробное описание команд МК PIC16F84A приведено в *Приложении 1* к лабораторной работе.

ИСЛЕДОВАНИЕ КОМАНД СЛОЖЕНИЯ И ВЫЧИТАНИЯ В МК PIC16F84A

Для исследования команды сложения двух операндов в MPLAB:

1. Создайте новый проект, если он еще не создан.
2. Добавьте к нему файл переименованной копии шаблона исходного файла.

Шаблон исходного файла template.asm для PIC16F84A

```
; ===== Секция заголовка =====
list   p=PIC16F84
include <P16F84.INC>

;===== Рабочая секция =====
; Начало исполняемого кода
   org   0
   goto  begin
   org   100h     ; при компиляции нижеследующий код будет размещен в
                  ; памяти программ, начиная с адреса 100h
begin
; Пример
   ; --- сюда вставляйте код примера ---
; Конец примера
   loop  goto loop
end
```

3. В секцию заголовка этого файла добавьте строки:

```
x   equ  0Ch
y   equ  0Dh
```

4. В рабочую секцию файла после комментария «сюда вставляйте код примера» добавьте следующий код

```
movlw  14H
movwf  x
addlw  0E0H
movwf  y
```

5. Скомпилируйте и выполните программу, изучите изменение состояний регистров W, x, y, STATUS.
6. Повторяйте этот эксперимент, варьируя значения переменной `x` и константы `k` в команде `addlw k`.

Рассмотрим подробно два примера выполнения этой программы.

1. Пусть (W) = 14h = 00010100_2, k = A0h = 10100000_2. Тогда результат вычислений равен B4h = 10110100_2, а значения флагов DC статусного регистра равно 00_2. Результат получается сложением значений (W) и k:

$$00010100_2 + 10100000_2 = 10110100_2 \text{ или } 14h + A0h = B4h$$

2. Пусть теперь (W) = 24h = 00100100_2, k = E0h = 11100000_2. Тогда результат вычислений равен 04h = 10110100_2, а значения флагов DC статусного регистра равно 01_2. Результат снова получается сложением значений (W) и k:

$$00100100_2 + 11100000_2 = 00000100_2 \text{ или } 24h + A0h = 04h + \langle\text{перенос}\rangle$$

Видим, что сложение двух операндов выполняется как арифметическое сложение по модулю 256. Если результирующая сумма превышает 255, то в регистре STATUS устанавливается флаг переноса.

Операция вычитания выполняется как сложение двух чисел, при этом второе число берется в дополнительном коде. В этом можно убедиться на следующем примере:

```
movlw  15H        ; 15h->(W), на флаги не влияет
movwf  x          ; (W)->x, на флаги не влияет
sublw  05H        ; 05h-(W)->(W), DC=10
movwf  y          ; (W)->y, на флаги не влияет
sublw  0          ; Вычисление модуля числа |05h-(W)|, DC=10
```

Действительно, $(15h)_{доп} = (00010101_2)_{доп} = 11101010_2 + 1_2 = 11101011_2$ и $5-21 = -16_{10} = 05h - 15h = 05h + (15h)_{доп} = 00000101_2 + 11101011_2 = 11110000_2 = F0h$. Тогда $00h - F0h = 00h + (F0h)_{доп} = 00000000_2 + 00010000_2 = 00010000_2 = 10h = 16_{10}$.

Заметим, что команду `sublw k` можно использовать для сравнения чисел k и (W): при выполнении неравенства k >= (W) всегда устанавливается флаг переноса C статусного регистра, в противном случае – нет.

Приложение к лабораторной работе содержит: 1) описание команд МК PIC16F84A с примерами их использования, 2) листинг заголовочного файла P16F84A.INC.

ЗАДАНИЯ ДЛЯ ВЫПОЛНЕНИЯ

1. Напишите программу на языке ассемблера, которая выводит в регистровый файл МК таблицу ASCII – кодов строчных букв английского алфавита.
2. Напишите программу на языке ассемблера, которая шифрует кодом сдвига (код Цезаря[1]) строку символов 'ABCD' и записывает криптограмму в регистровый файл МК. Величина сдвига L = 3.
3. Десятичное число -1 представьте в дополнительном коде, пользуясь только командами загрузки, пересылки, логических операций и инкрементирования. Проведите необходимые эксперименты и изучите, как в дополнительном коде представляются целые отрицательные числа от -127 до 0.

ПРИЛОЖЕНИЕ 1

Описание команд микроконтроллера PIC16F84A

Команды очистки регистров

Команда CLRF

Синтаксис:

[label] clrf f ; 00h→(f)

[1] В криптографическом шифре Цезаря каждый символ криптограммы заменяется на другой, который получается путем сдвига номеров букв в открытом алфавите на величину L. Например, при L = 3 (сдвиг на 3 символа вправо), открытой букве 'A' соответствует буква 'D', букве 'B' - буква 'E' криптограммы и т.д.

Очищает регистр f и устанавливает флаг Z регистра STATUS.

Пример:

```
x equ 0x0C
..........
clrf  x
```

Команда CLRW

Синтаксис:

[label] clrw ; 00h→(W)

Очищает регистр W и устанавливает флаг Z.

Пример:

```
clrw
```

Команды загрузки и пересылки

Команда MOVLW

Синтаксис:

[label] movlw k ; k→(W)

Загружает 8-битный литерал k в регистр W, на флаги влияния не оказывает.

Пример:

```
movlw 0xf9          movlw .255
movlw 0A5h          movlw 'a'
```

Команда MOVF

Синтаксис:

[label] movf f, d ; (f)→(destination)

Пересылает содержимое f в место назначения d: d=0 – место назначения W, d=1 – регистр f. Команда влияет на флаг Z. Значение d=1 используется для тестирования содержимого регистра f на нуль, т.к. команда влияет на флаг Z.

Команда MOVWF

Синтаксис:

[label] movwf f ; (W)→(f)

Пересылает содержимое W в регистр f, на флаги влияния не оказывает.

Пример:

```
x equ 0x0C
..........
movlw 0x00
movwf x
movf x      ;тестирование x на 0
movlw 0x11
movwf x
movf x      ;тестирование x на 0
```

Команды арифметических операций

Команда INCF

Синтаксис:

[label] incf f, d ; (f)+1→(destination)

Инкрементирует содержимое регистра f и пересылает результат в место назначения d: d=0 – аккумулятор W, d=1 – регистр f. Команда влияет на флаг Z.

Примечание. Команда incf W,0 не работает.

Пример:

```
x equ 0x0C
..........
incf x,0    ;(x)+1->W
```

Команда DECF

Синтаксис:

[label] decf f, d ; (f) -1→(destination)

Декрементирует содержимое регистра f и пересылает результат в место назначения d: d=0 – аккумулятор W, d=1 – регистр f. Команда влияет на флаг Z.

Примечание. Команда decf W,0 не работает.

Пример:

```
x equ 0x0C
..........
decf x      ;(x)-1->x
```

Команда ADDLW

Синтаксис:

[label] addlw k ; (W)+k→(W)

Прибавляет к содержимому регистра W значение литерала k. Команда влияет на флаги C, DC, Z регистра STATUS.

Пример:

```
addlw 0x0A
```

Команда ADDWF

Синтаксис:

[label] addwf f, d ; (W)+(f)→(destination)

Прибавляет к содержимому W содержимое регистра f и помещает результат в место назначения: d=0 – W, d=1 – f. Команда влияет на флаги C, DC, Z.

Пример:

```
    addwf x,1
    addwf x   ; 1 после запятой используется по
умолчанию
    addwf x,0
```

Команда SUBLW

Синтаксис:

[label] sublw k ; k - (W) →(W)

Вычитает из литерала k содержимое W, результат операции помещает в регистр W. Команда влияет на флаги C, DC, Z.

Пример:

```
sublw 0x0A
```

Команда SUBWF

Синтаксис:

[label] subwf f, d ; (f) - (W)→(destination)

Вычитает из содержимого f содержимое W и помещает результат в место назначения: d=0 – W, d=1 – f. Команда влияет на флаги C, DC, Z.

Пример:

```
subwf x          subwf x,0
```

<u>Команды логических операци</u>

Команда COMF

Синтаксис:

[label] comf f, d ; ~(f) →(destination)

Инвертирует побитово содержимое W и помещает результат в место назначения: d=0 – W, d=1 – f. Команда влияет на флаг Z.

Пример:

```
comf x,0
```

Команда ANDLW

Синтаксис:

[label] andlw k ; (W) .AND. (k)→(W)

Выполняет операцию побитового логического И над содержимым W и 8-битным литералом k, помещает результат в регистр W. Команда влияет на флаг Z.

Пример:

```
andlw 0x0A
```

Команда ANDWF

Синтаксис:

[label] andwf f, d ; (W) .AND. (f)→(destination)

Выполняет операцию побитового логического И над содержимым f и W, помещает результат в место назначения: d=0 – W, d=1 – f. Команда влияет на флаг Z.

Примеры:

```
andwf x          andwf x,0
```

Команда IORLW

Синтаксис:

[label] iorlw k ; (W) .OR. (k)→(W)

Выполняет операцию побитового логического ИЛИ над содержимым W и 8-битным литералом k, помещает результат в регистр W. Команда влияет на флаг Z.

Команда IORWF

Синтаксис:

[label] iorwf f, d ; (W) .OR. (f)→(destination)

Выполняет операцию побитового логического ИЛИ над содержимым f и W, помещает результат в место назначения: d=0 – W, d=1 – f. Команда влияет на флаг Z.

Команда XORLW

Синтаксис:

[label] xorlw k ; (W) .XOR. (k)→(W)

Выполняет операцию побитового исключающего ИЛИ над содержимым W и 8-битным литералом k, помещает результат в регистр W. Команда влияет на флаг Z.

Команда XORWF

Синтаксис:

[label] xorwf f, d ; (W) .XOR. (f)→(destination)

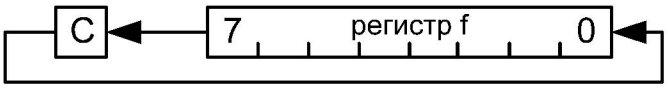
Выполняет операцию побитового исключающего ИЛИ над содержимым f и W, помещает результат в место назначения: d=0 – W, d=1 – f. Команда влияет на флаг Z.

Команда RLF

Синтаксис:

[label] rlf f, d

Побитово сдвигает содержимое регистра f влево через флаг переноса C, помещает результат в место назначения: d=0 – W, d=1 – f. Команда влияет на флаг C.

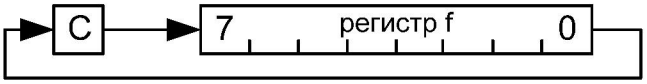


Команда RRF

Синтаксис:

[label] rrf f, d

Побитово сдвигает содержимое регистра f вправо через флаг переноса C, помещает результат в место назначения: d=0 – W, d=1 – f. Команда влияет на флаг C.



Команда SWAPF

Синтаксис:

[label] swapf f, d ; (f<3:0>)→(destination<7:4>)
; (f<7:4>)→(destination<3:0>)

Меняет местами младшую и старшую тетрады регистра f, помещает результат в место назначения: d=0 – W, d=1 – f. На флаги регистра STATUS влияния не оказывает.

Условные команды

Команда BTFSC

Синтаксис:

[label] btfsc f, b ; пропуск команды, если (f<b>)=0

Проверяет состояние бита b в регистре f. Если результат проверки равен 0, то вместо следующей команды выполняет команду NOP. На флаги регистра STATUS влияния не оказывает.

Команда используется для организации циклов и ветвлений в программе.

Команда BTFSS

Синтаксис:

[label] btfss f, b ; пропуск команды, если (f<b>)=1

Проверяет состояние бита b в регистре f. Если результат проверки равен 1, то вместо следующей команды выполняет команду NOP. На флаги регистра STATUS влияния не оказывает.

Команда используется для организации циклов и ветвлений в программе.

Команда DECFSZ

Синтаксис:

[label] decfsz f, d ; (f) -1→(destination)

Декрементирует содержимое регистра f и пересылает результат в место назначения d (d=0 – аккумулятор W, d=1 – регистр f). Проверяет состояние бита Z в регистре STATUS. Если Z=1, то выполняет следующую команду. Если Z=0, то вместо следующей команды выполняет команду NOP в течение двух тактов.

Команда используется для организации циклов и ветвлений в программе.

Команда INCFSZ

Синтаксис:

[label] incfsz f, d ; (f)+1→(destination)

Инкрементирует содержимое регистра f и пересылает результат в место назначения d: d=0 – аккумулятор W, d=1 – регистр f. Проверяет состояние бита Z в регистре STATUS. Если Z=1, то выполняет следующую команду. Если Z=0, то вместо следующей команды выполняет команду NOP в течение двух тактов.

Команда используется для организации циклов и ветвлений в программе.

Команды управления

Команда CLRWDT

Синтаксис:

[label] clrwdt ; 00h→WDT, 0→WDT prescaler, 1→~TO,
; 1→~PD

Инициализирует сторожевой таймер. Команда влияет на флаги ~TO, ~PD.

Команда BCF

Синтаксис:

[label] bcf f, b ; 0→(f<b>)

Сбрасывает в 0 бит b в регистре f. На флаги регистра STATUS влияния не оказывает.

Команда BSF

Синтаксис:

[label] bsf f, b ; 1→(f<b>)

Устанавливает в 1 бит b в регистре f. На флаги регистра STATUS влияния не оказывает.

Команда CALL

Синтаксис:

[label] call k ; (PC)+1→TOS, k→PC<10:0>,
; (PCLATCH<4:3>)→PC<12:11>

Вызов подпрограммы, выполняется за 2 цикла.

Команда GOTO

Синтаксис:

[label] goto k ; k→PC<10:0>
; (PCLATCH<4:3>)→PC<12:11>

Команда безусловного перехода, выполняется за 2 цикла.

Команда NOP

Синтаксис:

[label] nop

Холостая команда, ничего не делает, на флаги не влияет.

Команда RETFIE

Синтаксис:

[label] retfie ; 1→GIE, TOS→PC

Возврат из подпрограммы обработки прерываний. На статусные флаги влияния не оказывает.

Команда RETLW

Синтаксис:

[label] retlw k ; k→(W), TOS→PC

Возврат из подпрограммы с загруженным в регистр W литералом k. Команда выполняется за 2 такта.

Команда RETURN

Синтаксис:

[label] return ; TOS→PC

Возврат из подпрограммы, команда выполняется за 2 такта. На флаги статусного регистра влияния не оказывает.

Команда SLEEP

Синтаксис:

[label] sleep ; 00h→WDT, 0→WDT prescaler, 1→~TO, ; 0→~PD

Переводит процессор в состояние SLEEP с остановленным генератором. Команда влияет на флаги ~TO, ~PD.

ПРИЛОЖЕНИЕ 2

ЗАГОЛОВОЧНЫЙ ФАЙЛ P16F84A.INC

```
    LIST
; P16F84A.INC  Header File, Version 2.00  Microchip Technology, Inc.
    NOLIST

; Заголовочный файл определяет конфигурации, регистры и другие
; полезные биты информации для МК PIC16F84.
; Заметьте, что процессор должен быть выбран перед тем, как этот файл
; будет включен. Процессор может быть выбран следующими способами:
;       1. С помощью ключа командной строки:
;              C:\ MPASM MYFILE.ASM /PIC16F84A
;       2. С помощью директивы LIST в исходном файле
;              LIST   P=PIC16F84A
;       3. С помощью опции Processor Type в полноэкранном интерфейсе
;          MPASM
;==========================================================================
   IFNDEF __16F84A
    MESSG "Processor-header file mismatch.  Verify selected processor."
   ENDIF
;==========================================================================
;----- Register Definitions-----------------
W                           EQU     H'0000'
F                           EQU     H'0001'
;----- Register Files-----------------------
INDF                        EQU     H'0000'
TMR0                        EQU     H'0001'
PCL                         EQU     H'0002'
STATUS                      EQU     H'0003'
FSR                         EQU     H'0004'
PORTA                       EQU     H'0005'
PORTB                       EQU     H'0006'
EEDATA                      EQU     H'0008'
EEADR                       EQU     H'0009'
PCLATH                      EQU     H'000A'
INTCON                      EQU     H'000B'
```

```
OPTION_REG                      EQU     H'0081'
TRISA                           EQU     H'0085'
TRISB                           EQU     H'0086'
EECON1                          EQU     H'0088'
EECON2                          EQU     H'0089'

;----- STATUS Bits -------------------------
IRP                             EQU     H'0007'
RP1                             EQU     H'0006'
RP0                             EQU     H'0005'
NOT_TO                          EQU     H'0004'
NOT_PD                          EQU     H'0003'
Z                               EQU     H'0002'
DC                              EQU     H'0001'
C                               EQU     H'0000'

;----- INTCON Bits -------------------------
GIE                             EQU     H'0007'
EEIE                            EQU     H'0006'
T0IE                            EQU     H'0005'
INTE                            EQU     H'0004'
RBIE                            EQU     H'0003'
T0IF                            EQU     H'0002'
INTF                            EQU     H'0001'
RBIF                            EQU     H'0000'

;----- OPTION_REG Bits ---------------------
NOT_RBPU                        EQU     H'0007'
INTEDG                          EQU     H'0006'
T0CS                            EQU     H'0005'
T0SE                            EQU     H'0004'
PSA                             EQU     H'0003'
PS2                             EQU     H'0002'
PS1                             EQU     H'0001'
PS0                             EQU     H'0000'

;----- EECON1 Bits -------------------------
EEIF                            EQU     H'0004'
WRERR                           EQU     H'0003'
WREN                            EQU     H'0002'
WR                              EQU     H'0001'
RD                              EQU     H'0000'
;=========================================
;----- RAM Definition----------------------
        __MAXRAM H'CF'
        __BADRAM H'07', H'50'-H'7F', H'87'
;=========================================
;----- Configuration Bits------------------

_CP_ON                          EQU     H'000F'
_CP_OFF                         EQU     H'3FFF'
_PWRTE_ON                       EQU     H'3FF7'
_PWRTE_OFF                      EQU     H'3FFF'
_WDT_ON                         EQU     H'3FFF'
_WDT_OFF                        EQU     H'3FFB'
_LP_OSC                         EQU     H'3FFC'
```

```
_XT_OSC                     EQU     H'3FFD'
_HS_OSC                     EQU     H'3FFE'
_RC_OSC                     EQU     H'3FFF'
     LIST
```

Лабораторная работа № 2. Программирование микроконтроллеров PIC16F8xx на языке ассемблера MPASM

Цель работы: Изучить структуру управляющей программы на языке ассемблера MPASM, научиться реализовывать практически базовые конструкции структурированных языков программирования: последовательности, ветвления, циклы и вызовы подпрограмм.

СТРУКТУРА УПРАВЛЯЮЩЕЙ ПРОГРАММЫ НА ЯЗЫКЕ АССЕМБЛЕРА

Текст управляющей программы должен быть разбит на секции, чтобы его было легко читать, сопровождать, документировать и модифицировать. Применительно к PIC-микроконтроллерам структура текста программы может выглядеть так:

1. ***Блок определений***
   - секция заголовка (с информацией о назначении программы, имени автора, дате создания и внесенных изменений, применяемом контроллере, тактовой частоте и т.д.);
   - секция подключаемых файлов;
   - секция конфигурации (содержит определение битов конфигурации и IDLOC);
   - секция определения констант;
   - секция определения EEPROM-данных;
   - секция определения макросов;
   - секция объявления переменных.
2. ***Блок кода***
   - вектор сброса (обычно содержит только инструкцию безусловного перехода на код инициализации);
   - обработчик прерывания;
   - код инициализации;
   - основной цикл программы;
   - подпрограммы;
   - end

Для оформления секций внутри одного файла исходного текста есть свои правила:

- каждая секция должна содержать только те описания, которые ей соответствуют (т.е. не нужно в секции кода определять константы).
- секции должны быть едиными, а не разделенными на несколько фрагментов, разбросанных по всему тексту программы.
- каждой секции должен предшествовать хорошо заметный блок комментария, содержащий название секции.

Упрощенная структура управляющей программы для МК PIC16F84A на языке ассемблера MPASM имеет вид:

```
; ======= Блок определений =======
list       p=PIC16F84A
include    <P16F84A.INC>

; =========== Блок кода ============
; Начало исполняемого кода
     org   0x00       ; Адрес вектора сброса
     goto  start

     org   0x04       ; Адрес вектора прерываний
     ; Сюда вставляйте код обработчика прерываний
     retfie           ; Возврат из прерывания

; ------------- Начало управляющей программы -------------
start
     ; Сюда вставляйте код управляющей программы
     goto  $          ; Переход на текущую строку (бесконечный цикл)
; -------------- Конец управляющей программы -------------
subrt1
     ; Сюда вставляйте код подпрограммы 1
     return
subrt2
     ; Сюда вставляйте код подпрограммы 2
     return
     ; ..................................
end
```

где

`start` – метка, помечающая начало исполняемого кода программы;

`org` – директива компилятора, определяет адрес в памяти программ, с которого компилятор помещает код, следующий за директивой `org`;

`end` – директива компилятора, помечающая конец текста программы;

`list` - директива компилятора, задающая тип используемого МК;

`include` - директива компилятора, перед компиляцией добавляет в начало программы текст файла включений `P16F84A.INC`, содержащий определения констант и регистров для МК PIC16F84A.

Часто разработчики, пытаясь сделать свои программы самодостаточными, вручную определяют все регистры для данного МК. Это хорошо только при объяснении новичку взаимосвязи между текстом программы и адресами регистров конкретного МК. Но писать программы в таком стиле – дурной тон, приводящий к усложнению поддержки и модификации программы. Основной причиной некорректности такого подхода является то, что различные МК фирмы Microchip имеют регистры с одними и теми же именами, но расположенными по разным адресам. Типичный пример – расположение регистров управления EEPROM для PIC16F628 (адреса 9Ah-9Dh) и для PIC16F819 (10Ch-10Fh, 18Ch-18Dh). При замене контроллера (неважно, по каким причинам) можно случайно

пропустить такие различия, что приведет к неправильному поведению программы.

Поэтому не стоит пренебрегать файлами определений, поставляемых в комплекте с MPASM. Они не только избавят вас от рутины переопределения всех регистров и их служебных битов вручную, но и позволят легко и быстро перенастроить исходный текст под другой МК.

Неправильно:

```
INDF        equ   00h
TMR0        equ   01h
PCL         equ   02h
STATUS equ   03h
....................
```

Правильно:

```
include <P16F84A.INC>
```

ДИРЕКТИВЫ КОМПИЛЯТОРА MPASM

Перечислим другие полезные директивы компилятора, которые можно использовать в управляющих программах на языке ассемблера MPASM микроконтроллеров PIC.

Директива `equ`

Синтаксис:

```
<label> equ <expr>
```

Директива `equ` присваивает значение `<expr>` константе `<label>`.

Пример:

```
four equ 4 ;присваивает значение 4 константе four
```

Директива `banksel`

Синтаксис:

```
banksel <reg>
```

Директива `banksel` используется при генерации объектного файла, дает компилятору команду сгенерировать код выборки банка памяти данных для прямого обращения к регистру `<reg>`.

Пример:

```
x   equ        0x0C
    ................
    banksel    TRISB      ; выбор банка 1
    movf       TRISB,W    ; (TRISB)->(W)
    movlw      0x0F       ; <флаги настройки порта B> ->(W)
    movwf      TRISB      ; (W)->(TRISB)
    clrw                  ; очистка W
    movf       TRISB,W    ; (TRISB)->(W)
    banksel    x          ; выбор банка 0
    movwf      x          ; (W)->(x)
```

Директива `cblock`

Синтаксис:

```
cblock [<expr>]
   <label>[:<increment>][,<label>[:<increment>]]
```

```
endc
```

где

<label> - имена констант;

<expr> - указывает стартовый адрес для первой константы. Если адрес не указан, то используется заключительное значение предыдущего cblock. Если первый cblock не имеет никакого значения <expr>, то размещение начинается с нулевого адреса;

<increment> - указывает приращения адреса для текущей именованной константы. Именованные константы в одной строке разделяются запятыми.

Директива cblock используется для размещения именованных констант в памяти данных. Список именованных констант заканчивается директивой endc.

Пример:

```
cblock      0x0C        ; блок данных по адресу 0x0C
            x,y:3,z     ; переменной y выделено 3 байта
endc
....................
      ; очистка ячеек памяти
      clrf       x
      clrf       y
      clrf       y+1
      clrf       y+2
      clrw       z

      movlw      0x11
      movwf      x          ; запись в ячейку x
      movlw      0x22
      movwf      y          ; запись в ячейку y
      movlw      0x33
      movwf      y+1        ; запись в ячейку y+1
      movlw      0x44
      movwf      y+2        ; запись в ячейку y+2
      movlw      0x55
      movwf      z          ; запись в ячейку z
```

Директива constant

Синтаксис:

```
constant <label>[=<expr>,...,<label>[=<expr>]]
```

Директива constant – создает символьные константы для использования в выражениях MPASM.

Пример:

```
constant TEN=.10
................
Movlw  TEN+5
```

Директива radix

Синтаксис:

```
radix <default_radix>
```

Директива указывает системы счисления по умолчанию: hex, dec, oct. Первоначально установлена шестнадцатеричная система счисления (hex).

Пример:

```
list  p=PIC16F84A, r=dec
```

или

```
list  p=PIC16F84A
include <P16F84A.INC>
radix = dec
```

ПРЯМАЯ И КОСВЕННАЯ АДРЕСАЦИЯ РЕГИСТРОВ ДАННЫХ

В МК PICF84 обращение к регистрам общего назначения может быть выполнено прямой или косвенной адресацией (через регистры FSR и INDF).

Пример прямой и косвенной адресации:

```
xequ        0x0C
 ...............
 movlw      0x11
 movwf      x          ; прямая адресация к регистру x
 ................
 movlw      x          ; загрузка константы x в регистр W
 movwf      FSR        ; загрузка регистра косвенной адресации
 movlw      0x55       ; данные для записи
 movwf      INDF       ; косвенная запись в регистр x
 incf       FSR        ; инкремент косвенного адреса
 movlw      0x66       ; данные для записи
 movwf      INDF       ; косвенная запись в регистр x+1
```

РЕГИСТР СОСТОЯНИЯ МИКРОКОНТРОЛЛЕРА STATUS

Для реализации ветвлений и циклов в программе на языке ассемблера обычно анализируют флаги регистра специального назначения STATUS, имеющего следующий формат:

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

где

bit7 IRP - бит выбора банка памяти, применяемый при косвенной адресации. Этот бит не применяется в МК PIC16F8xx и всегда должен быть сброшенным.

Назначение бита:

bit7=0 - bank 0, 1 (00h-FFh)
bit7=1 - bank 2, 3 (100h-IFFh)

bit6-5 RP1, RP0 - биты выбора банка памяти, применяемые при прямой адресации.

Назначение битов:

00 - bank 0 (00h-7Fh)
01 - bank 1 (80h-FFh)
10 - bank 2 (100h-17Fh)
11 - bank 3 (130h-lFFh)

Каждый банк содержит 128 байт. В МК PIC16F8x используется только два банка (0 и 1), поэтому бит RP1 всегда должен быть сброшен, а для выбора банка используется бит RP0.

bit4 ~TO – инвертированный флаг срабатывания сторожевого таймера. Устанавливается в 1 при включении питания, а также командами CLRWDT и SLEEP. Сбрасывается в 0 по завершении выдержки сторожевого таймера.

bit3 ~PD – инвертированный флаг режима хранения данных. Устанавливается в 1 при включении питания или выполнении команды CLRWDT. Сбрасывается в 0 командой SLEEP.

bit2 Z - флаг нулевого результата. Устанавливается в 1, если результат арифметической или логической операции равен нулю. Сохраняет свое значение до следующей операции.

bit1 DC - флаг десятичного переноса. Используется командами ADDWF, ADDLW, SUBWF и SUBLW. Отслеживает перенос из четвертого в пятый разряд результата: 1 - произошел перенос при сложении; 0 - не произошел перенос при сложении.

bit0 C - флаг переноса. Используется командами ADDWF, ADDLW, SUBWF и SUBLW. Отслеживает перенос из старшего разряда в бит переноса при сложении: 1 - произошел перенос при сложении; 0 - не произошел перенос при сложении.

Вычитание в АЛУ МК выполняется посредством сложения кода первого операнда с дополнительным кодом второго операнда.

ОРГАНИЗАЦИЯ ВЕТВЛЕНИЙ В ПРОГРАММАХ ДЛЯ МК PIC16F84A

Для выполнения ветвлений в управляющих программах для МК PIC используются условные команды BTFSC, BTFSS. Поясним организацию условного ветвления на примере проверки состояния флага Z регистра STATUS.

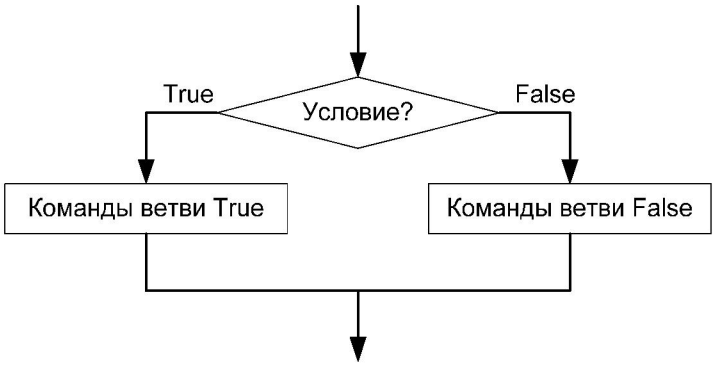


Рис.1. Программное ветвление по результату проверки условия.

Ветвления по результатам проверки условия Z=1 (результат предыдущей операции равен 0) реализуются с использованием следующего алгоритма:

```
                ..................
                <if Z=1 then Пропустить следующую команду>
                goto branch_False
branch_True
                ; команды ветви branch_True
                ..................
                goto labelI
                ; конец ветви branch_True
branch_False
```

```
                ; команды ветви branch_False
                .................
                ; конец ветви branch_False
labelI
                ; продолжение программы
                .................
```

Листинг демонстрационной программы, реализующий данный алгоритм:

```
; === Секция заголовка ===
list            p=PIC16F84
include         <P16F84.INC>

x     equ       0Ch

; === Рабочая секция ===
      org       0
      goto      start

start
      movlw     01h             ; введите тестовое значение
      movwf     x
      ; --- Ветвление в зависимости от значения x ---
      movf      x               ; воздействие на флаг Z
      btfss     STATUS,2        ; проверка x=0
      goto      br_False        ; переход, если Z=0
br_True
      ; команды ветви TRUE
      movlw     11h             ; (W)=11h - выполнены команды ветви TRUE
      goto      label_1
br_False
      ; команды ветви FALSE
      movlw     22h             ; (W)=22h - выполнены команды ветви FALSE
      ; --- конец ветвления ---
label_1
      ; --- продолжение программы ---
      goto $
end
```

ОРГАНИЗАЦИЯ ЦИКЛОВ В ПРОГРАММАХ ДЛЯ МК PIC16F84A

Для организации циклов в управляющих программах для МК PICmicro используются условные команды BTFSC, BTFSS, DECFSZ, INCFSZ.

Поясним организацию цикла в программе для МК PIC16F84A на примере проверки значения флага Z регистра STATUS.

Циклы, завершающиеся по условию Z=1 (результат предыдущей операции равен 0), реализуются с использованием следующего алгоритма:

```
                .............
                ; начало цикла
          loop_I
                ; команды тела цикла
                <if Z=1 then Пропустить следующую команду>
                goto loop_I     ; конец цикла
                .............
```

Листинг демонстрационной программы, реализующий данный алгоритм:

```
list        p=PIC16F84
include     <P16F84.INC>

iequ        0Ch        ; счетчик циклов
xequ        0Dh

org    0
goto   start

start
 movlw      .10
 movwf      i
 clrf       x
loop_1
 incf       x
 movf       x,W
 decf       i
 btfss      STATUS,2
 goto       loop_1
 ; продолжение программы
 goto $
end
```

ВЫЗОВЫ ПОДПРОГРАММ В ПРОГРАММАХ ДЛЯ МК PICmicro

Для вызова подпрограмм в управляющих программах для МК PICmicro используется команда CALL.

Листинг демонстрационной программы, вызывающей подпрограмму с именем `subrt1`:

```
; === Секция заголовка ===
list         p=PIC16F84A
include      <P16F84A.INC>

; === Рабочая секция ===
 org    0x00
 goto   start

start
 clrw
 call   subrt1
 goto   $         ; Бесконечный цикл
; Подпрограмма subrt1
subrt1
 movlw  0xFF
return
end
```

ЗАДАНИЯ ДЛЯ ВЫПОЛНЕНИЯ

1. Разработайте программу на языке ассемблера MPASM, реализующую расширенную форму оператора if:

```
If <условие1> Then
     <операторы1>
ElseIf <условие2> Then
     <операторы2>
ElseIf <условие3> Then
```

```
        <операторы3>
Else
        <операторы4>
End If
```

2. Разработайте программу на языке ассемблера MPASM, выводящую в регистровый файл номер дня недели (пн-1, вт-2, ср-3, чт-4, пт-5, сб-6, вс-7), который соответствует заданному пользователем дню недели при условии, что в месяце 31 день и первое число месяца – понедельник.
3. Разработайте программу на языке ассемблера MPASM, проверяющую, что значение переменной x, x ∈ [0,255], удовлетворяет неравенствам 20<=x<=50.
4. Открытый алфавит криптосистемы, шифрующей сообщения шифром замены Цезаря, образуют заглавные латинские буквы 'A', 'B','C', …, 'Z'. Шифралфавит криптосистемы получается путем циклического сдвига букв открытого алфавита на величину смещения L.

   Разработайте программу на языке ассемблера MPASM, вычисляющую по кодам букв открытого алфавита коды букв шифралфавита для любого смещения L=1..26, задаваемого пользователем.

Лабораторная работа №3. Программное управление портами ввода-вывода микроконтроллера *PIC16F84A*

Цель работы: Освоить базовые принципы управления портами ввода-вывода PIC-контроллеров, в среде MPLAB IDE научиться пользоваться асинхронными внешними воздействиями (стимулами) и использовать их для отладки программ работы с портами ввода-вывода.

МИКРОПРОЦЕССОРНОЕ УСТРОЙСТВО УПРАВЛЕНИЯ СВЕТОДИДОМ И ГЕНЕРАЦИИ ЗВУКА

Принципиальная схема модельного микропроцессорного устройства управления светодиодом и генерации звука приведена на рис.1.

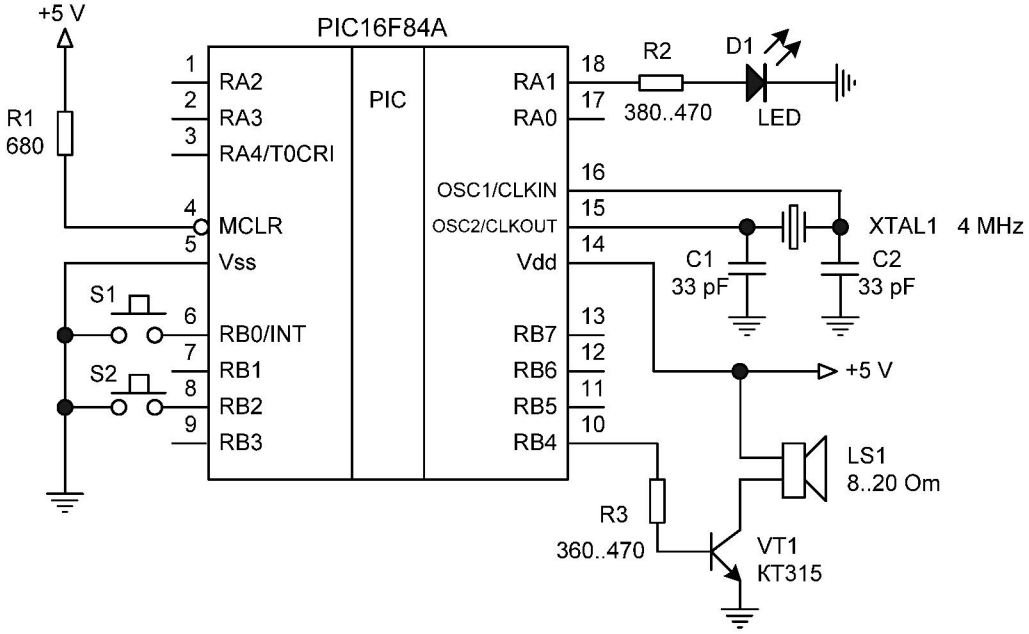


Рис.1. Принципиальная схема микропроцессорного устройства управления светодиодом и генерации звука.

Устройство состоит из двух кнопок S1 и S2, светодиода D1, генератора тактовых импульсов частотой 4 МГц с кварцевым резонатором XTAL1, динамика LS1, усилителя звуковых колебаний на базе транзистора VT1, токоограничивающих резисторов и частотозадающих конденсаторов. Управление работой устройства осуществляется с помощью микроконтроллера PIC16F84A. Заметим, что каждый вывод МК семейства PIC может непосредственно управлять подключенным к нему светодиодом без дополнительных усилителей.

Когда включается питание, МК по умолчанию устанавливает все разряды портов A и B на ввод и начинает выполнять управляющую программу с адреса 00h.

УПРАВЛЕНИЕ МИГАНИЕМ СВЕТОДИОДА

Ниже приведен базовый код программы мигания светодиода, написанной на языке ассемблера MPASM.

Базовый код программы управления светодиодом

```
; === Секция заголовка ===
list    p=PIC16F84A
include <P16F84A.INC>
__config _CP_OFF&_WDT_OFF&_PWRTE_ON&_XT_OSC

cnt1    equ     0x0C            ; счетчик циклов 1
cnt2    equ     0x0D            ; счетчик циклов 2
ledset  equ     0x0E            ; регистр светодиода

; === Рабочая секция ===
    org         0x00
    goto        start

    org         0x08
start
    clrf        PORTA           ; очистить порт A
    clrf        PORTB           ; очистить порт B
    bsf         STATUS,RP0      ; выбрать банк 1
    movlw       b'11111101'
    movwf       TRISA           ; установить линию RA1 порта A на выход
    movlw       b'00000101'
    movwf       TRISB       ; установить линии RP7:PR3 и RP1 порта B на выход
    bcf         OPTION_REG,7    ; включить подтягивающие резисторы
    bcf         STATUS,RP0      ; выбрать банк 0
    clrf        ledset          ; очистить регистр светодиода
    ; --- Бесконечный цикл включения/выключения светодиода ---
loop
    movlw       b'00000010'
    xorwf       ledset
    movf        ledset,W
    movwf       PORTA           ; включить/выключить светодиод
    ; --- Программная задержка ---
    ;movlw      0xFF
    movlw       0x03
    movwf       cnt1
cycle1
    ;movlw      0xFF
    movlw       0x03
    movwf       cnt2
cycle2
```

```
decfsz     cnt2
goto       cycle2
decfsz     cnt1
goto       cycle1
; --- Конец программной задержки ---
goto       loop            ; бесконечный цикл
end
```

где строка `__config _CP_OFF&_WDT_OFF&_PWRTE_ON&_XT_OSC` – директива ассемблера, под руководством которой в выходной файл заносится информация о слове состояния процессора. Она означает:

- _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).

Как работает эта программа? Указав ассемблеру тип процессора и соответствующий ему файл включений P16F84A.INC, мы описываем, в каких ячейках ОЗУ (регистрах общего назначения) будут храниться значения переменных программы, а затем настраиваем порты ввода-вывода.

Для этого сначала обнуляются значения в выходных защелках портов. В данной программе мы могли бы этого и не делать, но правильный стиль программирования PIC-микроконтроллеров требует, чтобы значения в выходных защелках были явно определены перед тем, как некоторые линии будут настроены на вывод.

Затем, установив в 1 бит RP0 регистра STATUS, получаем доступ к регистровому банку 1 и, обращаясь к регистрам состояния портов TRISA и TRISB, настраиваем часть линий на ввод, а часть на вывод согласно схеме микропроцессорного устройства (рис.1). После чего устанавливаем бит RP0 регистра STATUS обратно в 0, тем самым возвращаясь к банку памяти 0, и обнуляем переменную ledset.

Далее (метка loop) при помощи операции XOR 'Исключающее ИЛИ' инвертируем бит <1> переменной ledset, сохраняем полученное значение обратно в ledset (команда xorwf ledset) и, скопировав результат в аккумулятор (команда movf ledset,w), выводим значение ledset в выходную защелку порта A. В результате на линии RA1 микроконтроллера появится уровень напряжения, соответствующий текущему значению бита <1> переменной ledset.

Затем инициализируем переменную внешнего цикла cnt1. Внутрь этого цикла вложен еще один цикл с переменной cnt2. Назначение этих циклов - сформировать программную задержку, в течение которой на выводе RA1 порта A удерживается установленное значение ledset. Когда задержка исчерпана, в порт выводится инверсное значение, и цикл повторяется.

Когда вы наберете текст программы, сохраните его под произвольным именем в файле с расширением .asm, создайте в MPLAB новый проект и добавьте в него этот файл. Затем скомпилируйте программу (F10), выполните в команду

меню **Debugger/Animate** и наблюдайте, как изменяется состояние линии RA1 порта A во время выполнения программы.

Если вы введете во внутренний цикл три-четыре холостых команды NOP, то задержка заметно увеличится, соответственно уменьшится частота мигания светодиода.

Попробуйте также увеличить частоту мигания, уменьшая начальные значения переменных cnt1 и cnt2. Но учтите, что если установить частоту мигания больше 20...25 Гц, то из-за инерционности зрения будет казаться, что светодиод горит непрерывно, хотя программа работает правильно.

УПРАВЛЕНИЕ ГЕНЕРАЦИЕЙ ЗВУКА

Ниже приведен базовый код программы генерации звука, написанной на языке ассемблера MPASM. Поскольку для генерации импульсов звуковой частоты нужна относительно небольшая временная задержка, программа оказалась еще проще, чем для мигающего светодиода. Понадобилось сделать лишь небольшие изменения созданной ранее программы управления светодиодом.

Базовый код программы генерации звука

```
; === Секция заголовка ===
list    p=PIC16F84A
include <P16F84A.INC>
__config _CP_OFF&_WDT_OFF&_PWRTE_ON&_XT_OSC

cnt         equ     0x0D        ; счетчик циклов
ledset      equ     0x0E        ; регистр звука

; === Рабочая секция ===

    org     0x00
    goto    start

    org     0x08
start
    clrf    PORTA               ; очистить порт A
    clrf    PORTB               ; очистить порт B
    bsf     STATUS,RP0          ; выбрать банк 1
    movlw   b'11111101'
    movwf   TRISA               ; установить линию RA1 порта A на выход
    movlw   b'00000101'
    movwf   TRISB          ; установить линии RP7:PR3 и RP1 порта B на выход
    bcf     OPTION_REG,7        ; включить подтягивающие резисторы
    bcf     STATUS,RP0          ; выбрать банк 0
    clrf    ledset              ; очистить регистр звука
    ; --- Бесконечный цикл генерации звуковой частоты ---
loop
    movlw     b'00010000'
    xorwf     ledset
    movf      ledset,W
    movwf     PORTB             ; инвертировать бит RB4 порта звука
    ; --- Программная задержка ---
    ;movlw    0xA4
    movlw     0x0A              ; отладочное значение величины задержки
    movwf     cnt
cycle
    decfsz    cnt
```

```
goto        cycle
; --- Конец программной задержки ---
goto        loop            ; бесконечный цикл
end
```

В этой программе мы убрали одну переменную цикла, и оставили только один цикл программной задержки для формирования импульсов (меандра). Кроме того, изменено значение константы, при помощи которой инвертируется бит в ledset, таким образом, чтобы инвертировался бит <4>, а не бит <1>. Значение ledset теперь выводится на линию RB4 порта B. Начальное значение переменной цикла cnt подобрано так, чтобы при тактовой частоте 4МГц генерировались импульсы с частотой 1000 Гц. Изменяя значение этой переменной, вы можете изменять частоту генерации звука.

ОБРАБОТКА НАЖАТИЯ КНОПКИ

Теперь несколько усложним задачу. Построим программу так, чтобы в дежурном режиме мигал светодиод, а при нажатии на кнопку S2, подключенную к выводу RB2 порта B, программа генерировала звуковой сигнал. Для этого во время генерации импульсов низкой частоты, управляющей миганиями светодиода, будем периодически опрашивать состояние кнопки S2, и, если она нажата, вызывать подпрограмму генерации звука. Таким образом, мы получаем комбинацию двух ранее написанных программ, из которых одна является главной, а другая вызывается как подпрограмма.

Базовый код программы обработки нажатия кнопки

```
; === Секция заголовка ===
list  p=PIC16F84A
include <P16F84A.INC>
__config _CP_OFF&_WDT_OFF&_PWRTE_ON&_XT_OSC

cnt1        equ   0x0C              ; счетчик циклов 1
cnt2        equ   0x0D              ; счетчик циклов 2
cnt3        equ   0x0E              ; счетчик циклов 3
ledset      equ   0x0F              ; значение для вывода через порт

; === Рабочая секция ===

      org         0x00
      goto        start

      org         0x08
start

      clrf        PORTA             ; очистить порт A
      bsf         STATUS,RP0        ; выбрать банк 1
      movlw       b'11111011'
      movwf       TRISB             ; установить линию RB2 порта B на выход
      bcf         STATUS,RP0        ; выбрать банк 0
      clrf        PORTB             ; очистить порт B
      ;movlw      b'00000100'
      ;movwf      PORTB             ; установить бит RB2 порта B

      bsf         STATUS,RP0        ; выбрать банк 1
      movlw       b'11111101'
      movwf       TRISA             ; установить линию RA1 порта A на выход
      movlw       b'00000101'
```

```
      movwf       TRISB       ; установить линии RB2,RB0 порта B на вход
      bcf         OPTION_REG,7      ; включить подтягивающие резисторы
      bcf         STATUS,RP0        ; выбрать банк 0
      clrf        ledset            ; очистить порт светодиода
      ; --- Бесконечный цикл включения/выключения светодиода ---
loop
      movlw       b'00000010'
      xorwf       ledset
      movf        ledset,W
      movwf       PORTA             ; включить/выключить светодиод
      ; --- Программная задержка ---
      ;movlw      0xFF
      Movlw       0x02
      movwf       cnt1
cycle1
      ;movlw      0xF0
      Movlw       0x03
      movwf       cnt2
cycle2
      btfss       PORTB,2     ; проверяем нажатие кнопки S2
      call        beep        ; если нажата, вызываем процедуру CALL
      decfsz      cnt2        ; иначе продолжаем выполнение цикла
      goto        cycle2
      decfsz      cnt1
      goto        cycle1
      ; --- Конец программной задержки ---
      goto        loop        ; бесконечный цикл
      ; --- Подпрограмма beep ---
beep
      ;movlw            0xA0
      movlw       0x02
      movwf       cnt3        ; инициализируем переменную цикла
loop_b
      movlw       b'00010000'
      xorwf       ledset
      movf        ledset,W
      movwf       PORTB       ; инвертируем уровень на выходе RB4
      decfsz      cnt3
      goto        loop_b
      return
      end
```

Теперь, если нажать кнопку S2, светодиод будет мигать очень редко, и из динамика раздастся звук с частотой примерно 1000 Гц. После отпускания кнопки светодиод начнет мигать с обычной частотой, а звук прекратится.

К сожалению, при отладке программы в MPLAB SIM у нас нет возможности нажать кнопку S2, поэтому нажатие кнопки приходится имитировать. Приведенная выше программа настроена таким образом, будто кнопка S2 всегда нажата. Убедитесь в этом, создав в MPLAB проект, откомпилировав и выполнив команду **Debugger/Animate**.

Чтобы перенастроить программу таким образом, будто кнопка S2 всегда отпущена, раскомментируйте строки 25 и 26 программы (в листинге программы они выделены жирным шрифтом) и снова выполните команду **Debugger/Animate**.

Разумеется, данная программа написана не оптимально с точки зрения конечного результата, т.к. цикл управления светодиодом продолжает выполняться, несмотря на нажатие кнопки, звук прерывается легкими щелчками и

светодиод хоть и редко, но переключается. Эти проблемы легко устранить, но тогда программа не будет настолько простой и наглядной.

ВНЕШНИЕ АСИНХРОННЫЕ ВОЗДЕЙСТВИЯ (АСИНХРОННЫЕ СТИМУЛЫ)

Функции внешнего воздействия (стимулы) MPLAB предназначены для симуляции внешних событий и принудительного управления содержимым регистров, с их помощью можно имитировать появление на выводах МК сигналов высокого или низкого уровня и заносить значения непосредственно в регистры. Существует четыре режима внешних событий:

- асинхронное воздействие - при помощи интерактивного диалога, управляющего внешними уровнями на входах контроллера;
- файл внешних воздействий на выводы - текстовый файл, описывающий внешние сигналы на входах контроллера;
- файл внешних воздействий на регистры - текстовый файл, содержащий набор 8-битных значений для записи непосредственно в регистры;
- частотное воздействие - программируемый источник регулярных внешних импульсов.

Асинхронное внешнее воздействие осуществляется при помощи специального диалогового окна с настраиваемыми кнопками. Откройте диалоговое окно *Stimulus*, выполнив команду меню **Debugger/Stimulus/New Workbook** и перейдите на вкладку *Asynch.* Пользователю предоставлено 13 кнопок, параметры которых можно настроить индивидуально (рис.2).

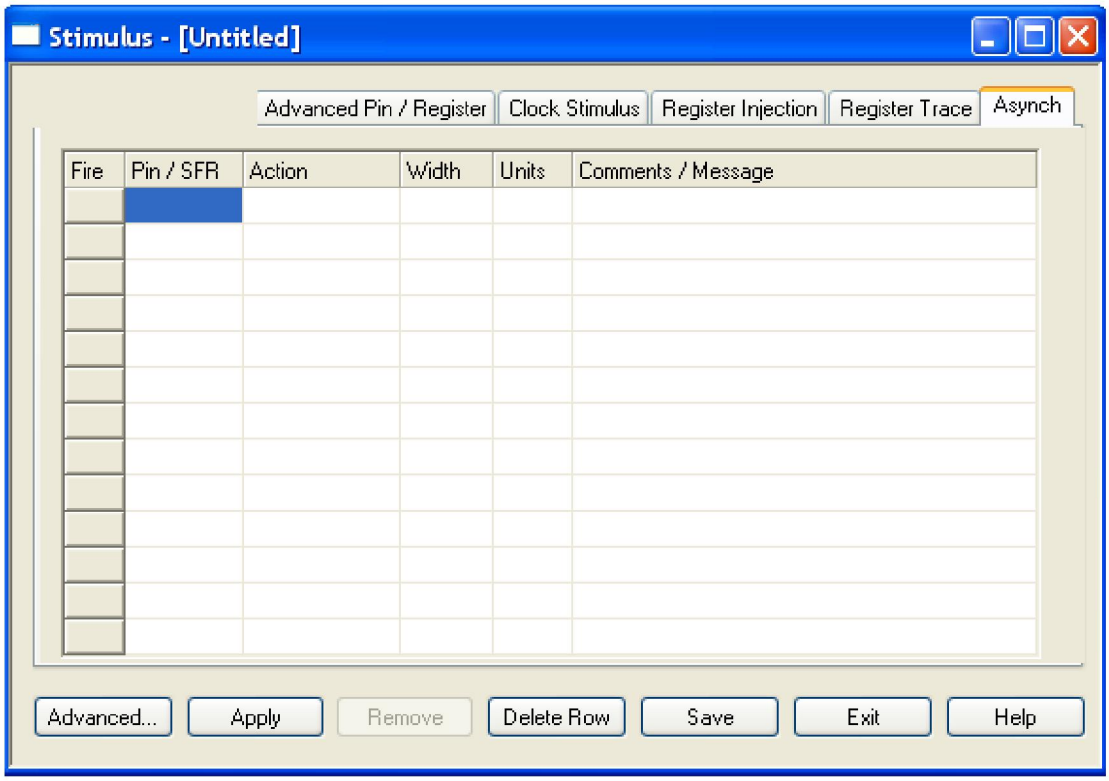


Рис.2. Диалоговое окно создания асинхронных стимулов.

Щелкните кнопкой мыши, например, на выделенном поле *Pin/SFR* первой строки и в появившемся списке доступных для внешнего воздействия выводов используемого МК выберите значение RB2.

Затем щелкните мышью на поле *Action* и в появившемся списке выберите, например, значение *Toggle* (Инвертировать). Вместе с *Toggle* доступны следующие формы внешнего асинхронного воздействия:

- *Set High* - устанавливает на указанном входе постоянный высокий уровень;
- *Set Low* - устанавливает на указанном входе постоянный низкий уровень;
- *Toggle* - изменяет уровень входного сигнала на противоположный и держит его;
- *Pulse High* - кратковременный импульс высокого уровня;
- *Pulse Low* - кратковременный импульс низкого уровня.

В поле *Comments/Message* первой строки добавьте поясняющий комментарий *Стимул входа RB2 порта B*. В результате этих действий будет создан первый асинхронный стимул (рис.3).

Аналогичным образом можно создать и другие асинхронные стимулы для данного и других входов МК. Отметим, что одному и тому же входу можно поставить в соответствие несколько стимулов и смоделировать самые различные внешние воздействия на него.

Закончив создание стимулов, скомпилируйте и запустите программу на исполнение в режиме *Debugger/Animate*. Нажимая мышью кнопки *Fire* > созданных асинхронных стимулов (рис.3), в соответствующих окнах отладки можно наблюдать изменение уровней на входах портов и реакцию программы на эти изменения.

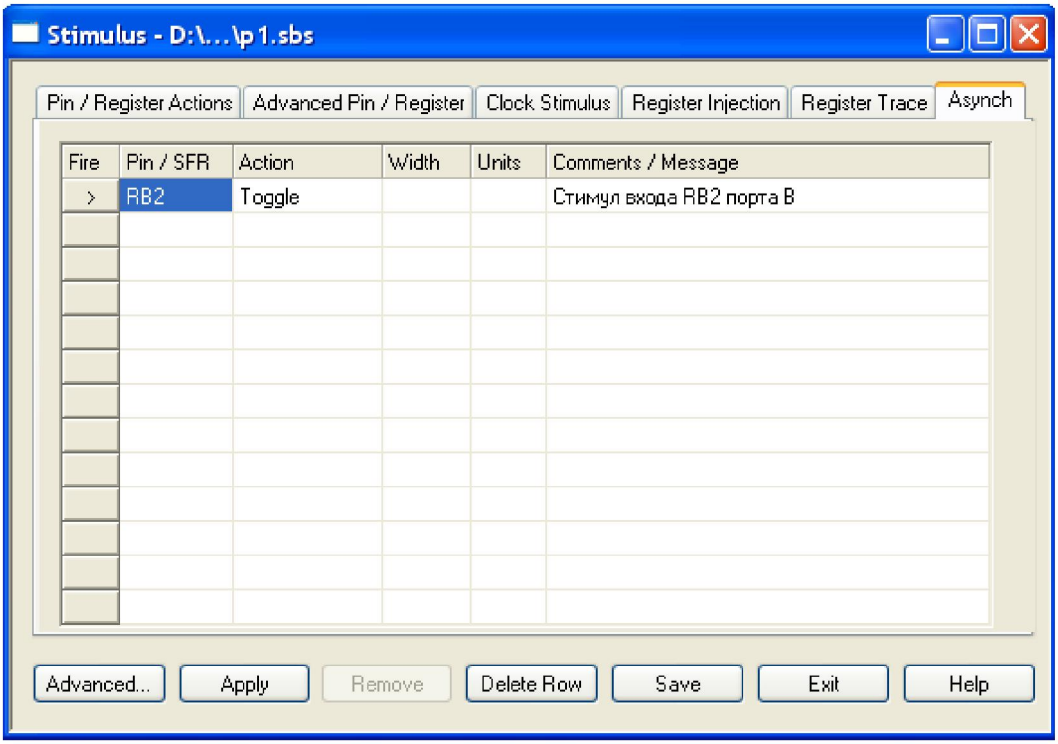


Рис.3. Создание асинхронного стимула для входа RB2 порта B микроконтроллера.

Чтобы сохранить рабочую книгу стимулов, выполните команду меню **Debugger/Stimulus/Save Workbook** и сохраните ее под именем вашего проекта.

Чтобы закрыть рабочую книгу стимулов, выполните команду меню **Debugger/Stimulus/Close Workbook**.

Чтобы воспользоваться стимулами вторично, выполните команду меню **Debugger/Stimulus/Open Workbook** и откройте файл сохраненной рабочей книги стимулов.

ЗАДАНИЯ ДЛЯ ВЫПОЛНЕНИЯ

1. Создайте в MPLAB проекты *Управление миганием светодиода*, *Управление генерацией звука*, *Обработка нажатия кнопки*, скомпилируйте их и отладьте с помощью инструмента MPLAB SIM.
2. В проекте *Обработка нажатия кнопки* создайте асинхронные стимулы, имитирующие нажатия кнопок S1 и S2, и в режиме **Debugger/Animate** убедитесь в их работоспособности.
3. Разработайте алгоритм и измените программу *Обработка нажатия кнопки* таким образом, чтобы при нажатии кнопки S2 генерировался звук, а мигания светодиода прекращались. И наоборот, при отпущенной кнопке S2 должны выполняться только мигания светодиода.
4. Еще раз измените программу *Обработка нажатия кнопки* таким образом, чтобы при нажатии кнопки S2 раздавался короткий сигнал *beep*, после чего программа возвращалась к нормальным миганиям светодиода независимо от состояния этой кнопки. При отпускании и повторном нажатии кнопки S2 все должно повторяться. Кратковременное нажатие кнопки S1 должно отключать генерацию сигнала *beep*, а повторное нажатие этой кнопки – снова включать ее.

Лабораторная работа №4. Система прерываний МК PIC16F84A. Использование прерываний для программного управления таймером TMR0

Цель работы: Изучить систему прерываний микроконтроллера PIC16F84A, научиться пользоваться встроенным счетчиком/таймером TMR0.

СИСТЕМА ПРЕРЫВАНИЙ МИКРОКОНТРОЛЛЕРА PIC16F84A

Микроконтроллер PIC16F84A имеет 4 источника прерываний:

- внешнее прерывание по входу RB0/INT;
- прерывание по переполнению таймера TMR0;
- прерывание по изменению состояния линий RB7:RB4 порта B;
- прерывание по завершению записи в EEPROM.

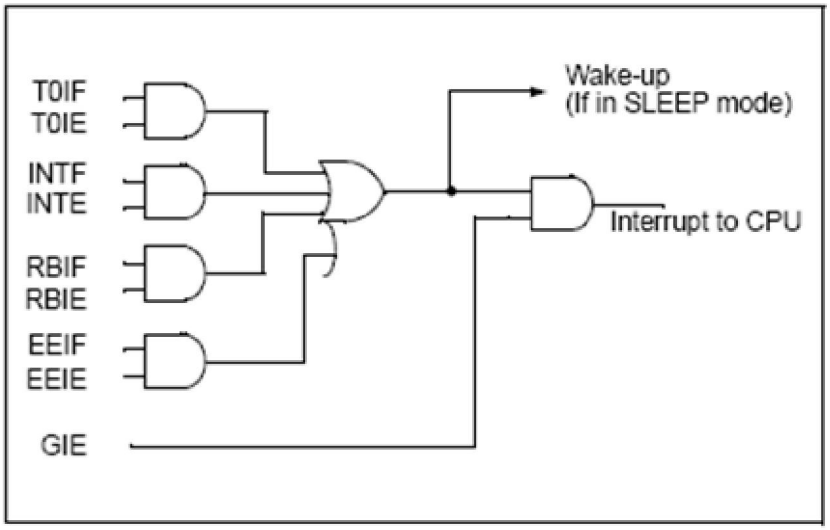


Рис.1. Логика прерываний микроконтроллера PIC16F84A.

Регистр управления прерываниями INTCON фиксирует индивидуальные запросы на прерывание в соответствующих битах – флагах, а также содержит индивидуальные и глобальный биты разрешения прерываний.

Регистр INTCON (адрес 0Bh, 8Bh) доступен для чтения и записи, содержит управляющие биты, разрешающие доступ к различным источникам прерываний.

Назначение битов регистра INTCON

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

bit7 GIE – бит глобального разрешения/запрета прерываний: 1 – разрешает немаскированные прерывания, 0 – запрещает все прерывания. Глобальный бит разрешения прерываний GIE разрешает (если установлен) все немаскируемые прерывания или отключает все прерывания. Бит GIE очищается при системном сбросе. Индивидуальные прерывания могут быть отключены посредством сброса соответствующих битов разрешения в регистре INTCON (EEIE, T0IE, INTE, RBIE);

bit6 EEIE – бит разрешения прерываний по окончанию записи в EEPROM: 1 –прерывание разрешено, 0 – прерывание запрещено;

bit5 T0IE – бит разрешения прерываний по переполнению таймера TMR0: 1 – прерывание разрешено, 0 – прерывание запрещено;

bit4 INTE – бит разрешения внешних прерываний по входу RB0/INT: 1 – прерывание разрешено, 0 – прерывание запрещено;

bit3 RBIE – бит разрешения прерываний по изменению состояния на входах RB7:RB4 порта B: 1 – прерывание разрешено, 0 – прерывание запрещено;

bit2 T0IF – флаг прерываний по переполнению таймера/счетчика TMR0: 1 – таймер переполнен, 0 – таймер не переполнен;

bit1 INTF – флаг внешних прерываний по входу RB0/INT: 1 – прерывание произошло (должен быть сброшен программно обработчиком прерывания), 0 – прерывание не произошло;

bit0 RBIF – флаг прерываний по изменению состояния на входах порта B: 1 – по крайней мере один из входов RB7:RB4 изменил свое состояние, 0 – ни один из входов RB7:RB4 не изменил свое состояние.

Флаги прерываний INTF, RBIF и T0IF устанавливаются аппаратно при появлении запросов на то или иное прерывание[2]. Процедура обработки прерывания IRS (Interrupt Service Routine) должна определить источник прерывания путем анализа состояния флагов прерываний. Флаги прерываний должны быть очищены программно перед разрешением прерываний, чтобы избежать зацикливания прерываний.

Когда прерывание начинает обрабатываться, то сбрасывается бит GIE, чтобы запретить любые другие прерывания, адрес возврата из процедуры отработки прерывания записывается в стек и в регистр PC записывается адрес подпрограммы обработки прерывания 0x0004.

Команда возврата из прерывания RETFIE предназначена для выхода из процедуры прерывания, а также устанавливает бит GIE, разрешая последующие прерывания.

Внешнее прерывание по входу RB0/INT управляется фронтом, если установлен бит OPTION_REG<6>, или спадом, если этот бит сброшен, сигнала. Когда на входе RB0/INT появляется заданный фронт, устанавливается бит INTF. Флаг INTF должен быть сброшен программно в процедуре IRS, перед тем как разрешить прерывания.

Прерывание INT может быть отключено путем сброса управляющего бита INTE (INTCON<4>).

Прерывание INT может пробуждать процессор из режима SLEEP, только если бит INTE был установлен перед тем, как перейти в режим SLEEP. Будет ли выполняться ветвь обработчика прерываний при пробуждении из режима SLEEP, определяется состоянием бита GIE.

Регистр OPTION_REG (адрес 81h) доступен для чтения и записи, содержит управляющие биты для управления делителем таймеров TMR0 и WDT, внешним прерыванием INT, таймером TMR0 и подтягивающими резисторами для порта B.

Назначение битов регистра OPTION_REG

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit7 RBPU – разрешение общего доступа к подтягивающим резисторам порта B: 1 – подтягивающие резисторы отключены, 0 – подтягивающие резисторы включены;

bit6 INTEDG – выбор фронта сигнала прерывания по входу RB0/INT: 1 - прерывание по положительному фронту (нарастанию), 0 - прерывание по отрицательному фронту (спаду);

bit5 T0CS – выбор источника синхронизации для таймера TMR0: 1 - сигнал на входе RA4/T0KI, 0 – внутренняя тактовая частота (CLKOUT);

---

[2] Микроконтроллер PIC16F84A имеет только один вектор прерываний по адресу 0x0004.

bit4 T0SE – выбор фронта сигнала, по которому срабатывает таймер TMR0 по сигналу на входе RA4/T0KI: 1 - инкремент по отрицательному фронту (спаду), 0 - инкремент по положительному фронту (нарастанию);

bit3 PSA – бит, управляющий подключением делителя (prescaler): 1 – делитель подключен к WDT, 0 - делитель подключен к TMR0.

bit2:bit0 PS2–PS0 – выбор коэффициента деления делителя для таймеров TMR0 и WDT:

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Ниже приведены блок-схема алгоритма (рис.2) и листинг демонстрационной программы обработки прерывания от таймера TMR0 для микроконтроллера PIC16F84A.

Программа очищает порт B и настраивает все его линии на вывод, разрешает глобальные прерывания и прерывания от таймера. Затем программа входит в бесконечный цикл `loop`, в котором она поочередно устанавливает на всех линиях порта B то высокий уровень напряжения на время `delay1`, то низкий уровень напряжения на время `delay2`.

Задержка времени `delay1` определяется временем работы таймера TMR0 до переполнения (прерывание T0INT), задержка времени `delay2` задается программным способом.

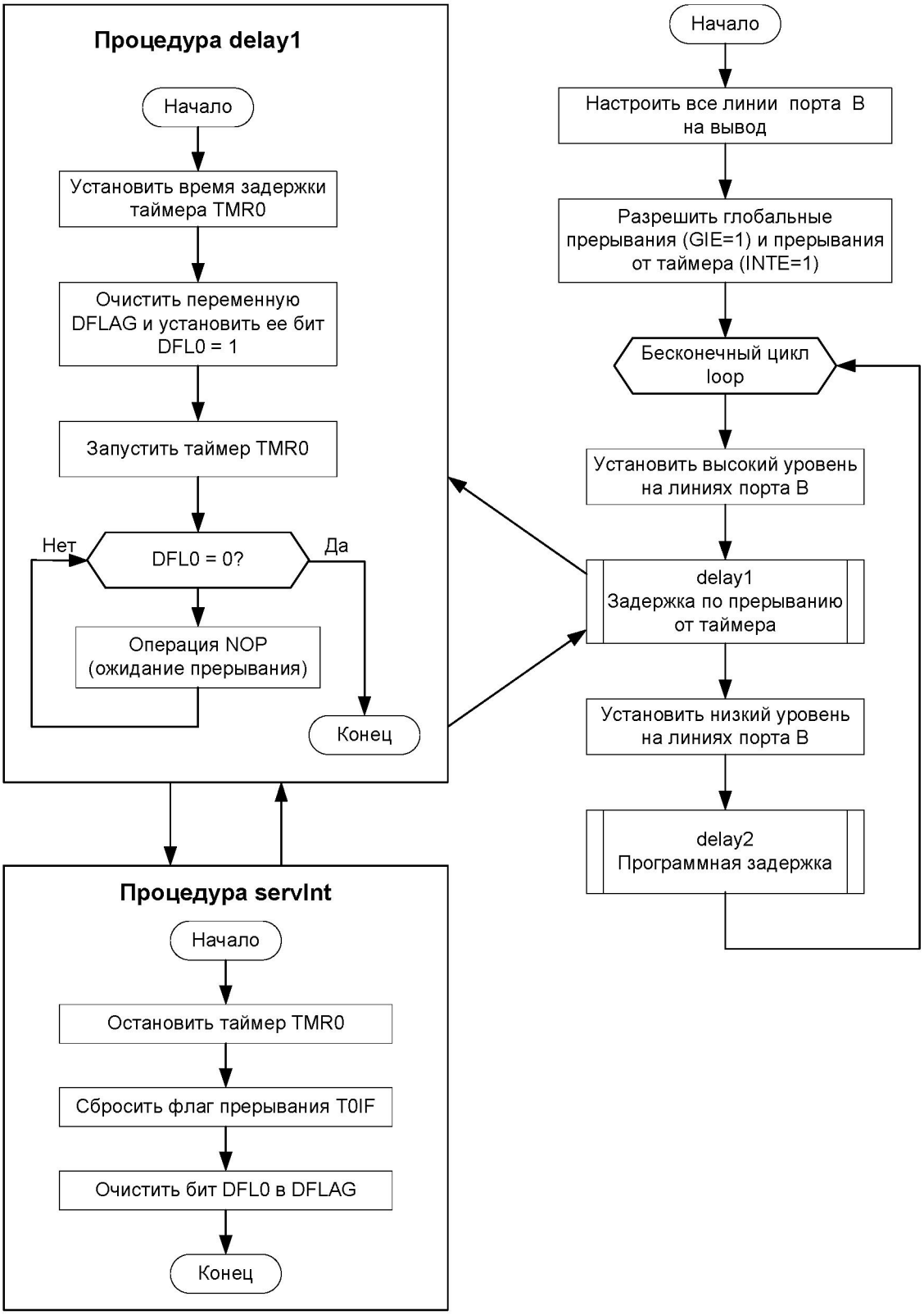


Рис.2. Блок-схема алгоритма программы обработки прерывания от таймера TMR0.

Листинг программы обработки прерываний от таймера (файл `intr.asm`):

```
;*************************************************
;* Чередование выходных линий порта B между 1 и 0 *
;*************************************************
list p=PIC16f84A
```

```
include p16f84A.inc
__config _CP_OFF&_WDT_OFF&_PWRTE_ON&_XT_OSC

DTEMP equ 0x0C
DFLAG equ 0x0D
DFL0  equ 0x00          ; в качестве флага выбран бит 0 переменной DFLAG
  org     0x00
  goto    start         ; переход на главную программу

  org 0x04              ; вектор прерываний
  goto servInt          ; переход на процедуру обработки прерываний


; --- Процедура обработки прерываний ---
servInt
  org 0x08
  banksel OPTION_REG          ; выбор банка 1
  bsf     OPTION_REG, T0CS    ; остановить таймер TMR0
  bcf     STATUS,RP0          ; выбор банка 0
  bcf     INTCON, T0IF        ; очистить флаг переполнения
  bcf     DFLAG, DFL0         ; очистить бит DFL0 переменной DFLAG
  retfie                      ; возврат из процедуры обработки прерываний

;*********************
;* Главная программа *
;*********************
start
  clrf    PORTB               ; очистить порт B
  banksel TRISB               ; выбрать банк  1
  clrf    TRISB               ; настроить все линии порта B на вывод
  banksel INTCON              ; выбрать банк 0
  bsf     INTCON, GIE         ; разрешить глобальные прерывания
  bsf     INTCON, T0IE        ; разрешить прерывания от таймера

loop
  movlw   0xFF
  movwf   PORTB               ; записать в порт B значение 0xFF
  call    delay1              ; задержка delay1 таймера TMR0
  clrf    PORTB               ; очистить порт B
  call    delay2              ; задержка delay2
  goto    loop                ; зацикливание

;**********************************
;* Процедура задержки таймера TMR0 *
;**********************************
delay1
  movlw   0xF0
  movwf   TMR0                ; установить время задержки таймера TMR0

  clrf    DFLAG               ; очистить переменную DFLAG
  bsf     DFLAG, DFL0         ; установить бит DFL0 переменной DFLAG
  banksel OPTION_REG          ; выбрать банк 1
  bcf     OPTION_REG, T0CS    ; запустить таймер TMR0
  banksel DFLAG               ; выбрать банк 0
tloop                         ; ожидание переполнения (0xFF->0x00), прерывание
  btfsc   DFLAG, DFL0         ; после прерывания DFL0 = 0
  goto    tloop               ; если DFL0 = 1, то зацикливание
  return                      ; конец процедуры delay1

;****************************
;* Процедура задержки delay2 *
;****************************
```

```
delay2
  movlw   0x0f
  movwf   DTEMP              ; установить значение переменной DTEMP

dloop                        ; цикл задержки
  decfsz  DTEMP, F
  goto    dloop
  return                     ; конец процедуры delay2

  goto $                     ; безусловный переход на текущую строку

  end                        ; конец программы
```

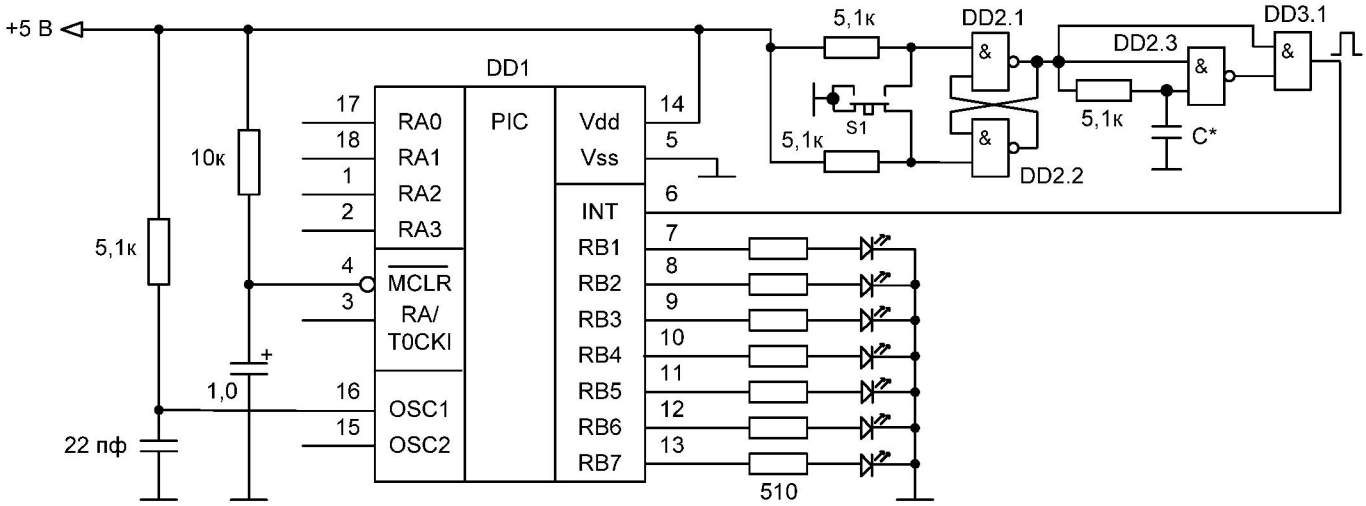
ЗАДАНИЯ ДЛЯ ВЫПОЛНЕНИЯ

1. В MPLAB создайте новый проект, добавьте к нему ассемблерный файл демонстрационной программы `intr.asm`, скомпилируйте проект и запустите его на выполнение командой меню **Debugger/Animate**. В процессе выполнения программы отслеживайте изменение содержимого порта B и флага - бита DFL0 переменной DFLAG.
2. Остановите выполнение программы (клавиша F5), сбросьте процессор (клавиша F6) и снова выполняйте программу, но теперь уже в пошаговом режиме (клавиша F7).

   Отслеживайте изменение содержимого регистра INTCON. Убедитесь в том, что в момент вызова процедуры обработки прерывания `servInt` в регистре INTCON сбрасывается флаг `GIE` и устанавливается флаг `T0IF`. Далее трассируйте процесс выполнения команд процедуры `servInt` и выясните, каким образом восстанавливается первоначальное состояние этих флагов.
3. В начало процедуры обработки прерывания `servInt` демонстрационной программы добавьте программный код, анализирующий состояние флагов `T0IF`, `INTF` и `RBIF` регистра INTCON, т.е. реализующий алгоритм:

```
If <Установлен флаг T0IF> Then
    <Обработать прерывание от TMR0>
ElseIf <Установлен флаг INTF> Then
    <Обработать внешнее прерывание INT>
ElseIf <Установлен флаг BRIF> Then
    <Обработать прерывание по изменению входов RB7:RB4>
Else
    <Обработать прерывание по окончанию записи в EEPROM>
End If
```

4. На языке ассемблера MPASM разработайте управляющую программу для микропроцессорной системы, представленной на рисунке.



Управляющая программа в цикле должна поочередно включать и выключать светодиоды по принципу «бегущего огня». Время свечения каждого светодиода, равное 1 сек., задавайте программной задержкой. Направление движения бегущего огня задавайте с помощью пользовательского флага.

При нажатии кнопки S1 формируется импульс внешнего запроса на прерывание INT. Процедура обработки этого прерывания должна инвертировать флаг направления движения бегущего огня и переключать направление движения «бегущего огня» в обратную сторону.

Отлаживая программу в MPLAB, для входа RB0 порта B создайте асинхронный стимул типа *Pulse High* длительностью в 10 машинных циклов и используйте его для имитации нажатия кнопки S1.

В процессе разработки программы рекомендуется сначала создать вспомогательные программы *Бегущий огонь* и *Внешние прерывания*, затем объединить их для решения поставленной задачи.

Учебно-методическое и информационное обеспечение дисциплины

Основная литература

1. Белов, А. В. Разработка устройств на микроконтроллерах AVR: шагаем от чайника до профи [Текст] : кн. + видеокурс / А. В. Белов. - СПб. : Наука и Техника, 2014. - 526 с.
2. Микропроцессорные системы [Электронный ресурс] : учеб. пособие для вузов по направлению подгот. бакалавров и магистров "Информатика и вычисл. техника" / под общ. ред. Д. В. Пузанкова. - СПб. : Политехника, 2012. - 935 с. - (Учебное пособие для вузов) - Режим доступа: http://www.iprbookshop.ru/16297.html.
3. Ревич, Ю. В. Практическое программирование микроконтроллеров Atmel AVR на языке ассемблера [Текст] / Ю. В. Ревич. - СПб. : БХВ-ПЕТЕРБУРГ, 2014. - 365 с.
4. Учебно-методический комплекс по дисциплине "Микропроцессорные системы" [Текст] : для студентов специальности 100101.65 "Сервис"

(специализация "Сервис компьютер. и микропроцессор. техники") / Поволж. гос. ун-т сервиса (ФГБОУ ВПО "ПВГУС"), Каф. "Информ. и электрон. сервис" ; сост.: В. Н. Будилов, В. И. Воловач. - Тольятти : ПВГУС, 2012. - 379 с.

5. Учебно-методический комплекс по дисциплине "Микропроцессорные системы" [Электронный ресурс] : для студентов специальности 100101.65 "Сервис" (специализация "Сервис компьютер. и микропроцессор. техники") / Поволж. гос. ун-т сервиса (ФГБОУ ВПО "ПВГУС"), Каф. "Информ. и электрон. сервис" ; сост.: В. Н. Будилов, В. И. Воловач. - Тольятти : ПВГУС, 2012. - 6,17 МБ, 379 с. - Режим доступа: http://elib.tolgas.ru.
6. Учебно-методический комплекс по дисциплине "Микропроцессорные системы" [Электронный ресурс] : для студентов направления подгот. 230100.68 "Информатика и вычисл. техника" / Поволж. гос. ун-т сервиса (ФГБОУ ВПО "ПВГУС"), Каф. "Информ. и электрон. сервис" ; сост. В. И. Аникин. - Тольятти : ПВГУС, 2012. - 4,12 МБ, 224 с. - Режим доступа: http://elib.tolgas.ru.

Дополнительная литература

7. Брей, Б. Применение микроконтроллеров PIC18. Архитектура, программирование и построение интерфейсов с применением С и ассемблера: пер. с англ. [Текст] / Б. Брей – К. : «МК Пресс»; СПб. : «КОРОНА-ВЕК», 2008. – 576 с.: ил.
8. Гуров, В. В. Архитектура микропроцессоров [Электронный ресурс] : учеб. пособие / В. В. Гуров. - М. : Интернет-Ун-т Информ. Технологий [и др.], 2010. - 272 с. - (Основы информационных технологий) - Режим доступа: http://www.iprbookshop.ru/15852.html.
9. Предко, М. PIC-микроконтроллеры: архитектура и программирование: пер. с англ. [Текст]. – М. : МДК Пресс, 2010. – 512 с.: ил.
10. Программирование на языке С для AVR и PIC микроконтроллеров [Текст] / сост. Ю. А. Шпак. – К. : «МК-Пресс», СПб. : «КОРОНА-ВЕК», 2011. – 544 с.
11. Угрюмов, Е. П. Цифровая схемотехника [Электронный ресурс] : учеб. пособие для вузов по направлению подгот. "Информатика и вычисл. техника" / Е. П. Угрюмов. - СПб. : БХВ-Петербург, 2010. - 816 с. - Режим доступа: http://znanium.com/bookread.php?book=350426.
12. Уилмсхерст, Т. Разработка встроенных систем с помощью микроконтроллеров PIC. Принципы и практические примеры: пер. с англ. [Текст] / Т. Уилмсхерст. - К. : «МК Пресс»; СПб. : «КОРОНА-ВЕК», 2008. – 544 с.: ил.
13. Хартов, В. Я. Микропроцессорные системы: учеб. пособие для студентов учреждений высш. проф. образования [Текст] / В. Я. Хартов. – М. : Изд. центр «Академия», 2010. – 352 с.

Программное обеспечение современных информационно-коммуникационных технологий и Интернет-ресурсы

Используемые при выполнении лабораторных работ программное обеспечение и Интернет - ресурсы:

- Microsoft Office;
- Microchip MP LAB;
- Поволжский государственный университет сервиса [Электронный ресурс]. - Режим доступа: http://www.tolgas.ru/.
- Microchip Technology Inc. [Электронный ресурс]. - Режим доступа: www.microchip.com.
- Atmel Corporation [Электронный ресурс]. - Режим доступа: www.atmel.com.