Welcome to 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	Obsolete
Core Processor	PIC
Core Size	8-Bit
Speed	40MHz
Connectivity	CANbus, I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, LVD, POR, PWM, WDT
Number of I/O	52
Program Memory Size	32KB (16K x 16)
Program Memory Type	EPROM, UV
EEPROM Size	-
RAM Size	1.5K x 8
Voltage - Supply (Vcc/Vdd)	2.5V ~ 5.5V
Data Converters	A/D 12x10b
Oscillator Type	External
Operating Temperature	0°C ~ 70°C (TA)
Mounting Type	Surface Mount
Package / Case	68-CLCC Window (J-Lead)
Supplier Device Package	68-CLCC (24.13x24.13)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18lc658-cl

Filename	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other RESETS(3)
TXB2D7	TXB2D77	TXB2D76	TXB2D75	TXB2D74	TXB2D73	TXB2D72	TXB2D71	TXB2D70	xxxx xxxx	uuuu uuuu
TXB2D6	TXB2D67	TXB2D66	TXB2D65	TXB2D64	TXB2D63	TXB2D62	TXB2D61	TXB2D60	xxxx xxxx	uuuu uuuu
TXB2D5	TXB2D57	TXB2D56	TXB2D55	TXB2D54	TXB2D53	TXB2D52	TXB2D51	TXB2D50	xxxx xxxx	uuuu uuuu
TXB2D4	TXB2D47	TXB2D46	TXB2D45	TXB2D44	TXB2D43	TXB2D42	TXB2D41	TXB2D40	xxxx xxxx	uuuu uuuu
TXB2D3	TXB2D37	TXB2D36	TXB2D35	TXB2D34	TXB2D33	TXB2D32	TXB2D31	TXB2D30	xxxx xxxx	uuuu uuuu
TXB2D2	TXB2D27	TXB2D26	TXB2D25	TXB2D24	TXB2D23	TXB2D22	TXB2D21	TXB2D20	xxxx xxxx	uuuu uuuu
TXB2D1	TXB2D17	TXB2D16	TXB2D15	TXB2D14	TXB2D13	TXB2D12	TXB2D11	TXB2D10	xxxx xxxx	uuuu uuuu
TXB2D0	TXB2D07	TXB2D06	TXB2D05	TXB2D04	TXB2D03	TXB2D02	TXB2D01	TXB2D00	xxxx xxxx	uuuu uuuu
TXB2DLC	—	TXRTR	—	—	DLC3	DLC2	DLC1	DLC0	0x00 xxxx	0u00 uuuu
TXB2EIDL	EID7	EID6	EID5	EID4	EID3	EID2	EID1	EID0	xxxx xxxx	uuuu uuuu
TXB2EIDH	EID15	EID14	EID13	EID12	EID11	EID10	EID9	EID8	xxxx xxxx	uuuu uuuu
TXB2SIDL	SID2	SID1	SID0	—	EXIDEN	—	EID17	EID16	xxx0 x0xx	uuu0 u0uu
TXB2SIDH	SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3	xxxx xxxx	uuuu uuuu
TXB2CON	—	TXABT	TXLARB	TXERR	TXREQ	—	TXPRI1	TXPRI0	0000 0000	0000 0000
RXM1EIDL	EID7	EID6	EID5	EID4	EID3	EID2	EID1	EID0	xxxx xxxx	uuuu uuuu
RXM1EIDH	EID15	EID14	EID13	EID12	EID11	EID10	EID9	EID8	xxxx xxxx	uuuu uuuu
RXM1SIDL	SID2	SID1	SID0	—	—	—	EID17	EID16	xxx- --xx	uuu- --uu
RXM1SIDH	SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3	xxxx xxxx	uuuu uuuu
RXM0EIDL	EID7	EID6	EID5	EID4	EID3	EID2	EID1	EID0	xxxx xxxx	uuuu uuuu
RXM0EIDH	EID15	EID14	EID13	EID12	EID11	EID10	EID9	EID8	xxxx xxxx	uuuu uuuu
RXM0SIDL	SID2	SID1	SID0	—	—	—	EID17	EID16	xxx- --xx	uuu- --uu
RXM0SIDH	SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3	xxxx xxxx	uuuu uuuu
RXF5EID0	EID7	EID6	EID5	EID4	EID3	EID2	EID1	EID0	xxxx xxxx	uuuu uuuu
RXF5EID8	EID15	EID14	EID13	EID12	EID11	EID10	EID9	EID8	xxxx xxxx	uuuu uuuu
RXF5SIDL	SID2	SID1	SID0	—	EXIDEN	—	EID17	EID16	xxx- x-xx	uuu- u-uu
RXF5SIDH	SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3	xxxx xxxx	uuuu uuuu
RXF4EID0	EID7	EID6	EID5	EID4	EID3	EID2	EID1	EID0	xxxx xxxx	uuuu uuuu
RXF4EID8	EID15	EID14	EID13	EID12	EID11	EID10	EID9	EID8	xxxx xxxx	uuuu uuuu
RXF4SIDL	SID2	SID1	SID0	—	EXIDEN	—	EID17	EID16	xxx- x-xx	uuu- u-uu
RXF4SIDH	SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3	xxxx xxxx	uuuu uuuu
RXF3EID0	EID7	EID6	EID5	EID4	EID3	EID2	EID1	EID0	xxxx xxxx	uuuu uuuu
RXF3EID8	EID15	EID14	EID13	EID12	EID11	EID10	EID9	EID8	xxxx xxxx	uuuu uuuu
RXF3SIDL	SID2	SID1	SID0	—	EXIDEN	—	EID17	EID16	xxx- x-xx	uuu- u-uu
RXF3SIDH	SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3	xxxx xxxx	uuuu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented, q = value depends on condition

Note 1: Bit 6 of PORTA, LATA and TRISA are enabled in ECIO and RCIO oscillator modes only. In all other oscillator modes, they are disabled and read '0'.

2: Bit 21 of the TBLPTRU allows access to the device configuration bits.

3: Other (non-power-up) RESETs include external RESET through MCLR and Watchdog Timer Reset.

4: These registers are reserved on PIC18C658.

4.13.1 RCON REGISTER

The Reset Control (RCON) register contains flag bits that allow differentiation between the sources of a device RESET. These flags include the $\overline{TO}$, $\overline{PD}$, $\overline{POR}$, $\overline{BOR}$ and $\overline{RI}$ bits. This register is readable and writable.

Note 1: If the BOREN configuration bit is set, $\overline{BOR}$ is '1' on Power-on Reset. If the BOREN configuration bit is clear, $\overline{BOR}$ is unknown on Power-on Reset.
The $\overline{BOR}$ status bit is a "don't care" and is not necessarily predictable if the brown-out circuit is disabled (the BOREN configuration bit is clear). $\overline{BOR}$ must then be set by the user and checked on subsequent RESETs to see if it is clear, indicating a brown-out has occurred.

2: It is recommended that the $\overline{POR}$ bit be set after a Power-on Reset has been detected, so that subsequent Power-on Resets may be detected.

REGISTER 4-3: RCON REGISTER

R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1	R/W-0	R/W-0
IPEN	LWRT	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

bit 7 **IPEN:** Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (16CXXX compatibility mode)

bit 6 **LWRT:** Long Write Enable bit
1 = Enable TBLWT to internal program memory
Once this bit is set, it can only be cleared by a POR or $\overline{MCLR}$ Reset
0 = Disable TBLWT to internal program memory; TBLWT only to external program memory

bit 5 **Unimplemented:** Read as '0'

bit 4 **$\overline{RI}$:** RESET Instruction Flag bit
1 = The RESET instruction was not executed
0 = The RESET instruction was executed causing a device RESET
(must be set in software after a Brown-out Reset occurs)

bit 3 **$\overline{TO}$:** Watchdog Time-out Flag bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 2 **$\overline{PD}$:** Power-down Detection Flag bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 1 **$\overline{POR}$:** Power-on Reset Status bit
1 = A Power-on Reset has not occurred
0 = A Power-on Reset occurred
(must be set in software after a Power-on Reset occurs)

bit 0 **$\overline{BOR}$:** Brown-out Reset Status bit
1 = A Brown-out Reset has not occurred
0 = A Brown-out Reset occurred
(must be set in software after a Brown-out Reset occurs)

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

TABLE 8-13: PORTG FUNCTIONS

Name	Bit#	Buffer Type	Function
RG0/CANTX0	bit0	ST	Input/output port pin or CAN bus transmit output.
RG1/CANTX1	bit1	ST	Input/output port pin or CAN bus complimentary transmit output or CAN bus bit time clock.
RG2/CANRX	bit2	ST	Input/output port pin or CAN bus receive input.
RG3	bit3	ST	Input/output port pin.
RG4	bit4	ST	Input/output port pin.

Legend: ST = Schmitt Trigger input

Note: Refer to "CAN Module", Section 17.0 for usage of CAN pin functions.

TABLE 8-14: SUMMARY OF REGISTERS ASSOCIATED WITH PORTG

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
TRISG	PORTG Data Direction Control Register								---1 1111	---1 1111
PORTG	Read PORTG pin / Write PORTG Data Latch								---x xxxx	---u uuuu
LATG	Read PORTG Data Latch/Write PORTG Data Latch								---x xxxx	---u uuuu
CIOCON	TX1SRC	TX1EN	ENDRHI	CANCAP	—	—	—	—	0000 ----	0000 ----

Legend: x = unknown, u = unchanged

FIGURE 9-2: PARALLEL SLAVE PORT WRITE WAVEFORMS

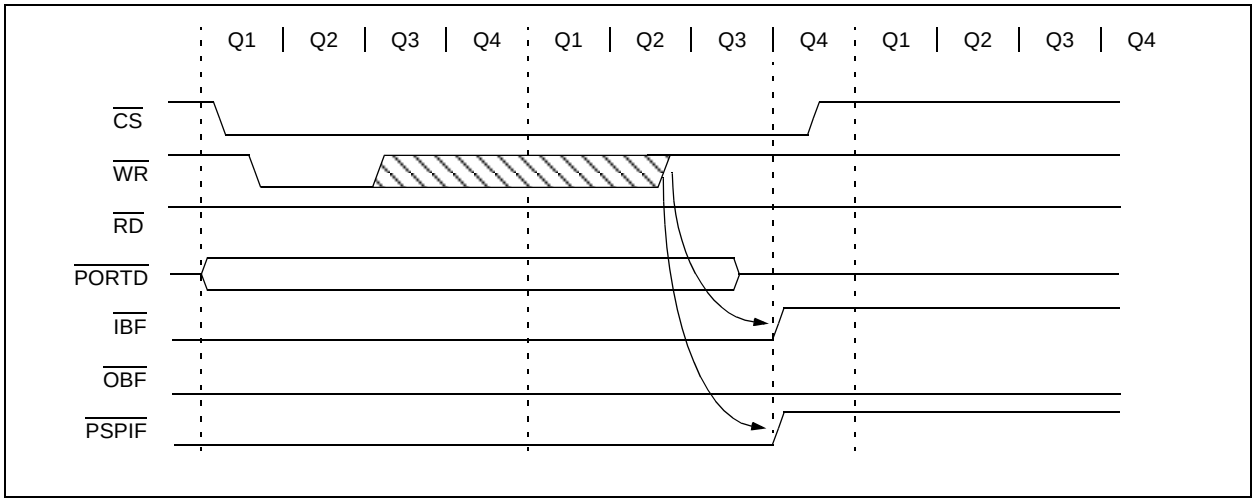


FIGURE 9-3: PARALLEL SLAVE PORT READ WAVEFORMS

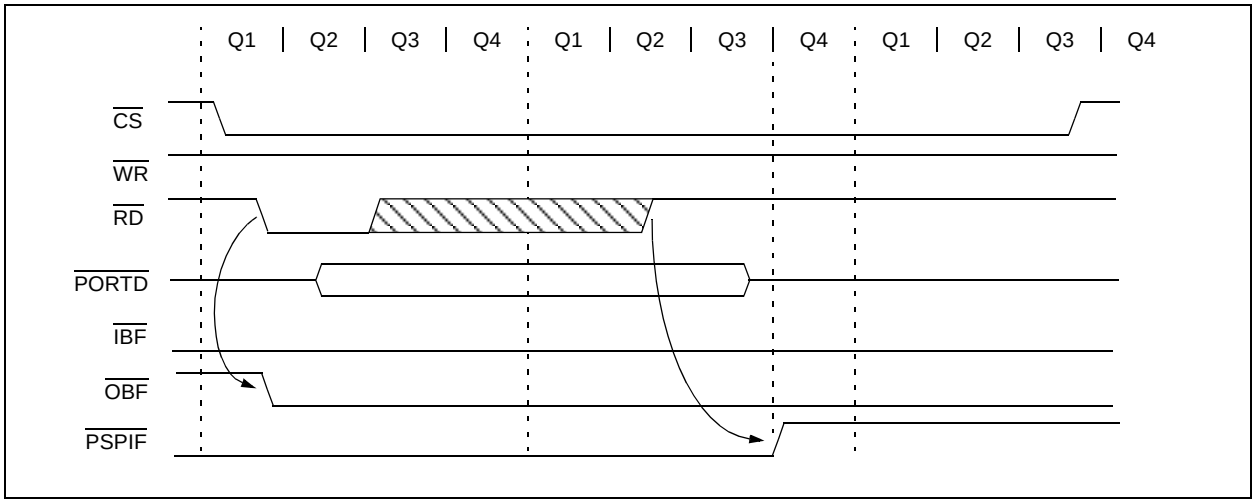


TABLE 9-1: REGISTERS ASSOCIATED WITH PARALLEL SLAVE PORT

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other RESETS
PORTD	Port data latch when written; port pins when read								xxxx xxxx	uuuu uuuu
LATD	LATD Data Output Bits								xxxx xxxx	uuuu uuuu
TRISD	PORTD Data Direction Bits								1111 1111	1111 1111
PORTE	RE7	RE6	RE5	RE4	RE3	RE2	RE1	RE0	0000 0000	0000 0000
LATE	LATE Data Output Bits								xxxx xxxx	uuuu uuuu
TRISE	PORTE Data Direction Bits								1111 1111	1111 1111
INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
IPR1	PSPIP	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	0000 0000	0000 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'.
Shaded cells are not used by the Parallel Slave Port.

13.0 TIMER3 MODULE

The Timer3 module timer/counter has the following features:

- 16-bit timer/counter
  (Two 8-bit registers: TMR3H and TMR3L)
- Readable and writable (both registers)
- Internal or external clock select
- Interrupt on overflow from FFFFh to 0000h
- RESET from CCP module trigger

Figure 13-1 is a simplified block diagram of the Timer3 module.

Register 13-1 shows the Timer3 Control Register. This register controls the operating mode of the Timer3 module and sets the CCP clock source.

Register 11-1 shows the Timer1 Control register. This register controls the operating mode of the Timer1 module, as well as contains the Timer1 oscillator enable bit (T1OSCEN), which can be a clock source for Timer3.

Note:	Timer3 is disabled on POR.

REGISTER 13-1: T3CON REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON
bit 7							bit 0

bit 7 **RD16:** 16-bit Read/Write Mode Enable
1 = Enables register Read/Write of Timer3 in one 16-bit operation
0 = Enables register Read/Write of Timer3 in two 8-bit operations

bit 6,3 **T3CCP2:T3CCP1:** Timer3 and Timer1 to CCPx Enable bits
1x = Timer3 is the clock source for compare/capture CCP modules
01 = Timer3 is the clock source for compare/capture of CCP2,
Timer1 is the clock source for compare/capture of CCP1
00 = Timer1 is the clock source for compare/capture CCP modules

bit 5-4 **T3CKPS1:T3CKPS0**: Timer3 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 2 **$\overline{\text{T3SYNC}}$:** Timer3 External Clock Input Synchronization Control bit
(Not usable if the system clock comes from Timer1/Timer3)

When TMR3CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input

When TMR3CS = 0:
This bit is ignored. Timer3 uses the internal clock when TMR3CS = 0.

bit 1 **TMR3CS:** Timer3 Clock Source Select bit
1 = External clock input from Timer1 oscillator or T1CKI (on the rising edge after the first falling edge)
0 = Internal clock (Fosc/4)

bit 0 **TMR3ON:** Timer3 On bit
1 = Enables Timer3
0 = Stops Timer3

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

14.0 CAPTURE/COMPARE/PWM (CCP) MODULES

Each CCP (Capture/Compare/PWM) module contains a 16-bit register that can operate as a 16-bit capture register, as a 16-bit compare register, or as a PWM Duty Cycle register. Table 14-1 shows the timer resources of the CCP module modes.

The operation of CCP1 is identical to that of CCP2, with the exception of the special event trigger and the CAN message timestamp received. (Refer to “CAN Module”, Section 17.0 for CAN operation.) Therefore, operation of a CCP module in the following sections is described with respect to CCP1.

Table 14-2 shows the interaction of the CCP modules.

Register 14-1 shows the CCPx Control registers (CCPxCON). For the CCP1 module, the register is called CCP1CON and for the CCP2 module, the register is called CCP2CON.

REGISTER 14-1: CCP1CON REGISTER
CCP2CON REGISTER

	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CCP1CON	—	—	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0
	bit 7							bit 0

	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CCP2CON	—	—	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0
	bit 7							bit 0

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

bit 5-4 **DCxB1:DCxB0**: PWM Duty Cycle bit1 and bit0

Capture Mode:
Unused

Compare Mode:
Unused

PWM Mode:
These bits are the two LSbs (bit1 and bit0) of the 10-bit PWM duty cycle. The upper eight bits (DCx9:DCx2) of the duty cycle are found in CCPRxL.

bit 3-0 **CCPxM3:CCPxM0**: CCPx Mode Select bits

0000 = Capture/Compare/PWM off (resets CCPx module)
0001 = Reserved
0010 = Compare mode, toggle output on match (CCPxIF bit is set)
0011 = Capture mode, CAN message received (CCP1 only)
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,
Initialize CCP pin Low, on compare match force CCP pin High (CCPIF bit is set)
1001 = Compare mode,
Initialize CCP pin High, on compare match force CCP pin Low (CCPIF bit is set)
1010 = Compare mode,
Generate software interrupt on compare match
(CCPIF bit is set, CCP pin is unaffected)
1011 = Compare mode,
Trigger special event (CCPIF bit is set, reset TMR1 or TMR3)
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

When the application software is expecting to receive valid data, the SSPBUF should be read before the next byte of data to transfer is written to the SSPBUF. The buffer full (BF) bit (SSPSTAT register) indicates when SSPBUF has been loaded with the received data (transmission is complete). When the SSPBUF is read, the BF bit is cleared. This data may be irrelevant if the SPI is only a transmitter. Generally, the MSSP Interrupt is used to determine when the transmission/reception has completed. The SSPBUF must be read and/or written. If the interrupt method is not going to be used, then software polling can be done to ensure that a write collision does not occur. Example 15-1 shows the loading of the SSPBUF (SSPSR) for data transmission.

The SSPSR is not directly readable or writable, and can only be accessed by addressing the SSPBUF register. Additionally, the MSSP status register (SSPSTAT register) indicates the various status conditions.

15.3.2 ENABLING SPI I/O

To enable the serial port, SSP Enable bit, SSPEN (SSPCON1 register), must be set. To reset or reconfigure SPI mode, clear the SSPEN bit, re-initialize the SSPCON registers, and then set the SSPEN bit. This configures the SDI, SDO, SCK, and $\overline{SS}$ pins as serial port pins. For the pins to behave as the serial port function, some must have their data direction bits (in the TRIS register) appropriately programmed. That is:

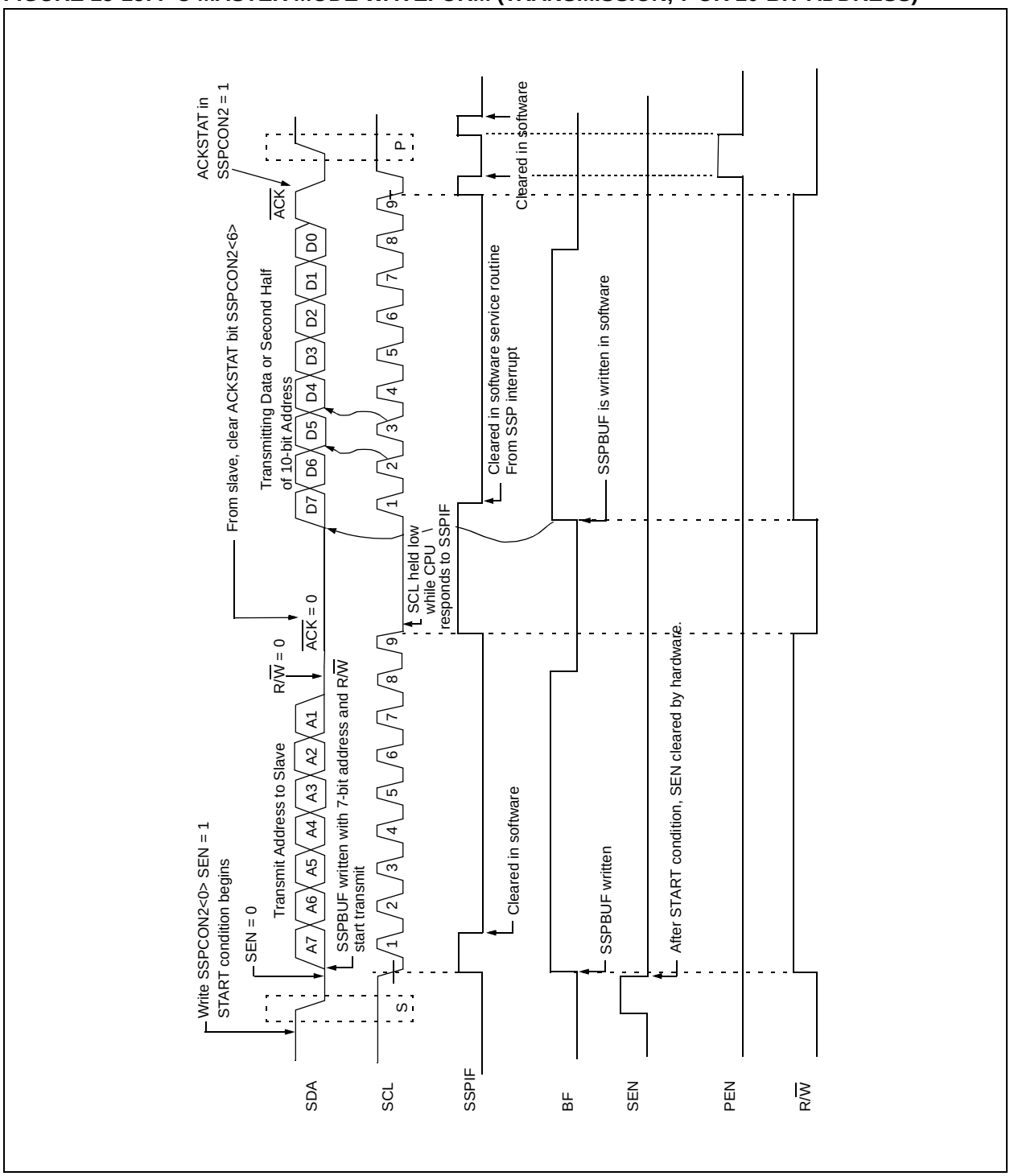
- SDI is automatically controlled by the SPI module
- SDO must have TRISC<5> bit cleared
- SCK (Master mode) must have TRISC<3> bit cleared
- SCK (Slave mode) must have TRISC<3> bit set
- $\overline{SS}$ must have TRISC<4> bit set

Any serial port function that is not desired may be overridden by programming the corresponding data direction (TRIS) register to the opposite value.

EXAMPLE 15-1: LOADING THE SSPBUF (SSPSR) REGISTER

```
LOOP BTFSS SSPSTAT, BF    ;Has data been received (transmit complete)?
     GOTO  LOOP           ;No
     MOVF  SSPBUF, W      ;WREG reg = contents of SSPBUF
     MOVWF RXDATA         ;Save in user RAM, if data is meaningful
     MOVF  TXDATA, W      ;W reg = contents of TXDATA
     MOVWF SSPBUF         ;New data to xmit
```


FIGURE 15-15: I²C MASTER MODE WAVEFORM (TRANSMISSION, 7 OR 10-BIT ADDRESS)



15.4.16.1 Bus Collision During a START Condition

During a START condition, a bus collision occurs if:

a) SDA or SCL are sampled low at the beginning of the START condition (Figure 15-21).

b) SCL is sampled low before SDA is asserted low (Figure 15-22).

During a START condition, both the SDA and the SCL pins are monitored.

If:

the SDA pin is already low
or the SCL pin is already low,

then:

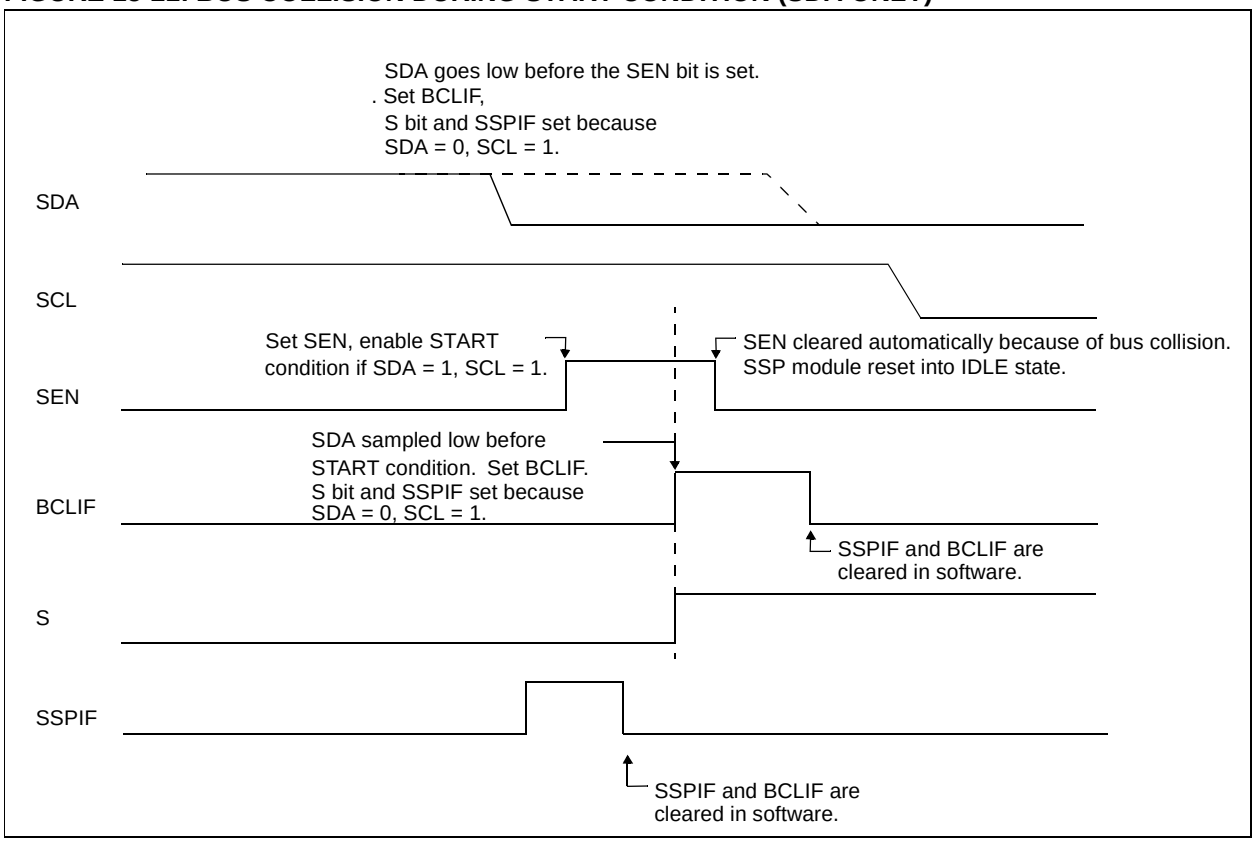
the START condition is aborted,
and the BCLIF flag is set,
and the MSSP module is reset to its IDLE state (Figure 15-21).

The START condition begins with the SDA and SCL pins de-asserted. When the SDA pin is sampled high, the baud rate generator is loaded from SSPADD<6:0> and counts down to 0. If the SCL pin is sampled low while SDA is high, a bus collision occurs, because it is assumed that another master is attempting to drive a data '1' during the START condition.

If the SDA pin is sampled low during this count, the BRG is reset and the SDA line is asserted early (Figure 15-23). If, however, a '1' is sampled on the SDA pin, the SDA pin is asserted low at the end of the BRG count. The baud rate generator is then reloaded and counts down to 0, and during this time, if the SCL pin is sampled as '0', a bus collision does not occur. At the end of the BRG count, the SCL pin is asserted low.

Note: The reason that bus collision is not a factor during a START condition is that no two bus masters can assert a START condition at the exact same time. Therefore, one master will always assert SDA before the other. This condition does not cause a bus collision, because the two masters must be allowed to arbitrate the first address following the START condition. If the address is the same, arbitration must be allowed to continue into the data portion, Repeated START or STOP conditions.

FIGURE 15-21: BUS COLLISION DURING START CONDITION (SDA ONLY)



22.3.2 WAKE-UP USING INTERRUPTS

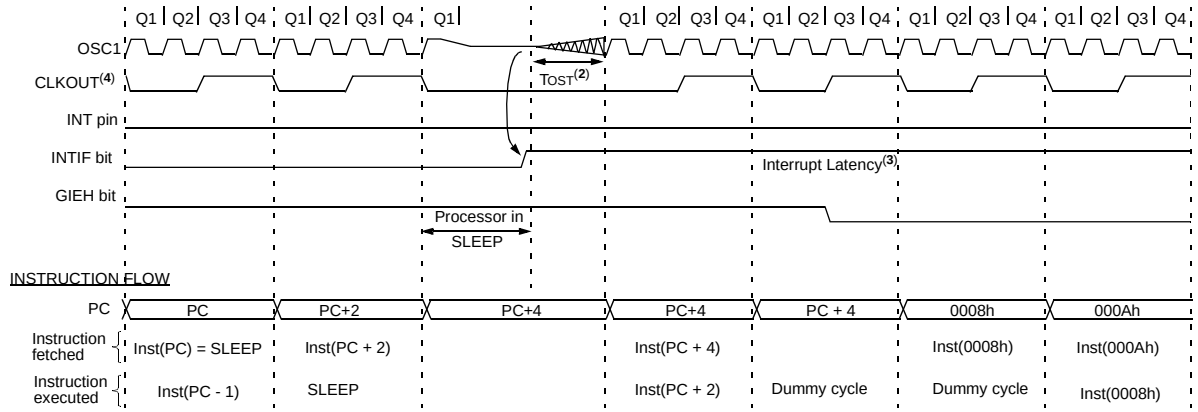
When global interrupts are disabled (GIE cleared) and any interrupt source has both its interrupt enable bit and interrupt flag bit set, one of the following will occur:

- If an interrupt condition (interrupt flag bit and interrupt enable bits are set) occurs **before** the execution of a SLEEP instruction, the SLEEP instruction will complete as a NOP. Therefore, the WDT and WDT postscaler will not be cleared, the $\overline{TO}$ bit will not be set and $\overline{PD}$ bits will not be cleared.
- If the interrupt condition occurs **during or after** the execution of a SLEEP instruction, the device will immediately wake-up from sleep. The SLEEP instruction will be completely executed before the wake-up. Therefore, the WDT and WDT postscaler will be cleared, the $\overline{TO}$ bit will be set and the $\overline{PD}$ bit will be cleared.

Even if the flag bits were checked before executing a SLEEP instruction, it may be possible for flag bits to become set before the SLEEP instruction completes. To determine whether a SLEEP instruction executed, test the $\overline{PD}$ bit. If the $\overline{PD}$ bit is set, the SLEEP instruction was executed as a NOP.

To ensure that the WDT is cleared, a CLRWDT instruction should be executed before a SLEEP instruction.

FIGURE 22-2: WAKE-UP FROM SLEEP THROUGH INTERRUPT(1,2)



Note 1: XT, HS or LP oscillator mode assumed.
2: GIE set is assumed. In this case, after wake- up, the processor jumps to the interrupt routine. If GIE is cleared, execution will continue in-line.
3: TOST = 1024TOSC (drawing not to scale). This delay will not occur for RC and EC osc modes.
4: CLKOUT is not available in these oscillator modes, but shown here for timing reference.

22.4 Program Verification/Code Protection

If the code protection bit(s) have not been programmed, the on-chip program memory can be read out for verification purposes.

Note:	Microchip Technology does not recommend code protecting windowed devices.

22.5 ID Locations

Five memory locations (200000h - 200004h) are designated as ID locations, where the user can store checksum or other code identification numbers. These locations are accessible during normal execution through the TBLRD instruction, or during program/verify. The ID locations can be read when the device is code protected.

22.6 In-Circuit Serial Programming

PIC18CXX8 microcontrollers can be serially programmed while in the end application circuit. This is simply done with two lines for clock and data, and three other lines for power, ground and the programming voltage. This allows customers to manufacture boards with unprogrammed devices, and then program the microcontroller just before shipping the product. This also allows the most recent firmware or a custom firmware to be programmed.

22.7 Device ID Bits

Device ID bits are located in program memory at 3FFFFEh and 3FFFFFh. The Device ID bits are used by programmers to retrieve part number and revision information about a device. These registers may also be accessed using a TBLRD instruction (Register 22-8 and Register 22-7).

REGISTER 22-7: DEVID1 ID REGISTER FOR THE PIC18CXX8 DEVICE (0x3FFFFE)

R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
DEV2	DEV1	DEV0	REV4	REV3	REV2	REV1	REV0
bit 7							bit 0

bit 7-5 **DEV2:DEV0**: Device ID bits
These bits are used with the DEV10:DEV3 bits in the Device ID register 2 to identify the part number

bit 4-0 **REV4:REV0**: Revision ID bits
These bits are used to indicate the revision of the device

Legend:	
R = Readable bit	P = Programmable bit
U = Unimplemented bit, read as ‘0’	- n = Unprogrammed Value (x = unknown)

REGISTER 22-8: DEVID2 ID REGISTER FOR THE PIC18CXX8 DEVICE (0x3FFFFF)

R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
DEV10	DEV9	DEV8	DEV7	DEV6	DEV5	DEV4	DEV3
bit 7							bit 0

bit 7-0 **DEV10:DEV3**: Device ID bits
These bits are used with the DEV2:DEV0 bits in the Device ID register 1 to identify the part number

Legend:	
R = Readable bit	P = Programmable bit
U = Unimplemented bit, read as ‘0’	- n = Unprogrammed Value (x = unknown)

TABLE 23-2: PIC18CXX8 INSTRUCTION SET (CONTINUED)

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb	LSb	Status Affected	Notes
CONTROL OPERATIONS							
BC	n	Branch if Carry	1 (2)	1110 0010 nnnn nnnn		None	
BN	n	Branch if Negative	1 (2)	1110 0110 nnnn nnnn		None	
BNC	n	Branch if Not Carry	1 (2)	1110 0011 nnnn nnnn		None	
BNN	n	Branch if Not Negative	1 (2)	1110 0111 nnnn nnnn		None	
BNOV	n	Branch if Not Overflow	1 (2)	1110 0101 nnnn nnnn		None	
BNZ	n	Branch if Not Zero	2	1110 0001 nnnn nnnn		None	
BOV	n	Branch if Overflow	1 (2)	1110 0100 nnnn nnnn		None	
BRA	n	Branch Unconditionally	1 (2)	1101 0nnn nnnn nnnn		None	
BZ	n	Branch if Zero	1 (2)	1110 0000 nnnn nnnn		None	
CALL	n, s	Call subroutine1st word	2	1110 110s kkkk kkkk		None	
		2nd word		1111 kkkk kkkk kkkk			
CLRWDT	—	Clear Watchdog Timer	1	0000 0000 0000 0100		$\overline{TO}$, $\overline{PD}$	
DAW	—	Decimal Adjust WREG	1	0000 0000 0000 0111		C	
GOTO	n	Go to address1st word	2	1110 1111 kkkk kkkk		None	
		2nd word		1111 kkkk kkkk kkkk			
NOP	—	No Operation	1	0000 0000 0000 0000		None	
NOP	—	No Operation **(Note 4)**	1	1111 xxxx xxxx xxxx		None	
POP	—	Pop top of return stack (TOS)	1	0000 0000 0000 0110		None	
PUSH	—	Push top of return stack (TOS)	1	0000 0000 0000 0101		None	
RCALL	n	Relative Call	2	1101 1nnn nnnn nnnn		None	
RESET		Software device RESET	1	0000 0000 1111 1111		All	
RETFIE	s	Return from interrupt enable	2	0000 0000 0001 000s		GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with literal in WREG	2	0000 1100 kkkk kkkk		None	
RETURN	s	Return from Subroutine	2	0000 0000 0001 001s		None	
SLEEP	—	Go into Standby mode	1	0000 0000 0000 0011		$\overline{TO}$, $\overline{PD}$	

Note 1: When a PORT register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

4: Some instructions are 2 word instructions. The second word of these instructions will be executed as a `NOP`, unless the first word of the instruction retrieves the information embedded in these 16-bits. This ensures that all program memory locations have a valid instruction.

5: If the table write starts the write cycle to internal memory, the write will continue until terminated.

6: Microchip Assembler MASM automatically defaults destination bit 'd' to '1', while access bit 'a' defaults to '1' or '0' according to address of register being used.

ADDWFC **ADD WREG and Carry bit to f**

Syntax: [*label*] ADDWFC f [,d [,a]]

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (WREG) + (f) + (C) → dest

Status Affected: N,OV, C, DC, Z

Encoding:

0010	00da	ffff	ffff

Description: Add WREG, the Carry Flag and data memory location 'f'. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed in data memory location 'f'. If 'a' is 0, the Access Bank will be selected. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `ADDWFC REG, W`

Before Instruction
C = 1
REG = 0x02
WREG = 0x4D
N = ?
OV = ?
DC = ?
Z = ?

After Instruction
C = 0
REG = 0x02
WREG = 0x50
N = 0
OV = 0
DC = 0
Z = 0

ANDLW **AND literal with WREG**

Syntax: [*label*] ANDLW k

Operands: $0 \le k \le 255$

Operation: (WREG) .AND. k → WREG

Status Affected: N,Z

Encoding:

0000	1011	kkkk	kkkk

Description: The contents of WREG are AND'ed with the 8-bit literal 'k'. The result is placed in WREG.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `ANDLW 0x5F`

Before Instruction
WREG = 0xA3
N = ?
Z = ?

After Instruction
WREG = 0x03
N = 0
Z = 0

BZ **Branch if Zero**

Syntax: [*label*] BZ n

Operands: -128 ≤ n ≤ 127

Operation: if Zero bit is '1'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1110	0000	nnnn	nnnn

Description: If the Zero bit is '1', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BZ Jump`

Before Instruction
PC = address `(HERE)`
After Instruction
If Zero = 1;
PC = address `(Jump)`
If Zero = 0;
PC = address `(HERE+2)`

CALL **Subroutine Call**

Syntax: [*label*] CALL k [,s]

Operands: 0 ≤ k ≤ 1048575
s ∈ [0,1]

Operation: (PC) + 4 → TOS,
k → PC<20:1>,
if s = 1
(WREG) → WS,
(STATUS) → STATUSS,
(BSR) → BSRS

Status Affected: None

Encoding:

1st word (k<7:0>)	1110	110s	$k_7$kkk	kkkk$_0$
2nd word(k<19:8>)	1111	k_{19}kkk	kkkk	kkkk$_8$

Description: Subroutine call of entire 2M byte memory range. First, return address (PC+ 4) is pushed onto the return stack. If 's' = 1, the WREG, STATUS and BSR registers are also pushed into their respective shadow registers, WS, STATUSS and BSRS. If 's' = 0, no update occurs (default). Then the 20-bit value 'k' is loaded into PC<20:1>. `CALL` is a two-cycle instruction.

Words: 2

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>,	Push PC to stack	Read literal 'k'<19:8>, Write to PC
No operation	No operation	No operation	No operation

Example: `HERE CALL THERE, FAST`

Before Instruction
PC = Address `(HERE)`
After Instruction
PC = Address `(THERE)`
TOS = Address `(HERE + 4)`
WS = WREG
BSRS = BSR
STATUSS = STATUS

POP **Pop Top of Return Stack**

Syntax: [*label*] POP

Operands: None

Operation: (TOS) → bit bucket

Status Affected: None

Encoding:

0000	0000	0000	0110

Description: The TOS value is pulled off the return stack and is discarded. The TOS value then becomes the previous value that was pushed onto the return stack.

This instruction is provided to enable the user to properly manage the return stack to incorporate a software stack.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	Pop TOS value	No operation

Example:

```
POP
GOTO    NEW
```

Before Instruction
TOS = 0031A2h
Stack (1 level down) = 014332h

After Instruction
TOS = 014332h
PC = NEW

PUSH **Push Top of Return Stack**

Syntax: [*label*] PUSH

Operands: None

Operation: (PC+2) → TOS

Status Affected: None

Encoding:

0000	0000	0000	0101

Description: The PC+2 is pushed onto the top of the return stack. The previous TOS value is pushed down on the stack.

This instruction allows implementing a software stack by modifying TOS, and then push it onto the return stack.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Push PC+2 onto return stack	No operation	No operation

Example:

```
PUSH
```

Before Instruction
TOS = 00345Ah
PC = 000124h

After Instruction
PC = 000126h
TOS = 000126h
Stack (1 level down) = 00345Ah

RETFIE **Return from Interrupt**

Syntax: [*label*] RETFIE [s]

Operands: s ∈ [0,1]

Operation: (TOS) → PC,
1 → GIE/GIEH or PEIE/GIEL,
if s = 1
(WS) → W,
(STATUSS) → STATUS,
(BSRS) → BSR,
PCLATU, PCLATH are unchanged.

Status Affected: None

Encoding:

0000	0000	0001	000s

Description: Return from Interrupt. Stack is popped and Top-of-Stack (TOS) is loaded into the PC. Interrupts are enabled by setting the either the high or low priority global interrupt enable bit. If 's' = 1, the contents of the shadow registers WS, STATUSS and BSRS are loaded into their corresponding registers, WREG, STATUS and BSR. If 's' = 0, no update of these registers occurs (default).

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	Pop PC from stack Set GIEH or GIEL
No operation	No operation	No operation	No operation

Example: `RETFIE 1`

After Interrupt
PC = TOS
WREG = WS
BSR = BSRS
STATUS = STATUSS
GIE/GIEH, PEIE/GIEL = 1

RETLW **Return Literal to WREG**

Syntax: [*label*] RETLW k

Operands: 0 ≤ k ≤ 255

Operation: k → W,
(TOS) → PC,
PCLATU, PCLATH are unchanged

Status Affected: None

Encoding:

0000	1100	kkkk	kkkk

Description: W is loaded with the eight bit literal 'k'. The program counter is loaded from the top of the stack (the return address). The high address latch (PCLATH) remains unchanged.

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Pop PC from stack, write to W
No operation	No operation	No operation	No operation

Example:

```
   CALL TABLE  ; WREG contains table
               ;  offset value
               ;  WREG now has
               ;  table value
   :
TABLE
   ADDWF PCL   ; WREG = offset
   RETLW k0    ; Begin table
   RETLW k1    ;
   :
   :
   RETLW kn    ; End of table
```

Before Instruction
WREG = 0x07

After Instruction
WREG = value of kn

RLNCF **Rotate Left f (no carry)**

Syntax: [*label*] RLNCF f [,d [,a]]

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

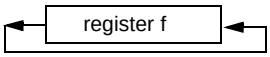
Operation: (f<n>) → dest<n+1>,
(f<7>) → dest<0>

Status Affected: N,Z

Encoding:

0100	01da	ffff	ffff

Description: The contents of register 'f' are rotated one bit to the left. If 'd' is 0 the result is placed in WREG. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.



Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: RLNCF REG

Before Instruction
REG = 1010 1011
N = ?
Z = ?

After Instruction
REG = 0101 0111
N = 0
Z = 0

RRCF **Rotate Right f through Carry**

Syntax: [*label*] RRCF f [,d [,a]]

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

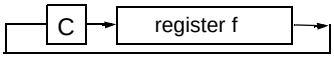
Operation: (f<n>) → dest<n-1>,
(f<0>) → C,
(C) → dest<7>

Status Affected: C,N,Z

Encoding:

0011	00da	ffff	ffff

Description: The contents of register 'f' are rotated one bit to the right through the Carry Flag. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.



Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: RRCF REG, W

Before Instruction
REG = 1110 0110
C = 0
N = ?
Z = ?

After Instruction
REG = 1110 0110
WREG = 0111 0011
C = 0
N = 0
Z = 0

SUBFWB (Cont.)

Example 1: `SUBFWB REG`

Before Instruction
REG = 3
WREG = 2
C = 1

After Instruction
REG = 0xFF
WREG = 2
C = 0
Z = 0
N = 1 ; result is negative

Example 2: `SUBFWB REG`

Before Instruction
REG = 2
WREG = 5
C = 1

After Instruction
REG = 2
WREG = 3
C = 1
Z = 0
N = 0 ; result is positive

Example 3: `SUBFWB REG`

Before Instruction
REG = 1
WREG = 2
C = 0

After Instruction
REG = 0
WREG = 2
C = 1
Z = 1 ; result is zero
N = 0

SUBLW Subtract WREG from literal

Syntax: [*label*] SUBLW k

Operands: $0 \le k \le 255$

Operation: k – (WREG) → WREG

Status Affected: N,OV, C, DC, Z

Encoding:

0000	1000	kkkk	kkkk

Description: WREG is subtracted from the eight bit literal 'k'. The result is placed in WREG.

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
WREG = 1
C = ?

After Instruction
WREG = 1
C = 1 ; result is positive
Z = 0
N = 0

Example 2: `SUBLW 0x02`

Before Instruction
WREG = 2
C = ?

After Instruction
WREG = 0
C = 1 ; result is zero
Z = 1
N = 0

Example 3: `SUBLW 0x02`

Before Instruction
WREG = 3
C = ?

After Instruction
WREG = 0xFF ; (2's complement)
C = 0 ; result is negative
Z = 0
N = 1

APPENDIX C: DEVICE MIGRATIONS

This section is intended to describe the functional and electrical specification differences when migrating between functionally similar devices (such as from a PIC16C74A to a PIC16C74B).

Not Applicable

APPENDIX D: MIGRATING FROM OTHER PICMICRO DEVICES

This discusses some of the issues in migrating from other PICmicro devices to the PIC18CXXX family of devices.

D.1 PIC16CXXX to PIC18CXXX

See application note AN716.

D.2 PIC17CXXX to PIC18CXXX

See application note AN726.

NOTES:

