



Welcome to [E-XFL.COM](https://www.e-xfl.com)

### What is "[Embedded - Microcontrollers](#)"?

"[Embedded - Microcontrollers](#)" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

### Applications of "[Embedded - Microcontrollers](#)"

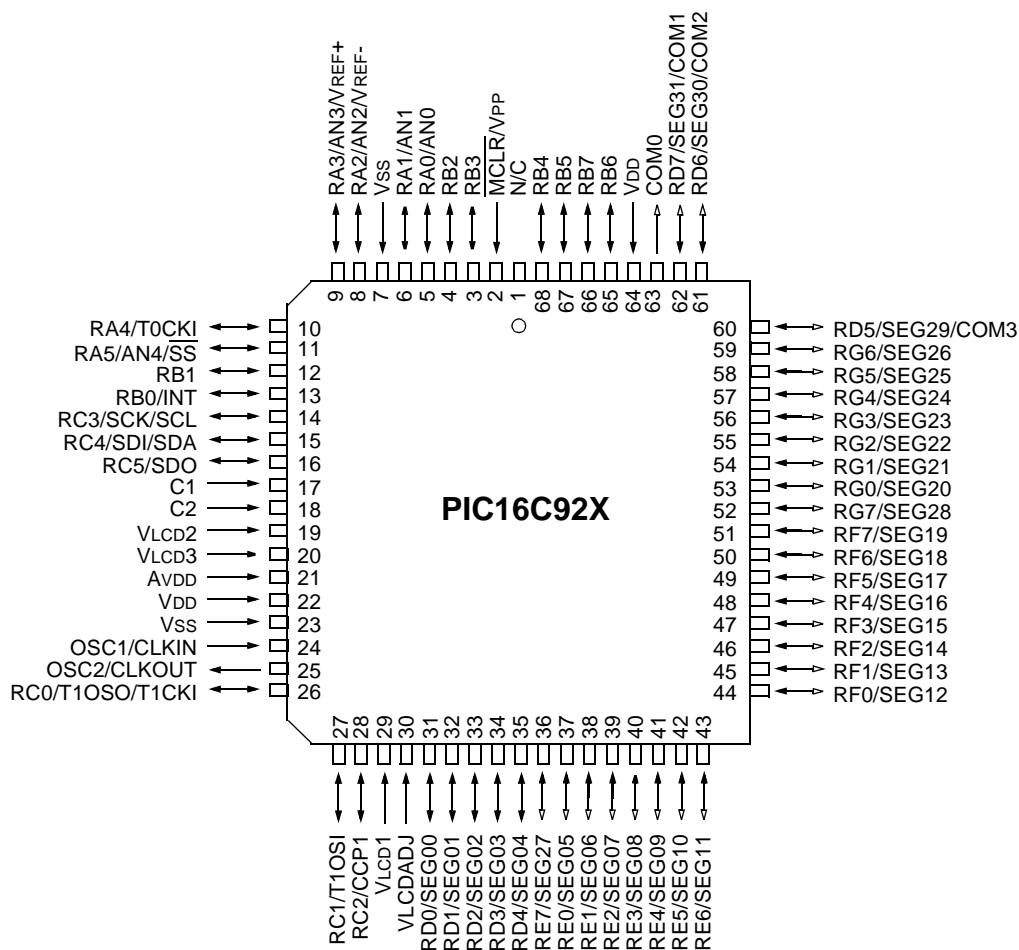
#### Details

Product Status	Active
Core Processor	PIC
Core Size	8-Bit
Speed	20MHz
Connectivity	I <sup>2</sup> C, SPI
Peripherals	Brown-out Detect/Reset, LCD, POR, PWM, WDT
Number of I/O	25
Program Memory Size	14KB (8K x 14)
Program Memory Type	OTP
EEPROM Size	-
RAM Size	336 x 8
Voltage - Supply (Vcc/Vdd)	4V ~ 5.5V
Data Converters	A/D 5x10b
Oscillator Type	External
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	68-LCC (J-Lead)
Supplier Device Package	68-PLCC (24.23x24.23)
Purchase URL	<a href="https://www.e-xfl.com/product-detail/microchip-technology/pic16c926-i-l">https://www.e-xfl.com/product-detail/microchip-technology/pic16c926-i-l</a>

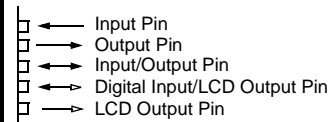
# PIC16C925/926

## Pin Diagrams

### PLCC, CLCC




#### LEGEND:



**FIGURE 2-3: REGISTER FILE MAP — DSTEMP**

File Address		File Address		File Address		File Address	
Indirect addr.(*)	00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h
TMR0	01h	OPTION	81h	TMR0	101h	OPTION	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h	PORTF	107h	TRISF	187h
PORTD	08h	TRISD	88h	PORTG	108h	TRISG	188h
PORTE	09h	TRISE	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	PMCON1	10Ch	PMDATA	18Ch
	0Dh		8Dh	LCDSE	10Dh	PMADR	18Dh
TMR1L	0Eh	PCON	8Eh	LCDPS	10Eh	PMDATH	18Eh
TMR1H	0Fh		8Fh	LCDDCON	10Fh	PMADRH	18Fh
T1CON	10h		90h	LCDD00	110h		190h
TMR2	11h		91h	LCDD01	111h		191h
T2CON	12h	PR2	92h	LCDD02	112h		192h
SSPBUF	13h	SSPADD	93h	LCDD03	113h		193h
SSPCON	14h	SSPSTAT	94h	LCDD04	114h		194h
CCPR1L	15h		95h	LCDD05	115h		195h
CCPR1H	16h		96h	LCDD06	116h		196h
CCP1CON	17h		97h	LCDD07	117h		197h
	18h		98h	LCDD08	118h		198h
	19h		99h	LCDD09	119h		199h
	1Ah		9Ah	LCDD10	11Ah		19Ah
	1Bh		9Bh	LCDD11	11Bh		19Bh
	1Ch		9Ch	LCDD12	11Ch		19Ch
	1Dh		9Dh	LCDD13	11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh	LCDD14	11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh	LCDD15	11Fh		19Fh
	20h		A0h		120h		1A0h
General Purpose Register		General Purpose Register					
			EFh		16Fh		1EFh
			F0h		170h		1F0h
			FFh		17Fh		1FFh
Bank 0		Bank 1		Bank 2		Bank 3	

 Unimplemented data memory locations, read as '0'.  
 \* Not a physical register.

**TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)**

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Details on page
<b>Bank 2</b>											
100h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	26
101h	TMR0	Timer0 Module Register								xxxx xxxx	41
102h	PCL	Program Counter (PC) Least Significant Byte								0000 0000	25
103h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	19
104h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	26
105h	—	Unimplemented								—	—
106h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	31
107h	PORTF	PORTF pins when read								0000 0000	37
108h	PORTG	PORTG pins when read								0000 0000	38
109h	—	Unimplemented								—	—
10Ah	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the PC					---0 0000	25
10Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	21
10Ch	PMCON1	reserved	—	—	—	—	—	—	RD	1--- ---0	27
10Dh	LCDSE	SE29	SE27	SE20	SE16	SE12	SE9	SE5	SE0	1111 1111	94
10Eh	LCDPS	—	—	—	—	LP3	LP2	LP1	LP0	---- 0000	84
10Fh	LCDCON	LCDEN	SLPEN	—	VGEN	CS1	CS0	LMUX1	LMUX0	00-0 0000	83
110h	LCDD00	SEG07 COM0	SEG06 COM0	SEG05 COM0	SEG04 COM0	SEG03 COM0	SEG02 COM0	SEG01 COM0	SEG00 COM0	xxxx xxxx	92
111h	LCDD01	SEG15 COM0	SEG14 COM0	SEG13 COM0	SEG12 COM0	SEG11 COM0	SEG10 COM0	SEG09 COM0	SEG08 COM0	xxxx xxxx	92
112h	LCDD02	SEG23 COM0	SEG22 COM0	SEG21 COM0	SEG20 COM0	SEG19 COM0	SEG18 COM0	SEG17 COM0	SEG16 COM0	xxxx xxxx	92
113h	LCDD03	SEG31 COM0	SEG30 COM0	SEG29 COM0	SEG28 COM0	SEG27 COM0	SEG26 COM0	SEG25 COM0	SEG24 COM0	xxxx xxxx	92
114h	LCDD04	SEG07 COM1	SEG06 COM1	SEG05 COM1	SEG04 COM1	SEG03 COM1	SEG02 COM1	SEG01 COM1	SEG00 COM1	xxxx xxxx	92
115h	LCDD05	SEG15 COM1	SEG14 COM1	SEG13 COM1	SEG12 COM1	SEG11 COM1	SEG10 COM1	SEG09 COM1	SEG08 COM1	xxxx xxxx	92
116h	LCDD06	SEG23 COM1	SEG22 COM1	SEG21 COM1	SEG20 COM1	SEG19 COM1	SEG18 COM1	SEG17 COM1	SEG16 COM1	xxxx xxxx	92
117h	LCDD07	SEG31 COM1 <sup>(1)</sup>	SEG30 COM1	SEG29 COM1	SEG28 COM1	SEG27 COM1	SEG26 COM1	SEG25 COM1	SEG24 COM1	xxxx xxxx	92
118h	LCDD08	SEG07 COM2	SEG06 COM2	SEG05 COM2	SEG04 COM2	SEG03 COM2	SEG02 COM2	SEG01 COM2	SEG00 COM2	xxxx xxxx	92
119h	LCDD09	SEG15 COM2	SEG14 COM2	SEG13 COM2	SEG12 COM2	SEG11 COM2	SEG10 COM2	SEG09 COM2	SEG08 COM2	xxxx xxxx	92
11Ah	LCDD10	SEG23 COM2	SEG22 COM2	SEG21 COM2	SEG20 COM2	SEG19 COM2	SEG18 COM2	SEG17 COM2	SEG16 COM2	xxxx xxxx	92
11Bh	LCDD11	SEG31 COM2 <sup>(1)</sup>	SEG30 COM2 <sup>(1)</sup>	SEG29 COM2	SEG28 COM2	SEG27 COM2	SEG26 COM2	SEG25 COM2	SEG24 COM2	xxxx xxxx	92
11Ch	LCDD12	SEG07 COM3	SEG06 COM3	SEG05 COM3	SEG04 COM3	SEG03 COM3	SEG02 COM3	SEG01 COM3	SEG00 COM3	xxxx xxxx	92
11Dh	LCDD13	SEG15 COM3	SEG14 COM3	SEG13 COM3	SEG12 COM3	SEG11 COM3	SEG10 COM3	SEG09 COM3	SEG08 COM3	xxxx xxxx	92
11Eh	LCDD14	SEG23 COM3	SEG22 COM3	SEG21 COM3	SEG20 COM3	SEG19 COM3	SEG18 COM3	SEG17 COM3	SEG16 COM3	xxxx xxxx	92
11Fh	LCDD15	SEG31 COM3 <sup>(1)</sup>	SEG30 COM3 <sup>(1)</sup>	SEG29 COM3 <sup>(1)</sup>	SEG28 COM3	SEG27 COM3	SEG26 COM3	SEG25 COM3	SEG24 COM3	xxxx xxxx	92

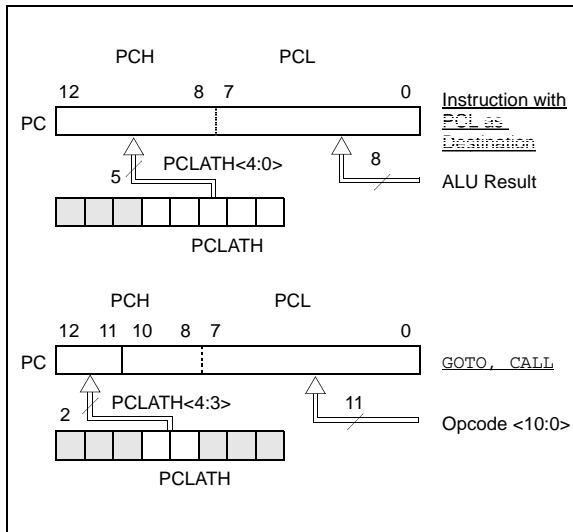
Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0'.  
Shaded locations are unimplemented, read as '0'.

**Note 1:** These pixels do not display, but can be used as general purpose RAM.

## 2.4 PCL and PCLATH

The program counter (PC) is 13-bits wide. The low byte comes from the PCL register, which is a readable and writable register. The upper bits (PC<12:8>) are not readable, but are indirectly writable through the PCLATH register. On any RESET, the upper bits of the PC will be cleared. Figure 2-5 shows the two situations for the loading of the PC. The upper example in the figure shows how the PC is loaded on a write to PCL (PCLATH<4:0> → PCH). The lower example in the figure shows how the PC is loaded during a CALL or GOTO instruction (PCLATH<4:3> → PCH).

**FIGURE 2-5: LOADING OF PC IN DIFFERENT SITUATIONS**



### 2.4.1 COMPUTED GOTO

A computed GOTO is accomplished by adding an offset to the program counter (ADDWF PCL). When doing a table read using a computed GOTO method, care should be exercised if the table location crosses a PCL memory boundary (each 256 byte block). Refer to the application note “Implementing a Table Read” (AN556).

### 2.4.2 STACK

The PIC16CXX family has an 8-level deep x 13-bit wide hardware stack. The stack space is not part of either program or data space and the stack pointer is not readable or writable. The PC is PUSHed onto the stack when a CALL instruction is executed, or an interrupt causes a branch. The stack is POPed in the event of a RETURN, RETLW or a RETFIE instruction execution. PCLATH is not affected by a PUSH or POP operation.

The stack operates as a circular buffer. This means that after the stack has been PUSHed eight times, the ninth push overwrites the value that was stored from the first push. The tenth push overwrites the second push (and so on).

**Note 1:** There are no status bits to indicate stack overflow or stack underflow conditions.

**2:** There are no instructions/mnemonics called PUSH or POP. These are actions that occur from the execution of the CALL, RETURN, RETLW, and RETFIE instructions, or the vectoring to an interrupt address.

## 2.5 Program Memory Paging

PIC16C925/926 devices are capable of addressing a continuous 8K word block of program memory. The CALL and GOTO instructions provide only 11-bits of address to allow branching within any 2K program memory page. When doing a CALL or GOTO instruction, the upper 2-bits of the address are provided by PCLATH<4:3>. When doing a CALL or GOTO instruction, the user must ensure that the page select bits are programmed so that the desired program memory page is addressed. If a return from a CALL instruction (or interrupt) is executed, the entire 13-bit PC is pushed onto the stack. Therefore, manipulation of the PCLATH<4:3> bits is not required for the RETURN instructions (which POPs the address from the stack).

**Note:** The contents of the PCLATH register are unchanged after a RETURN or RETFIE instruction is executed. The user must rewrite the PCLATH for any subsequent CALL or GOTO instructions.

Example 2-1 shows the calling of a subroutine in page 1 of the program memory. This example assumes that PCLATH is saved and restored by the Interrupt Service Routine (if interrupts are used).

### EXAMPLE 2-1: CALL OF A SUBROUTINE IN PAGE 1 FROM PAGE 0

```

ORG 0x500
BCF    PCLATH,4
BSF    PCLATH,3 ;Select page 1 (800h-FFFh)
CALL   SUB1_P1 ;Call subroutine in
            ;page 1 (800h-FFFh)
            :
            :
            :
ORG 0x900
SUB1_P1:      ;called subroutine
            :      ;page 1 (800h-FFFh)
            :
            :
RETURN      ;return to Call subroutine
            ;in page 0 (000h-7FFh)
    
```

2.6 Indirect Addressing, INDF and FSR Registers

The INDF register is not a physical register. Addressing the INDF register will cause indirect addressing.

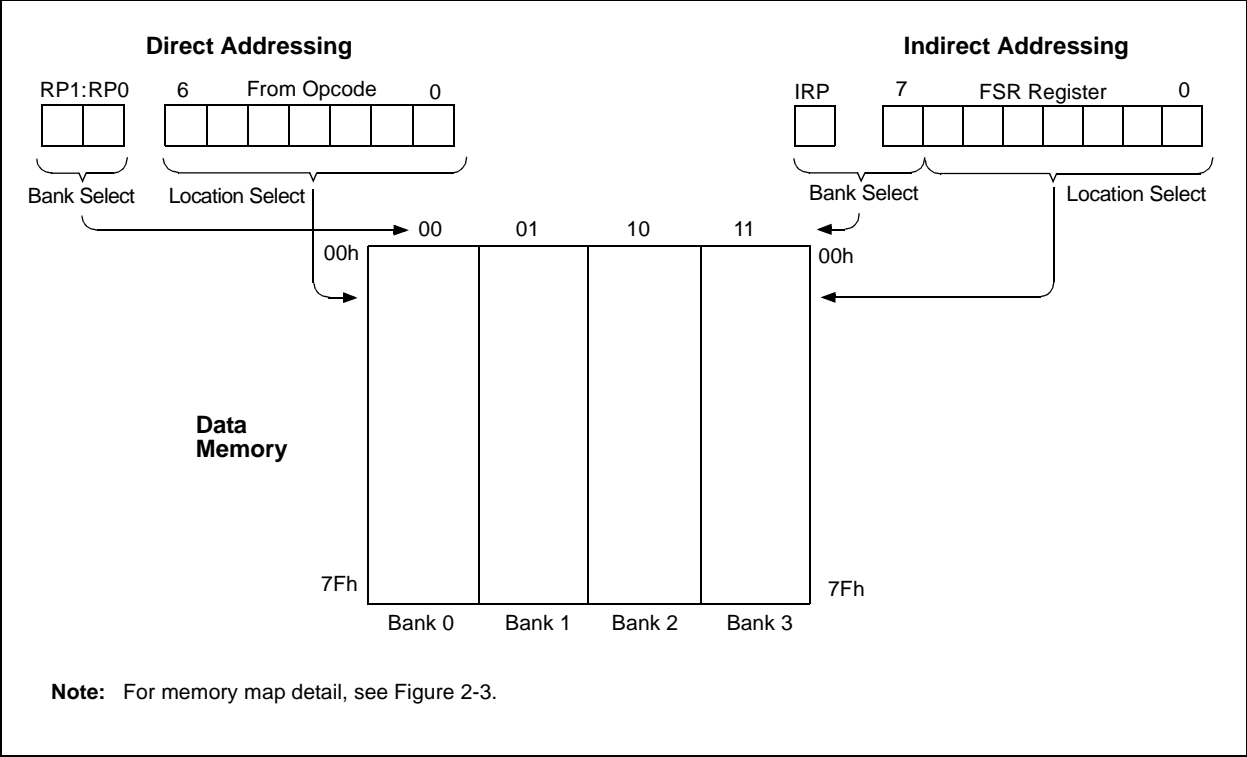
Indirect addressing is possible by using the INDF register. Any instruction using the INDF register actually accesses the register pointed to by the File Select Register (FSR). Reading the INDF register itself, indirectly (FSR = '0'), will produce 00h. Writing to the INDF register indirectly results in a no operation (although status bits may be affected). An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (STATUS<7>), as shown in Figure 2-6.

A simple program to clear RAM locations 20h-2Fh using indirect addressing is shown in Example 2-2.

**EXAMPLE 2-2: INDIRECT ADDRESSING**

```
MOV LW 0x20 ;initialize pointer
MOV WF FSR ;to RAM
NEXT CLRF INDF ;clear INDF register
      INCF FSR,F ;inc pointer
      BTFSS FSR,4 ;all done?
      GOTO NEXT ;no clear next
CONTINUE
      : ;yes continue
```

FIGURE 2-6: DIRECT/INDIRECT ADDRESSING



## 4.6 PORTF and TRISF Register

PORTF is a digital input only port. Each pin is multiplexed with an LCD segment driver. These pins have Schmitt Trigger input buffers.

**Note 1:** On a Power-on Reset, these pins are configured as LCD segment drivers.

**2:** To configure the pins as a digital port, the corresponding bits in the LCDSE register must be cleared. Any bit set in the LCDSE register overrides any bit settings in the corresponding TRIS register.

### EXAMPLE 4-6: INITIALIZING PORTF

```
BCF STATUS, RP0      ;Select Bank2
BSF STATUS, RP1      ;
BCF LCDSE, SE16       ;Make all PORTF
BCF LCDSE, SE12       ;digital inputs
```

FIGURE 4-8: PORTF BLOCK DIAGRAM

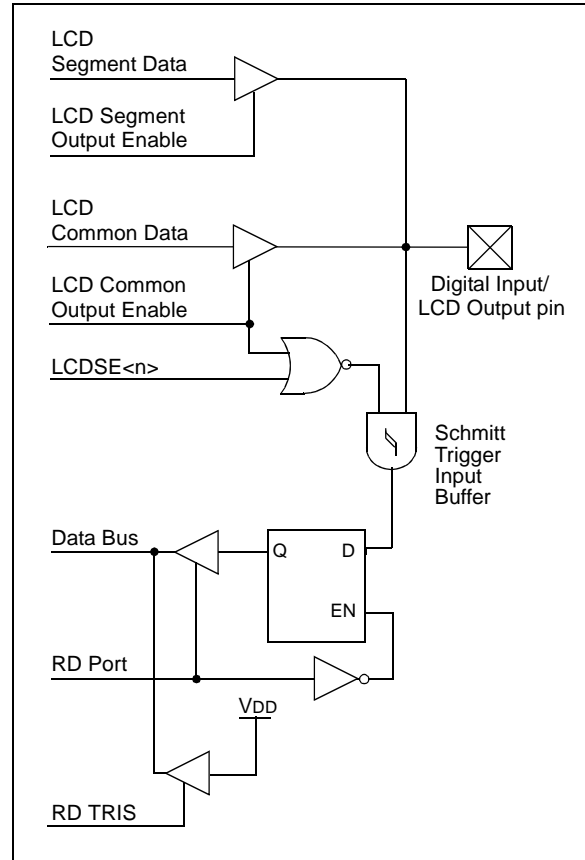


TABLE 4-11: PORTF FUNCTIONS

Name	Bit#	Buffer Type	Function
RF0/SEG12	bit0	ST	Digital input or Segment Driver12.
RF1/SEG13	bit1	ST	Digital input or Segment Driver13.
RF2/SEG14	bit2	ST	Digital input or Segment Driver14.
RF3/SEG15	bit3	ST	Digital input or Segment Driver15.
RF4/SEG16	bit4	ST	Digital input or Segment Driver16.
RF5/SEG17	bit5	ST	Digital input or Segment Driver17.
RF6/SEG18	bit6	ST	Digital input or Segment Driver18.
RF7/SEG19	bit7	ST	Digital input or Segment Driver19.

Legend: ST = Schmitt Trigger input

TABLE 4-12: SUMMARY OF REGISTERS ASSOCIATED WITH PORTF

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other RESETS
107h	PORTF	RF7	RF6	RF5	RF4	RF3	RF2	RF1	RF0	0000 0000	0000 0000
187h	TRISF	PORTF Data Direction Control Register								1111 1111	1111 1111
10Dh	LCDSE	SE29	SE27	SE20	SE16	SE12	SE9	SE5	SE0	1111 1111	1111 1111

Legend: Shaded cells are not used by PORTF.

# PIC16C925/926

## 4.7 PORTG and TRISG Register

PORTG is a digital input only port. Each pin is multiplexed with an LCD segment driver. These pins have Schmitt Trigger input buffers.

**Note 1:** On a Power-on Reset, these pins are configured as LCD segment drivers.

**2:** To configure the pins as a digital port, the corresponding bits in the LCDSE register must be cleared. Any bit set in the LCDSE register overrides any bit settings in the corresponding TRIS register.

### EXAMPLE 4-7: INITIALIZING PORTG

```
BCF STATUS, RP0 ;Select Bank2
BSF STATUS, RP1 ;
BCF LCDSE, SE27 ;Make all PORTG
BCF LCDSE, SE20 ;and PORTE<7>
                    ;digital inputs
```

FIGURE 4-9: PORTG BLOCK DIAGRAM

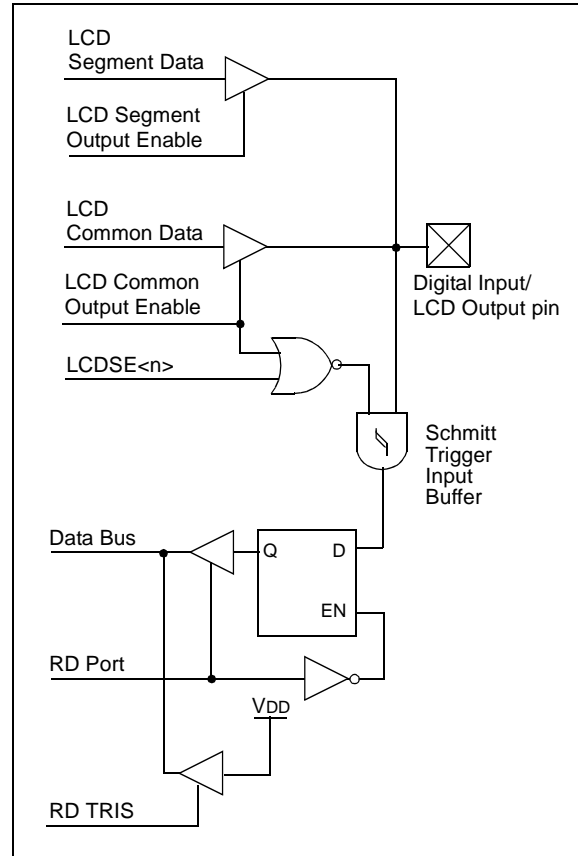


TABLE 4-13: PORTG FUNCTIONS

Name	Bit#	Buffer Type	Function
RG0/SEG20	bit0	ST	Digital input or Segment Driver20.
RG1/SEG21	bit1	ST	Digital input or Segment Driver21.
RG2/SEG22	bit2	ST	Digital input or Segment Driver22.
RG3/SEG23	bit3	ST	Digital input or Segment Driver23.
RG4/SEG24	bit4	ST	Digital input or Segment Driver24.
RG5/SEG25	bit5	ST	Digital input or Segment Driver25.
RG6/SEG26	bit6	ST	Digital input or Segment Driver26.
RG7/SEG28	bit7	ST	Digital input or Segment Driver28 (not available on 64-pin devices).

Legend: ST = Schmitt Trigger input

TABLE 4-14: SUMMARY OF REGISTERS ASSOCIATED WITH PORTG

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other RESETS
108h	PORTG	RG7	RG6	RG5	RG4	RG3	RG2	RG1	RG0	0000 0000	0000 0000
188h	TRISG	PORTG Data Direction Control Register								1111 1111	1111 1111
10Dh	LCDSE	SE29	SE27	SE20	SE16	SE12	SE9	SE5	SE0	1111 1111	1111 1111

Legend: Shaded cells are not used by PORTG.



## 5.2 Using Timer0 with an External Clock

When an external clock input is used for Timer0, it must meet certain requirements. The requirements ensure the external clock can be synchronized with the internal phase clock (Tosc). Also, there is a delay in the actual incrementing of Timer0 after synchronization.

### 5.2.1 EXTERNAL CLOCK SYNCHRONIZATION

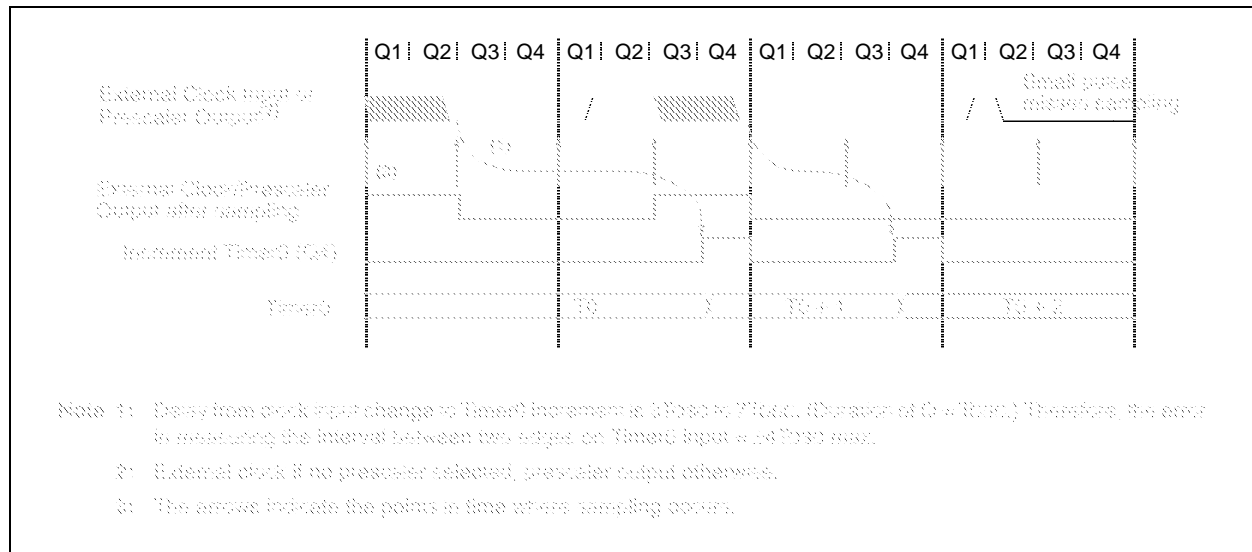
When no prescaler is used, the external clock input is the same as the prescaler output. The synchronization of T0CKI with the internal phase clocks is accomplished by sampling the prescaler output on the Q2 and Q4 cycles of the internal phase clocks (Figure 5-5). Therefore, it is necessary for T0CKI to be high for at least  $2T_{osc}$  (and a small RC delay of 20 ns) and low for at least  $2T_{osc}$  (and a small RC delay of 20 ns). Refer to the electrical specification of the desired device.

When a prescaler is used, the external clock input is divided by the asynchronous ripple counter type prescaler, so that the prescaler output is symmetrical. For the external clock to meet the sampling requirement, the ripple counter must be taken into account. Therefore, it is necessary for T0CKI to have a period of at least  $4T_{osc}$  (and a small RC delay of 40 ns) divided by the prescaler value. The only requirement on T0CKI high and low time is that they do not violate the minimum pulse width requirement of 10 ns. Refer to parameters 40, 41 and 42 in the electrical specification of the desired device.

### 5.2.2 TMR0 INCREMENT DELAY

Since the prescaler output is synchronized with the internal clocks, there is a small delay from the time the external clock edge occurs to the time the Timer0 module is actually incremented. Figure 5-5 shows the delay from the external clock edge to the timer incrementing.

**FIGURE 5-5: TIMER0 TIMING WITH EXTERNAL CLOCK**



## 6.0 TIMER1 MODULE

Timer1 is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L), which are readable and writable. The TMR1 Register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. The TMR1 Interrupt, if enabled, is generated on overflow, which is latched in interrupt flag bit, TMR1IF (PIR1<0>). This interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit, TMR1IE (PIE1<0>).

Timer1 can operate in one of two modes:

- As a timer
- As a counter

The operating mode is determined by the clock select bit, TMR1CS (T1CON<1>).

In Timer mode, Timer1 increments every instruction cycle. In Counter mode, it increments on every rising edge of the external clock input.

Timer1 can be turned on and off using the control bit TMR1ON (T1CON<0>).

Timer1 also has an internal "RESET input". This RESET can be generated by the CCP module (Section 8.0). Register 6-1 shows the Timer1 control register.

When the Timer1 oscillator is enabled (T1OSCEN is set), the RC1/T1OSI and RC0/T1OSO/T1CKI pins become inputs, regardless of the TRISC<1:0>. RC1 and RC0 will be read as '0'.

### REGISTER 6-1: T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
bit 7		bit 0					

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **T1CKPS1:T1CKPS0:** Timer1 Input Clock Prescale Select bits

11 = 1:8 Prescale value  
10 = 1:4 Prescale value  
01 = 1:2 Prescale value  
00 = 1:1 Prescale value

bit 3 **T1OSCEN:** Timer1 Oscillator Enable Control bit

1 = Oscillator is enabled  
0 = Oscillator is shut-off

**Note:** The oscillator inverter and feedback resistor are turned off to eliminate power drain.

bit 2 **T1SYNC:** Timer1 External Clock Input Synchronization Control bit

**TMR1CS = 1:**

1 = Do not synchronize external clock input  
0 = Synchronize external clock input

**TMR1CS = 0:**

This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS:** Timer1 Clock Source Select bit

1 = External clock from pin T1CKI (on the rising edge)  
0 = Internal clock (Fosc/4)

bit 0 **TMR1ON:** Timer1 On bit

1 = Enables Timer1  
0 = Stops Timer1

Legend:

R = Readable bit

W = Writable bit

U = Unimplemented bit, read as '0'

- n = Value at POR

'1' = Bit is set

'0' = Bit is cleared

x = Bit is unknown

## 6.3 Timer1 Operation in Asynchronous Counter Mode

If control bit  $\overline{T1SYNC}$  (T1CON<2>) is set, the external clock input is not synchronized. The timer continues to increment asynchronously to the internal phase clocks. The timer will continue to run during SLEEP and can generate an interrupt-on-overflow which will wake-up the processor. However, special precautions in software are needed to read from, or write to the Timer1 register pair (TMR1H:TMR1L) (Section 6.3.2).

In Asynchronous Counter mode, Timer1 cannot be used as a time-base for capture or compare operations.

### 6.3.1 EXTERNAL CLOCK INPUT TIMING WITH UNSYNCHRONIZED CLOCK

If control bit  $\overline{T1SYNC}$  is set, the timer will increment completely asynchronously. The input clock must meet certain minimum high time and low time requirements, as specified in timing parameters 45, 46, and 47.

### 6.3.2 READING AND WRITING TMR1 IN ASYNCHRONOUS COUNTER MODE

Reading TMR1H or TMR1L, while the timer is running from an external asynchronous clock, will ensure a valid read (taken care of in hardware). However, the user should keep in mind that reading the 16-bit timer in two 8-bit values itself, poses certain problems, since the timer may overflow between the reads.

For writes, it is recommended that the user simply stop the timer and write the desired values. A write contention may occur by writing to the timer registers while the register is incrementing. This may produce an unpredictable value in the timer register.

Reading the 16-bit value requires some care. Example 6-1 is an example routine to read the 16-bit timer value. This is useful if the timer cannot be stopped.

#### EXAMPLE 6-1: READING A 16-BIT FREE-RUNNING TIMER

```
; All interrupts are disabled
;
    MOVF    TMR1H, W    ;Read high byte
    MOVWF   TMPH        ;
    MOVF    TMR1L, W    ;Read low byte
    MOVWF   TMPL        ;
    MOVF    TMR1H, W    ;Read high byte
    SUBWF   TMPH, W     ;Sub 1st read with 2nd read
    BTFSC   STATUS, Z   ;Is result = 0
    GOTO    CONTINUE    ;Good 16-bit read
;
; TMR1L may have rolled over between the read of the high and low bytes.
; Reading the high and low bytes now will read a good value.
;
    MOVF    TMR1H, W    ;Read high byte
    MOVWF   TMPH        ;
    MOVF    TMR1L, W    ;Read low byte
    MOVWF   TMPL        ;
; Re-enable the Interrupt (if required)
;
CONTINUE                                ;Continue with your code
```

## 8.0 CAPTURE/COMPARE/PWM (CCP) MODULE

The CCP (Capture/Compare/PWM) module contains a 16-bit register which can operate as a 16-bit capture register, as a 16-bit compare register, or as a PWM master/slave duty cycle register. Table 8-1 shows the timer resources used by the CCP module.

The Capture/Compare/PWM Register1 (CCPR1) is comprised of two 8-bit registers: CCPR1L (low byte) and CCPR1H (high byte). The CCP1CON register controls the operation of CCP1. All three are readable and writable.

Register 8-1 shows the CCP1CON register.

For use of the CCP module, refer to the *Embedded Control Handbook*, "Using the CCP Modules" (AN594).

**TABLE 8-1: CCP MODE - TIMER RESOURCE**

CCP Mode	Timer Resource
Capture	Timer1
Compare	Timer1
PWM	Timer2

### REGISTER 8-1: CCP1CON REGISTER (ADDRESS 17h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **CCP1X:CCP1Y:** PWM Least Significant bits

Capture mode:

Unused

Compare mode:

Unused

PWM mode:

These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPR1L.

bit 3-0 **CCP1M3:CCP1M0:** CCP1 Mode Select bits

0000 = Capture/Compare/PWM disabled (resets CCP1 module)

0100 = Capture mode, every falling edge

0101 = Capture mode, every rising edge

0110 = Capture mode, every 4th rising edge

0111 = Capture mode, every 16th rising edge

1000 = Compare mode, set output on match (bit CCP1IF is set)

1001 = Compare mode, clear output on match (bit CCP1IF is set)

1010 = Compare mode, generate software interrupt-on-match (bit CCP1IF is set, CCP1 pin is unaffected)

1011 = Compare mode, trigger special event (CCP1IF bit is set; CCP1 resets TMR1)

11xx = PWM mode

Legend:

R = Readable bit

W = Writable bit

U = Unimplemented bit, read as '0'

- n = Value at POR

'1' = Bit is set

'0' = Bit is cleared

x = Bit is unknown

# PIC16C925/926

To enable the serial port, SSP enable bit, SSPEN (SSPCON<5>) must be set. To reset or reconfigure SPI mode, clear bit SSPEN, re-initialize the SSPCON register, and then set bit SSPEN. This configures the SDI, SDO, SCK, and  $\overline{SS}$  pins as serial port pins. For the pins to behave as the serial port function, they must have their data direction bits (in the TRISC register) appropriately programmed. That is:

- SDI must have TRISC<4> set
- SDO must have TRISC<5> cleared
- SCK (Master mode) must have TRISC<3> cleared
- SCK (Slave mode) must have TRISC<3> set
- $\overline{SS}$  must have TRISA<5> set and ADCON must be configured such that RA5 is a digital I/O

Any serial port function that is not desired may be overridden by programming the corresponding data direction (TRIS) register to the opposite value. An example would be in Master mode, where you are only sending data (to a display driver), then both SDI and  $\overline{SS}$  could be used as general purpose outputs by clearing their corresponding TRIS register bits.

Figure 9-2 shows a typical connection between two microcontrollers. The master controller (Processor 1) initiates the data transfer by sending the SCK signal. Data is shifted out of both shift registers on their programmed clock edge, and latched on the opposite edge of the clock. Both processors should be programmed to same Clock Polarity (CKP), then both controllers would send and receive data at the same time. Whether the data is meaningful (or dummy data), depends on the application software. This leads to three scenarios for data transmission:

- Master sends data — Slave sends dummy data
- Master sends data — Slave sends data

- Master sends dummy data — Slave sends data

The master can initiate the data transfer at any time because it controls the SCK. The master determines when the slave (Processor 2) is to broadcast data by the firmware protocol.

In Master mode, the data is transmitted/received as soon as the SSPBUF register is written to. If the SPI is only going to receive, the SCK output could be disabled (programmed as an input). The SSPSR register will continue to shift in the signal present on the SDI pin at the programmed clock rate. As each byte is received, it will be loaded into the SSPBUF register as if a normal received byte (interrupts and status bits appropriately set). This could be useful in receiver applications as a “line activity monitor” mode.

In Slave mode, the data is transmitted and received as the external clock pulses appear on SCK. When the last bit is latched, the interrupt flag bit SSPIF (PIR1<3>) is set.

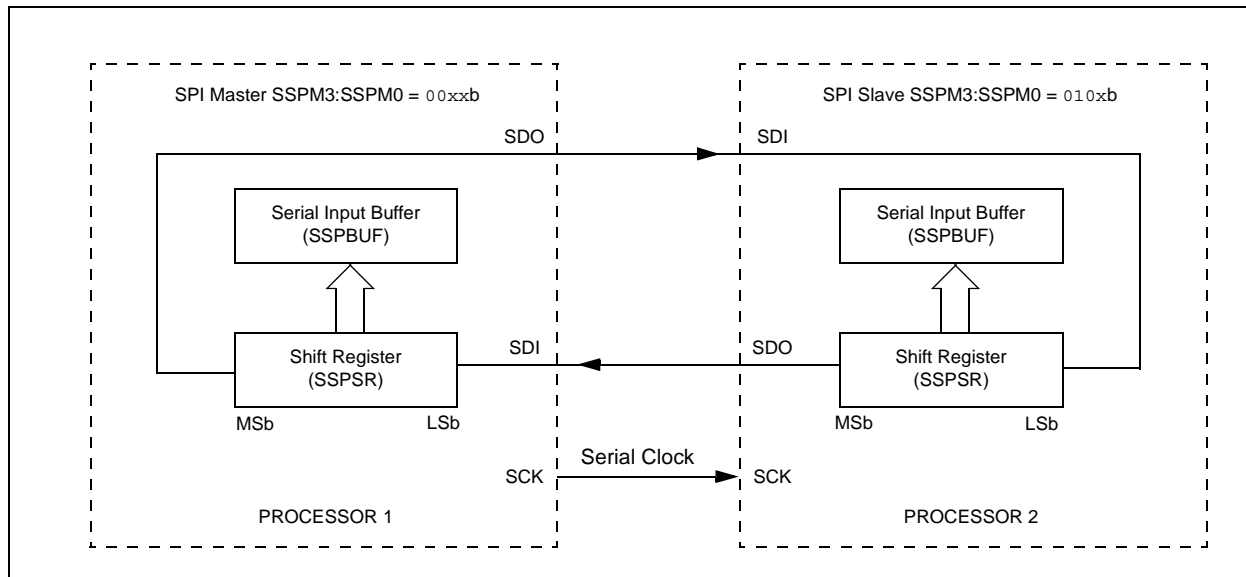
The clock polarity is selected by appropriately programming bit CKP (SSPCON<4>). This then, would give waveforms for SPI communication as shown in Figure 9-3, Figure 9-4, and Figure 9-5, where the MSB is transmitted first. In Master mode, the SPI clock rate (bit rate) is user programmable to be one of the following:

- Fosc/4 (or Tcy)
- Fosc/16 (or 4 • Tcy)
- Fosc/64 (or 16 • Tcy)
- Timer2 output/2

This allows a maximum bit clock frequency (at 8 MHz) of 2 MHz. When in Slave mode, the external clock must meet the minimum high and low times.

In SLEEP mode, the slave can transmit and receive data and wake the device from SLEEP.

**FIGURE 9-2: SPI MASTER/SLAVE CONNECTION**



# PIC16C925/926

## 9.3.2 MASTER MODE

Master mode of operation is supported, in firmware, using interrupt generation on the detection of the START and STOP conditions. The STOP (P) and START (S) bits are cleared from a RESET, or when the SSP module is disabled. The STOP and START bits will toggle based on the START and STOP conditions. Control of the I<sup>2</sup>C bus may be taken when the P bit is set, or the bus is idle with both the S and P bits clear.

In Master mode, the SCL and SDA lines are manipulated by clearing the corresponding TRISC<4:3> bit(s). The output level is always low, irrespective of the value(s) in PORTC<4:3>. So when transmitting data, a '1' data bit must have the TRISC<4> bit set (input) and a '0' data bit must have the TRISC<4> bit cleared (output). The same scenario is true for the SCL line with the TRISC<3> bit.

The following events will cause SSP Interrupt Flag bit, SSPIF, to be set (SSP Interrupt if enabled):

- START condition
- STOP condition
- Data transfer byte transmitted/received

Master mode of operation can be done with either the Slave mode idle (SSPM3:SSPM0 = 1011), or with the slave active. When both Master and Slave modes are enabled, the software needs to differentiate the source(s) of the interrupt.

## 9.3.3 MULTI-MASTER MODE

In Multi-Master mode, the interrupt generation on the detection of the START and STOP conditions allows the determination of when the bus is free. The STOP (P) and START (S) bits are cleared from a RESET or when the SSP module is disabled. The STOP and START bits will toggle based on the START and STOP conditions. Control of the I<sup>2</sup>C bus may be taken when bit P (SSPSTAT<4>) is set, or the bus is idle, with both the S and P bits clear. When the bus is busy, enabling the SSP interrupt will generate the interrupt when the STOP condition occurs.

In multi-master operation, the SDA line must be monitored to see if the signal level is the expected output level. This check only needs to be done when a high level is output. If a high level is expected and a low level is present, the device needs to release the SDA and SCL lines (set TRISC<4:3>). There are two stages where this arbitration can be lost, they are:

- Address Transfer
- Data Transfer

When the slave logic is enabled, the slave continues to receive. If arbitration was lost during the address transfer stage, communication to the device may be in progress. If addressed, an  $\overline{\text{ACK}}$  pulse will be generated. If arbitration was lost during the data transfer stage, the device will need to re-transfer the data at a later time.

**TABLE 9-4: REGISTERS ASSOCIATED WITH I<sup>2</sup>C OPERATION**

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other RESETS
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	LCDIF	ADIF	—	—	SSPIF	CCP1IF	TMR2IF	TMR1IF	00-- 0000	00-- 0000
8Ch	PIE1	LCDIE	ADIE	—	—	SSPIE	CCP1IE	TMR2IE	TMR1IE	00-- 0000	00-- 0000
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	uuuu uuuu
93h	SSPADD	Synchronous Serial Port (I <sup>2</sup> C mode) Address Register								0000 0000	0000 0000
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	0000 0000
94h	SSPSTAT	SMP	CKE	D $\overline{\text{A}}$	P	S	R $\overline{\text{W}}$	UA	BF	0000 0000	0000 0000
87h	TRISC	—	—	PORTC Data Direction Control Register						--11 1111	--11 1111

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by SSP in I<sup>2</sup>C mode.

NOTES:

## BCF Bit Clear f

Syntax:	[ <i>label</i> ] BCF f [,b]			
Operands:	$0 \leq f \leq 127$ $0 \leq b \leq 7$			
Operation:	$0 \rightarrow (f<b>)$			
Status Affected:	None			
Encoding:	01	00bb	bfff	ffff
Description:	Bit 'b' in register 'f' is cleared.			
Words:	1			
Cycles:	1			
Q Cycle Activity:	Q1	Q2	Q3	Q4
	Decode	Read register 'f'	Process data	Write register 'f'

Example      BCF      FLAG\_REG, 7

Before Instruction:

FLAG\_REG = 0xC7

After Instruction:

FLAG\_REG = 0x47

## BSF Bit Set f

Syntax:	[ <i>label</i> ] BSF f [,b]			
Operands:	$0 \leq f \leq 127$ $0 \leq b \leq 7$			
Operation:	$1 \rightarrow (f<b>)$			
Status Affected:	None			
Encoding:	01	01bb	bfff	ffff
Description:	Bit 'b' in register 'f' is set.			
Words:	1			
Cycles:	1			
Q Cycle Activity:	Q1	Q2	Q3	Q4
	Decode	Read register 'f'	Process data	Write register 'f'

Example      BSF      FLAG\_REG, 7

Before Instruction:

FLAG\_REG = 0x0A

After Instruction:

FLAG\_REG = 0x8A

## BTFSC Bit Test, Skip if Clear

Syntax:	[ <i>label</i> ] BTFSC f [,b]			
Operands:	$0 \leq f \leq 127$ $0 \leq b \leq 7$			
Operation:	skip if (f<b>) = 0			
Status Affected:	None			
Encoding:	01	10bb	bfff	ffff
Description:	If bit 'b' in register 'f' is '1', then the next instruction is executed. If bit 'b' in register 'f' is '0', then the next instruction is discarded, and a NOP is executed instead, making this a 2TCY instruction.			
Words:	1			
Cycles:	1(2)			
Q Cycle Activity:	Q1	Q2	Q3	Q4
	Decode	Read register 'f'	Process data	No Operation

If Skip: (2nd Cycle)

Q1	Q2	Q3	Q4
No Operation	No Operation	No Operation	No Operation

Example

```

HERE   BTFSC  FLAG, 1
FALSE  GOTO   PROCESS_CODE
TRUE   •
        •
        •
  
```

Before Instruction:

PC = address HERE

After Instruction:

if FLAG<1> = 0,

PC = address TRUE

if FLAG<1> = 1,

PC = address FALSE



## DECF Decrement f

Syntax:	[ <i>label</i> ] DECF f [,d]			
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$			
Operation:	$(f) - 1 \rightarrow (\text{destination})$			
Status Affected:	Z			
Encoding:	00	0011	dfff	ffff
Description:	Decrement register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.			
Words:	1			
Cycles:	1			
Q Cycle Activity:	Q1	Q2	Q3	Q4

Decode	Read register 'f'	Process data	Write to destination
--------	-------------------	--------------	----------------------

Example      DECF      CNT, 1

Before Instruction:

CNT = 0x01  
Z = 0

After Instruction:

CNT = 0x00  
Z = 1

## DECFSZ Decrement f, Skip if 0

Syntax:	[ <i>label</i> ] DECFSZ f [,d]			
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$			
Operation:	$(f) - 1 \rightarrow (\text{destination});$ skip if result = 0			
Status Affected:	None			
Encoding:	00	1011	dfff	ffff
Description:	The contents of register 'f' are decremented. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'. If the result is 1, the next instruction is executed. If the result is 0, then a NOP is executed instead, making it a 2TCY instruction.			

Words: 1

Cycles: 1(2)

Q Cycle Activity:      Q1      Q2      Q3      Q4

Decode	Read register 'f'	Process data	Write to destination
--------	-------------------	--------------	----------------------

If Skip: (2nd Cycle)

Q1      Q2      Q3      Q4

No Operation	No Operation	No Operation	No Operation
--------------	--------------	--------------	--------------

Example      HERE      DECFSZ      CNT, 1  
                              GOTO      LOOP  
CONTINUE •  
                              •  
                              •

Before Instruction:

PC = address HERE

After Instruction:

CNT = CNT - 1  
if CNT = 0,  
PC = address CONTINUE  
if CNT  $\neq$  0,  
PC = address HERE+1

# PIC16C925/926

## GOTO Unconditional Branch

Syntax:	[ <i>label</i> ] GOTO k			
Operands:	$0 \leq k \leq 2047$			
Operation:	$k \rightarrow PC<10:0>$ $PCLATH<4:3> \rightarrow PC<12:11>$			
Status Affected:	None			
Encoding:	10	1kkk	kkkk	kkkk
Description:	GOTO is an unconditional branch. The eleven-bit immediate value is loaded into PC bits <10:0>. The upper bits of PC are loaded from PCLATH<4:3>. GOTO is a two-cycle instruction.			
Words:	1			
Cycles:	2			
Q Cycle Activity:	Q1	Q2	Q3	Q4
1st Cycle	Decode	Read literal 'k'	Process data	Write to PC
2nd Cycle	No Operation	No Operation	No Operation	No Operation

Example           GOTO THERE

After Instruction:

PC = Address THERE

## INCF Increment f

Syntax:	[ <i>label</i> ] INCF f [,d]			
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$			
Operation:	$(f) + 1 \rightarrow (\text{destination})$			
Status Affected:	Z			
Encoding:	00	1010	dfff	ffff
Description:	The contents of register 'f' are incremented. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'.			
Words:	1			
Cycles:	1			
Q Cycle Activity:	Q1	Q2	Q3	Q4
	Decode	Read register 'f'	Process data	Write to destination

Example           INCF     CNT, 1

Before Instruction:

CNT = 0xFF

Z = 0

After Instruction:

CNT = 0x00

Z = 1

## SUBLW Subtract W from Literal

Syntax: [label] SUBLW k

Operands:  $0 \leq k \leq 255$

Operation:  $k - (W) \rightarrow (W)$

Status Affected: C, DC, Z

Encoding:

11	110x	kkkk	kkkk
----	------	------	------

Description: The W register is subtracted (2's complement method) from the eight-bit literal 'k'. The result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W

Example 1: SUBLW 0x02

Before Instruction:

W = 1

C = ?

Z = ?

After Instruction:

W = 1

C = 1; result is positive

Z = 0

Example 2:

Before Instruction:

W = 2

C = ?

Z = ?

After Instruction:

W = 0

C = 1; result is zero

Z = 1

Example 3:

Before Instruction:

W = 3

C = ?

Z = ?

After Instruction:

W = 0xFF

C = 0; result is negative

Z = 0

## SUBWF Subtract W from f

Syntax: [label] SUBWF f[,d]

Operands:  $0 \leq f \leq 127$   
 $d \in [0,1]$

Operation:  $(f) - (W) \rightarrow (\text{destination})$

Status Affected: C, DC, Z

Encoding:

00	0010	dfff	ffff
----	------	------	------

Description: Subtract (2's complement method) W register from register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1: SUBWF REG1, 1

Before Instruction:

REG1 = 3

W = 2

C = ?

Z = ?

After Instruction:

REG1 = 1

W = 2

C = 1; result is positive

Z = 0

Example 2:

Before Instruction:

REG1 = 2

W = 2

C = ?

Z = ?

After Instruction:

REG1 = 0

W = 2

C = 1; result is zero

Z = 1

Example 3:

Before Instruction:

REG1 = 1

W = 2

C = ?

Z = ?

After Instruction:

REG1 = 0xFF

W = 2

C = 0; result is negative

Z = 0

TABLE 14-1: DEVELOPMENT TOOLS FROM MICROCHIP

	PIC12CXXX	PIC14000	PIC16C5X	PIC16C6X	PIC16CXXX	PIC16F62X	PIC16C7X	PIC16C7XX	PIC16C8X	PIC16F8XX	PIC16C9XX	PIC17C4X	PIC17C7XX	PIC18CXX2	24CXX/ 25CXX/ 93CXX	HCXXX	MCRFXXX	MCP2510
Tools	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓				
Software Tools	MPLAB® Integrated Development Environment																	
Software Tools	MPLAB® C17 C Compiler																	
Software Tools	MPLAB® C18 C Compiler																	
Emulators	MPASM™ Assembler/ MPLINK™ Object Linker	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
Emulators	MPLAB® ICE In-Circuit Emulator	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
Emulators	ICEPIC™ In-Circuit Emulator	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
Debugger	MPLAB® ICD In-Circuit Debugger			✓*			✓*			✓								
Programmers	PICSTART® Plus Entry Level Development Programmer	✓	✓	✓	✓	✓**	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
Programmers	PRO MATE® II Universal Device Programmer	✓	✓	✓	✓	✓**	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
Demo Boards and Eval Kits	PICDEM™ 1 Demonstration Board		✓				†		✓			✓						
Demo Boards and Eval Kits	PICDEM™ 2 Demonstration Board				†		†							✓				
Demo Boards and Eval Kits	PICDEM™ 3 Demonstration Board										✓							
Demo Boards and Eval Kits	PICDEM™ 14A Demonstration Board	✓																
Demo Boards and Eval Kits	PICDEM™ 17 Demonstration Board											✓						
Demo Boards and Eval Kits	KEELOQ® Evaluation Kit														✓			
Demo Boards and Eval Kits	KEELOQ® Transponder Kit														✓			
Demo Boards and Eval Kits	microID™ Programmer's Kit																✓	
Demo Boards and Eval Kits	125 kHz microID™ Developer's Kit																✓	
Demo Boards and Eval Kits	125 kHz Anticollision microID™ Developer's Kit																✓	
Demo Boards and Eval Kits	13.56 MHz Anticollision microID™ Developer's Kit																✓	
Demo Boards and Eval Kits	MCP2510 CAN Developer's Kit																✓	✓

\* Contact the Microchip Technology Inc. web site at [www.microchip.com](http://www.microchip.com) for information on how to use the MPLAB® ICD In-Circuit Debugger (DV164001) with PIC16C62, 63, 64, 65, 72, 73, 74, 76, 77.

\*\* Contact Microchip Technology Inc. for availability date.

† Development tool is available on select devices.

# PIC16C925/926

**TABLE 15-8: SPI MODE REQUIREMENTS**

Param No.	Symbol	Characteristic	Min	Typ†	Max	Units	Conditions
70	TssL2scH, TssL2scL	$\overline{SS}\downarrow$ to SCK $\downarrow$ or SCK $\uparrow$ input	Tcy	—	—	ns	
71	TscH	SCK input high time (Slave mode)	Continuous	1.25Tcy + 30	—	—	ns
71A		Single Byte	40	—	—	ns	
72	TscL	SCK input low time (Slave mode)	Continuous	1.25Tcy + 30	—	—	ns
72A		Single Byte	40	—	—	ns	
73	TdiV2scH, TdiV2scL	Setup time of SDI data input to SCK edge	50	—	—	ns	
74	Tsch2diL, TscL2diL	Hold time of SDI data input to SCK edge	50	—	—	ns	
75	TdoR	SDO data output rise time	—	10	25	ns	
76	TdoF	SDO data output fall time	—	10	25	ns	
77	TssH2doZ	$\overline{SS}\uparrow$ to SDO output hi-impedance	10	—	50	ns	
78	TscR	SCK output rise time (Master mode)	—	10	25	ns	
79	TscF	SCK output fall time (Master mode)	—	10	25	ns	
80	Tsch2doV, TscL2doV	SDO data output valid after SCK edge	—	—	50	ns	
81	TdoV2scH, TdoV2scL	SDO data output setup to SCK edge	Tcy	—	—	ns	
82	TssL2doV	SDO data output valid after $\overline{SS}\downarrow$ edge	—	—	50	ns	
83	Tsch2ssH, TscL2ssH	$\overline{SS}\uparrow$ after SCK edge	1.5Tcy + 40	—	—	ns	
84	Tb2b	Delay between consecutive bytes	1.5Tcy + 40	—	—	ns	

† Data in "Typ" column is at 5V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.