МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра биомедицинской инженерии

Утверждаю
Проректор по учебной работе
О.Г. Локтионова
«25» 09 2023 г.

АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ РАСЧЕТА И ПРОЕКТИРОВАНИЯ ЭЛЕКТРОННЫХ СХЕМ

Методические рекомендации по выполнению самостоятельных работ для студентов направления подготовки 12.03.04 – «Биотехнические системы и технологии» (бакалавр)

Курск 2023

УДК 621.(076.1)

Составители: А.А.Кузьмин

Рецензент:
Кандидат технических наук, доцент *Т.Н. Конаныхина*

Автоматизированные системы расчета и проектирования электронных схем: методические рекомендации по выполнению самостоятельных работ для студентов направления подготовки 12.03.04 – «Биотехнические системы и технологии» (бакалавр) / Юго-Зап. гос. ун-т; сост.: А.А.Кузьмин. - Курск, 2023. -31 с.

Содержат методические рекомендации к проведению самостоятельных работ по дисциплине «Автоматизированные системы расчета и проектирования электронных схем». Методические указания по структуре, содержанию и стилю изложения материала соответствуют методическим и научным требования, предъявляемым к учебным и методическим пособиям.

Предназначены для студентов направления подготовки 12.03.04 – «Биотехнические системы и технологии» (бакалавр)

Текст печатается в авторской редакции

Подписано в печать 25.09.23 Формат 60х84 1/16
Усо.печ.л. 1,8 . Уч.-изд.л. 1,6 .Тираж 30 экз. Заказ: 1092 .Бесплатно.
Юго-Западный государственный университет.
305040. г. Курск, ул. 50 лет Октября, 94.

Самостоятельная работа №1

Система команд микроконтроллеров Microchip PIC16. Объявления переменных, пересылки, вычисления.

1 Краткие теоретические сведения

Микроконтроллеры семейства PIC16 имеют очень эффективную систему команд, состоящую всего из 35 инструкций. Все инструкции выполняются за один цикл, за исключением условных переходов и команд, изменяющих программный счетчик, которые выполняются за 2 цикла. Один цикл выполнения инструкции состоит из 4 периодов тактовой частоты. Таким образом, при частоте 4 МГц, время выполнения инструкции составляет 1 мксек. Каждая инструкция состоит из 14 бит, делящихся на код операции и операнд (возможна манипуляция с регистрами, ячейками памяти и непосредственными данными). Система команд микроконтроллеров PIC16 представлена в таблице 1.1.

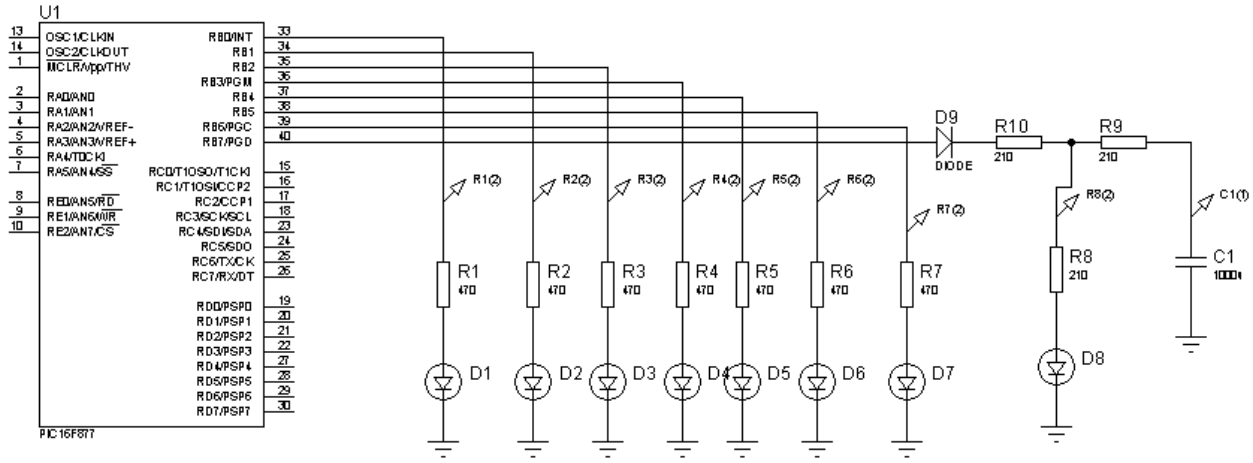

Рисунок 1.1 – Схема включения установки Самостоятельнуюой работы №1 (осциллятор и питание микроконтроллера в схеме не показаны)

Принципиальная схема устройства индикации, которое мы будем использовать для демонстрации работы основных команд PIC16, приведена на рисунке 1.1. Устройство состоит из 8 светодиодов с токоограничивающими резисторами (диод D9 включен с параллельным светодиоду конденсатором для демонстрации эффекта «плавного» включения-выключения). Каждый вывод микроконтроллеров семейства PIC может непосредственно управлять светодиодом без дополнительных усилителей.

Начнем с описания базового кода, который будет использован в наших примерах. Весь код до строки с выражением ORG 0 условно

называется секцией заголовка. В секции заголовка или определяются логические имена для всех используемых в проекте ресурсов - портов, битовых и байтовых переменных и регистров, или подключаются стандартные файлы определений. Например, можно непосредственно определить логические имена так:

```
; описание операционных регистров
TMR0          EQU     01h
PC            EQU     02h
STATUS        EQU     03h
FSR           EQU     04h
```

А можно включить стандартные описания командой

```
    #include p16f877.inc
```

Базовый код программы для микроконтроллера показан ниже:

```
;Назначение - ….
;Схема включения - ….
;Автор программы =…..

listp=16f877     ;Листинг для команд
                 ;микроконтроллера 16F877
#include p16f877.inc ;Определения для
                          ;микроконтроллера 16F877
;-----------------------------------------------
;Определение переменных
; Например так:
SCRATCH          EQU     0x20  ;Регистр
;-----------------------------------------------
org 0x00
nop      ;Первая операция - nop для
             ;внутрисхемных отладчиков
goto start   ;"Стандартное" начало

org 0x8
start        ;Метка начала программы
;-------Инициализация портов и т.д.--------------
bsf STATUS,RP0        ;Регистр TRISB не в
                     ;нулевой странице!
movlw 0x00        ;все выводы PORTB как выходы
movwf TRISB
bcf STATUS,RP0        ;Возвращаемся к
                         ;нулевой странице

;------Главный цикл-------------------------
;Здесь непосредственно программа
```

```
;-----------------------------------------
End       ;Конец программы
```

Все строки, начинающиеся со знака ";", воспринимаются ассемблером как комментарии. Разберем выражение «SCRATCH EQU 0x20». Ассемблер подставляет значение 0x20 всякий раз, когда встретится слово SCRATCH. Слово "EQU" означает эквивалентность. Таким образом, присваивается символам SCRATCH значение 0x20. Пользовательские регистры в микроконтроллере PIC16F877 начинаются с адреса 0x20. Использование символьных имен устраняет двусмысленность и позволяет облегчить чтение исходного текста.

Перед тем, как начать исполняемый код, необходимо задать выражение ORG 0. Это указатель для ассемблера, что код, следующий за этим выражением, начинается с нулевого адреса ЭППЗУ. Выражение "ORG" используется для размещения сегментов кода по различным адресам в пределах размеров ЭППЗУ. Еще одно выражение ORG находится перед меткой start, имеющей адрес 0x8, как задано выражением ORG 0x8. Исполняемый код должен заканчиваться директивой END, означающей, что за этой директивой отсутствуют исполняемые команды. При включении питания PIC16F877 переходит на адрес 000h. Первая инструкция, которая будет выполнена процессором, это команда NOP.

Таблица 1.1 – Список команд микроконтроллеров PIC16

Мнемоника команды		Описание	Циклов	14-разрядный код		Изм. флаги	Прим.
				Бит 13	Бит 0		
Байт ориентированные команды							
ADDWF	**f,d**	Сложение W и f	1	00 0111 dfff ffff		C,DC,Z	1,2
ANDWF	**f,d**	Побитное 'И' W и f	1	00 0101 dfff ffff		Z	1,2
CLRF	**f**	Очистить f	1	00 0001 1fff ffff		Z	2
CLRW	**-**	Очистить W	1	00 0001 0xxx xxxx		Z	
COMF	**f,d**	Инвертировать f	1	00 1001 dfff ffff		Z	1,2
DECF	**f,d**	Вычесть 1 из f	1	00 0011 dfff ffff		Z	1,2
DECFSZ	**f,d**	Вычесть 1 из f и пропустить если 0	1(2)	00 1011 dfff ffff			1,2,3
INCF	**f,d**	Прибавить 1 к f	1	00 1010 dfff ffff		Z	1,2
INCFSZ	**f,d**	Прибавить 1 к f и пропустить если 0	1(2)	00 1111 dfff ffff			1,2,3
IORWF	**f,d**	Побитное 'ИЛИ' W и f	1	00 0100 dfff ffff		Z	1,2
MOVF	**f,d**	Переслать f	1	00 1000 dfff ffff		Z	1,2
MOVWF	**f**	Переслать W в f	1	00 0000 1fff ffff			
NOP	**-**	Нет операции	1	00 0000 0xx0 0000			
RLF	**f,d**	Циклический сдвиг f влево через перенос	1	00 1101 dfff ffff		C	1,2
RRF	**f,d**	Циклический сдвиг f вправо через перенос	1	00 1100 dfff ffff		C	1,2
SUBWF	**f,d**	Вычесть W из f	1	00 0010 dfff ffff		C,DC,Z	1,2
SWAPF	**f,d**	Поменять местами полубайты в регистре f	1	00 1110 dfff ffff			1,2
XORWF	**f,d**	Побитное 'исключающее ИЛИ' W и f	1	00 0110 dfff ffff		Z	1,2
Бит ориентированные команды							
BCF	**f,b**	Очистить бит b в регистре f	1	01 00bb bfff ffff			1,2
BSF	**f,b**	Установить бит b в регистре f	1	01 01bb bfff ffff			1,2
BTFSC	**f,b**	Проверить бит b в регистре f, пропустить если 0	1(2)	01 10bb bfff ffff			3
BTFSS	**f,b**	Проверить бит b в регистре f, пропустить если 1	1(2)	01 11bb bfff ffff			3
Команды управления и операций с константами							
ADDLW	**k**	Сложить константу с W	1	11 111x kkkk kkkk		C,DC,Z	
ANDLW	**k**	Побитное 'И' константы и W	1	11 1001 kkkk kkkk		Z	
CALL	**k**	Вызов подпрограммы	2	10 0kkk kkkk kkkk			
CLRWDT	**-**	Очистить WDT	1	00 0000 0110 0100		-TO,-PD	
GOTO	**k**	Безусловный переход	2	10 1kkk kkkk kkkk			
IORLW	**k**	Побитное 'ИЛИ' константы и W	1	11 1000 kkkk kkkk		Z	
MOVLW	**k**	Переслать константу в W	1	11 00xx kkkk kkkk			
RETFIE	**-**	Возврат из подпрограммы с разрешением прерываний	2	00 0000 0000 1001			
RETLW	**k**	Возврат из подпрограммы с загрузкой константы в W	2	11 01xx kkkk kkkk			
RETURN	**-**	Возврат из подпрограммы	2	00 0000 0000 1000			
SLEEP	**-**	Перейти в режим SLEEP	1	00 0000 0110 0011		-TO,-PD	
SUBLW	**k**	Вычесть W из константы	1	11 110x kkkk kkkk		C,DC,Z	
XORLW	**k**	Побитное 'исключающее ИЛИ' константы и W	1	11 1010 kkkk kkkk		Z	

Примечания:

1. 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'.
2. При выполнении записи в TMR0 (и d=1) предделитель TMR0 сбрасывается, если он подключен к модулю TMR0.
3. Если условие истинно или изменяется значение счетчика команд PC, то инструкция выполняется за два цикла. Во втором цикле выполняется команда NOP.

Эта команда ничего не выполняет, и включение ее первой является требованием для использования внутрисхемных отладчиков. В конечном варианте программы эту команду можно исключить. Вторая команда - GOTO start, которая передает управление на адрес 0x8, и дальнейшая работа будет продолжаться с этого адреса. start - это выбираемое пользователем имя метки (метки всегда должны начинаться с первой позиции строки), которое ассемблер использует в качестве адресной ссылки. В процессе работы ассемблер определяет расположение метки start и запоминает, что если это имя будет встречено еще раз, вместо него будет подставлен адрес метки. Команды CALL и GOTO используют метки для ссылок в исходном тексте.

Следующая команда bsf STATUS,RP0 переключает страницы памяти, так как регистр TRISB, к которому мы хотим обратиться, находится в первой странице, а при включении микроконтроллера по умолчанию устанавливается нулевая страница памяти. Возврат к нулевой странице происходит после выполнения команды bcf STATUS,RP0. Команда movlw 0x00 загружает в рабочий регистр W значение, 0x00. Это значение может быть задано и в двоичном формате как B'00000000'.

Символы B' означают, что данные заданы в двоичном формате. Можно было бы написать в этом же месте 0h, 0x0 (шестнадцатеричный) и получить тот же самый результат. Десятичные значения должны начинаться с точки, т.е. число пятьдесят должно записываться как .50. По умолчанию, если число записывается без уточняющих формат символов, считается, что число записано в шестнадцатеричном формате! Эта особенность ассемблера является источником многочисленных ошибок!

Двоичное представление удобнее использовать в тех случаях, когда предполагается операция с битами в регистре. Следующая команда MOVWF TRISB загружает значение из рабочего регистра W в регистр управления конфигурацией порта B TRISB. Задание 0 в разряде этого регистра определяет, что соответствующий разряд порта B является выходом. В нашем случае все разряды порта B устанавливаются выходами. Если бы мы захотели, например, установить младший разряд порта B как вход, мы бы задали в регистре TRISB значение B'00000001'.

ПЕРВАЯ ПРОГРАММА

Для первой программы нам хватит всего трех команд:

```
MOVLW   k
MOVWF   f
GOTO    k
```

Мы уже использовали эти команды в заголовке нашего базового кода. Команда MOVLW загружает байтовый литерал или константу в рабочий регистр W. Следующая команда MOVWF пересылает байт из рабочего регистра W в заданный регистр f. Команда GOTO передает управление на адрес k. Следующая программа записывает в рабочий регистр W значение 01010101 и затем выдает его содержимое на порт B. После запуска этой программы Вы увидите свечение четырех светодиодов.

```
MOVLW   B'01010101'  ;загрузить 01010101
                      ;в регистр W
MOVWF   PORTB        ;записать W в порт B
GOTO    $            ;зациклиться навсегда
```

Директива ассемблера "$" означает текущее значение программного счетчика (PC). Поэтому команда GOTO $ означает переход туда, где мы в данный момент находимся. Такой цикл бесконечен, поскольку не существует способа (кроме прерывания) выйти из него. Команда GOTO $ часто применяется для остановки кода при отладке. В реальных программах для микроконтроллеров такая команда практически не применяется.

НАБОР КОМАНД PIC

Перейдем к описанию основного набора команд микроконтроллеров семейства PIC. Мы по-прежнему будем ориентироваться на PIC16F877, хотя почти все, о чем мы будем говорить, применимо и к другим микроконтроллерам семейства PIC. По ходу описания мы будем составлять короткие программы, чтобы лучше понять, как работают те или иные команды.

NOP

Начнем наше описание с команды NOP. Посмотреть результат выполнения этой команды трудно, поскольку она не делает ничего. Эта инструкция обычно используется в циклах временной задержки или для точной настройки времени выполнения определенного участка программы.

CLRW

Эта команда очищает рабочий регистр W. Добавим одну строчку в наш пример и увидим, что все светодиоды потухнут.

```
MOVLW   B'01010101' ;загрузить 01010101 в регистр W
CLRW                ;очистить регистр W
MOVWF   PORTB       ;записать W в порт B
GOTO    $           ;зациклиться навсегда
```

CLRF f

CLRF делает для любого регистра то же, что CLRW делает для рабочего регистра W. Следующая команда установит порт B в 0h.

```
CLRF    PORTB    ;очистить порт B (DATAPORT)
```

SUBWF f,d

Вычесть рабочий регистр W из любого регистра f. Эта команда также устанавливает признаки CARRY, DIGIT CARRY и ZERO в регистре STATUS. После выполнения команды можно проверить эти признаки и определить, является ли результат нулевым, положительным или отрицательным. Символ d после запятой означает адрес, куда будет помещен результат выполнения команды. Если d=0, то результат помещается в рабочий регистр W, а если d=1, то результат записывается в использованный в команде регистр f. Допускается также игнорировать написание символа d. В таком случае подразумевается, что d=1, т.е. результат поместится в регистр f. А вместо указания d=0 допускается писать символ W – так код выглядит намного нагляднее. Например, следующие две строчки эквивалентны:

```
SUBWF   SCRATCH,0    ;здесь надо помнить, что 0-это
                     ;признак рабочего регистра W
SUBWF   SCRATCH,W    ;Так намного нагляднее!
```

И эти две строчки тоже эквивалентны:

```
SUBWF   SCRATCH,1    ;здесь надо помнить, что 1-это
                     ;признак регистра-источника
SUBWF   SCRATCH      ;Так намного нагляднее!
```

В нашем примере в регистр SCRATCH загружается значение 0FFh, а в регистр W значение 01h. Затем выполняется команда SUBWF и результат отображается на светодиодах.

```
MOVLW   0FFh         ;загрузить 0FFh в регистр W
MOVWF   SCRATCH      ;загрузить содержимое W в
                     ;регистр SCRATCH
MOVLW   01h          ;загрузить 01h в регистр W
SUBWF   SCRATCH,W    ;выполнить вычитание
MOVWF   PORTB        ;записать W в порт B
GOTO    $            ;зациклиться навсегда
```

Светодиоды должны отобразить 11111110, где 0 соответствует потушенному светодиоду, а 1 - горящему.

Обратите внимание, что перед значением FFh в примере вычитания стоит "0". Символ "0" для ассемблера означает, что это число, а не метка. Если бы символа 0 не было, то ассемблер начал бы искать метку с именем FFh, которой в этой программе не существует и, соответственно, возникла бы ошибка. символ "h", следующий за значением 0FF, означает,

что значение задано в шестнадцатеричном формате. Также шестнадцатеричный формат можно задавать как в языке Си в таком виде: 0xFF

ADDWF f,d

Команда ADDWF работает аналогично SUBWF f,d, прибавляя рабочий регистр W к любому регистру f и устанавливая те же признаки. Следующий пример демонстрирует работу команды ADDWF.

```
MOVLW   0h          ;загрузить 0 в регистр W
MOVWF   SCRATCH     ;загрузить содержимое W
                    ;в регистр SCRATCH
MOVLW   1h          ;загрузить 01h в регистр W
ADDWF   SCRATCH,W   ;выполнить сложение
MOVWF   PORTB       ;записать W в порт B
GOTO    $           ;зациклиться навсегда
```

Светодиоды должны отобразить 00000001

SUBLW k

ADDLW k

Эти две команды работают совершенно аналогично вышеописанным, за тем исключением, что операция производится между рабочим регистром W и байтовой константой, заданной в команде. Команда SUBLW вычитает рабочий регистр W из константы k, а команда ADDLW добавляет рабочий регистр W к константе k. Эти команды также устанавливают признаки CARRY, DIGIT CARRY и ZERO. Результат выполнения команды помещается в рабочий регистр W. Следующий пример уменьшит SCRATCH на 5.

```
MOVLW   0FFh        ;загрузить 0FFh в регистр W
MOVWF   SCRATCH     ;загрузить содержимое W
                    ; в регистр SCRATCH
SUBLW   05h         ;вычесть 5 из рабочего регистра
MOVWF   SCRATCH     ; записать W в регистр SCRATCH
MOVWF   PORTB       ;записать W в порт B
GOTO    $           ;зациклиться навсегда
```

Светодиоды должны отобразить 11111010.

DECF f,d

INCF f,d

Команда DECF уменьшает заданный регистр на 1, а INCF увеличивает заданный регистр на 1. Результат может быть помещен обратно в заданный регистр (при d=1) либо в рабочий регистр W (при d=0). В результате выполнения этих команд может установиться признак ZERO в регистре STATUS. Вот пример использования этих команд:

```
MOVLW   0FFh        ;загрузить 0FFh в регистр W
MOVWF   SCRATCH     ;загрузить содержимое W
```

```
                    ;в регистр SCRATCH
DECF    SCRATCH  ;уменьшить SCRATCH на 1
```

Этот пример увеличит SCRATCH с 0 до 1.

```
CLRF    SCRATCH         ;очистить SCRATCH
INCF    SCRATCH      ;увеличить SCRATCH на 1
```

IORWF f,d

ANDWF f,d

XORWF f,d

Эти три команды выполняют логические действия ИЛИ, И и ИСКЛЮЧАЮЩЕЕ ИЛИ. Операция логического сложения ИЛИ чаще всего используется для установки отдельных битов в регистрах. Сбрасываются эти биты затем операцией логического умножения И. Когда над одинаковыми битами выполняется операция ИСКЛЮЧАЮЩЕЕ ИЛИ, результат равен 0. Поэтому операция ИСКЛЮЧАЮЩЕЕ ИЛИ часто используется для проверки состояния (установлены или сброшены) определенных бит в регистре. Следующая процедура установит бит 1 в порте B при помощи команды IORWF:

```
MOVLW   B'00000000'  ;загрузить 0h в регистр W
MOVWF   PORTB        ;установить все биты в порте B
MOVLW   B'00000010'  ;установить маску в регистре W
IORWF   PORTB    ;установить биты в порте B
                    ; по маске W
GOTO    $               ;зациклиться навсегда
```

Светодиоды должны показать 00000010. А теперь сбросим 2 бита при помощи команды ANDWF:

```
MOVLW   B'11111111'  ;загрузить 0FFh в регистр W
MOVWF   PORTB        ;установить все биты в порте B
MOVLW   B'00000101'  ;установить маску в регистре W
ANDWF   PORTB  ,1    ;очистить биты в порте B
                    ; по маске W
GOTO    $               ;зациклиться навсегда
```

Светодиоды должны показать 00000101. Поэкспериментируйте с начальными значениями в порту B и со значениями маски.

Предположим, что мы использовали регистр SCRATCH и хотим знать, равен ли он значению 04h. Это удобный случай использовать команду XORWF:

```
MOVLW   04h         ;загрузить 04h в регистр W
MOVWF   SCRATCH     ;загрузить регистр W в SCRATCH
XORWF   SCRATCH,W   ;проверить равенство W и SCRATCH
MOVWF   PORTB       ;записать W в порт B
GOTO    $           ;зациклиться навсегда
```

Поскольку SCRATCH и W равны, результат выполнения операции XORWF равен нулю (все светодиоды потухли). В регистре STATUS установится бит ZERO, который реальная программа затем может проверить и обработать.

IORLW k

ANDLW k

XORLW k

Эти три команды выполняют те же действия, что и их вышеописанные аналоги, за тем исключением, что операция производится между рабочим регистром W и маской, заданной в команде. Результат выполнения команды помещается в рабочий регистр W. Например:

```
MOVLW   0FFh  ;загрузить 0FFh в регистр W
ANDLW   040h  ;оставить 6-й бит
MOVWF   PORTB      ;Светодиоды покажут 01000000.

MOVLW   10h     ;загрузить 10h в регистр W
IORLW   09h  ;установить 0-й и 3-й биты
MOVWF   PORTB      ;Светодиоды покажут 01000000.
MOVLW   B'00100000'     ;загрузить 40h в регистр W
XORLW   B'11111111'     ;проинвертировать W
MOVWF   PORTB      ;Светодиоды покажут 01000000.
```

MOVF f,d

Эта команда в основном используется для пересылки регистра в рабочий регистр W (d=0). Если же установить d=1, то эта команда загрузит регистр сам в себя, но при этом бит ZERO в регистре STATUS установится в соответствии с содержимым регистра. Например, мы хотим загрузить в регистр SCRATCH 0Fh, а потом загрузить регистр SCRATCH в рабочий регистр W.

```
MOVLW   0Fh       ;загрузить 0Fh в рабочий регистр W
MOVWF   SCRATCH   ;загрузить регистр W в SCRATCH
CLRW              ;очистить W
MOVF    SCRATCH,W ;загрузить SCRATCH в регистр W
```

Если в процессе выполнения программы мы хотим проверить регистр SCRATCH на ноль, мы можем выполнить следующую команду:

```
MOVF    SCRATCH
```

Бит ZERO регистра STATUS будет установлен, если условие будет выполнено (SCRATCH = 0h).

COMF f,d

Эта команда инвертирует любой заданный регистр. При d=0 результат заносится в рабочий регистр W, а при d=1 инвертируется содер-

жимое заданного регистра. В качестве примера проинвертируем значение 01010101:

```
MOVLW   B'01010101' ;загрузить 01010101 в регистр W
MOVWF   SCRATCH     ;загрузить регистр W в SCRATCH
COMF    SCRATCH,W   ;инвертировать SCRATCH
MOVWF   PORTB       ;записать W в порт B
GOTO    $           ;зациклиться навсегда
```

Светодиоды покажут 10101010.

2. Цель работы

Целью работы является приобретение навыков в проектировании простейших программ для микроконтроллеров PIC16 и эмуляции выполнения этих программ в системах автоматизированного проектирования.

3. Порядок выполнения работы

3.1 Написать программу, которая решает вычислительную задачу, согласно выданному варианту.

3.2 Промоделировать работу микроконтроллера с разработанной программой на эмуляторе микроконтроллера.

3.3 Составить контрольные примеры и вычислить при помощи ЭВМ контрольные результаты. По результатам проверить правильность выполнения разработанной программы.

4. Содержание отчета:

Титульный лист с названием и номером работы, а также с фамилией исполнителя.

Цель работы.

Задание на работу.

Контрольные примеры, результат расчета контрольных примеров на ЭВМ и на эмуляторе микроконтроллера.

Листинг программы.

Выводы.

5 Контрольные вопросы

5.1 Напишите подпрограмму, которая при частоте работы микроконтроллера 4МГц, выполнялась бы ровно 10 микросекунд.

5.2 Напишите программу, которая настраивает первые четыре вывода порта B на вывод информации, а остальные четыре вывода на ввод.

5.3 Напишите программу, которая вводит в рабочий регистр W число двадцать.

5.4 Напишите программу, которая складывает два числа.

5.5 Напишите программу, которая вычитает два числа.

5.6 Напишите программу, которая сбрасывает три младших бита числа в ноль.

5.7 Напишите программу, которая устанавливает три старших бита числа в единицу.

5.8 Напишите программу, которая инвертирует три старших бита числа.

Самостоятельная работа №2

Система команд микроконтроллеров Microchip. Подпрограммы, циклы, ветвления.

1 Краткие теоретические сведения

Продолжим разбирать основные команды микроконтроллеров Microchip.

DECFSZ f,d

INCFSZ f,d

При d=1 команда DECFSZ уменьшает на единицу, а INCFZ увеличивает на единицу заданный регистр и пропускает следующую команду, если регистр стал равен нулю. При d=0 результат записывается в регистр W и следующая команда пропускается, если рабочий регистр W стал равным нулю. Эти команды используются для формирования временных задержек, счетчиков, циклов и т.д.

Вот типичный пример использования цикла:

```
START
MOVLW   0FFh            ;загрузить FFh в регистр W
MOVWF   SCRATCH         ;загрузить регистр W в SCRATCH
LOOP
DECFSZ  SCRATCH,1       ;уменьшать SCRATCH на 1
GOTO    LOOP            ;и переходить обратно, пока
                        ; не станет = 0
MOVF    DIGIT,W         ;загрузить регистр DIGIT в W
MOVWF   PORTB           ;вывести на светодиоды
DECF    DIGIT,1         ;уменьшить регистр DIGIT на 1
GOTO    START           ;перейти на начало
```

В результате светодиоды будут мигать с различной частотой. Светодиод младшего разряда будет мигать чаще всего, а светодиод старшего разряда реже всего. При тактовой частоте 4 МГц частота миганий светодиода старшего разряда будет примерно 8 Гц, а каждый следующий будет мигать вдвое чаще. Теперь разберемся, как это у нас получилось. Команда DECFSZ здесь работает в цикле задержки, состоящем из двух команд - DECFSZ и GOTO LOOP. Поскольку мы предварительно загрузили в регистр SCRATCH значение 0FFh, этот цикл выполнится 255 раз, пока SCRATCH не станет равным нулю. При тактовой частоте 4 МГц это дает задержку 1 мксек/команду * 2 команды * 255 = 510 мксек. В регистр DIGIT мы предварительно ничего не записывали, поэтому там могло находиться любое значение, которое и выводится на светодиоды на первом проходе. Затем регистр DIGIT уменьшается на 1 и цикл по-

вторяется сначала. В результате регистр DIGIT перебирает все значения за 256 циклов, т.е. за примерно за 130 мсек. Тот же код можно использовать и с командой INCFSZ, заменив загружаемое в регистр SCRATCH значение с FFh на 0h. Светодиоды будут мигать точно так же и если заменить команду DECF на команду INCF.

SWAPF f,d

Эта команда меняет местами полубайты в любом регистре. Как и для других команд, при d=0 результат записывается в рабочий регистр W, а при d=1 остается в регистре. Вот простой пример использования этой команды:

```
MOVLW   B'00001111'   ;загрузить 0Fh в регистр W
MOVWF   SCRATCH       ;загрузить регистр W в SCRATCH
SWAPF   SCRATCH,0     ;поменять полубайты
```

Светодиоды покажут 11110000.

RRF f,d

RLF f,d

В ассемблере PIC имеется две команды сдвига - сдвиг вправо через бит CARRY любого регистра RRF и сдвиг влево через бит CARRY любого регистра RLF. Как и для других команд, при d=0 результат сдвига записывается в регистр W, а при d=1 остается в регистре. Инструкции сдвига используются для выполнения операций умножения и деления, для последовательной передачи данных и для других целей. Во всех случаях бит, сдвигаемый из 8-битного регистра, записывается в бит CARRY в регистре STATUS, а бит CARRY записывается в другой конец регистра, в зависимости от направления сдвига. При сдвиге влево RLF CARRY записывается в младший бит регистра, а при сдвиге вправо RRF CARRY записывается в старший бит регистра.

```
CLRF    STATUS        ;очистить регистр STATUS
MOVLW   0FFh          ;загрузить 0FFh в регистр W
MOVWF   SCRATCH       ;загрузить регистр W в SCRATCH
RRF     SCRATCH,0     ;сдвинуть вправо
```

Светодиоды должны показать 01111111, поскольку CARRY загрузился в старший бит. Теперь сдвинем влево:

```
CLRF    STATUS       ;очистить регистр STATUS
MOVLW   0FFH         ;загрузить 0FFh в регистр W
MOVWF   SCRATCH      ;загрузить регистр W в SCRATCH
RLF     SCRATCH,1    ;сдвинуть влево
```

Светодиоды должны показать 11111110.

BCF f,b
BSF f,b

Команды очистки бита BCF и установки бита BSF используются для работы с отдельными битами в регистрах. Параметр b означает номер бита, с которым производится операция, и может принимать значения от 0 до 7. Попробуем включить светодиод, используя команду BCF:

```
MOVLW   0h              ;загрузить 0h в регистр W
MOVWF   PORTB           ;выключить светодиоды
BSF     PORTB ,7        ;установить бит 7 в порте B
GOTO    $               ;зациклиться навсегда
```

В результате загорится светодиод, соответствующий биту 7. Вспомните, мы делали аналогичные вещи при помощи использования маски и команды ANDWF. Разница в том, что команды ANDWF и IORWF требуют предварительного формирования маски и хранения ее в каком-либо регистре, но в то же время способны одновременно установить или очистить несколько бит. Команды же BCF и BSF оперируют только с одним битом. Кроме того, команды BCF и BSF не изменяют регистр состояния STATUS, поэтому они часто используются в тех случаях, когда не требуется последующая проверка регистра состояния.

BTFSC f,b
BTFSS f,b

Команды условных переходов BTFSC и BTFSS проверяют состояние заданного бита в любом регистре и в зависимости от результата пропускают или нет следующую команду. Команда BTFSC пропускает команду, если заданный бит сброшен, а команда BTFSS - если установлен. Вот простой пример:

```
MOVLW   0h              ;загрузить 0h в регистр W
MOVWF   DATAPORT        ;выключить светодиоды
LOOP
BTFSS   PORTA,0     ;проверить бит 0 в PORTA
GOTO    LOOP        ;ждать, пока бит 0 не установится
BSF     DATAPORT,7   ;включить светодиод
GOTO    $            ;зациклиться навсегда
```

В этом примере проверяется разряд 0 порта A (вывод 17 микросхемы) и, если этот вывод установлен в высокий уровень, включается светодиод. Попробуйте заменить BTFSS на BTFSC в этом примере. Свето-

диод будет включаться, когда разряд 0 порта A установится в низкий уровень. Ранее мы упоминали о возможности проверки битов состояния в регистре STATUS. Это также делается при помощи команд BTFSS и BTFSC:

```
                    ;Проверка бита CARRY
BTFSS   STATUS,C ;если C установлен, пропустить GOTO
GOTO    WHERE_EVER
```

Аналогично проверяется бит ZERO:

```
;Проверка бита ZERO
BTFSS   STATUS,Z ;если Z установлен, пропустить GOTO
GOTO    WHERE_EVER
```

Проверка битов Z и C в регистре STATUS позволяет определить факт равенства содержимого двух регистров (определяется по биту Z), а также того, что содержимое одного регистра больше или меньше содержимого другого (определяется по биту C).

CALL k
RETURN

Эти две команды предназначены для работы с подпрограммами. Команда CALL используется для перехода на подпрограмму по адресу, задаваемому в команде, а команда RETURN - для возврата из подпрограммы. Обе команды выполняются за 2 цикла. Адрес, на котором находилась команда CALL, запоминается в специально организованных регистрах, называемых стеком. Эти регистры недоступны для обращений и используются только при вызовах подпрограмм и возвратах. Глубина стека, т.е. число специальных регистров - 8. Поэтому из основной программы можно сделать не более 8 вложенных вызовов подпрограмм. После возврата из подпрограммы выполнение продолжается со следующей после CALL команды. Регистр W и регистр STATUS при вызове подпрограммы не сохраняются, поэтому, если необходимо, их можно сохранить в отдельных ячейках памяти. Вот простой пример использования подпрограммы:

```
START
BCF     DATAPORT,7      ;выключить светодиод
CALL    TURNON          ;вызвать подпрограмму
GOTO    START           ;перейти на начало
```

```
TURNON
BSF     DATAPORT,7      ;включить светодиод
RETURN                  ;вернуться из подпрограммы
```

В результате светодиод будет мигать с частотой около 150 кГц.

RETLW k

RETFIE

Существуют еще две команды, предназначенные для возврата из подпрограмм. Команда RETLW возвращает в рабочем регистре W константу, заданную в этой команде, а команда RETFIE разрешает прерывания. Команда RETLW часто используется для создания таблиц значений. Пусть в рабочем регистре W содержится смещение от начала таблицы. Тогда получить нужный элемент можно следующей процедурой:

```
MOVLW   02h         ;задать смещение
CALL    SHOWSYM     ;вызвать подпрограмму
MOVWF   PORTB       ;вывести элемент таблицы в порт B
GOTO    $           ;зациклиться навсегда
SHOWSYM
ADDWF   PC        ;вычислить смещение в таблице
                  ;со счетчиком команд
RETLW   0AAh      ;1-й элемент таблицы
RETLW   0BBh      ;2-й элемент таблицы
RETLW   0CCh      ;3-й элемент таблицы
```

Светодиоды должны отобразить 10111011 (0BBh).

СПЕЦИАЛЬНЫЕ КОМАНДЫ

Осталось упомянуть о двух специальных командах - **CLRWDT** и **SLEEP**. Команда CLRWDT предназначена для сброса сторожевого таймера, назначение которого мы уже обсуждали. Эта команда должна присутствовать в таких участках программы, чтобы время выполнения программы между двумя соседними командами CLRWDT не превышало времени срабатывания сторожевого таймера.

Команда SLEEP предназначена для перевода процессора в режим пониженного энергопотребления. После выполнения этой команды тактовый генератор процессора выключается и обратно в рабочий режим процессор можно перевести либо по входу сброса, либо по срабатыванию сторожевого таймера, либо по прерыванию.

2 Цель работы

Целью работы является усовершенствование навыков в написании программ для микроконтроллеров PIC16.

3 Порядок выполнения работы

3.1 Написать программу, которая решает задачу, согласно выданному варианту.

3.2 Промоделировать работу микроконтроллера с разработанной программой на эмуляторе микроконтроллера.

3.3 Составить контрольные примеры и вычислить при помощи ЭВМ контрольные результаты. По результатам проверить правильность выполнения разработанной программы.

4 Содержание отчета:

Титульный лист с названием и номером работы, а также с фамилией исполнителя.

Цель работы.

Задание на работу.

Контрольные примеры, результат расчета контрольных примеров на ЭВМ и на эмуляторе микроконтроллера.

Листинг программы.

Выводы.

5 Контрольные вопросы

5.1 Напишите подпрограмму умножения двух чисел, используя команды цикла DECFSZ f,d или INCFSZ f,d и команду сложения.

5.2 Напишите подпрограмму целочисленного деления числа на четыре, используя команды сдвига.

5.3 Как программно определить остаток от целочисленного деления числа на два?

5.4 Напишите подпрограммы очистки двух младших битов в регистре с использованием битовой маски и логических команд, и с помощью команды BCF. Какая из этих подпрограмм эффективнее?

5.5 Напишите подпрограмму проверки равенства содержимого двух регистров.

5.6 Напишите подпрограмму проверки того, что содержимое регистра reg1 больше содержимого регистра reg2.

5.7 Напишите подпрограмму проверки того, что содержимое регистра reg1 меньше содержимого регистра reg2.

Самостоятельная работа №3
Видеовывод в микроконтроллерных устройствах. Реализация знакогенератора.

1 Краткие теоретические сведения

Знакогенератор выполняет функцию замены кода символа его графическим изображением и выводом этого изображения на средство отображения информации (в нашем случае – ЖКИ). Графические изображения символов, как правило, хранятся в статических массивах. В микроконтроллерах Microchip (без использования дополнительных микросхем) статические массивы можно хранить в памяти программ и в EEPROM.

Для организации статического массива в EEPROM в микроконтроллере PIC16F877 достаточно объявить массив байт по адресу 0x2100 в листинге. Т.е. для сохранения в EEPROM массива из элементов [0,1,2,3,4,5] необходимо написать следующий кусок кода (как правило, в конце программы перед директивой END):

```
org 0x2100                  ; Инициализация EEPROM
de 0x0,0x1,0x2,0x3,0x4,0x5    ;массив
```

Заметим, что существуют процедуры записи в EEPROM во время выполнения программы.

Для чтения из EEPROM памяти необходимо только записать адрес в регистр EEADR и сбросить бит EEPGD в '0'. После установки в '1' бита RD данные будут доступны в регистре EEDATA на следующем машинном цикле. Данные в регистре EEDATA сохраняются до выполнения следующей операции чтения или записи в EEDATA.

Рекомендованная последовательность действий при чтении из EEPROM памяти данных:

1. Записать адрес в регистр EEADR. Проверьте, что записанный адрес корректен для данного типа микроконтроллера.

2. Сбросить в '0' бит EEPGD для обращения к EEPROM памяти данных.

3. Инициализировать операцию чтения установкой бита RD в '1'.

4. Прочитать данные из регистра EEDATA.

Следующая процедура читает данные из EEPROM:

```
;-------------------------------------------------
;Функция работы с EEPROM
;-------------------------------------------------
;Вход - регистр eepcount – адрес ячейки
```

```
;Выход - значение eeprom в W
;
geteeprom
    MOVF eepcount,W  ; Записать адрес
    BSF STATUS,RP1   ;
    BCF STATUS,RP0   ; Выбрать банк 2
    MOVWF EEADR      ; ячейки
    BSF STATUS,RP0   ; Выбрать банк 3
    BCF EECON1,EEPGD     ; Выбрать EEPROM память
    BSF EECON1,RD        ; Инициализировать чтение
    BCF STATUS,RP0   ; Выбрать банк 2
    MOVF EEDATA,W        ; W = EEDATA
    BCF STATUS,RP1   ;
 return
```

Для организации статического массива в памяти программ в микроконтроллере PIC16F877 существует несколько способов. Во-первых, для чтения и записи 14-битных значений памяти программ можно использовать процедуры, основанные на работе с регистрами EEADRH:EEADR и EEDATH:EEDATA (см. документацию).

А для организации статических 8-битных массивов можно воспользоваться процедурой, основанной на команде RETLW.

Для этого объявляют подпрограмму следующего вида:

```
str0              ;имя подпрограммы-массива
    retlw "P"         ;здесь могут быть любые
    retlw "A"         ;байтовые данные
    retlw "Б"
    retlw "O"
    retlw "T"
    retlw "A"
    retlw 0
```

А читают данные из такой подпрограммы с помощью процедуры:

```
;---------------------------------------
;Процедура доступа к данным в памяти программ
;Использует регистры tptrh и tptrl - указатель на
;объявленный статический массив
;вызов:
;movlw   str0 / 0x100     ;Настройка указателей
;movwf   tptrh            ;Старший байт
;movlw   str0 % 0x100
;movwf   tptrl            ;Младший байт
;call    getdata          ;Нулевой элемент - в W
;.........................
;incf    tptrl       ;Переходим к следующему элементу
```

```
;call    getdata                ;Второй элемент – в W

; код подпрограммы:
getdata
   movf    tptrh,W    ; Настройка  PCLATH для прыжка
   movwf   PCLATH
   movf    tptrl,W    ; Младший адрес указателя
   movwf   PCL        ;Пересылаем в PCL, совершая
                  ;тем самым переход на инструкцию
                  ; retlw
```

Разработаем изображение символа «А» размером 5x8. Для этого составим следующую сетку:

		1	1	
	1			1
	1			1
	1	1	1	1
	1			1
	1			1
	1			1
0	252	18	18	252

Где пустые значения соответствуют нулям, единицы – закрашенным точкам, в нижней строке вычислен (согласно правилам отображения данных в ЖКИ МТ-6116) десятичный код столбца, которому соответствуют данные нули и единицы. Т.е. если в память ЖКИ переслать последовательно цифры 0, 252, 18, 18, 252, то на экране нарисуется символ «А». Таким образом, изображение символа «А» можно записать в EEPROM:

```
org 0x2100              ; Инициализация EEPROM
de 0,.252,.18,.18,.252
```

Или изображение можно записать в память программ:

```
Znaki          ;имя подпрограммы-массива
    retlw 0
    retlw .252
    retlw .18
    retlw .18
    retlw .252
```

2 Цель работы: научиться хранить и использовать статические массивы информации (калибровочная информация, знакогенераторы и т.п.) в памяти EEPROM и в памяти программ; реализовывать на низком уровне вывод символьной информации на ЖКИ.

3 Порядок выполнения работы

3.1 Решить практическую задачу по программированию внешнего контроллера, обеспечивающего графический вывод на LCD дисплей, согласно своему варианту.

3.2 Проверить работоспособность разработанной программы в системе Proteus.

3.3 Проверить работоспособность разработанной программы на лабораторном стенде.

4 Содержание отчета:

Титульный лист с названием и номером работы, а также с фамилией исполнителя

Цель работы

Задание на работу

Контрольные примеры, результат выполнения микроконтроллером контрольных примеров.

Листинг программы.

Выводы.

5 Контрольные вопросы

5.1 Какие значения необходимо переслать в видеопамять, чтобы получить изображение буквы «Б»?

5.2 Напишите программу для хранения в EEPROM памяти изображения буквы «Б».

5.3 Напишите программу для хранения в памяти программ изображения буквы «Б».

5.4 Изображения букв в EEPROM памяти хранятся последовательно, причем каждая буква занимает 5 байт. Напишите подпрограмму, которая по коду буквы пересылает в видеопамять 5 байт изображения буквы.

5.5 Напишите алгоритм вывода изображения содержимого регистра в десятичном виде.

5.6 Напишите алгоритм вывода изображения содержимого регистра в двоичном виде.

5.7 Напишите алгоритм вывода изображения отрицательного содержимого регистра в десятичном виде.

5.8 Какова минимальная ширина букв в пикселах для реализации русского знакогенератора?

5.9 Сколько минимально необходимо байт памяти для реализации русского знакогенератора с постоянным количеством байт изображения на символ?

5.10 Как можно построить знакогенератор с переменным количеством байт изображения на символ?

5.11 Сколько минимально необходимо байт памяти для реализации русского знакогенератора с переменным количеством байт изображения на символ?

Самостоятельная работа №4
Программирование генератора аналоговых сигналов произвольной формы на основе таймера.

1 Краткие теоретические сведения

Программирование генератора аналоговых сигналов необходимо начинать с процедур обмена с ЦАП. У каждой микросхемы и схемы подключения будут свои процедуры обмена. Написание процедур обмена микроконтроллера с ЦАП, как правило, не сложнее, чем с ЖКИ и подобными устройствами.

При готовых процедурах обмена с ЦАП программирование вывода аналоговых сигналов не представляет собой сложной задачи – достаточно загружать цифровые значения, соответствующие отсчетам генерируемого сигнала, в регистр ЦАП. Одной из проблем, которую придется решать разработчику такого генератора, является точное задание частоты дискретизации сигнала.

Для задания временных интервалов дискретизации можно использовать программные задержки, как это было показано в предыдущих работах, т.е. интервал дискретизации делился бы на две основные части – часть вывода информации в ЦАП и часть программной задержки. Понятно, что для обеспечения заданного интервала дискретизации величина программной задержки напрямую зависит от длительности процедуры вывода информации в ЦАП. При изменении микросхемы ЦАП или, в общем, при изменении процедуры вывода информации в ЦАП, величину программной задержки каждый раз необходимо было бы изменять. Поэтому более эффективный путь для формирования точной частоты дискретизации сигнала, который лишен этих недостатков – это использование таймеров.

В микроконтроллере PIC16F877 есть три таймера – TMR0, TMR1, TMR2. Для примера разберем механизмы работы с таймером TMR0.

TMR0 с точки зрения программиста – это просто 8-ми разрядный регистр, который может увеличивать свое значение по определенному фронту внешнего для микроконтроллера сигнала (и тогда он работает просто как счетчик внешних импульсов), или может увеличивать свое значение по сигналу от внутреннего тактового генератора (и тогда он работает как таймер или формирователь временных интервалов).

TMR0 – таймер/счетчик, имеет следующие особенности:

· 8-разрядный таймер/счетчик;

· Возможность чтения и записи текущего значения счетчика;

· 8-разрядный программируемый предделитель;
· Внутренний или внешний источник тактового сигнала;
· Выбор активного фронта внешнего тактового сигнала;
· Прерывания при переполнении (переход от FFh к 00h).

Когда бит T0CS сброшен в ‘0’ (OPTION_REG<5>), TMR0 работает от внутреннего тактового сигнала. Приращение счетчика TMR0 происходит в каждом машинном цикле, т.е. с частотой Fosc/4 (если предделитель отключен). После записи в регистр TMR0 приращение счетчика запрещено два следующих цикла. Пользователь должен скорректировать эту задержку перед записью нового значения в TMR0.

Если бит T0CS установлен в ‘1’ (OPTION_REG<5>), TMR0 работает от внешнего источника тактового сигнала с входа RA4/T0CKI. Активный фронт внешнего тактового сигнала выбирается битом T0SE в регистре OPTION_REG<4> (T0SE=0 – активным является передний фронт сигнала).

Предделитель может быть включен перед сторожевым таймером WDT или перед таймером TMR0, в зависимости от состояния бита PSA (OPTION_REG<3>). Использование предделителя пред TMR0 означает, что WDT работает без предделителя, и наоборот.

Коэффициент деления предделителя определяется битами PSA и PS2:PS0 в регистре OPTION_REG<3:0>. Если предделитель включен перед TMR0, любые команды записи в TMR0 (например, CLRF 1, MOVWF 1, BSF 1,x и т.д.) сбрасывают предделитель. Когда предделитель подключен к WDT, команда CLRWDT сбросит предделитель вместе с WDT. Предделитель также очищается при сбросе микроконтроллера. Предделитель недоступен для чтения/записи.

На рисунке 4.1 показаны основные конфигурационные биты регистра OPTION_REG, связанные с таймером TMR0.

Регистр OPTION_REG (адрес 81h или 181h)

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**
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

бит 7: **-RBPU**:

бит 6: **INTEDG**:

бит 5: **T0CS**: Выбор тактового сигнала для TMR0
1 = внешний тактовый сигнал с вывода RA4/T0CKI
0 = внутренний тактовый сигнал CLKOUT

бит 4: **T0SE**: Выбор фронта приращения TMR0 при внешнем тактовом сигнале
1 = приращение по заднему фронту сигнала (с высокого к низкому уровню) на выводе RA4/T0CKI
0 = приращение по переднему фронту сигнала (с низкого к высокому уровню) на выводе RA4/T0CKI

бит 3: **PSA**: Выбор включения предделителя
1 = предделитель включен перед WDT
0 = предделитель включен перед TMR0

биты 2-0: **PS2: PS0**: Установка коэффициента деления предделителя

Значение	Для TMR0	Для WDT
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

Рисунок 4.1 - Основные конфигурационные биты регистра OPTION_REG, связанные с таймером TMR0

При переполнении счетчика TMR0, т.е. при переходе его значения от FFh к 00h возникают прерывания. При возникновении прерывания устанавливается в ‘1’ бит T0IF(INTCON<2>). Само прерывание может быть разрешено/запрещено установкой/сбросом бита T0IE (INTCON<5>). Флаг прерывания от TMR0 T0IF (INTCON<2>) должен быть сброшен в подпрограмме обработки прерываний. В SLEEP режиме микроконтроллера модуль TMR0 выключен и не может генерировать прерывания.

Даже при запрещенных прерываниях от таймера TMR0 при переполнении счетчика устанавливается в ‘1’ бит T0IF(INTCON<2>). Поэтому факт переполнения счетчика можно установить путем опроса этого бита.

Суммируя все вышесказанное, напишем процедуру формирования относительно больших временных задержек с использованием таймера TMR0.

В начале программы необходимо проинициализировать конфигурацию таймера:

```
;-----------------------------------------------
;Процедура инициализации таймера0
;Версия для работы без прерываний!
;прерывания - запрещены, срабатывание определяем
;по биту T0IF в INTCON
;установим предделитель к таймеру 0
;Настроим предделитель на деление 1:256
InitTmr0
    bcf INTCON,T0IE  ;Запретим прерывания
                     ;от таймера 0
    bsf STATUS,RP0   ;OPTION_REG - в первой странице
    bcf OPTION_REG,5 ;0 = внутренний тактовый
                     ; сигнал CLKOUT
    bcf OPTION_REG,3 ;0 = предделитель включен
                     ;перед TMR0
    movlw 0x7        ;установим три младших бита -
iorwf OPTION_REG     ; Настроим предделитель
                     ;на деление 1:256
    bcf STATUS, RP0
    clrf TMR0            ;очистим TMR0
    return
```

Теперь временной интервал между двумя срабатываниями таймера можно определить, например, по опросу бита T0IF(INTCON<2>). Ниже приведен фрагмент программы, который формирует задержку в 3.6 сек при частоте кварца у микроконтроллера в 20 МГц. Для этого используется дополнительный регистр tmr0count, который дополнительно делит формируемую таймером частоту в 76.3 Гц еще на 256. Если необходима частота больше, чем 0.3 Гц, но меньше чем 76.3 Гц, то необходимо инициализировать регистр tmr0count ненулевым значением. Если необходима частота больше, чем 76.3 Гц, то можно уменьшить значение коэффициента деления предделителя, или после переполнения таймера записать в него (в регистр TMR0) ненулевое значение.

```
mainloop         ;Основной цикл программы

;Ждем срабатывания таймера 0
;таймер настроен на 20МГц/4/256(предделитель)/256
;(регистр таймера)=76.3 Гц
;Вывод информации будет осуществляться с частотой
; 76.3Гц/tmr0count=76.3/256=0.3 Гц
; (или раз за 3,6 сек)
;В протеусе частота процентов на 20
```

```
;оказывается меньшей! -
;проверено на эксперименте!

    clrw tmr0count        ;Максимальное число в
                          ;tmr0count (256)
    ;……….
    ;тут можно выводить информацию в ЦАП и т.д.
;……….

waitmr0 btfss INTCON,T0IF
    goto waitmr0      ;Опрос бита T0IF
    bcf INTCON,T0IF   ;программный сброс бита-
                      ;признака переполнения
    decfsz tmr0count ;Уменьшаем наш регистр
    goto waitmr0

    ; Дождались!

    goto mainloop
```

2 Цель работы: научиться пользоваться таймером и формировать аналоговые сигналы с помощью ЦАП.

3 Порядок выполнения работы

3.1 Решить практическую задачу по программированию генератора аналоговых сигналов произвольной формы, согласно своему варианту.

3.2 Проверить работоспособность разработанной программы в системе Proteus.

3.3 Проверить работоспособность разработанной программы на лабораторном стенде.

4 Содержание отчета:

Титульный лист с названием и номером работы, а также с фамилией исполнителя

Цель работы

Задание на Самостоятельную работу

Осциллограммы выводимых сигналов.

Листинг программы.

Выводы.

5 Контрольные вопросы

5.1 Разработайте алгоритм формирования треугольного сигнала.

5.2 Разработайте алгоритм формирования пилообразного сигнала.

5.3 Разработайте алгоритм формирования синусоидального сигнала.

5.4 Какую максимальную и минимальную задержку можно организовать на определенной частоте работы микроконтроллера при помощи таймера TMR0 без использования дополнительных регистров?

5.5 Как с помощью микроконтроллера можно сформировать сигнал ЭКГ?

5.6 Рассчитайте максимальную частоту дискретизации сигнала, который можно сформировать на основе определенной конкретной микросхемы ЦАП.

5.7 Какие существуют возможные пути повышения максимальной частоты дискретизации сигнала?