Виталий Антонов
kaligraf@yandex.ru

Создание ассемблерных модулей

Как создать перемещаемый код - исследуем вопрос
http://progcode.narod.ru

Содержание:

Введение

Прежде всего, звучит вопрос: «Зачем всё это нужно?»

Отвечу на него в меру своего понимания. Когда вы только начинаете осваивать программирование на ассемблере для PIC микроконтроллеров и делаете свои первые проекты, модули вовсе не нужны, и вполне можно обходиться без них.

Если продолжаете программировать, и счёт вашим проектам идёт уже на десятки, то, возможно, вы начнёте замечать, что часто повторяете одни и те же действия со своим собственным кодом. То есть создаёте новый проект в MpLab, копируете в него пустой шаблон-заготовку для будущей программы, копируете в текст свои готовые программы и подпрограммы, а потом часть программы доделываете уже под конкретные задачи конкретного устройства. Чтобы избежать рутинных операций с копированием готового ассемблерного кода в новый проект, разработано несколько инструментов.

- Макроопределения (макросы)
- Перемещаемые модули
- Библиотеки подпрограмм
- Языки высокого уровня

Модули применяются для облегчения
ПОВТОРНОГО ИСПОЛЬЗОВАНИЯ КОДА

Разработчики MpLab, в частности - ассемблера Mpasm, позаботились о том, чтобы облегчить программистам труд и создали систему, пользуясь которой, возможно создавать библиотеки своих собственных программ и легко подключать их к новым проектам.

В этой статье будет рассмотрен способ оформления и использования модулей ассемблерного кода для микроконтроллеров Microchip среднего семейства.

Создание перемещаемого модуля

При создании единичного проекта переменные задаются в фиксированных регистрах при помощи директив **equ** и **cblock**, а программа и подпрограммы сразу же размещаются в фиксированных адресах при помощи директивы **org**.

При создании перемещаемого кода ситуация другая. Адреса для переменных и адреса размещения модулей кода назначаются автоматически в процессе компиляции проекта.

Это основное преимущество перемещаемого кода – он может располагаться в любых адресах и работать с регистрами в любых банках микроконтроллера (речь идёт о средней серии PIC16Fxxx, в которой присутствует страничная адресация ОЗУ).

Основное преимущество оборачивается и некоторой избыточностью. В перемещаемом коде нужно обеспечивать выбор нужного банка используемых регистров и учесть, что код может оказаться в любых страницах памяти программ. То есть нужно предустанавливать биты **PR0, PR1, IRP** регистра **STATUS** при работе с переменными и регистр **PCLATH** при вызове подпрограмм.

К примеру, имеется две готовых подпрограммы, осуществляющих математические операции, и мы задаёмся целью разместить их в отдельном модуле, чтобы впоследствии его можно было включить в любой проект.

Исходные подпрограммы

Первая подпрограмма преобразует число в диапазоне 0…99 из двоичного кода в десятичный. Вторая делает обратное преобразование и из десятичного вида переводит число в двоичный код.

Подпрограммы сделаны для устройства с двухзначной динамической индикацией для ввода с клавиатуры и вывода на дисплей числа в диапазоне 0…99. Рассчитаны они на работу с нулевым банком регистров и в нулевой странице памяти программ.

Вот эти подпрограммы:

```
; двоичное число(0...99) в аккумуляторе в десятичное(e_rez/d_rez)
; Вход:
; в аккумуляторе преобразуемое число 0...99
; В программе определить ячейки e_rez d_rez
; - dd_preo
;=============================================================
dd_preo    movwf       e_rez          ; загружаем число в ячейку единиц
                                      ; делим число на 10 - получим десятки
d_10       clrf        d_rez
           movlw       0Ah
           bcf         STATUS,C
d_10_cikl  subwf       e_rez,F        ; вычитаем из преобразуемого числа 10
           btfss       STATUS,C
           goto        dd_ed          ; перенос (результат отрицательный),
                                      ; прекращаем вычитание переноса нет
           incf        d_rez,F        ; увеличиваем на 1 регистр десятков
           goto        d_10_cikl
dd_ed      addwf       e_rez,F        ; прибавив 10, получим единицы
           return


; Подпрограмма преобразования числа в двоичный формат
; из двух десятичных разрядов в диапазоне 0...99
; Вход:
; в регистре FSR адрес младшего (старший - следующий адрес)
; Выход
; в аккумуляторе - число в двоичном виде
; - decb_preo
;=============================================================
```

```
decb_preo  incf        FSR,F          ; адрес старшего
           movf        INDF,W
           movwf       temp           ; старший во временную ячейку - умножаем на 10
           bcf         STATUS,C
           rlf         temp,F
           rlf         temp,F
           rlf         temp,F         ; умножили на 8
           addwf       temp,F
           addwf       temp,F         ; умножили на 10
           decf        FSR,F          ; адрес младшего
           movf        INDF,W
           addwf       temp,W
           return
           end
```

Наша задача сформировать из этих подпрограмм перемещаемый ассемблерный модуль. Учесть, что регистры, с которыми будут работать подпрограммы, могут оказаться в любых адресах и банках. Сам код подпрограмм также может быть размещён при компиляции в любых страницах памяти программ.

Чтобы сделать этот код перемещаемым, его придётся переделать и дополнить.

Начало модуля

Необходимо определить однобайтные переменные e_rez, d_rez, temp. Для этого воспользуемся директивой **udata**. Кроме того, чтобы основная программа могла получить доступ к переменным e_rez, d_rez, их необходимо сделать глобальными, то есть применить к ним директиву **global**.

Итак, начало нашего будущего модуля:

```
           global      e_rez, d_rez
           udata
d_rez      res         1
e_rez      res         1
temp       res         1
```

Директива **udata** объявляет (говорит ассемблеру о начале) секцию переменных. Директива **res** резервирует место под переменные, то есть транслятор (ассемблер) выделяет столько байт, сколько указано.

В нашем случае мы объявили три переменные e_rez, d_rez, temp, под каждую из которых будет выделен один байт памяти.

Ассемблер Mpasm может быть использован двумя способами:

- для генерации абсолютного кода, который может быть напрямую исполнен микроконтроллером

- для генерации перемещаемого кода, который может быть объединён с другими отдельно скомпилированными файлами

Мы задействуем его во втором варианте, для объединения нескольких файлов в один проект.

Когда файлов в проекте больше одного, то в дело включается линкер. Ассемблер генерирует из каждого ACM файла объектный файл. Линкер собирает отдельные объектные файлы в цельный проект, пересчитывает адреса размещения кода и адреса меток в коде, а также размещает переменные в RAM области с назначением им конкретных адресов.

Почему модуль кода называется перемещаемым?

Он может быть размещён линкером в любых адресах памяти программ и работать с регистрами в любых адресах памяти данных микроконтроллера.

В объектном файле находится готовый код и информация об используемых данных. Адрес начала этого кода в будущем проекте можем определить заранее или оставить его определение на усмотрение линкера, что обычно и делается. В некоторых случаях необходимо зафиксировать размещение кода.

Директива **code** объявляет начало секции кода и позволяет при необходимости зафиксировать секцию кода в конкретных адресах памяти программ микроконтроллера.

Наращиваем «шапку» нашего перемещаемого модуля.

```
            global      e_rez, d_rez
            udata
e_rez       res         1
d_rez       res         1
temp        res         1

            code
```

Если бы мы хотели разместить начало кода в определённых адресах памяти, например с адреса 0x0000, то пришлось бы сделать запись:

```
            code        0x0000
```

Применяется фиксация адреса в перемещаемом коде в редких случаях. Когда необходимо указать на начало программы, при объявлении начала адреса программы прерывания, размещении загрузчика или таблицы с данными. Фиксирование адреса в перемещаемом коде в других случаях практически не применяется, так как фиксированные адреса могут привести к несовместимости модулей.

Каждую секцию кода можно назвать отдельным именем. Впоследствии, если будет необходимость включить подпрограммы в библиотечный файл, это пригодится. Так и сделаем:

```
            global      e_rez, d_rez
            udata
e_rez       res         1
d_rez       res         1
temp        res         1

bin99dec_seg    code
```

Именование секций, подпрограмм, переменных и других элементов при программировании с возможностью повторного использования кода – отдельная тема. Имя подпрограммы должно отражать её функциональное назначение и способствовать тому, чтобы, всего лишь взглянув на имя подпрограммы, модуля или файла, можно было легко вспомнить и понять функциональное назначение кода. Подробнее с правилами оформления кода и именования меток можно ознакомиться в статье: «MPASM. Как правильно оформлять программы на ассемблере для PIC-контроллеров»

Переделка исходных подпрограмм

Для того чтобы определить, в каких именно банках находятся регистры с переменными, в ассемблере имеется директива **banksel**.

В абсолютном коде заранее известно, в каком банке находятся переменные. В перемещаемом коде необходимо выбрать банк регистров, соответствующий конкретной переменной. Директива **banksel** предустанавливает биты **RP0, RP1** регистра **STATUS** и производит выбор адресации между банками:

Банк 0 - адреса - 0x000…0x07F
Банк 1 - адреса - 0x080…0x0FF
Банк 2 - адреса - 0x100…0x17F
Банк 3 - адреса - 0x180…0x1FF

Предустанавливать биты выбора банка для каждой из переменных, объявленных в одной секции **udata** не требуется. Достаточно выбрать банк единожды для одной секции. Линкер размещает переменные из одной секции в одном и том же банке.

Работая с регистрами **INDF** и **FSR** при использовании косвенной адресации в перемещаемом коде, необходимо определить бит **IRP** регистра **STATUS**. В микроконтроллерах PIC16Fxxx два банка косвенной адресации. Нулевой банк 0x000…0x0FF и первый банк 0x100…0x1FF. Бит **IRP** регистра **STATUS** определяет, к какому именно банку регистров мы обращаемся. Для выбора банка косвенной адресации в перемещаемом коде служит директива **bankisel**.

```
bin99dec_seg code

;=================================================================
; - Bin99Dec
; пробразование двоичного числа в десятичное
; Параметры:
;   w - двоичное число(0...99)
; Результат:
;   e_rez - единицы
;   d_rez - десятки
;=================================================================

Bin99Dec
            banksel     e_rez           ; выставляем биты RP0,RP1 регистра STATUS для выбора
банка
            movwf       e_rez

d_10        clrf        d_rez
            movlw       0Ah
            bcf         STATUS,C

d_10_loop   subwf       e_rez,F         ; делим число на 10 - получим десятки
            btfss       STATUS,C
            goto        dd_ed           ; перенос (результат отрицательный),
                                        ; прекращаем вычитание

            incf        d_rez,F         ; увеличиваем десятки
            goto        d_10_loop

dd_ed       addwf       e_rez,F         ; прибавив 10, получим единицы
            return
;-----------------------------------

dec99bin_seg code

;=================================================================
; - Dec99Bin
; преобразование десятичного числа в двоичное
; Вход:
; в регистре FSR адрес младшего(старший - следующий адрес)
; (бит IRP – номер банка косвенной адресации)
; Результат:
;   W - число в двоичном виде
;=================================================================

Dec99Bin
            incf        FSR,F           ; адрес старшего
            movf        INDF,W

            banksel     temp
            movwf       temp            ; старший во временную ячейку

            bcf         STATUS,C
```

```
            rlf         temp,F
            rlf         temp,F
            addwf       temp,F
            rlf         temp,F        ; умножили на 10

            decf        FSR,F
            movf        INDF,W
            addwf       temp,W        ; прибавили младший
            return
;-----------------------------------
```

Постоянное применение директив **banksel** и **bankisel** создаёт ту самую избыточность перемещаемого кода, которой обычно удаётся избежать при создании кода абсолютного. Перемещаемый код требует несколько больше места в памяти программ контроллера. Это относится в основном к микроконтроллерам средней серии типа PIC16Fxxx. В 18-й серии и более новых микроконтроллерах Microchip большей разрядности ОЗУ устроено по–другому, и перемещаемый код более компактен.

Сборка модуля

Следующий необходимый шаг – объявить подпрограммы Bin99Dec, Dec99Bin глобальными, чтобы основная программа могла их вызвать. Что происходит при объявлении метки глобальной, вы наверное уже догадались. Ассемблер в объектном файле создаёт таблицу меток, к которым разрешён доступ извне. Если основная программа обращается к глобальной метке внешнего модуля, то всё отлично, доступ разрешён. Если же метка глобальной не является, то доступа к ней нет. При попытке обращения к внутренней метке модуля ассемблер выдаст ошибку, что идентификатор неопределен.

Это позволяет использовать одинаковые имена меток в разных модулях одного проекта и обеспечивает разделение регистров между модулями. Ячейки, которые глобальными не являются, используются только внутри модуля и к ним нет доступа из других модулей программы.

Для окончательной сборки модуля остаётся добавить в него строку подключения файла определений для микроконтроллера среднего семейства и завершить всё директивой **end**

```
            include   <p16F628a.inc>  ;подключили файл определений

            global      e_rez, d_rez
            global      Bin99Dec, Dec99Bin

;==================
; Секция данных
;==================

            udata

e_rez       res         1             ; единицы
d_rez       res         1             ; десятки

temp        res         1

;==============
; Секция кода
;==============

bin99dec_seg code

;=============================================================
; - Bin99Dec
; пробразование двоичного числа в десятичное
; Параметры:
;   w - двоичное число(0...99)
; Результат:
;   e_rez - единицы
;   d_rez - десятки
;=============================================================

Bin99Dec
```

```
            banksel     e_rez           ; выставляем биты RP0,RP1 регистра STATUS
                                        ; для выбора банка
            movwf       e_rez

d_10        clrf        d_rez
            movlw       0Ah
            bcf         STATUS,C

d_10_loop   subwf       e_rez,F         ; делим число на 10 - получим десятки
            btfss       STATUS,C
            goto        dd_ed           ; перенос (результат отрицательный),
                                        ; прекращаем вычитание

            incf        d_rez,F         ; увеличиваем десятки
            goto        d_10_loop

dd_ed       addwf       e_rez,F         ; прибавив 10 получим единицы
            return
;-----------------------------------

dec99bin_seg code

;=============================================================
; - Dec99Bin
; преобразование десятичного числа в двоичное
; Вход:
; в регистре FSR адрес младшего(старший - следующий адрес)
; (бит IRP – номер банка косвенной адресации)
; Результат:
;   W - число в двоичном виде
;=============================================================

Dec99Bin
            incf        FSR,F           ; адрес старшего
            movf        INDF,W

            banksel     temp
            movwf       temp            ; старший во временную ячейку

            bcf         STATUS,C
            rlf         temp,F
            rlf         temp,F
            addwf       temp,F
            rlf         temp,F          ; умножили на 10

            decf        FSR,F
            movf        INDF,W
            addwf       temp,W          ; прибавили младший
            return
;-----------------------------------

            end
```

Настройка всех модулей проекта на один контроллер

Если модулей в проекте несколько, то придётся настроить их все на один и тот же тип микроконтроллера. Можно для этого во всех модулях переделать строку:

```
            include    <p16F628a.inc>
```

Чтобы не исправлять эту строку во всех готовых модулях, входящих в проект, вместо неё включаем файл настроек для всех модулей проекта:

```
            include    <hardware_profile.inc>
```

Создаём файл “hardware_profile.inc” и прописываем в нём строки, которые окажутся включенными во все модули проекта:

```
            list        p=16f628a
            include     <p16F628a.inc>
```

Впоследствии, внеся изменения в этот файл, можно внести изменения и перенастроить проект на другой микроконтроллер, не изменяя код в файлах модулей.

Файл модуля примет вид:

```
            #include     <hardware_profile.inc> ; настройка

            global      e_rez, d_rez
            global      Bin99Dec, Dec99Bin

;=================
; Секция данных
;=================

            udata

e_rez       res         1               ; единицы
d_rez       res         1               ; десятки

temp        res         1

;==============
; Секция кода
;==============

bin99dec_seg code

;=============================================================
; - Bin99Dec
; пробразование двоичного числа в десятичное
; Параметры:
;   w - двоичное число(0...99)
; Результат:
;   e_rez - единицы
;   d_rez - десятки
;=============================================================

Bin99Dec
            banksel     e_rez           ; выставляем биты RP0,RP1 регистра STATUS для выбора
банка
            movwf       e_rez

d_10        clrf        d_rez
            movlw       0Ah
            bcf         STATUS,C

d_10_loop   subwf       e_rez,F         ; делим число на 10 - получим десятки
            btfss       STATUS,C
            goto        dd_ed           ; перенос (результат отрицательный), прекращаем
вычитание

            incf        d_rez,F         ; увеличиваем десятки
            goto        d_10_loop

dd_ed       addwf       e_rez,F         ; прибавив 10 получим единицы
            return
;-----------------------------------

dec99bin_seg code

;=============================================================
; - Dec99Bin
; преобразование десятичного числа в двоичное
; Вход:
; в регистре FSR адрес младшего(старший - следующий адрес)
; (бит IRP – номер банка косвенной адресации)
; Результат:
;   W - число в двоичном виде
;=============================================================

Dec99Bin
            incf        FSR,F           ; адрес старшего
            movf        INDF,W

            banksel     temp
            movwf       temp            ; старший во временную ячейку

            bcf         STATUS,C
            rlf         temp,F
            rlf         temp,F
            addwf       temp,F
            rlf         temp,F          ; умножили на 10

            decf        FSR,F
            movf        INDF,W
            addwf       temp,W          ; прибавили младший
            return
;-----------------------------------

            end
```

Подключение перемещаемого ассемблерного модуля к проекту

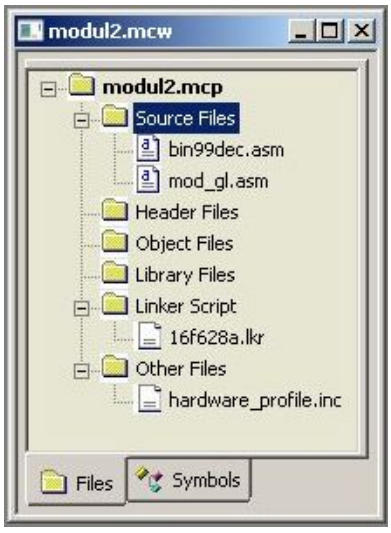
Так как речь идёт о применении в проекте нескольких файлов, то нужно создать ещё один файл – основной файл проекта. Создадим простейшую «пустышку». Для понимания того, как задействовать модули, этого достаточно. Вот код основного файла проекта:

```
;=========================
;  Подключаемые файлы
;=========================
           #include     <hardware_profile.inc> ; настройка

; =====Конфигурация==========
           __CONFIG    _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC & _LVP_OFF & _BODEN_OFF &
_MCLRE_OFF


;================================
; Секция основного кода программы
;================================

reset_seg  code        0x0000
reset      lgoto       start

int_seg    code        0x0004
           nop
           return

start_seg  code
start      nop

;==============================
; Главный цикл программы
; - gl_cikl
;==============================
gl_cikl    nop
           goto        gl_cikl


           end
```

Сохраняем этот код в файл с именем «mod_gl.asm».

Создаём для проекта новую папку и размещаем в неё подготовленные файлы. При сборке и компиляции проекта задействуется линкер, поэтому в папку проекта копируем файл скрипта линкера для микроконтроллера PIC16F628A «16f628a.lkr».

Воспользовавшись Project Wizard, создаем новый проект. Первоначально у нас задействовано четыре файла, и окно проекта выглядит так:



Жмём кнопку «Build All». Всё компилируется.

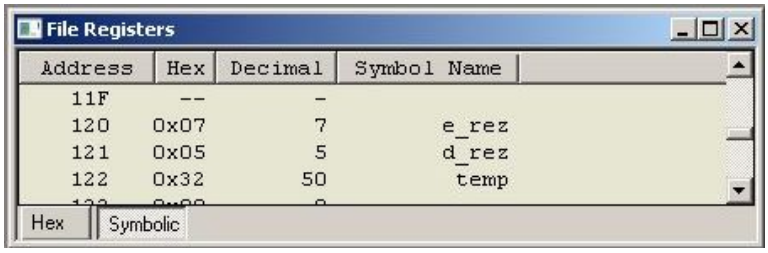
Проверим, что получилось в памяти программ.

```
Line Address Opcode  Label    Disassembly


   1   000     2821 reset     GOTO start
   2   001     3400           RETLW 0
   3   002     3400           RETLW 0
   4   003     3FFF
   5   004     0000           NOP
   6   005     0008           RETURN
   7   006     0A84 Dec99Bin  INCF FSR, F
   8   007     0800           MOVF INDF, W
   9   008     1283           BCF STATUS, 0x5
  10   009     1703           BSF STATUS, 0x6
  11   00A     00A2           MOVWF 0x22
  12   00B     1003           BCF STATUS, 0
  13   00C     0DA2           RLF 0x22, F
  14   00D     0DA2           RLF 0x22, F
  15   00E     07A2           ADDWF 0x22, F
  16   00F     0DA2           RLF 0x22, F
  17   010     0384           DECF FSR, F
  18   011     0800           MOVF INDF, W
  19   012     0722           ADDWF 0x22, W
  20   013     0008           RETURN
  21   014     1283 Bin99Dec  BCF STATUS, 0x5
  22   015     1703           BSF STATUS, 0x6
  23   016     00A0           MOVWF 0x20
  24   017     01A1 d_10      CLRF 0x21
  25   018     300A           MOVLW 0xa
  26   019     1003           BCF STATUS, 0
  27   01A     02A0 d_10_loop SUBWF 0x20, F
  28   01B     1C03           BTFSS STATUS, 0
  29   01C     281F           GOTO dd_ed
  30   01D     0AA1           INCF 0x21, F
  31   01E     281A           GOTO d_10_loop
  32   01F     07A0 dd_ed     ADDWF 0x20, F
  33   020     0008           RETURN
  34   021     0000 start     NOP
  35   022     0000 gl_cikl   NOP
  36   023     2822           GOTO gl_cikl
```

Сразу видим, что секции кода (подпрограммы Bin99Dec, Dec99Bin и основная программа) поменялись местами. На будущее нужно учесть, что при сборке проекта отдельные секции кода могут быть размещены линкером в любых адресах. Ну, да на то они и перемещаемые.

Линкер разместил код модуля с адреса 0x009, то есть сразу же за главным циклом основной программы.

Если заглянуть в файл регистров проекта (в символьном виде), то ячейки e_rez, d_rez, temp обнаружим в адресах 0x120, 0x121, 0x122, то есть во втором банке ОЗУ.



Директива EXTERN
Задействуем модуль в основной программе

Для того чтобы основной файл смог использовать внешние метки необходимо добавить в него строку:

```
        extern      e_rez, d_rez, Bin99Dec, Dec99Bin
```

Директива **extern** даёт понять ассемблеру, что метки, указанные за ней, внешние и находятся в других файлах проекта. Линкеру предстоит найти их при компиляции в других объектных файлах и добавить в таблицу используемых меток основного файла.

В первом примере подпрограммы модуля не вызываются, и обращения к ячейкам e_rez, d_rez из основной программы нет. Добавим несколько обращений к программам модуля, усложним немного основную программу.

```
;========================
;  Подключаемые файлы
;========================
           #include     <hardware_profile.inc> ; настройка

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

           extern      e_rez, d_rez, Bin99Dec, Dec99Bin


; =====Конфигурация=========
           __CONFIG    _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC & _LVP_OFF & _BODEN_OFF &
_MCLRE_OFF


;================================
; Секция основного кода программы
;================================

reset_seg  code        0x0000
reset      lgoto       start

int_seg    code        0x0004
           nop
           return

start_seg  code
start      nop
           movlw       .32
           lcall       Bin99Dec      ; преобразуем число 32 в десятичный вид
           pagesel     $

           bankisel    e_rez         ; выбор банка косвенной адресации - бит IRP
                                     ; регистра STATUS
           movlw       e_rez
           movwf       FSR

           banksel     e_rez         ; выставляем биты RP0,RP1 регистра STATUS
                                     ; для выбора банка
           movlw       .7
           movwf       e_rez
           movlw       .5
           movwf       d_rez         ; загрузили число 57

           lcall       Dec99Bin      ; преобразуем число в e_rez, d_rez в двоичный вид
           pagesel     $

;==============================
; Главный цикл программы
; - gl_cikl
;==============================
gl_cikl    nop
           goto        gl_cikl

           end
```

Так как модуль может оказаться в любой странице памяти программ, при вызове подпрограмм используется встроенный макрос **lcall**. Макрос **lcall** перед вызовом подпрограммы настраивает соответствующие биты регистра **PCLATH** на ту страницу памяти программ, где расположена вызываемая подпрограмма.

При возврате из подпрограммы регистр **PCLATH** восстанавливается макрокомандой **pagesel** и настраивает биты **PCLATH** в соответствие с текущей страницей памяти программ.

```
           pagesel     $
```

Применение этих команд в перемещаемом коде необходимо и также создаёт некоторую избыточность перемещаемого кода по сравнению с кодом абсолютным.

После пошагового прогона скомпилированного проекта в отладчике убеждаемся, что ошибок при преобразовании нет. Модуль готов. Единственное, что может смутить на данном этапе, чтобы основной файл смог обращаться к подпрограммам модуля, в него необходимо добавить строку, в которой определены внешние метки подключенного перемещаемого кода.

```
          extern      e_rez, d_rez, Bin99Dec, Dec99Bin
```

Допустим мы собрали несколько таких модулей, в каждом решили конкретные задачи и разместили по несколько подпрограмм, каждая из которых работает со своими переменными и решает определённую задачу. Например, в одном модуле все подпрограммы, обеспечивающие связь по USART, в другом решается задача измерения температуры с датчика DS18B20, в третьем - подпрограммы работы с внешней памятью, типа 24Cxx, и т.д.

Для каждого из модулей придётся добавить в основной файл проекта строки с определением внешних меток директивой **extern**.

Подключаемый заголовочный файл

Основной способ, применяемый при подключении модуля к проекту, – использование заголовочного файла.

Создаём ещё один файл с названием «bin99dec.inc».

В этом файле прописываем:

```
; bin99dec.inc – заголовочный файл к модулю
; bin99dec.asm – подпрограммы математических преобразований

; Переменные
        extern      d_rez, e_rez ; ячейки для хранения десятичного числа в диапазоне 0…99

; Подпрограммы
        extern      Bin99Dec  ; число в аккумуляторе в десятичный вид,
                              ; результат в d_rez, e_rez
        extern      Dec99Bin  ; десятичное 0…99 в двоичный вид, результат в аккумуляторе
                              ; вход: в FSR адрес младшего (старший - следующий адрес)
```

В файле фактически дублируется строка:

```
          extern      e_rez, d_rez, Bin99Dec, Dec99Bin
```

Комментарии позволят разобраться в функциональном назначении модуля, не заглядывая в код самого модуля, только лишь заглянув в заголовочный файл. Кроме того, в заголовочный файл можно включить макроопределения (макросы) и константы, необходимые для работы с модулем.

Создав файл «bin99dec.inc», подключаем его к основному проекту при помощи директивы **include**.

Основной файл проекта в этом случае будет выглядеть так:

```
;=======================
;  Подключаемые файлы
;=======================
          #include     <hardware_profile.inc> ; настройка
          #include     <bin99dec.inc> ; заголовок к модулю bin_dec преобразований

; =====Конфигурация=========
          __CONFIG     _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC & _LVP_OFF & _BODEN_OFF &
```

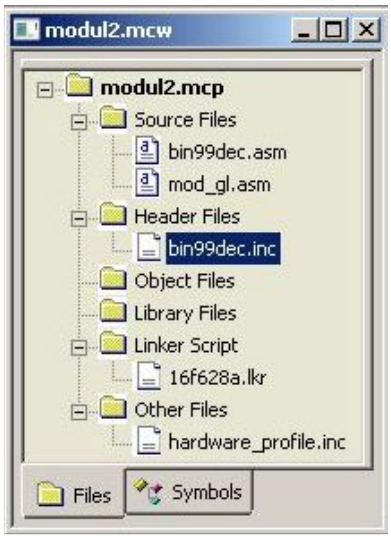
```
_MCLRE_OFF


;================================
; Секция основного кода программы
;================================

reset_seg  code         0x0000
reset      lgoto        start

int_seg    code         0x0004
           nop
           return

start_seg  code
start      nop
           movlw        .32
           lcall        Bin99Dec      ; преобразуем число 32 в десятичный вид
           pagesel      $

           bankisel     e_rez         ; выбор банка косвенной адресации - бит IRP
                                      ; регистра STATUS
           movlw        e_rez
           movwf        FSR

           banksel      e_rez         ; выставляем биты RP0,RP1 регистра STATUS
                                      ; для выбора банка
           movlw        .7
           movwf        e_rez
           movlw        .5
           movwf        d_rez         ; загрузили число 57

           lcall        Dec99Bin      ; преобразуем число в e_rez, d_rez в двоичный вид
           pagesel      $

;==============================
; Главный цикл программы
; - gl_cikl
;==============================
gl_cikl    nop
           goto         gl_cikl

           end
```

При компиляции, если модуль подключен к проекту с использованием заголовочного файла, получается тот же результат, что и со строкой объявления внешних меток.

```
           extern       e_rez, d_rez, Bin99Dec, Dec99Bin
```

Директива **include** работает по тому же принципу, что и макрос, но в приложении к целому файлу, являясь удобным способом подстановки готового кода всего лишь одной строкой.

В окно основного проекта можно добавить файл заголовка, в пункт «Header Files», чтобы можно было одним кликом заглянуть в него при необходимости:



Блокировка повторного включения заголовочных файлов

Когда модули становятся основным способом программирования, могут возникнуть довольно сложные ситуации, когда один модуль использует подпрограммы другого модуля. Возможно включение в заголовочный файл одного модуля заголовочного файла другого модуля и повтор включенных заголовков. Если произойдёт дублированное включение одного и того же заголовка, то неизбежна ошибка при компиляции:

«Дублирование меток».

Чтобы этого избежать, применяется блокировка повторного включения файла заголовка в один и тот же файл. Для блокировки повторного включения применяются директивы условной компиляции:

```
            #ifndef    BIN99DEC_H     ; Блокируем повторное включение заголовка
            #define    BIN99DEC_H

; Здесь размещается код заголовка

            #endif
```

С помощью этих директив исключается повторное включение заголовка в один тот же файл проекта. Заголовочный файл в нашем примере примет такой вид:

```
            #ifndef    BIN99DEC_H     ; Блокируем повторное включение заголовка
            #define    BIN99DEC_H

; bin99dec.inc – заголовочный файл к модулю
; bin99dec.asm – подпрограммы математических преобразований

; Переменные

            extern      d_rez, e_rez  ; ячейки для хранения десятичного числа в диапазоне
0…99

; Подпрограммы

            extern      Bin99Dec      ; число в аккумуляторе в десятичный вид, результат в
d_rez, e_rez
            extern      Dec99Bin      ; десятичное 0…99 в двоичный вид, результат в
аккумуляторе
                                      ; вход: в FSR адрес младшего (старший - следующий
адрес)

            #endif
```

Вот и весь принцип построения модулей и подключения их к проектам. Создавать модули несколько сложнее, чем единичный проект. Но выигрыш от такого подхода хорошо осознаётся со временем, когда программ и проектов становится много, и что-то найти в собственных исходниках иногда сложнее, чем переписать весь код по-новой. Готовая собственная библиотека модулей - отличное подспорье, если вы много программируете.

Литература:

1. MPASM. Как правильно оформлять программы на ассемблере для PIC-контроллеров (пособие для начинающих)
http://progcode.narod.ru/stati/asm_standart/mpasm_formatting.pdf
2. Как оформлять модули
http://www.pic24.ru/lib/exe/fetch.php/osa/articles/c_modules.pdf
3. MPASM. Руководство пользователя
http://www.microchip.ru/files/d-sheets-rus/mpasm.pdf