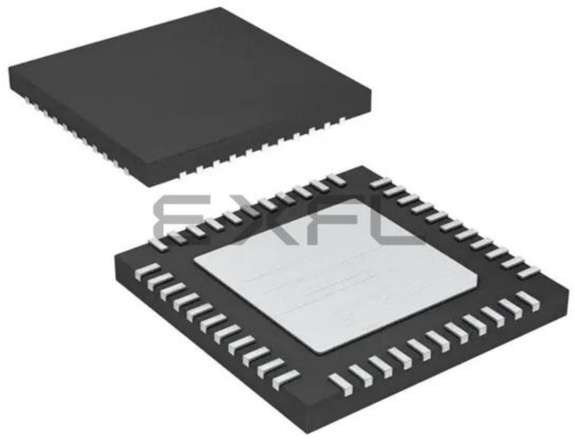


Welcome to E-XFL.COM

## What is "Embedded - Microcontrollers"?

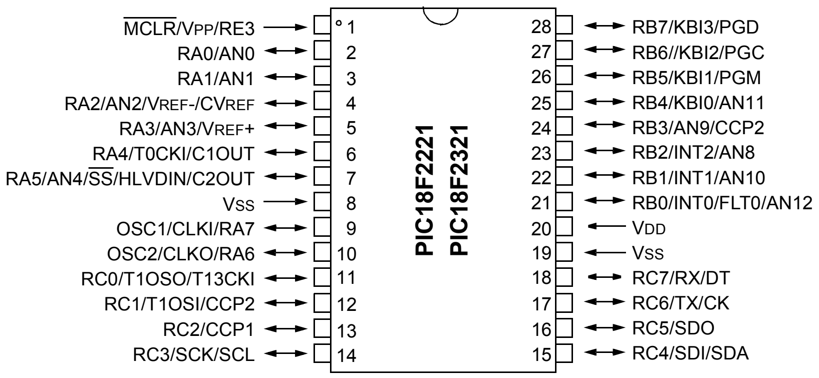
"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" 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"

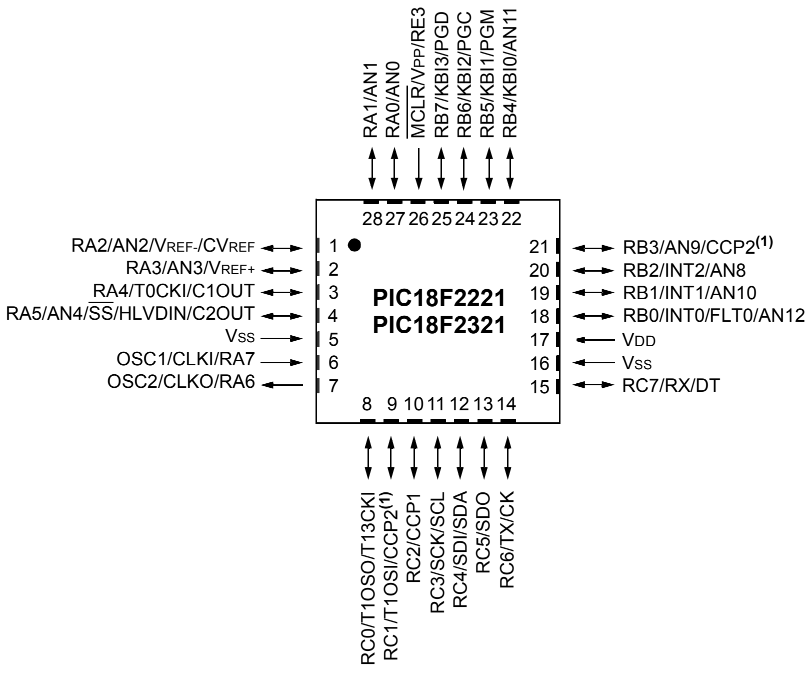
| Details | |
|---|---|
| Product Status | Obsolete |
| Core Processor | PIC |
| Core Size | 8-Bit |
| Speed | 40MHz |
| Connectivity | I²C, SPI, UART/USART |
| Peripherals | Brown-out Detect/Reset, HLVD, POR, PWM, WDT |
| Number of I/O | 36 |
| Program Memory Size | 4KB (2K x 16) |
| Program Memory Type | FLASH |
| EEPROM Size | 256 x 8 |
| RAM Size | 512 x 8 |
| Voltage - Supply (Vcc/Vdd) | 4.2V ~ 5.5V |
| Data Converters | A/D 13x10b |
| Oscillator Type | Internal |
| Operating Temperature | -40°C ~ 85°C (TA) |
| Mounting Type | Surface Mount |
| Package / Case | 44-VQFN Exposed Pad |
| Supplier Device Package | 44-QFN (8x8) |
| Purchase URL | https://www.e-xfl.com/product-detail/microchip-technology/pic18f4221t-i-ml |

## Pin Diagrams

### 28-Pin SPDIP, SOIC, SSOP



### 28-Pin QFN



**Note 1:** RB3 is the alternate pin for CCP2 multiplexing.

### 6.1.1 PROGRAM COUNTER

The Program Counter (PC) specifies the address of the instruction to fetch for execution. The PC is 21 bits wide and is contained in three separate 8-bit registers. The low byte, known as the PCL register, is both readable and writable. The high byte, or PCH register, contains the PC<15:8> bits; it is not directly readable or writable. Updates to the PCH register are performed through the PCLATH register. The upper byte is called PCU. This register contains the PC<20:16> bits; it is also not directly readable or writable. Updates to the PCU register are performed through the PCLATU register.

The contents of PCLATH and PCLATU are transferred to the program counter by any operation that writes PCL. Similarly, the upper two bytes of the program counter are transferred to PCLATH and PCLATU by an operation that reads PCL. This is useful for computed offsets to the PC (see **Section 6.1.4.1 "Computed GOTO"**).

The PC addresses bytes in the program memory. To prevent the PC from becoming misaligned with word instructions, the Least Significant bit of PCL is fixed to a value of '`0`'. The PC increments by 2 to address sequential instructions in the program memory.

The `CALL`, `RCALL`, `GOTO` and program branch instructions write to the program counter directly. For these instructions, the contents of PCLATH and PCLATU are not transferred to the program counter.

### 6.1.2 RETURN ADDRESS STACK

The return address stack allows any combination of up to 31 program calls and interrupts to occur. The PC is pushed onto the stack when a `CALL` or `RCALL` instruction is executed or an interrupt is Acknowledged. The PC value is pulled off the stack on a `RETURN`, `RETLW` or a `RETFIE` instruction. PCLATU and PCLATH are not affected by any of the `RETURN` or `CALL` instructions.

The stack operates as a 31-word by 21-bit RAM and a 5-bit Stack Pointer, STKPTR. The stack space is not part of either program or data space. The Stack Pointer is readable and writable and the address on the top of the stack is readable and writable through the Top-of-Stack Special Function Registers. Data can also be pushed to, or popped from the stack, using these registers.

A `CALL` type instruction causes a push onto the stack; the Stack Pointer is first incremented and the location pointed to by the Stack Pointer is written with the contents of the PC (already pointing to the instruction following the `CALL`). A `RETURN` type instruction causes a pop from the stack; the contents of the location pointed to by the STKPTR are transferred to the PC and then the Stack Pointer is decremented.

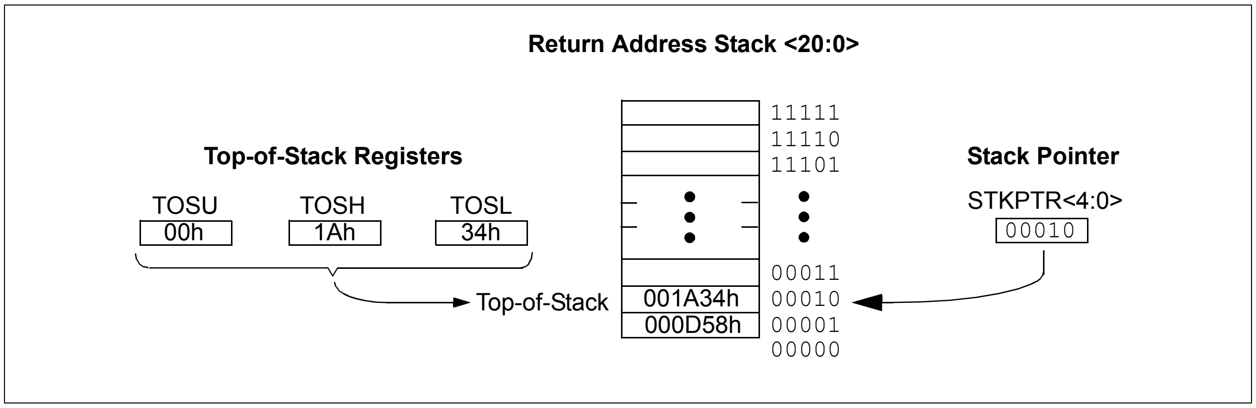
The Stack Pointer is initialized to '`00000`' after all Resets. There is no RAM associated with the location corresponding to a Stack Pointer value of '`00000`'; this is only a Reset value. Status bits indicate if the stack is full or has overflowed or has underflowed.

#### 6.1.2.1 Top-of-Stack Access

Only the top of the return address stack (TOS) is readable and writable. A set of three registers, TOSU:TOSH:TOSL, hold the contents of the stack location pointed to by the STKPTR register (Figure 6-2). This allows users to implement a software stack if necessary. After a `CALL`, `RCALL` or interrupt, the software can read the pushed value by reading the TOSU:TOSH:TOSL registers. These values can be placed on a user-defined software stack. At return time, the software can return these values to TOSU:TOSH:TOSL and do a return.

The user must disable the global interrupt enable bits while accessing the stack to prevent inadvertent stack corruption.

**FIGURE 6-2: RETURN ADDRESS STACK AND ASSOCIATED REGISTERS**



### 6.3.4 SPECIAL FUNCTION REGISTERS

The Special Function Registers (SFRs) are registers used by the CPU and peripheral modules for controlling the desired operation of the device. These registers are implemented as static RAM. SFRs start at the top of data memory (FFFh) and extend downward to occupy the top half of Bank 15 (F80h to FFFh). A list of these registers is given in Table 6-1 and Table 6-2.

The SFRs can be classified into two sets: those associated with the "core" device functionality (ALU, Resets and interrupts) and those related to the peripheral functions. The reset and interrupt registers are described in their respective chapters, while the ALU's STATUS register is described later in this section. Registers related to the operation of a peripheral feature are described in the chapter for that peripheral.

The SFRs are typically distributed among the peripherals whose functions they control. Unused SFR locations are unimplemented and read as '0's.

**TABLE 6-1: SPECIAL FUNCTION REGISTER MAP FOR PIC18F2221/2321/4221/4321 FAMILY DEVICES**

| Address | Name | Address | Name | Address | Name | Address | Name |
|---|---|---|---|---|---|---|---|
| FFFh | TOSU | FDFh | INDF2(1) | FBFh | CCPR1H | F9Fh | IPR1 |
| FFEh | TOSH | FDEh | POSTINC2(1) | FBEh | CCPR1L | F9Eh | PIR1 |
| FFDh | TOSL | FDDh | POSTDEC2(1) | FBDh | CCP1CON | F9Dh | PIE1 |
| FFCh | STKPTR | FDCh | PREINC2(1) | FBCh | CCPR2H | F9Ch | —(2) |
| FFBh | PCLATU | FDBh | PLUSW2(1) | FBBh | CCPR2L | F9Bh | OSCTUNE |
| FFAh | PCLATH | FDAh | FSR2H | FBAh | CCP2CON | F9Ah | —(2) |
| FF9h | PCL | FD9h | FSR2L | FB9h | —(2) | F99h | —(2) |
| FF8h | TBLPTRU | FD8h | STATUS | FB8h | BAUDCON | F98h | —(2) |
| FF7h | TBLPTRH | FD7h | TMR0H | FB7h | ECCP1DEL(3) | F97h | —(2) |
| FF6h | TBLPTRL | FD6h | TMR0L | FB6h | ECCP1AS(3) | F96h | TRISE(3) |
| FF5h | TABLAT | FD5h | T0CON | FB5h | CVRCON | F95h | TRISD(3) |
| FF4h | PRODH | FD4h | —(2) | FB4h | CMCON | F94h | TRISC |
| FF3h | PRODL | FD3h | OSCCON | FB3h | TMR3H | F93h | TRISB |
| FF2h | INTCON | FD2h | HLVDCON | FB2h | TMR3L | F92h | TRISA |
| FF1h | INTCON2 | FD1h | WDTCON | FB1h | T3CON | F91h | —(2) |
| FF0h | INTCON3 | FD0h | RCON | FB0h | SPBRGH | F90h | —(2) |
| FEFh | INDF0(1) | FCFh | TMR1H | FAFh | SPBRG | F8Fh | —(2) |
| FEEh | POSTINC0(1) | FCEh | TMR1L | FAEh | RCREG | F8Eh | —(2) |
| FEDh | POSTDEC0(1) | FCDh | T1CON | FADh | TXREG | F8Dh | LATE(3) |
| FECh | PREINC0(1) | FCCh | TMR2 | FACh | TXSTA | F8Ch | LATD(3) |
| FEBh | PLUSW0(1) | FCBh | PR2 | FABh | RCSTA | F8Bh | LATC |
| FEAh | FSR0H | FCAh | T2CON | FAAh | —(2) | F8Ah | LATB |
| FE9h | FSR0L | FC9h | SSPBUF | FA9h | EEADR | F89h | LATA |
| FE8h | WREG | FC8h | SSPADD | FA8h | EEDATA | F88h | —(2) |
| FE7h | INDF1(1) | FC7h | SSPSTAT | FA7h | EECON2(1) | F87h | —(2) |
| FE6h | POSTINC1(1) | FC6h | SSPCON1 | FA6h | EECON1 | F86h | —(2) |
| FE5h | POSTDEC1(1) | FC5h | SSPCON2 | FA5h | —(2) | F85h | —(2) |
| FE4h | PREINC1(1) | FC4h | ADRESH | FA4h | —(2) | F84h | PORTE |
| FE3h | PLUSW1(1) | FC3h | ADRESL | FA3h | —(2) | F83h | PORTD(3) |
| FE2h | FSR1H | FC2h | ADCON0 | FA2h | IPR2 | F82h | PORTC |
| FE1h | FSR1L | FC1h | ADCON1 | FA1h | PIR2 | F81h | PORTB |
| FE0h | BSR | FC0h | ADCON2 | FA0h | PIE2 | F80h | PORTA |

**Note 1:** This is not a physical register.
**2:** Unimplemented registers are read as '0'.
**3:** This register is not available on 28-pin devices.

**REGISTER 10-5: PIR2: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 2**

| R/W-0 | R/W-0 | U-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| OSCFIF | CMIF | — | EEIF | BCLIF | HLVDIF | TMR3IF | CCP2IF |
| bit 7 | | | | | | | bit 0 |

bit 7 **OSCFIF:** Oscillator Fail Interrupt Flag bit
1 = Device oscillator failed, clock input has changed to INTOSC (must be cleared in software)
0 = Device clock operating

bit 6 **CMIF:** Comparator Interrupt Flag bit
1 = Comparator input has changed (must be cleared in software)
0 = Comparator input has not changed

bit 5 **Unimplemented:** Read as ‘0’

bit 4 **EEIF:** Data EEPROM/Flash Write Operation Interrupt Flag bit
1 = The write operation is complete (must be cleared in software)
0 = The write operation is not complete or has not been started

bit 3 **BCLIF:** Bus Collision Interrupt Flag bit
1 = A bus collision occurred (must be cleared in software)
0 = No bus collision occurred

bit 2 **HLVDIF:** High/Low-Voltage Detect Interrupt Flag bit
1 = A high/low-voltage condition occurred; direction determined by VDIRMAG bit (HLVDCON<7>)
0 = A high/low-voltage condition has not occurred

bit 1 **TMR3IF:** TMR3 Overflow Interrupt Flag bit
1 = TMR3 register overflowed (must be cleared in software)
0 = TMR3 register did not overflow

bit 0 **CCP2IF:** CCP2 Interrupt Flag bit
<u>Capture mode:</u>
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
<u>Compare mode:</u>
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
<u>PWM mode:</u>
Unused in this 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 |

**REGISTER 10-7: PIE2: PERIPHERAL INTERRUPT ENABLE REGISTER 2**

| R/W-0 | R/W-0 | U-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| OSCFIE | CMIE | — | EEIE | BCLIE | HLVDIE | TMR3IE | CCP2IE |
| bit 7 | | | | | | | bit 0 |

| | |
|---|---|
| bit 7 | **OSCFIE:** Oscillator Fail Interrupt Enable bit<br>1 = Enabled<br>0 = Disabled |
| bit 6 | **CMIE:** Comparator Interrupt Enable bit<br>1 = Enabled<br>0 = Disabled |
| bit 5 | **Unimplemented:** Read as '0' |
| bit 4 | **EEIE:** Data EEPROM/Flash Write Operation Interrupt Enable bit<br>1 = Enabled<br>0 = Disabled |
| bit 3 | **BCLIE:** Bus Collision Interrupt Enable bit<br>1 = Enabled<br>0 = Disabled |
| bit 2 | **HLVDIE:** High/Low-Voltage Detect Interrupt Enable bit<br>1 = Enabled<br>0 = Disabled |
| bit 1 | **TMR3IE:** TMR3 Overflow Interrupt Enable bit<br>1 = Enabled<br>0 = Disabled |
| bit 0 | **CCP2IE:** CCP2 Interrupt Enable bit<br>1 = Enabled<br>0 = Disabled |

| **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 |

## 10.5 RCON Register

The RCON register contains flag bits which are used to determine the cause of the last Reset or wake-up from Idle or Sleep modes. RCON also contains the IPEN bit which enables interrupt priorities.

The operation of the SBOREN bit and the Reset flag bits is discussed in more detail in **Section 5.1 "RCON Register"**.

**REGISTER 10-10: RCON: RESET CONTROL REGISTER**

| R/W-0 | R/W-1[(1)] | U-0 | R/W-1 | R-1 | R-1 | R/W-0[(2)] | R/W-0 |
|---|---|---|---|---|---|---|---|
| IPEN | SBOREN | — | $\overline{\text{RI}}$ | $\overline{\text{TO}}$ | $\overline{\text{PD}}$ | $\overline{\text{POR}}$ | $\overline{\text{BOR}}$ |
| bit 7 | | | | | | | bit 0 |

bit 7 **IPEN:** Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (PIC16XXX Compatibility mode)

bit 6 **SBOREN:** Software BOR Enable bit[(1)]
For details of bit operation, see Register 5-1.

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

bit 4 **$\overline{\text{RI}}$:** RESET Instruction Flag bit
For details of bit operation, see Register 5-1.

bit 3 **$\overline{\text{TO}}$:** Watchdog Time-out Flag bit
For details of bit operation, see Register 5-1.

bit 2 **$\overline{\text{PD}}$:** Power-down Detection Flag bit
For details of bit operation, see Register 5-1.

bit 1 **$\overline{\text{POR}}$:** Power-on Reset Status bit[(2)]
For details of bit operation, see Register 5-1.

bit 0 **$\overline{\text{BOR}}$:** Brown-out Reset Status bit
For details of bit operation, see Register 5-1.

**Note 1:** If SBOREN is enabled, its Reset state is '1'; otherwise, it is '0'.
**2:** Actual Reset values are determined by device configuration and the nature of the device Reset. See Register 5-1 for additional information.

| **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 |

## 12.3 Prescaler

An 8-bit counter is available as a prescaler for the Timer0 module. The prescaler is not directly readable or writable; its value is set by the PSA and T0PS<2:0> bits (T0CON<3:0>) which determine the prescaler assignment and prescale ratio.

Clearing the PSA bit assigns the prescaler to the Timer0 module. When it is assigned, prescale values from 1:2 through 1:256 in power-of-2 increments are selectable.

When assigned to the Timer0 module, all instructions writing to the TMR0 register (e.g., `CLRF TMR0`, `MOVWF TMR0`, `BSF TMR0`, etc.) clear the prescaler count.

| **Note:** | Writing to TMR0 when the prescaler is assigned to Timer0 will clear the prescaler count but will not change the prescaler assignment. |
|---|---|

### 12.3.1 SWITCHING PRESCALER ASSIGNMENT

The prescaler assignment is fully under software control and can be changed "on-the-fly" during program execution.

## 12.4 Timer0 Interrupt

The TMR0 interrupt is generated when the TMR0 register overflows from FFh to 00h in 8-bit mode, or from FFFFh to 0000h in 16-bit mode. This overflow sets the TMR0IF flag bit. The interrupt can be masked by clearing the TMR0IE bit (INTCON<5>). Before re-enabling the interrupt, the TMR0IF bit must be cleared in software by the Interrupt Service Routine.

Since Timer0 is shut down in Sleep mode, the TMR0 interrupt cannot awaken the processor from Sleep.

**TABLE 12-1: REGISTERS ASSOCIATED WITH TIMER0**

| Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Reset Values on page |
|---|---|---|---|---|---|---|---|---|---|
| TMR0L | Timer0 Register Low Byte | | | | | | | | 56 |
| TMR0H | Timer0 Register High Byte | | | | | | | | 56 |
| INTCON | GIE/GIEH | PEIE/GIEL | TMR0IE | INT0IE | RBIE | TMR0IF | INT0IF | RBIF | 55 |
| T0CON | TMR0ON | T08BIT | T0CS | T0SE | PSA | T0PS2 | T0PS1 | T0PS0 | 56 |
| TRISA | RA7(1) | RA6(1) | RA5 | RA4 | RA3 | RA2 | RA1 | RA0 | 58 |

**Legend:** Shaded cells are not used by Timer0.

**Note 1:** PORTA<7:6> and their direction bits are individually configured as port pins based on various primary oscillator modes. When disabled, these bits read as '0'.

**FIGURE 18-8: I²C™ SLAVE MODE TIMING WITH SEN = 0 (RECEPTION, 7-BIT ADDRESSING)**

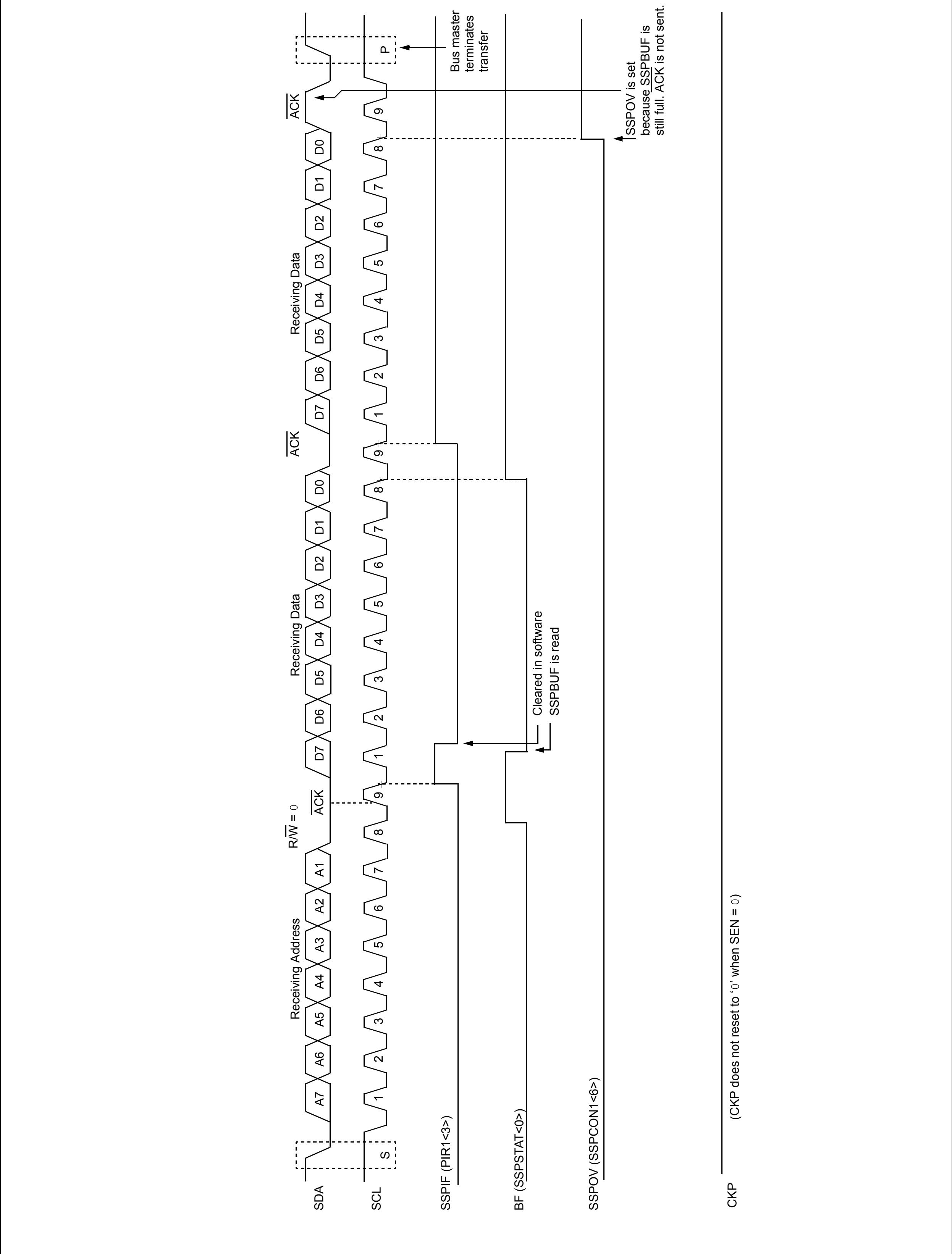
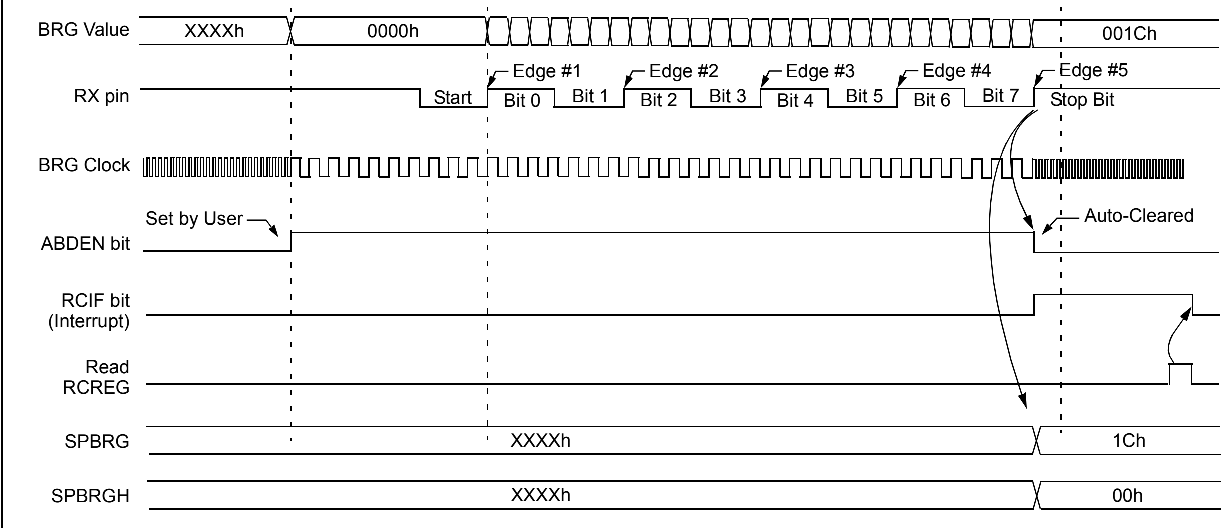


**TABLE 18-4: REGISTERS ASSOCIATED WITH I$^2$C™ OPERATION**

| Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Reset Values on page |
|---|---|---|---|---|---|---|---|---|---|
| INTCON | GIE/GIEH | PEIE/GIEL | TMR0IE | INT0IE | RBIE | TMR0IF | INT0IF | RBIF | 55 |
| PIR1 | PSPIF | ADIF | RCIF | TXIF | SSPIF | CCP1IF | TMR2IF | TMR1IF | 58 |
| PIE1 | PSPIE | ADIE | RCIE | TXIE | SSPIE | CCP1IE | TMR2IE | TMR1IE | 58 |
| IPR1 | PSPIP | ADIP | RCIP | TXIP | SSPIP | CCP1IP | TMR2IP | TMR1IP | 58 |
| PIR2 | OSCFIF | CMIF | — | EEIF | BCLIF | HLVDIF | TMR3IF | CCP2IF | 58 |
| PIE2 | OSCFIE | CMIE | — | EEIE | BCLIE | HLVDIE | TMR3IE | CCP2IE | 58 |
| IPR2 | OSCFIP | CMIP | — | EEIP | BCLIP | HLVDIP | TMR3IP | CCP2IP | 58 |
| TRISC | TRISC7 | TRISC6 | TRISC5 | TRISC4 | TRISC3 | TRISC2 | TRISC1 | TRISC0 | 58 |
| TRISD | TRISD7 | TRISD6 | TRISD5 | TRISD4 | TRISD3 | TRISD2 | TRISD1 | TRISD0 | 58 |
| SSPBUF | MSSP Receive Buffer/Transmit Register | | | | | | | | 56 |
| SSPADD | ADD7 | ADD6 | ADD5 | ADD4 | ADD3 | ADD2 | ADD1 | ADD0 | 56 |
| TMR2 | Timer2 Register | | | | | | | | 56 |
| PR2 | Timer2 Period Register | | | | | | | | 56 |
| SSPCON1 | WCOL | SSPOV | SSPEN | CKP | SSPM3 | SSPM2 | SSPM1 | SSPM0 | 56 |
| SSPCON2 | GCEN | ACKSTAT | ACKDT/ ADMSK5 | ACKEN/ ADMSK5 | RCEN/ ADMSK5 | PEN/ ADMSK5 | RSEN/ ADMSK5 | SEN | 56 |
| SSPSTAT | SMP | CKE | D/$\overline{A}$ | P | S | R/$\overline{W}$ | UA | BF | 56 |

**Legend:** — = unimplemented, read as '0'. Shaded cells are not used by the MSSP module in I$^2$C mode.

**FIGURE 19-1: AUTOMATIC BAUD RATE CALCULATION**



**Note:** The ABD sequence requires the EUSART module to be configured in Asynchronous mode and WUE = 0.

**FIGURE 19-2: BRG OVERFLOW SEQUENCE**

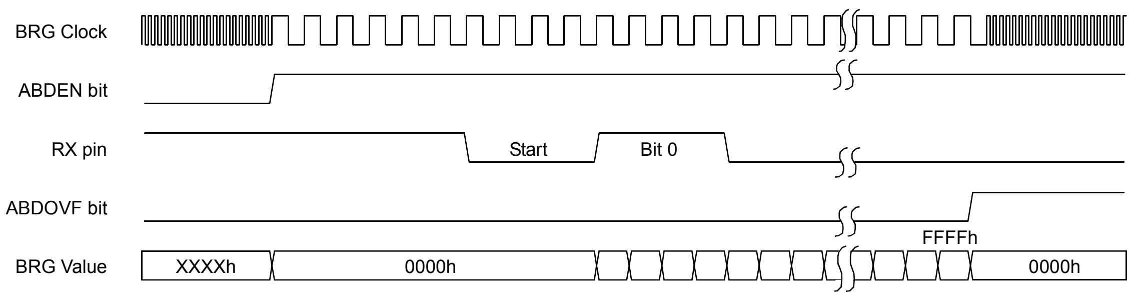


**FIGURE 19-3: EUSART TRANSMIT BLOCK DIAGRAM**

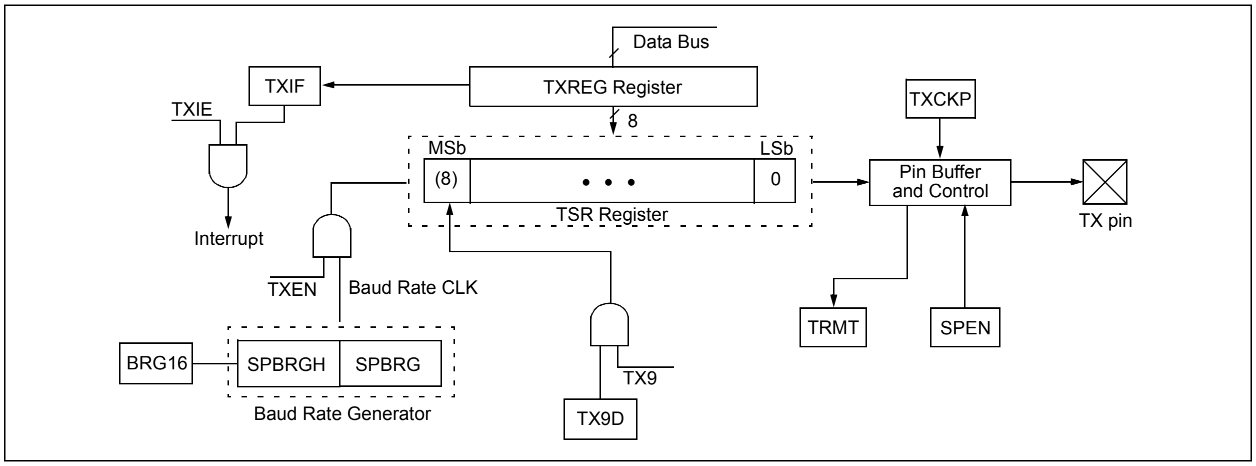


**FIGURE 19-4: ASYNCHRONOUS TRANSMISSION, TXCKP = 0 (TX NOT INVERTED)**

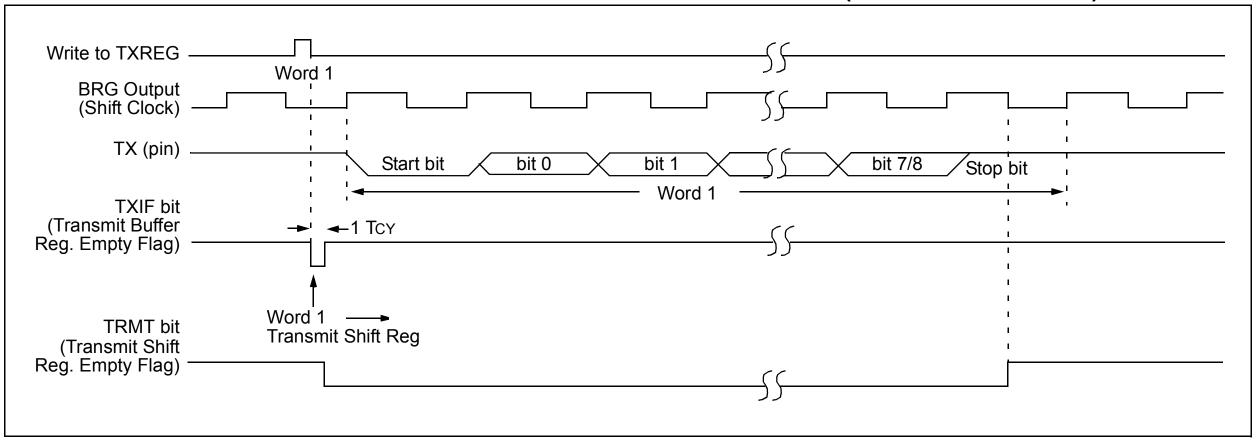
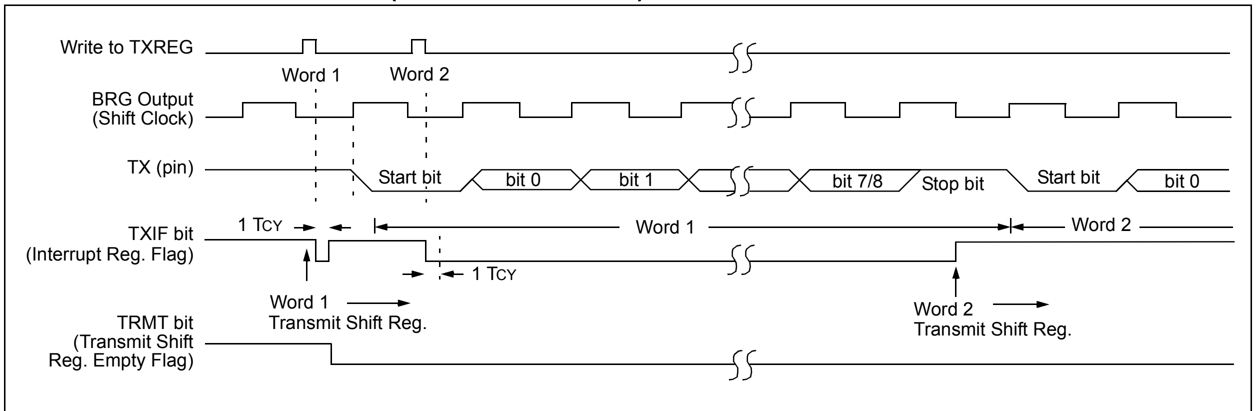
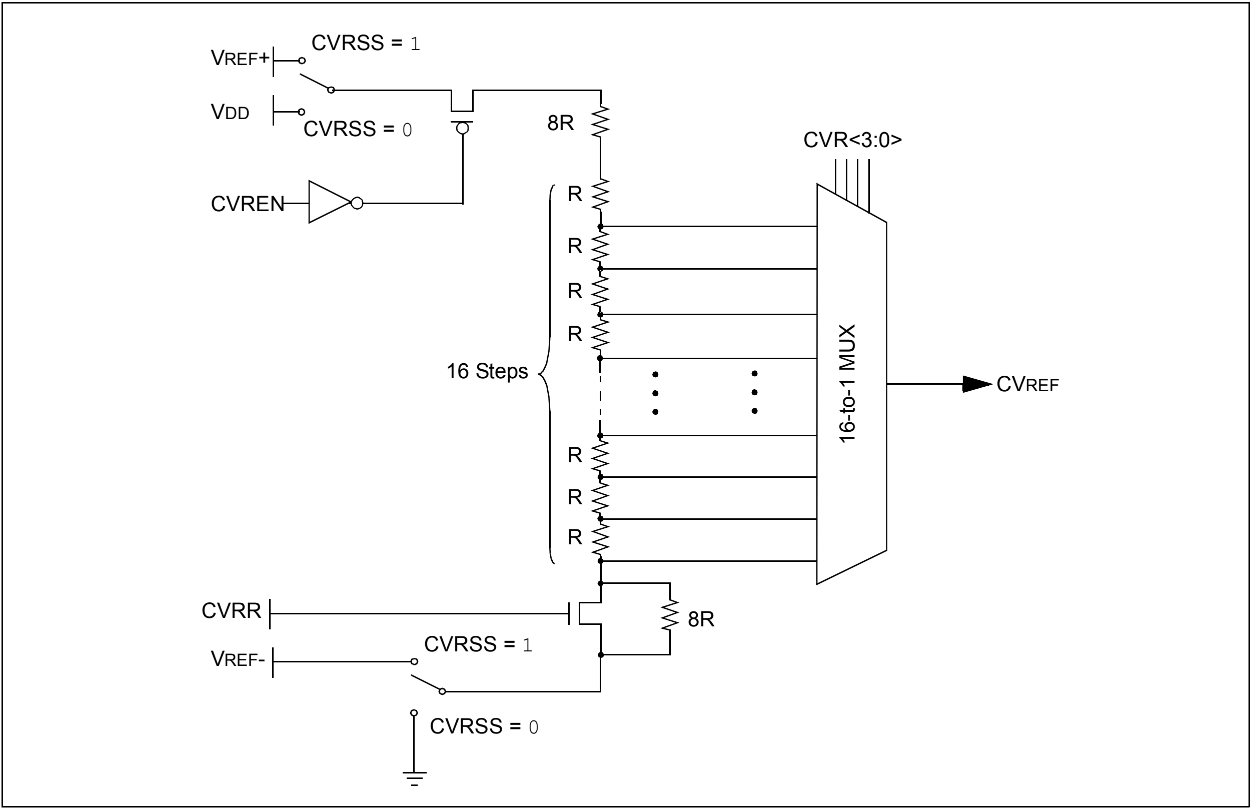

Write to TXREG
Word 1
BRG Output (Shift Clock)
TX (pin)
Start bit
bit 0
bit 1
bit 7/8
Stop bit
Word 1
TXIF bit (Transmit Buffer Reg. Empty Flag)
1 TCY
Word 1 Transmit Shift Reg
TRMT bit (Transmit Shift Reg. Empty Flag)

**FIGURE 19-5: ASYNCHRONOUS TRANSMISSION (BACK TO BACK), TXCKP = 0 (TX NOT INVERTED)**



**Note:** This timing diagram shows two consecutive transmissions.

**FIGURE 22-1: COMPARATOR VOLTAGE REFERENCE BLOCK DIAGRAM**



## 22.2 Voltage Reference Accuracy/Error

The full range of voltage reference cannot be realized due to the construction of the module. The transistors on the top and bottom of the resistor ladder network (Figure 22-1) keep CVREF from approaching the reference source rails. The voltage reference is derived from the reference source; therefore, the CVREF output changes with fluctuations in that source. The tested absolute accuracy of the voltage reference can be found in **Section 27.0 “Electrical Characteristics”**.

## 22.3 Operation During Sleep

When the device wakes up from Sleep through an interrupt or a Watchdog Timer time-out, the contents of the CVRCON register are not affected. To minimize current consumption in Sleep mode, the voltage reference should be disabled.

## 22.4 Effects of a Reset

A device Reset disables the voltage reference by clearing bit, CVREN (CVRCON<7>). This Reset also disconnects the reference from the RA2 pin by clearing bit, CVROE (CVRCON<6>) and selects the high-voltage range by clearing bit, CVRR (CVRCON<5>). The CVR value select bits are also cleared.

## 22.5 Connection Considerations

The voltage reference module operates independently of the comparator module. The output of the reference generator may be connected to the RA2 pin if the CVROE bit is set. Enabling the voltage reference output onto RA2 when it is configured as a digital input will increase current consumption. Connecting RA2 as a digital output with CVRSS enabled will also increase current consumption.

The RA2 pin can be used as a simple D/A output with limited drive capability. Due to the limited current drive capability, a buffer must be used on the voltage reference output for external connections to VREF. Figure 22-2 shows an example buffering technique.

## ANDWF — AND W with f

| | |
|---|---|
| Syntax: | ANDWF f {,d {,a}} |
| Operands: | 0 ≤ f ≤ 255<br>d ∈ [0,1]<br>a ∈ [0,1] |
| Operation: | (W) .AND. (f) → dest |
| Status Affected: | N, Z |
| Encoding: | `0001` `01da` `ffff` `ffff` |
| Description: | The contents of W are ANDed with register 'f'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'f' (default).<br>If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).<br>If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 25.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details. |
| Words: | 1 |
| Cycles: | 1 |

Q Cycle Activity:

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

Example: `ANDWF REG, 0, 0`

Before Instruction

| | | |
|---|---|---|
| W | = | 17h |
| REG | = | C2h |

After Instruction

| | | |
|---|---|---|
| W | = | 02h |
| REG | = | C2h |

## BC — Branch if Carry

| | |
|---|---|
| Syntax: | BC n |
| Operands: | -128 ≤ n ≤ 127 |
| Operation: | If Carry bit is '1',<br>(PC) + 2 + 2n → PC |
| Status Affected: | None |
| Encoding: | `1110` `0010` `nnnn` `nnnn` |
| Description: | If the Carry bit is '1', then the program will branch.<br>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 BC 5`

Before Instruction

| | | |
|---|---|---|
| PC | = | address `(HERE)` |

After Instruction

| | | |
|---|---|---|
| If Carry | = | 1; |
| PC | = | address `(HERE + 12)` |
| If Carry | = | 0; |
| PC | = | address `(HERE + 2)` |

## BCF — Bit Clear f

| | |
|---|---|
| Syntax: | BCF f, b {,a} |
| Operands: | $0 \le f \le 255$<br>$0 \le b \le 7$<br>$a \in [0,1]$ |
| Operation: | `0` → f<b> |
| Status Affected: | None |
| Encoding: | `1001` `bbba` `ffff` `ffff` |
| Description: | Bit 'b' in register 'f' is cleared.<br>If 'a' is '`0`', the Access Bank is selected. If 'a' is '`1`', the BSR is used to select the GPR bank (default).<br>If 'a' is '`0`' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 25.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details. |
| 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, 0`

```
Before Instruction
    FLAG_REG  =  C7h
After Instruction
    FLAG_REG  =  47h
```

## BN — Branch if Negative

| | |
|---|---|
| Syntax: | BN n |
| Operands: | $-128 \le n \le 127$ |
| Operation: | If Negative bit is '`1`',<br>(PC) + 2 + 2n → PC |
| Status Affected: | None |
| Encoding: | `1110` `0110` `nnnn` `nnnn` |
| Description: | If the Negative bit is '`1`', then the program will branch.<br>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 BN Jump`

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

**CLRF — Clear f**

| | |
|---|---|
| Syntax: | CLRF f {,a} |
| Operands: | $0 \le f \le 255$<br>$a \in [0,1]$ |
| Operation: | 000h → f,<br>1 → Z |
| Status Affected: | Z |

Encoding:

| | | | |
|---|---|---|---|
| 0110 | 101a | ffff | ffff |

Description: Clears the contents of the specified register.
If ‘a’ is ‘0’, the Access Bank is selected.
If ‘a’ is ‘1’, the BSR is used to select the GPR bank (default).
If ‘a’ is ‘0’ and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 25.2.3 “Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode”** for details.

Words: 1

Cycles: 1

Q Cycle Activity:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| Decode | Read register ‘f’ | Process Data | Write register ‘f’ |

Example: `CLRF FLAG_REG, 1`

```
Before Instruction
    FLAG_REG  =  5Ah
After Instruction
    FLAG_REG  =  00h
```

**CLRWDT — Clear Watchdog Timer**

| | |
|---|---|
| Syntax: | CLRWDT |
| Operands: | None |
| Operation: | 000h → WDT,<br>000h → WDT postscaler,<br>1 → $\overline{TO}$,<br>1 → $\overline{PD}$ |
| Status Affected: | $\overline{TO}$, $\overline{PD}$ |

Encoding:

| | | | |
|---|---|---|---|
| 0000 | 0000 | 0000 | 0100 |

Description: `CLRWDT` instruction resets the Watchdog Timer. It also resets the postscaler of the WDT. Status bits, $\overline{TO}$ and $\overline{PD}$, are set.

Words: 1

Cycles: 1

Q Cycle Activity:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| Decode | No operation | Process Data | No operation |

Example: `CLRWDT`

Before Instruction
- WDT Counter = ?

After Instruction
- WDT Counter = 00h
- WDT Postscaler = 0
- $\overline{TO}$ = 1
- $\overline{PD}$ = 1

## COMF Complement f

| | |
|---|---|
| Syntax: | COMF f {,d {,a}} |
| Operands: | $0 \le f \le 255$<br>$d \in$ [0,1]<br>$a \in$ [0,1] |
| Operation: | $(\bar{f}) \rightarrow$ dest |
| Status Affected: | N, Z |

Encoding:

| | | | |
|---|---|---|---|
| 0001 | 11da | ffff | ffff |

Description: The contents of register ‘f’ are complemented. If ‘d’ is ‘0’, the result is stored in W. If ‘d’ is ‘1’, the result is stored back in register ‘f’ (default).
If ‘a’ is ‘0’, the Access Bank is selected. If ‘a’ is ‘1’, the BSR is used to select the GPR bank (default).
If ‘a’ is ‘0’ and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 25.2.3 “Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode”** for details.

Words: 1

Cycles: 1

Q Cycle Activity:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| Decode | Read register ‘f’ | Process Data | Write to destination |

Example:
```
COMF     REG,  0,  0
```

Before Instruction
REG = 13h
After Instruction
REG = 13h
W = ECh

## CPFSEQ Compare f with W, Skip if f = W

| | |
|---|---|
| Syntax: | CPFSEQ f {,a} |
| Operands: | $0 \le f \le 255$<br>$a \in$ [0,1] |
| Operation: | (f) – (W),<br>skip if (f) = (W)<br>(unsigned comparison) |
| Status Affected: | None |

Encoding:

| | | | |
|---|---|---|---|
| 0110 | 001a | ffff | ffff |

Description: Compares the contents of data memory location ‘f’ to the contents of W by performing an unsigned subtraction.
If ‘f’ = W, then the fetched instruction is discarded and a `NOP` is executed instead, making this a two-cycle instruction.
If ‘a’ is ‘0’, the Access Bank is selected. If ‘a’ is ‘1’, the BSR is used to select the GPR bank (default).
If ‘a’ is ‘0’ and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 25.2.3 “Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode”** for details.

Words: 1

Cycles: 1(2)
**Note:** 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| Decode | Read register ‘f’ | Process Data | No operation |

If skip:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| No operation | No operation | No operation | No operation |

If skip and followed by 2-word instruction:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| No operation | No operation | No operation | No operation |
| No operation | No operation | No operation | No operation |

Example:
```
HERE      CPFSEQ REG, 0
NEQUAL    :
EQUAL     :
```

Before Instruction
PC Address = `HERE`
W = ?
REG = ?
After Instruction
If REG = W;
PC = Address `(EQUAL)`
If REG ≠ W;
PC = Address `(NEQUAL)`

## RETFIE — Return from Interrupt

| | |
|---|---|
| Syntax: | RETFIE {s} |
| Operands: | s ∈ [0,1] |
| Operation: | (TOS) → PC,<br>1 → GIE/GIEH or PEIE/GIEL;<br>if s = 1,<br>(WS) → W,<br>(STATUSS) → STATUS,<br>(BSRS) → BSR,<br>PCLATU, PCLATH are unchanged |
| Status Affected: | GIE/GIEH, PEIE/GIEL |
| 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 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, W, 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<br>Set GIEH or GIEL |
| No operation | No operation | No operation | No operation |

Example: `RETFIE 1`

After Interrupt

| | | |
|---|---|---|
| PC | = | TOS |
| W | = | WS |
| BSR | = | BSRS |
| STATUS | = | STATUSS |
| GIE/GIEH, PEIE/GIEL | = | 1 |

## RETLW — Return Literal to W

| | |
|---|---|
| Syntax: | RETLW k |
| Operands: | 0 ≤ k ≤ 255 |
| Operation: | k → W,<br>(TOS) → PC,<br>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 ; W contains table
               ; offset value
               ; W now has
               ; table value
   :
TABLE
    ADDWF PCL  ; W = offset
    RETLW k0   ; Begin table
    RETLW k1   ;
   :
   :
    RETLW kn   ; End of table
```

Before Instruction

| | | |
|---|---|---|
| W | = | 07h |

After Instruction

| | | |
|---|---|---|
| W | = | value of kn |

## 25.2 Extended Instruction Set

In addition to the standard 75 instructions of the PIC18 instruction set, PIC18F2221/2321/4221/4321 family devices also provide an optional extension to the core CPU functionality. The added features include eight additional instructions that augment indirect and indexed addressing operations and the implementation of Indexed Literal Offset Addressing mode for many of the standard PIC18 instructions.

The additional features of the extended instruction set are disabled by default. To enable them, users must set the XINST Configuration bit.

The instructions in the extended set (with the exception of `CALLW`, `MOVSF` and `MOVSS`) can all be classified as literal operations, which either manipulate the File Select Registers, or use them for indexed addressing. Two of the instructions, `ADDFSR` and `SUBFSR`, each have an additional special instantiation for using FSR2. These versions (`ADDULNK` and `SUBULNK`) allow for automatic return after execution.

The extended instructions are specifically implemented to optimize re-entrant program code (that is, code that is recursive or that uses a software stack) written in high-level languages, particularly C. Among other things, they allow users working in high-level languages to perform certain operations on data structures more efficiently. These include:

- Dynamic allocation and deallocation of software stack space when entering and leaving subroutines
- Function Pointer invocation
- Software Stack Pointer manipulation
- Manipulation of variables located in a software stack

A summary of the instructions in the extended instruction set is provided in Table 25-3. Detailed descriptions are provided in **Section 25.2.2 "Extended Instruction Set"**. The opcode field descriptions in Table 25-1 (page 280) apply to both the standard and extended PIC18 instruction sets.

| **Note:** | The instruction set extension and the Indexed Literal Offset Addressing mode were designed for optimizing applications written in C; the user may likely never use these instructions directly in the assembler. The syntax for these commands is provided as a reference for users who may be reviewing code that has been generated by a compiler. |
|---|---|

### 25.2.1 EXTENDED INSTRUCTION SYNTAX

Most of the extended instructions use indexed arguments, using one of the File Select Registers and some offset to specify a source or destination register. When an argument for an instruction serves as part of indexed addressing, it is enclosed in square brackets ("[ ]"). This is done to indicate that the argument is used as an index or offset. The MPASM™ Assembler will flag an error if it determines that an index or offset value is not bracketed.

When the extended instruction set is enabled, brackets are also used to indicate index arguments in byte-oriented and bit-oriented instructions. This is in addition to other changes in their syntax. For more details, see **Section 25.2.3.1 "Extended Instruction Syntax with Standard PIC18 Commands"**.

| **Note:** | In the past, square brackets have been used to denote optional arguments in the PIC18 and earlier instruction sets. In this text and going forward, optional arguments are denoted by braces ("{ }"). |
|---|---|

**TABLE 25-3: EXTENSIONS TO THE PIC18 INSTRUCTION SET**

| Mnemonic, | Operands | Description | Cycles | 16-Bit Instruction Word MSb | | | LSb | Status Affected |
|---|---|---|---|---|---|---|---|---|
| ADDFSR | f, k | Add Literal to FSR | 1 | 1110 | 1000 | ffkk | kkkk | None |
| ADDULNK | k | Add Literal to FSR2 and Return | 2 | 1110 | 1000 | 11kk | kkkk | None |
| CALLW | | Call Subroutine using WREG | 2 | 0000 | 0000 | 0001 | 0100 | None |
| MOVSF | $z_s$, $f_d$ | Move $z_s$ (source) to 1st Word | 2 | 1110 | 1011 | 0zzz | zzzz | None |
| | | $f_d$ (destination) 2nd Word | | 1111 | ffff | ffff | ffff | |
| MOVSS | $z_s$, $z_d$ | Move $z_s$ (source) to 1st word | 2 | 1110 | 1011 | 1zzz | zzzz | None |
| | | $z_d$ (destination) 2nd Word | | 1111 | xxxx | xzzz | zzzz | |
| PUSHL | k | Store Literal at FSR2, Decrement FSR2 | 1 | 1110 | 1010 | kkkk | kkkk | None |
| SUBFSR | f, k | Subtract Literal from FSR | 1 | 1110 | 1001 | ffkk | kkkk | None |
| SUBULNK | k | Subtract Literal from FSR2 and Return | 2 | 1110 | 1001 | 11kk | kkkk | None |

### 26.11 PICkit 2 Development Programmer/Debugger and PICkit 2 Debug Express

The PICkit™ 2 Development Programmer/Debugger is a low-cost development tool with an easy to use interface for programming and debugging Microchip's Flash families of microcontrollers. The full featured Windows® programming interface supports baseline (PIC10F, PIC12F5xx, PIC16F5xx), midrange (PIC12F6xx, PIC16F), PIC18F, PIC24, dsPIC30, dsPIC33, and PIC32 families of 8-bit, 16-bit, and 32-bit microcontrollers, and many Microchip Serial EEPROM products. With Microchip's powerful MPLAB Integrated Development Environment (IDE) the PICkit™ 2 enables in-circuit debugging on most PIC® microcontrollers. In-Circuit-Debugging runs, halts and single steps the program while the PIC microcontroller is embedded in the application. When halted at a breakpoint, the file registers can be examined and modified.

The PICkit 2 Debug Express include the PICkit 2, demo board and microcontroller, hookup cables and CDROM with user's guide, lessons, tutorial, compiler and MPLAB IDE software.

### 26.12 MPLAB PM3 Device Programmer

The MPLAB PM3 Device Programmer is a universal, CE compliant device programmer with programmable voltage verification at VDDMIN and VDDMAX for maximum reliability. It features a large LCD display (128 x 64) for menus and error messages and a modular, detachable socket assembly to support various package types. The ICSP™ cable assembly is included as a standard item. In Stand-Alone mode, the MPLAB PM3 Device Programmer can read, verify and program PIC devices without a PC connection. It can also set code protection in this mode. The MPLAB PM3 connects to the host PC via an RS-232 or USB cable. The MPLAB PM3 has high-speed communications and optimized algorithms for quick programming of large memory devices and incorporates an MMC card for file storage and data applications.

### 26.13 Demonstration/Development Boards, Evaluation Kits, and Starter Kits

A wide variety of demonstration, development and evaluation boards for various PIC MCUs and dsPIC DSCs allows quick application development on fully functional systems. Most boards include prototyping areas for adding custom circuitry and provide application firmware and source code for examination and modification.

The boards support a variety of features, including LEDs, temperature sensors, switches, speakers, RS-232 interfaces, LCD displays, potentiometers and additional EEPROM memory.

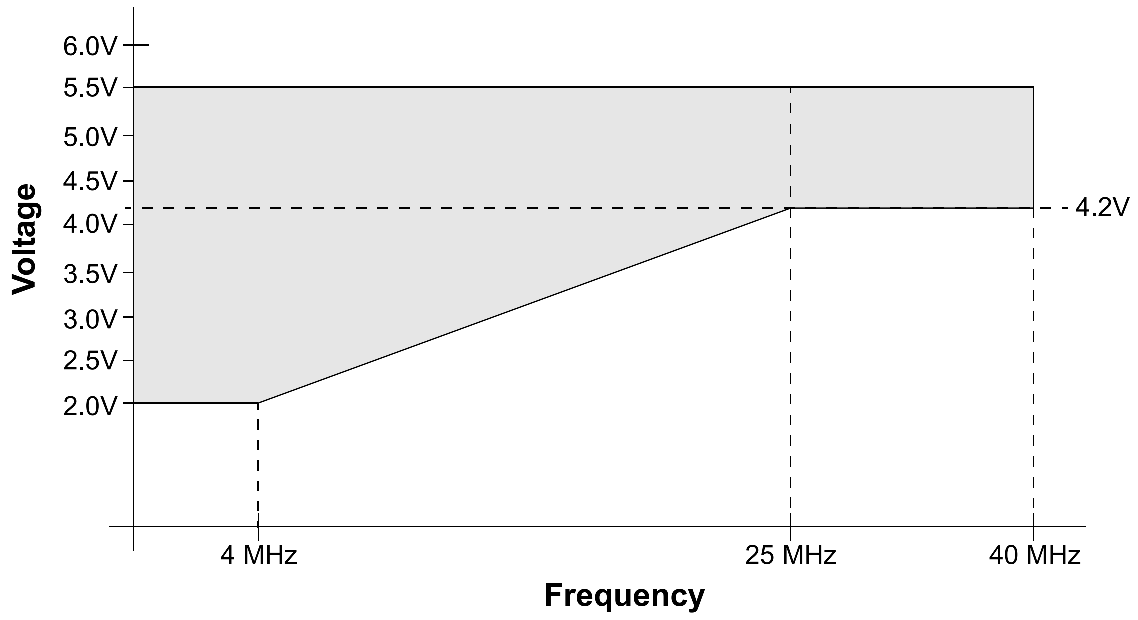
The demonstration and development boards can be used in teaching environments, for prototyping custom circuits and for learning about various microcontroller applications.

In addition to the PICDEM™ and dsPICDEM™ demonstration/development board series of circuits, Microchip has a line of evaluation kits and demonstration software for analog filter design, KEELOQ® security ICs, CAN, IrDA®, PowerSmart battery management, SEEVAL® evaluation system, Sigma-Delta ADC, flow rate sensing, plus many more.

Also available are starter kits that contain everything needed to experience the specified device. This usually includes a single application and debug capability, all on one board.

Check the Microchip web page (www.microchip.com) for the complete list of demonstration, development and evaluation kits.

**FIGURE 27-3: PIC18LF2221/2321/4221/4321 VOLTAGE-FREQUENCY GRAPH (INDUSTRIAL)**



FMAX = (9.54 MHz/V) (VDDAPPMIN – 2.0V) + 4 MHz

**Note:** VDDAPPMIN is the minimum voltage of the PIC® device in the application.