

## 1、程序的基本格式

先介绍二条伪指令：

EQU ——标号赋值伪指令

ORG ——地址定义伪指令

PIC16C5X在RESET后指令计算器PC被置为全“1”，所以PIC16C5X几种型号芯片的复位地址为：

PIC16C54/55 : 1FFH

PIC16C56 : 3FFH

PIC16C57/58 : 7FFH

一般来说，PIC的源程序并没有要求统一的格式，大家可以根据自己的风格来编写。但这里我们推荐一种清晰明了的格式：

```
TITLE This is ..... ;程序标题
;-----
;名称定义和变量定义
;-----
FO      EQU  0
RTCC    EQU  1
PC      EQU  2
STATUS  EQU  3
FSR     EQU  4
RA      EQU  5
RB      EQU  6
RC      EQU  7

PIC16C54 EQU 1FFH ;芯片复位地址
PIC16C56 EQU 3FFH
PIC16C57 EQU 7FFH
;-----
ORG PIC16C54 GOTO MAIN ;在复位地址处转入主程序
ORG 0 ;在0000H开始存放程序
;-----
;子程序区
;-----
DELAY MOVLW 255

        RETLW 0
;-----
;主程序区
;-----
MAIN
        MOVLW B'00000000'
        TRIS RB ;RB已由伪指令定义为6，即B口

LOOP
        BSF RB, 7 CALL DELAY
        BCF RB, 7 CALL DELAY

        GOTO LOOP
;-----
        END ;程序结束
```

注：MAIN标号一定要处在0页面内。

## 2、程序设计基础

### 1) 设置 I/O 口的输入/输出方向

PIC16C5X的I/O 口皆为双向可编程，即每一根I/O 端线都可分别单独地由程序设置为输入或输出。这个过程由写I/O 控制，写入值为“1”，则为输入；写入值为“0”，则为输出。

```
MOVLW 0FH      ; 0000 1111 (0FH)
                输入 输出
TRIS 6         ; 将W中的0FH写入B口控制器，
                ; B口高4位为输出，低4位为输入。
MOVLW 0C0H    ; 11 000000 (0C0H)
                RB4, RB5输出0 RB6, RB7输出1
```

### 2) 检查寄存器是否为零

如果要判断一个寄存器内容是否为零，很简单，现以寄存器F10为例：

```
MOVF 10, 1     ; F10 F10, 结果影响零标记状态位Z
BTFSS STATUS, Z ; F10为零则跳
GOTO NZ        ; Z=0即F10不为零转入标号NZ处程序
                ; Z=1即F10=0处理程序
```

### 3) 比较二个寄存器的大小

要比较二个寄存器的大小，可以将它们做减法运算，然后根据状态位C来判断。注意，相减的结果放入W，则不会影响二例如F8和F9二个寄存器要比较大小：

```
MOVF 8, 0      ; F8 W
SUBWF 9, 0     ; F9—W (F8) W
BTFSC STATUS, Z ; 判断F8=F9否
GOTO F8=F9
BTFSC STATUS, C ; C=0则跳
GOTO F9>F8     ; C=1相减结果为正，F9>F8
GOTO F9<
F9             ; C=0相减结果为负，F9<F8
```

### 4) 循环n次的程序

如果要使某段程序循环执行n次，可以用一个寄存器作计数器。下例以F10做计数器，使程序循环8次。

```
COUNT EQU 10   ; 定义F10名称为COUNT (计数器)

MOVLW 8
MOVWF COUNT LOOP ; 循环体
LOOP

DECFSZ COUNT, 1 ; COUNT减1, 结果为零则跳
GOTO LOOP      ; 结果不为零, 继续循环
                ; 结果为零, 跳出循环
```

### 5) “IF.....THEN.....”格式的程序

下面以“IF X=Y THEN GOTO NEXT”格式为例。

```
MOVF X, 0      ; X W
SUBWF Y, 0     ; Y—W (X) W
BTFSC STATUS, Z ; X=Y 否
GOTO NEXT     ; X=Y, 跳到NEXT去执行。
                ; X Y
```

### 6) “FOR.....NEXT”格式的程序

“FOR.....NEXT”程序使循环在某个范围内进行。下例是“FOR X=0 TO 5”格式的程序。F10放X的初值，F11放X的终值。

```
START EQU 10
DAEND EQU 11

MOVLW 0
```

```

MOVWF START      ; 0 START (F10)
MOVLW 5
MOVWF DAEND      ; 5 DAEND (F11)
LOOP

INCF START, 1    ; START值加1
MOVWF START, 0
SUBWF DAEND, 0   ; START=DAEND ? (X=5否)
BTFSS STATUS, Z
GOTO LOOP       ; X < 5, 继续循环
                ; X = 5, 结束循环

```

### 7) “DO WHILE.....END” 格式的程序

“DO WHILE.....END” 程序是在符合条件下执行循环。下例是“DO WHILE X=1” 格式的程序。F10放X的值。

```

X EQU 10

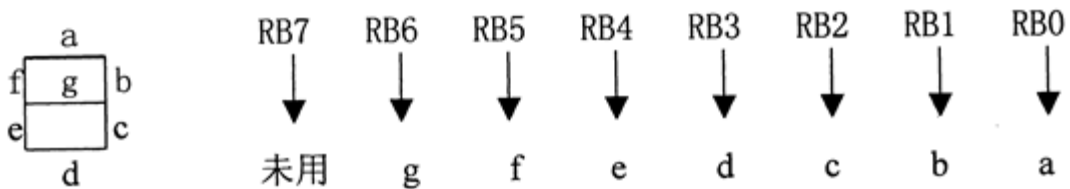
MOVLW 1
MOVWF X          ; 1 X (F10), 作为初值
LOOP

MOVLW 1
SUBWF X, 0
BTFSS STATUS, Z ; X = 1否?
GOTO LOOP       ; X = 1继续循环
                ; X = 1跳出循环

```

### 8) 查表程序

查表是程序中经常用到的一种操作。下例是将十进制0~9转换成7段LED数字显示值。若以B口的RB0~RB6来驱动LED的a~系：



设LED为共阳，则0~9数字对应的线段值如下表：

十进数	线段值	十进数	线段值
0	C0H	5	92H
1	C9H	6	82H
2	A4H	7	F8H
3	B0H	8	80H
4	99H	9	90H

十进数 线段值 十进数 线段值

```

0 C0H 5 92H
1 C9H 6 82H
2 A4H 7 F8H
3 B0H 8 80H
4 99H 9 90H

```

PIC的查表程序可以利用子程序带值返回的特点来实现。具体是在主程序中先取表数据地址放入W，接着调用子程序，子置入PC，则程序跳到数据地址的地方，再由“RETLW”指令将数据放入W返回到主程序。下面程序以F10放表头地址。

```

MOVLW TABLE      ;表头地址 F10
MOVWF 10

MOVLW 1            ;1 W,准备取“1”的线段值
ADDWF 10,1        ;F10+W = “1”的数据地址
CALL CONVERT
MOVWF 6           ;线段值置到B口,点亮LED

CONVERT MOVWF 2    ;W PC TABLE
RETLW 0C0H       ;“0”线段值
RETLW 0F9H       ;“1”线段值

RETLW 90H        ;“9”线段值

```

#### 9) “READ.....DATA, RESTORE”格式程序

“READ.....DATA”程序是每次读取数据表的一个数据，然后将数据指针加1，准备取下一个数据。下例程序中以F10为数据指针。

```

POINTER EQU 11    ;定义F11名称为POINTER

MOVLW DATA
MOVWF 10          ;数据表头地址 F10
CLRF POINTER     ;数据指针清零

MOVF POINTER,0
ADDWF 10,0       ;W =F10+POINTER

INCF POINTER,1   ;指针加1
CALL CONVERT     ;调子程序,取表格数据

CONVERT MOVWF 2   ;数据地址 PC
DATA RETLW 20H   ;数据

RETLW 15H       ;数据

```

如果要执行“RESTORE”，只要执行一条“CLRF POINTER”即可。

#### 10) 延时程序

如果延时时间较短，可以让程序简单地连续执行几条空操作指令“NOP”。如果延时时间长，可以用循环来实现。下例以执行100次。

```

MOVLW D'100'
MOVWF 10
LOOP DECFSZ 10,1 ;F10—1 F10,结果为零则跳
GOTO LOOP

```

延时程序中计算指令执行的时间和即为延时时间。如果使用4MHz振荡，则每个指令周期为1μS。所以单周期指令时间为1μS。在上例的LOOP循环延时时间即为： $(1+2) * 100 + 2 = 302 (\mu S)$ 。在循环中插入空操作指令即可延长延时时间：

```

MOVLW D'100'
MOVWF 10
LOOP NOP
NOP
NOP

```

```

DECFSZ 10, 1
GOTO LOOP

```

延时时间 = (1+1+1+1+2) \* 100 + 2 = 602 (μs)。

用几个循环嵌套的方式可以大大延长延时时间。下例用2个循环来做延时：

```

        MOVLW    D'100'
        MOVWF    10
LOOP    MOVLW    D'16'
        MOVWF    11
LOOP1   DECFSZ   11, 1
        GOTO    LOOP1
        DECFSZ   10, 1
        GOTO    LOOP

```

延时时间 = 1+1+[1+1+(1+2)\*16-1+1+2]\*100-1=5201 (μs)

### 11) RTCC计数器的使用

RTCC是一个脉冲计数器，它的计数脉冲有二个来源，一个是从RTCC引脚输入的外部信号，一个是内部的指令时钟信号。一个信号源作为输入。RTCC可被程序用作计时之用；程序读取RTCC寄存器值以计算时间。当RTCC作为内部计时器使用时需以减少干扰和耗电流。下例程序以RTCC做延时：

```

RTCC EQU 1

        CLRF    RTCC        ; RTCC清0
        MOVLW   07H
        OPTION  ; 选择预设倍数1:256 RTCC
LOOP    MOVLW   255        ; RTCC计数终值
        SUBWF   RTCC, 0
        BTFSS  STATUS, Z    ; RTCC=255?
        GOTO   LOOP

```

这个延时程序中，每过256个指令周期RTCC寄存器增1（分频比=1:256），设芯片使用4MHz振荡，则：

延时时间 = 256 \* 256 = 65536 (μs)

RTCC是自振式的，在它计数时，程序可以去别的事情，只要隔一段时间去读取它，检测它的计数值即可。

### 12) 寄存器体 (BANK) 的寻址

对于PIC16C54/55/56，寄存器有32个，只有一个体 (BANK)，故不存在体寻址问题，对于PIC16C57/58来说，寄存器则有 (BANK0-BANK3)。在对F4 (FSR) 的说明中可知，F4的bit6和bit5是寄存器体寻址位，其对应关系如下：

Bit 6	Bit 5	BANK	物理地址
0	0	BANK0	10H ~ 1FH
0	1	BANK1	30H ~ 3FH
1	0	BANK2	50H ~ 5FH
1	1	BANK3	70H ~ 7FH

Bit 6	Bit 5	BANK	物理地址
0	0	BANK0	10H ~ 1FH
0	1	BANK1	30H ~ 3FH
1	0	BANK2	50H ~ 5FH
1	1	BANK3	70H ~ 7FH

当芯片上电RESET后，F4的bit6, bit5是随机的，非上电的RESET则保持原先状态不变。

下面的例子对BANK1和BANK2的30H及50H寄存器写入数据。

例1. (设目前体选为BANK0)

```
BSF    4, 5      ; 置位bit5=1, 选择BANK1
MOVLW DATA
MOVWF  10H      ; DATA 30H
BCF    4, 5
BSF    4, 6      ; bit6=1, bit5=0选择BANK2
MOVWF  10H      ; DATA 50H
```

从上例中我们看到,对某一体(BANK)中的寄存器进行读写,首先要先对F4中的体寻址位进行操作。实际应用中一般上和bit5为0,使之指向BANK0,以后再根据需要使其指向相应的体。

注意,在例子中对30H寄存器(BANK1)和50H寄存器(BANK2)写数时,用的指令“MOVWF 10H”中寄存器地址写的都是期的“MOVWF 30H”和“MOVWF 50H”,为什么?

让我们回顾一下指令表。在PIC16C5X的所有有关寄存器的指令码中,寄存寻址位都只占5个位:ffff,只能寻址32个(以要选址80个寄存器,还要再用二位体选址位PA1和PA0。当我们设置好体寻址位PA1和PA0,使之指向一个BANK,那么指令“容置入这个BANK中的相应寄存器内(10H,30H,50H,或70H)。

有些设计者第一次接触体选址的概念,难免理解上有出入,下面是一个例子:

例2: (设目前体选为BANK0)

```
MOVLW  55H
MOVWF  30H      ; 欲把55H 30H寄存器
MOVLW  66H
MOVWF  50H      ; 欲把66H 50H寄存器
```

以为“MOVWF 30H”一定能把W置入30H,“MOVWF 50H”一定能把W置入50H,这是错误的。因为这两条指令的实际效果是上面已经说明过了。所以例2这段程序最后结果是F10H=66H,而真正的F30H和F50H并没有被操作到。

建议:为使体选址的程序清晰明了,建议多用名称定义符来写程序,则不易混淆。 例3:假设在程序中用到BANK0,寄存器如下:

BANK0	地址	BANK1	地址	BANK2	地址	BANK3	地址
A	10H	B	30H	C	50H	.	70H
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

BANK0 地址 BANK1 地址 BANK2 地址 BANK3 地址

A 10H B 30H C 50H . 70H

. . . . .

```
A    EQU  10H    ; BANK0
B    EQU  10H    ; BANK1
C    EQU  10H    ; BANK2
```

```
FSR  EQU  4
Bi t6 EQU  6
Bi t5 EQU  5
DATA EQU  55H
```

```
MOVLW DATA
MOVWF  A
BSF   FSR, Bi t5
MOVWF B      ; DATA F30H
```

```

BCF    FSR, Bi t5
BSF    FSR, Bi t6
MOVWF  C          ; DATA F50H

```

程序这样书写，相信体选址就不容易错了。

### 13) 程序跨页面跳转和调用

下面介绍PIC16C5X的程序存储区的页面概念和F3寄存器中的页面选址位PA1和PA0两位应用的实例。

#### (1) “GOTO”跨页面

例：设目前程序在0页面（PAGE0），欲用“GOTO”跳转到1页面的某个地方KEY（PAGE1）。

```

STATUS EQU 3
PA1     EQU 6
PA0     EQU 5

BSF    STATUS, PA0    ; PA0=1, 选择PAGE页面
GOTO   KEY           ; 跨页跳转到1页面的KEY

KEY    NOP           ; 1页面的程序

```

#### (2) “CALL”跨页面

例：设目前程序在0页面（PAGE0），现在要调用——放在1页面（PAGE1）的子程序DELAY。

```

BSF    STATUS, PA0    ; PA0=1, 选择PAGE1页面
CALL   DELAY         ; 跨页调用
BCF    STATUS, PA0    ; 恢复0页面地址

DELAY NOP           ; 1页面的子程序

```

注意：程序为跨页CALL而设了页面地址，从子程序返回后一定要恢复原来的页面地址。

#### (3) 程序跨页跳转和调用的编写

读者看到这里，一定要问：我写源程序（.ASM）时，并不去注意每条指令的存放地址，我怎么知道这个GOTO是要跨页面的？的确，开始写源程序时并不知道何时会发生跨页面跳转或调用，不过当你将源程序汇编时，就会自动给出。当汇编结束

X X X (地址) “GOTO out of Range”

X X X (地址) “CALL out of Range”

这表明你的程序发生了跨页面的跳转和调用，而你的程序中在这些跨页GOTO和CALL之前还未设置好相应的页面地址。这的.LST文件，找到这些GOTO和CALL，并查看它们要跳转去的地址处在什么页面，然后再回到源程序（.ASM）做必要的修改。通过（0 Errors and Warnings）。

#### (4) 程序页面的连接

程序4个页面连接处应该做一些处理。一般建议采用下面的格式：即在进入另一个页面后，马上设置相应的页面地址位处理是PIC16C5X编程中最麻烦的部分，不过并不难。只要做了一次实际的编程练习后，就能掌握了。