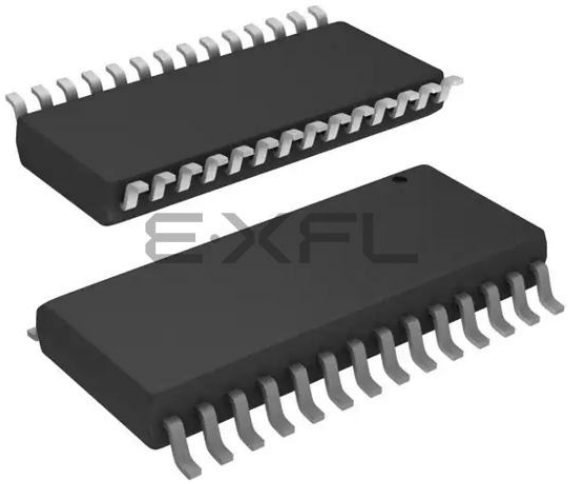


Welcome to E-XFL.COM

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	Active
Core Processor	PIC
Core Size	8-Bit
Speed	64MHz
Connectivity	I²C, LINbus, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, LVD, POR, PWM, WDT
Number of I/O	25
Program Memory Size	32KB (16K x 16)
Program Memory Type	FLASH
EEPROM Size	256 x 8
RAM Size	2K x 8
Voltage - Supply (Vcc/Vdd)	2.3V ~ 5.5V
Data Converters	A/D 35x10b; D/A 1x5b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	28-SOIC (0.295", 7.50mm Width)
Supplier Device Package	28-SOIC
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18f25k40t-i-so

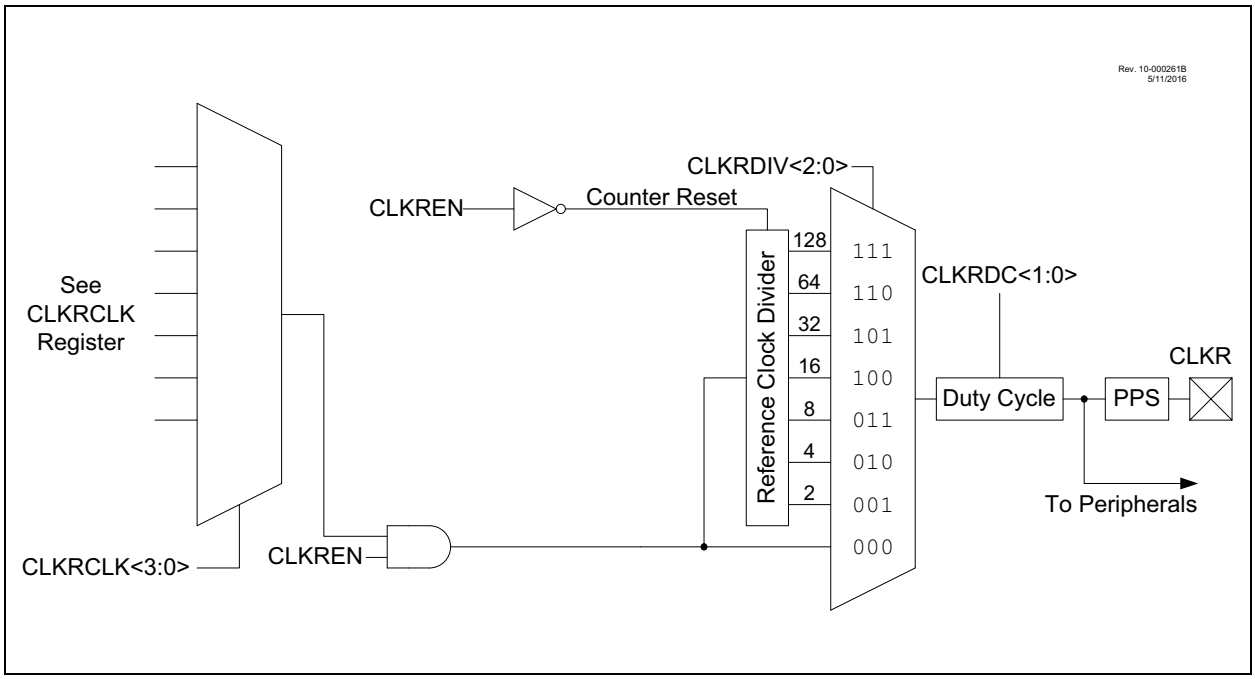
5.0 REFERENCE CLOCK OUTPUT MODULE

The reference clock output module provides the ability to send a clock signal to the clock reference output pin (CLKR). The reference clock output can also be used as a signal for other peripherals, such as the Data Signal Modulator (DSM), Memory Scanner and Timer module.

The reference clock output module has the following features:

- Selectable clock source using the CLKRCLK register
- Programmable clock divider
- Selectable duty cycle

FIGURE 5-1: CLOCK REFERENCE BLOCK DIAGRAM



10.0 MEMORY ORGANIZATION

There are three types of memory in PIC18 enhanced microcontroller devices:

- Program Memory
- Data RAM
- Data EEPROM

As Harvard architecture devices, the data and program memories use separate buses; this allows for concurrent access of the two memory spaces. The data EEPROM, for practical purposes, can be regarded as a peripheral device, since it is addressed and accessed through a set of control registers.

Additional detailed information on the operation of the Program Flash Memory and Data EEPROM Memory is provided in **Section 11.0 "Nonvolatile Memory (NVM) Control"**.

10.1 Program Memory Organization

PIC18 microcontrollers implement a 21-bit program counter, which is capable of addressing a 2 Mbyte program memory space. Accessing a location between the upper boundary of the physically implemented memory and the 2 Mbyte address will return all '0's (a NOP instruction).

These devices contains the following:

- PIC18(L)F24K40: 16 Kbytes of Flash memory, up to 8,192 single-word instructions
- PIC18(L)F25K40: 32 K bytes of Flash memory, up to 16,384 single-word instructions

PIC18 devices have two interrupt vectors. The Reset vector address is at 0000h and the interrupt vector addresses are at 0008h and 0018h.

Note:	For memory information on this family of devices, see Table 10-1 and Table 10-2.

TABLE 10-5: REGISTER FILE SUMMARY FOR PIC18(L)F24/25K40 DEVICES (CONTINUED)

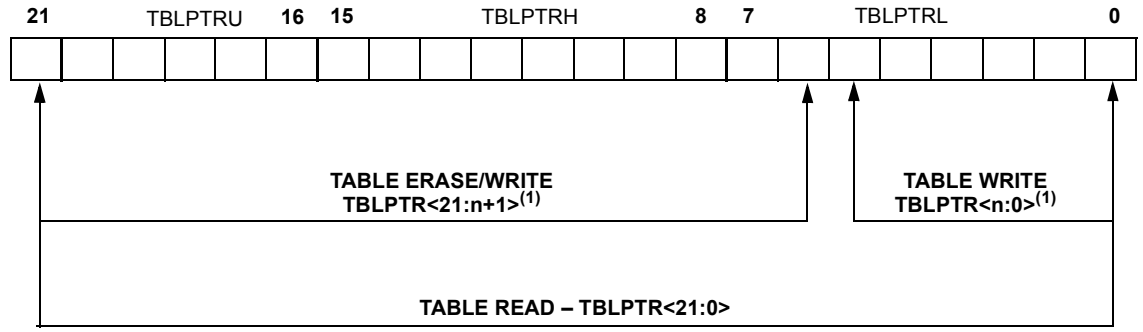
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR
F26h to F22h	—	Unimplemented								—
F21h	ANSELC	ANSELC7	ANSELC6	ANSELC5	ANSELC4	ANSELC3	ANSELC2	ANSELC1	ANSELC0	11111111
F20h	WPUC	WPUC7	WPUC6	WPUC5	WPUC4	WPUC3	WPUC2	WPUC1	WPUC0	00000000
F1Fh	ODCONC	ODCC7	ODCC6	ODCC5	ODCC4	ODCC3	ODCC2	ODCC1	ODCC0	00000000
F1Eh	SLRCONC	SLRC7	SLRC6	SLRC5	SLRC4	SLRC3	SLRC2	SLRC1	SLRC0	11111111
F1Dh	INLVLC	INLVLC7	INLVLC6	INLVLC5	INLVLC4	INLVLC3	INLVLC2	INLVLC1	INLVLC0	11111111
F1Ch	IOCCP	IOCCP7	IOCCP6	IOCCP5	IOCCP4	IOCCP3	IOCCP2	IOCCP1	IOCCP0	00000000
F1Bh	IOCCN	IOCCN7	IOCCN6	IOCCN5	IOCCN4	IOCCN3	IOCCN2	IOCCN1	IOCCN0	00000000
F1Ah	IOCCF	IOCCF7	IOCCF6	IOCCF5	IOCCF4	IOCCF3	IOCCF2	IOCCF1	IOCCF0	00000000
F19h	ANSELB	ANSELB7	ANSELB6	ANSELB5	ANSELB4	ANSELB3	ANSELB2	ANSELB1	ANSELB0	11111111
F18h	WPUB	WPUB7	WPUB6	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0	00000000
F17h	ODCONB	ODCB7	ODCB6	ODCB5	ODCB4	ODCB3	ODCB2	ODCB1	ODCB0	00000000
F16h	SLRCONB	SLRB7	SLRB6	SLRB5	SLRB4	SLRB3	SLRB2	SLRB1	SLRB0	11111111
F15h	INLVLB	INLVLB7	INLVLB6	INLVLB5	INLVLB4	INLVLB3	INLVLB2	INLVLB1	INLVLB0	11111111
F14h	IOCBP	IOCBP7	IOCBP6	IOCBP5	IOCBP4	IOCBP3	IOCBP2	IOCBP1	IOCBP0	00000000
F13h	IOCBN	IOCBN7	IOCBN6	IOCBN5	IOCBN4	IOCBN3	IOCBN2	IOCBN1	IOCBN0	00000000
F12h	IOCBF	IOCBF7	IOCBF6	IOCBF5	IOCBF4	IOCBF3	IOCBF2	IOCBF1	IOCBF0	00000000
F11h	ANSELA	ANSELA7	ANSELA6	ANSELA5	ANSELA4	ANSELA3	ANSELA2	ANSELA1	ANSELA0	11111111
F10h	WPUA	WPUA7	WPUA6	WPUA5	WPUA4	WPUA3	WPUA2	WPUA1	WPUA0	00000000
F0Fh	ODCONA	ODCA7	ODCA6	ODCA5	ODCA4	ODCA3	ODCA2	ODCA1	ODCA0	00000000
F0Eh	SLRCONA	SLRA7	SLRA6	SLRA5	SLRA4	SLRA3	SLRA2	SLRA1	SLRA0	11111111
F0Dh	INLVLA	INLVLA7	INLVLA6	INLVLA5	INLVLA4	INLVLA3	INLVLA2	INLVLA1	INLVLA0	11111111
F0Ch	IOCAP	IOCAP7	IOCAP6	IOCAP5	IOCAP4	IOCAP3	IOCAP2	IOCAP1	IOCAP0	00000000
F0Bh	IOCAN	IOCAN7	IOCAN6	IOCAN5	IOCAN4	IOCAN3	IOCAN2	IOCAN1	IOCAN0	00000000
F0Ah	IOCAF	IOCAF7	IOCAF6	IOCAF5	IOCAF4	IOCAF3	IOCAF2	IOCAF1	IOCAF0	00000000
F09h to EFFh	—	Unimplemented								—
EFEh	RC7PPS	—	—	—	RC7PPS<4:0>					---00000
EFDh	RC6PPS	—	—	—	RC6PPS<4:0>					---00000
EFCh	RC5PPS	—	—	—	RC5PPS<4:0>					---00000
EFBh	RC4PPS	—	—	—	RC4PPS<4:0>					---00000
EFAh	RC3PPS	—	—	—	RC3PPS<4:0>					---00000
EF9h	RC2PPS	—	—	—	RC2PPS<4:0>					---00000
EF8h	RC1PPS	—	—	—	RC1PPS<4:0>					---00000
EF7h	RC0PPS	—	—	—	RC0PPS<4:0>					---00000
EF6h	RB7PPS	—	—	—	RB7PPS<4:0>					---00000
EF5h	RB6PPS	—	—	—	RB6PPS<4:0>					---00000
EF4h	RB5PPS	—	—	—	RB5PPS<4:0>					---00000
EF3h	RB4PPS	—	—	—	RB4PPS<4:0>					---00000

Legend: x = unknown, u = unchanged, — = unimplemented, q = value depends on condition
Note 1: Not available on LF devices.

TABLE 11-3: TABLE POINTER OPERATIONS WITH TBLRD AND TBLWT INSTRUCTIONS

Example	Operation on Table Pointer
TBLRD* TBLWT*	TBLPTR is not modified
TBLRD*+ TBLWT*+	TBLPTR is incremented after the read/write
TBLRD*- TBLWT*-	TBLPTR is decremented after the read/write
TBLRD+* TBLWT+*	TBLPTR is incremented before the read/write

FIGURE 11-3: TABLE POINTER BOUNDARIES BASED ON OPERATION



Note 1: Refer to Table 11-3 for the row size values.

REGISTER 11-2: NVMCON2: NONVOLATILE MEMORY CONTROL 2 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
NVMCON2<7:0>							
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
x = Bit is unknown	'0' = Bit is cleared	'1' = Bit is set
-n = Value at POR		

bit 7-0 **NVMCON2<7:0>:**
Refer to **Section 11.1.4 "NVM Unlock Sequence"**.

Note 1: This register always reads zeros, regardless of data written.

Register 11-3: NVMADRL: Data EEPROM Memory Address Low

R/W-x/0	R/W-x/0	R/W-x/0	R/W-x/0	R/W-x/0	R/W-x/0	R/W-x/0	R/W-x/0
NVMADR<7:0>							
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
x = Bit is unknown	'0' = Bit is cleared	'1' = Bit is set
-n = Value at POR		

bit 7-0 **NVMADR<7:0>:** EEPROM Read Address bits

REGISTER 11-4: NVMADRH: DATA EEPROM MEMORY ADDRESS HIGH(1)

U-0	U-0	U-0	U-0	U-0	U-0	R/W-x/u	R/W-x/u
—	—	—	—	—	—	NVMADR<9:8>	
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
x = Bit is unknown	'0' = Bit is cleared	'1' = Bit is set
-n = Value at POR		

bit 7-2 **Unimplemented:** Read as '0'
bit 1-0 **NVMADR<9:8>:** EEPROM Read Address bits

Note 1: The NVMADRH register is not implemented on PIC18(L)F24/25K40.

15.2.5 SLEW RATE CONTROL

The SLRCONx register (Register 15-7) controls the slew rate option for each port pin. Slew rate for each port pin can be controlled independently. When an SLRCONx bit is set, the corresponding port pin drive is slew rate limited. When an SLRCONx bit is cleared, The corresponding port pin drive slews at the maximum rate possible.

15.2.6 INPUT THRESHOLD CONTROL

The INLVLx register (Register 15-8) controls the input voltage threshold for each of the available PORTx input pins. A selection between the Schmitt Trigger CMOS or the TTL compatible thresholds is available. The input threshold is important in determining the value of a read of the PORTx register and also the level at which an interrupt-on-change occurs, if that feature is enabled. See Table 37-8 for more information on threshold levels.

Note:	Changing the input threshold selection should be performed while all peripheral modules are disabled. Changing the threshold level during the time a module is active may inadvertently generate a transition associated with an input pin, regardless of the actual voltage level on that pin.

15.2.7 WEAK PULL-UP CONTROL

The WPUx register (Register 15-5) controls the individual weak pull-ups for each port pin.

15.2.8 EDGE SELECTABLE INTERRUPT-ON-CHANGE

An interrupt can be generated by detecting a signal at the port pin that has either a rising edge or a falling edge. Any individual pin can be configured to generate an interrupt. The interrupt-on-change module is present on all the pins that are common between 28-pin and 40/44-pin devices. For further details about the IOC module refer to **Section 16.0 “Interrupt-on-Change”**.

15.3 PORTE Registers

EXAMPLE 15-2: INITIALIZING PORTE

```
CLRF    PORTE   ; Initialize PORTE by
                ; clearing output
                ; data latches
CLRF    LATE    ; Alternate method
                ; to clear output
                ; data latches
CLRF    ANSELE  ; Configure analog pins
                ; for digital only
MOVLW   05h     ; Value used to
                ; initialize data
                ; direction
MOVWF   TRISE   ; Set RE<0> as input
                ; RE<1> as output
                ; RE<2> as input
```

15.3.1 PORTE ON 28-PIN DEVICES

For PIC18(L)F2xK40 devices, PORTE is only available when Master Clear functionality is disabled (MCLRE = 0). In this case, PORTE is a single bit, input-only port comprised of RE3 only. The pin operates as previously described. RE3 in PORTE register is a read-only bit and will read ‘1’ when MCLRE = 1 (i.e., Master Clear enabled).

15.3.2 RE3 WEAK PULL-UP

The port RE3 pin has an individually controlled weak internal pull-up. When set, the WPUE3 bit enables the RE3 pin pull-up. When the RE3 port pin is configured as $\overline{\text{MCLR}}$, (CONFIG2L, MCLRE = 1 and CONFIG4H, LVP = 0), or configured for Low-Voltage Programming, (MCLRE = x and LVP = 1), the pull-up is always enabled and the WPUE3 bit has no effect.

15.3.3 INTERRUPT-ON-CHANGE

The interrupt-on-change feature is available only on the RE3 pin for all devices. For further details refer to **Section 14.11 “Interrupt-on-Change”**.

19.7 Timer1/3/5 16-Bit Read/Write Mode

Timer1/3/5 can be configured to read and write all 16 bits of data, to and from, the 8-bit TMRxL and TMRxH registers, simultaneously. The 16-bit read and write operations are enabled by setting the RD16 bit of the TxCON register.

To accomplish this function, the TMRxH register value is mapped to a buffer register called the TMRxH buffer register. While in 16-Bit mode, the TMRxH register is not directly readable or writable and all read and write operations take place through the use of this TMRxH buffer register.

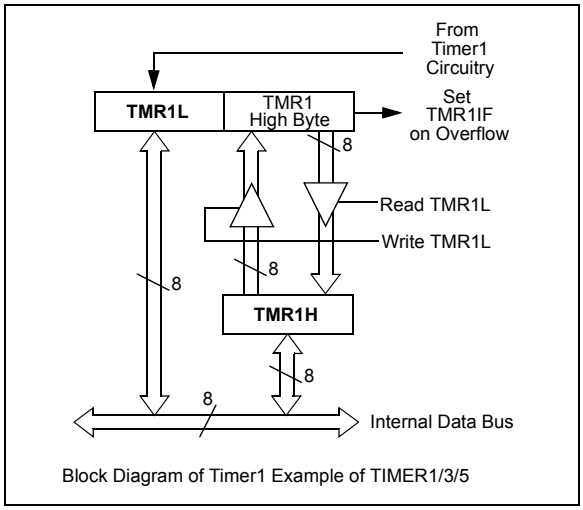
When a read from the TMRxL register is requested, the value of the TMRxH register is simultaneously loaded into the TMRxH buffer register. When a read from the TMRxH register is requested, the value is provided from the TMRxH buffer register instead. This provides the user with the ability to accurately read all 16 bits of the Timer1/3/5 value from a single instance in time. Reference the block diagram in Figure 19-2 for more details.

In contrast, when not in 16-Bit mode, the user must read each register separately and determine if the values have become invalid due to a rollover that may have occurred between the read operations.

When a write request of the TMRxL register is requested, the TMRxH buffer register is simultaneously updated with the contents of the TMRxH register. The value of TMRxH must be preloaded into the TMRxH buffer register prior to the write request for the TMRxL register. This provides the user with the ability to write all 16 bits to the TMRxL:TMRxH register pair at the same time.

Any requests to write to the TMRxH directly does not clear the Timer1/3/5 prescaler value. The prescaler value is only cleared through write requests to the TMRxL register.

FIGURE 19-2: TIMER1/3/5 16-BIT READ/WRITE MODE BLOCK DIAGRAM



Block Diagram of Timer1 Example of TIMER1/3/5

19.8 Timer1/3/5 Gate

Timer1/3/5 can be configured to count freely or the count can be enabled and disabled using Timer1/3/5 gate circuitry. This is also referred to as Timer1/3/5 gate enable.

Timer1/3/5 gate can also be driven by multiple selectable sources.

19.8.1 TIMER1/3/5 GATE ENABLE

The Timer1/3/5 Gate Enable mode is enabled by setting the TMRxGE bit of the TxGCON register. The polarity of the Timer1/3/5 Gate Enable mode is configured using the TxGPOL bit of the TxGCON register.

When Timer1/3/5 Gate Enable mode is enabled, Timer1/3/5 will increment on the rising edge of the Timer1/3/5 clock source. When Timer1/3/5 Gate signal is inactive, the timer will not increment and hold the current count. Enable mode is disabled, no incrementing will occur and Timer1/3/5 will hold the current count. See Figure 19-4 for timing details.

TABLE 19-3: TIMER1/3/5 GATE ENABLE SELECTIONS

TMRxCLK	TxGPOL	TxG	Timer1/3/5 Operation
↑	1	1	Counts
↑	1	0	Holds Count
↑	0	1	Holds Count
↑	0	0	Counts

TABLE 19-4: SUMMARY OF REGISTERS ASSOCIATED WITH TIMER1/3/5 AS A TIMER/COUNTER

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	IPEN	—	—	INT2EDG	INT1EDG	INT0EDG	166
PIE4	—	—	TMR6IE	TMR5IE	TMR4IE	TMR3IE	TMR2IE	TMR1IE	179
PIE5	—	—	—	—	—	TMR5GIE	TMR3GIE	TMR1GIE	180
PIR4	—	—	TMR6IF	TMR5IF	TMR4IF	TMR3IF	TMR2IF	TMR1IF	170
PIR5	—	—	—	—	—	TMR5GIF	TMR3GIF	TMR1GIF	171
IPR4	—	—	TMR6IP	TMR5IP	TMR4IP	TMR3IP	TMR2IP	TMR1IP	187
IPR5	—	—	—	—	—	TMR5GIP	TMR3GIP	TMR1GIP	188
PMD1	—	TMR6MD	TMR5MD	TMR4MD	TMR3MD	TMR2MD	TMR1MD	TMR0MD	65
T1CON	—	—	CKPS<1:0>		—	$\overline{\text{SYNC}}$	RD16	ON	223
T1GCON	GE	GPOL	GTM	GSPM	GO/$\overline{\text{DONE}}$	GVAL	—	—	224
T3CON	—	—	CKPS<1:0>		—	$\overline{\text{SYNC}}$	RD16	ON	223
T3GCON	GE	GPOL	GTM	GSPM	GO/$\overline{\text{DONE}}$	GVAL	—	—	224
T5CON	—	—	CKPS<1:0>		—	$\overline{\text{SYNC}}$	RD16	ON	223
T5GCON	GE	GPOL	GTM	GSPM	GO/$\overline{\text{DONE}}$	GVAL	—	—	224
TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								227
TMR1L	Least Significant Byte of the 16-bit TMR1 Register								227
TMR3H	Holding Register for the Most Significant Byte of the 16-bit TMR3 Register								227
TMR3L	Least Significant Byte of the 16-bit TMR3 Register								227
TMR5H	Holding Register for the Most Significant Byte of the 16-bit TMR5 Register								227
TMR5L	Least Significant Byte of the 16-bit TMR5 Register								227
T1CKIPPS	—	—	—	T1CKIPPS<4:0>					211
T1GPPS	—	—	—	T1GPPS<4:0>					211
T3CKIPPS	—	—	—	T3CKIPPS<4:0>					211
T3GPPS	—	—	—	T3GPPS<4:0>					211
T5CKIPPS	—	—	—	T5CKIPPS<4:0>					211
T5GPPS	—	—	—	T5GPPS<4:0>					211

Legend: — = Unimplemented location, read as '0'. Shaded cells are not used by TIMER1/3/5.

REGISTER 26-2: SSPxCON1: MSSPx CONTROL REGISTER 1 (SPI MODE)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV(1)	SSPEN(2)	CKP	SSPM3(4)	SSPM2(4)	SSPM1(4)	SSPM0(4)
bit 7							bit 0

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

bit 7 **WCOL:** Write Collision Detect bit
- 1 = The SSPxBUF register is written while it is still transmitting the previous word (must be cleared in software)
- 0 = No collision

bit 6 **SSPOV:** Receive Overflow Indicator bit(1)

SPI Slave mode:
- 1 = A new byte is received while the SSPxBUF register is still holding the previous data. In case of overflow, the data in SSPxSR is lost. Overflow can only occur in Slave mode. The user must read the SSPxBUF, even if only transmitting data, to avoid setting overflow (must be cleared in software).
- 0 = No overflow

bit 5 **SSPEN:** Master Synchronous Serial Port Enable bit(2)
- 1 = Enables serial port and configures SCKx, SDOx, SDIx and $\overline{SSx}$ as serial port pins
- 0 = Disables serial port and configures these pins as I/O port pins

bit 4 **CKP:** Clock Polarity Select bit
- 1 = Idle state for the clock is a high level
- 0 = Idle state for the clock is a low level

bit 3-0 **SSPM<3:0>:** Master Synchronous Serial Port Mode Select bits(4)
- 1010 = SPI Master mode: Clock = FOSC/(4 * (SSPxADD + 1))(3)
- 0101 = SPI Slave mode: Clock = SCKx pin; $\overline{SSx}$ pin control is disabled; $\overline{SSx}$ can be used as I/O pin
- 0100 = SPI Slave mode: Clock = SCKx pin; $\overline{SSx}$ pin control is enabled
- 0011 = SPI Master mode: Clock = TMR2 output/2
- 0010 = SPI Master mode: Clock = FOSC/64
- 0001 = SPI Master mode: Clock = FOSC/16
- 0000 = SPI Master mode: Clock = FOSC/4

Note 1: In Master mode, the overflow bit is not set since each new reception (and transmission) is initiated by writing to the SSPxBUF register.

2: When enabled, these pins must be properly configured as inputs or outputs.

3: SSPxADD = 0 is not supported.

4: Bit combinations not specifically listed here are either reserved or implemented in I^2C mode only.

26.5.4 SLAVE SELECT SYNCHRONIZATION

The Slave Select can also be used to synchronize communication. The Slave Select line is held high until the master device is ready to communicate. When the Slave Select line is pulled low, the slave knows that a new transmission is starting.

If the slave fails to receive the communication properly, it will be reset at the end of the transmission, when the Slave Select line returns to a high state. The slave is then ready to receive a new transmission when the Slave Select line is pulled low again. If the Slave Select line is not used, there is a risk that the slave will eventually become out of sync with the master. If the slave misses a bit, it will always be one bit off in future transmissions. Use of the Slave Select line allows the slave and master to align themselves at the beginning of each transmission.

The $\overline{SS}$ pin allows a Synchronous Slave mode. The SPI must be in Slave mode with $\overline{SS}$ pin control enabled (SSPxCON1<3:0> = 0100).

When the $\overline{SS}$ pin is low, transmission and reception are enabled and the SDO pin is driven.

When the $\overline{SS}$ pin goes high, the SDO pin is no longer driven, even if in the middle of a transmitted byte and becomes a floating output. External pull-up/pull-down resistors may be desirable depending on the application.

Note 1:	When the SPI is in Slave mode with $\overline{SS}$ pin control enabled (SSPxCON1<3:0> = 0100), the SPI module will reset if the $\overline{SS}$ pin is set to VDD.
2:	When the SPI is used in Slave mode with CKE set; the user must enable $\overline{SS}$ pin control.
3:	While operated in SPI Slave mode the SMP bit of the SSPxSTAT register must remain clear.

When the SPI module resets, the bit counter is forced to '0'. This can be done by either forcing the $\overline{SS}$ pin to a high level or clearing the SSPEN bit.

FIGURE 26-5: SPI DAISY-CHAIN CONNECTION

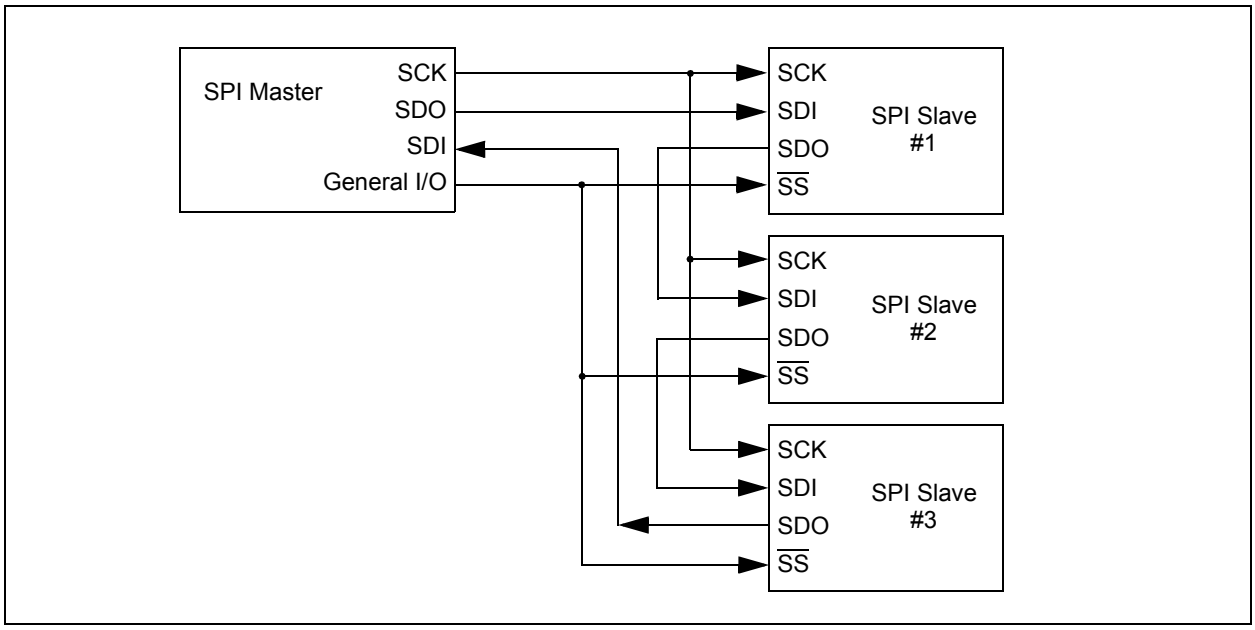


TABLE 27-7: SUMMARY OF REGISTERS ASSOCIATED WITH SYNCHRONOUS MASTER TRANSMISSION

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register on Page
BAUDxCON	ABDOVF	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN	389
INTCON	GIE/GIEH	PEIE/GIEL	IPEN	—	—	INT2EDG	INT1EDG	INT0EDG	166
PIE3	—	—	RC1IE	TX1IE	—	—	BCL1IE	SSP1IE	178
PIR3	—	—	RC1IF	TX1IF	—	—	BCL1IF	SSP1IF	170
IPR3	—	—	RC1IP	TX1IP	—	—	BCL1IP	SSP1IP	186
RCxSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	388
RxyPPS	—	—	—	RxyPPS<4:0>					213
TXxPPS	—	—	—	TXPPS<4:0>					211
SPxBRGH	EUSARTx Baud Rate Generator, High Byte								398*
SPxBRGL	EUSARTx Baud Rate Generator, Low Byte								398*
TXxREG	EUSARTx Transmit Data Register								390*
TXxSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D	387

Legend: — = unimplemented location, read as '0'. Shaded cells are not used for synchronous master transmission.
* Page provides register information.

TABLE 31-1: ADC CLOCK PERIOD (T_{AD}) VS. DEVICE OPERATING FREQUENCIES(1,4)

ADC Clock Period (T_{AD})		Device Frequency (F_{OSC})						
ADC Clock Source	ADCS<5:0>	64 MHz	32 MHz	20 MHz	16 MHz	8 MHz	4 MHz	1 MHz
F_{OSC}/2	000000	31.25 ns(2)	62.5 ns(2)	100 ns(2)	125 ns(2)	250 ns(2)	500 ns(2)	2.0 μs
F_{OSC}/4	000001	62.5 ns(2)	125 ns(2)	200 ns(2)	250 ns(2)	500 ns(2)	1.0 μs	4.0 μs
F_{OSC}/6	000010	125 ns(2)	187.5 ns(2)	300 ns(2)	375 ns(2)	750 ns(2)	1.5 μs	6.0 μs
F_{OSC}/8	000011	187.5 ns(2)	250 ns(2)	400 ns(2)	500 ns(2)	1.0 μs	2.0 μs	8.0 μs(3)
...	...	...	...	...	...	...	...	...
F_{OSC}/16	000100	250 ns(2)	500 ns(2)	800 ns(2)	1.0 μs	2.0 μs	4.0 μs	16.0 μs(3)
...	...	...	...	...	...	...	...	...
F_{OSC}/128	111111	2.0 μs	4.0 μs	6.4 μs	8.0 μs	16.0 μs(3)	32.0 μs(2)	128.0 μs(2)
FRC	ADCS(ADCON0<4>) = 1	1.0-6.0 μs	1.0-6.0 μs	1.0-6.0 μs	1.0-6.0 μs	1.0-6.0 μs	1.0-6.0 μs	1.0-6.0 μs

Legend: Shaded cells are outside of recommended range.

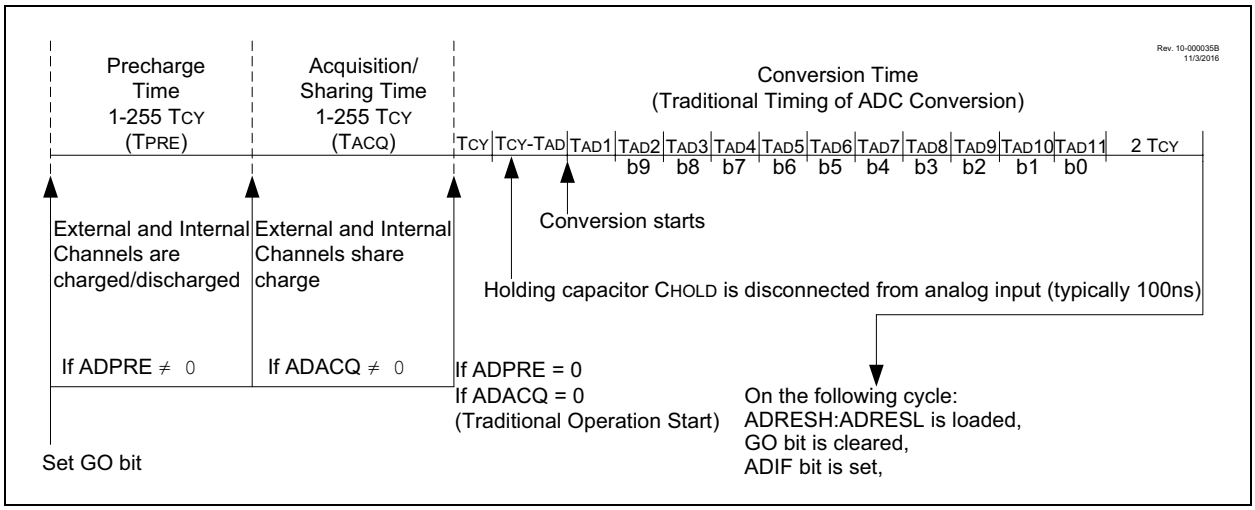
Note 1: See T_{AD} parameter for FRC source typical T_{AD} value.

2: These values violate the required T_{AD} time.

3: Outside the recommended T_{AD} time.

4: The ADC clock period (T_{AD}) and total ADC conversion time can be minimized when the ADC clock is derived from the system clock F_{OSC}. However, the FRC oscillator source must be used when conversions are to be performed with the device in Sleep mode.

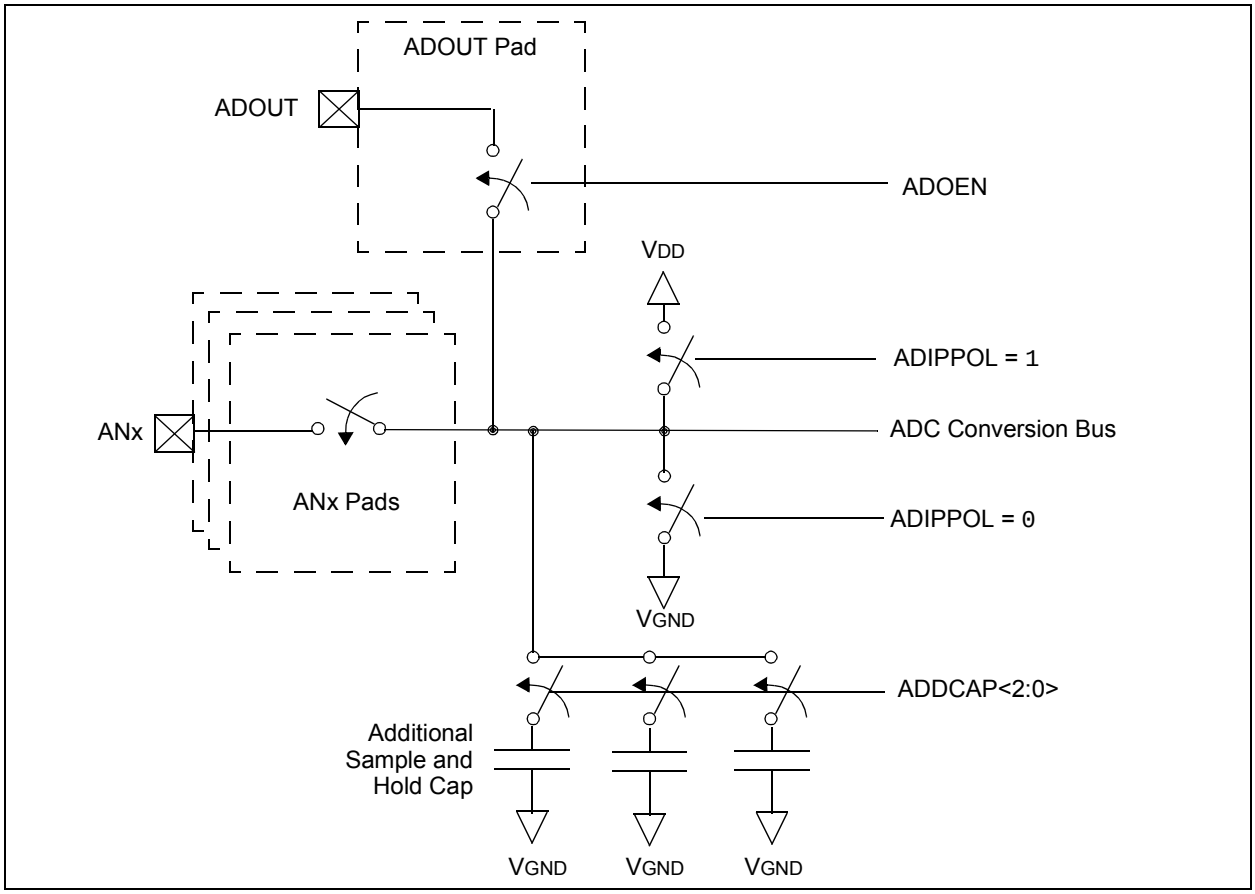
FIGURE 31-2: ANALOG-TO-DIGITAL CONVERSION T_{AD} CYCLES



31.4 Capacitive Voltage Divider (CVD) Features

The ADC module contains several features that allow the user to perform a relative capacitance measurement on any ADC channel using the internal ADC sample and hold capacitance as a reference. This relative capacitance measurement can be used to implement capacitive touch or proximity sensing applications. Figure 31-6 shows the basic block diagram of the CVD portion of the ADC module.

FIGURE 31-6: HARDWARE CAPACITIVE VOLTAGE DIVIDER BLOCK DIAGRAM



35.0 INSTRUCTION SET SUMMARY

PIC18(L)F2x/4xK40 devices incorporate the standard set of 75 PIC18 core instructions, as well as an extended set of eight new instructions, for the optimization of code that is recursive or that utilizes a software stack. The extended set is discussed later in this section.

35.1 Standard Instruction Set

The standard PIC18 instruction set adds many enhancements to the previous PIC® MCU instruction sets, while maintaining an easy migration from these PIC® MCU instruction sets. Most instructions are a single program memory word (16 bits), but there are four instructions that require two program memory locations.

Each single-word instruction is a 16-bit word divided into an opcode, which specifies the instruction type and one or more operands, which further specify the operation of the instruction.

The instruction set is highly orthogonal and is grouped into four basic categories:

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal** operations
- **Control** operations

The PIC18 instruction set summary in Table 35-2 lists **byte-oriented**, **bit-oriented**, **literal** and **control** operations. Table 35-1 shows the opcode field descriptions.

Most **byte-oriented** instructions have three operands:

1. The file register (specified by ‘f’)
2. The destination of the result (specified by ‘d’)
3. The accessed memory (specified by ‘a’)

The file register designator ‘f’ specifies which file register is to be used by the instruction. The destination designator ‘d’ specifies where the result of the operation is to be placed. If ‘d’ is zero, the result is placed in the WREG register. If ‘d’ is one, the result is placed in the file register specified in the instruction.

All **bit-oriented** instructions have three operands:

1. The file register (specified by ‘f’)
2. The bit in the file register (specified by ‘b’)
3. The accessed memory (specified by ‘a’)

The bit field designator ‘b’ selects the number of the bit affected by the operation, while the file register designator ‘f’ represents the number of the file in which the bit is located.

The literal instructions may use some of the following operands:

- A literal value to be loaded into a file register (specified by ‘k’)
- The desired FSR register to load the literal value into (specified by ‘f’)
- No operand required (specified by ‘—’)

The control instructions may use some of the following operands:

- A program memory address (specified by ‘n’)
- The mode of the CALL or RETURN instructions (specified by ‘s’)
- The mode of the table read and table write instructions (specified by ‘m’)
- No operand required (specified by ‘—’)

All instructions are a single word, except for four double-word instructions. These instructions were made double-word to contain the required information in 32 bits. In the second word, the four MSbs are ‘1’s. If this second word is executed as an instruction (by itself), it will execute as a NOP.

All single-word instructions are executed in a single instruction cycle, unless a conditional test is true or the program counter is changed as a result of the instruction. In these cases, the execution takes two instruction cycles, with the additional instruction cycle(s) executed as a NOP.

The double-word instructions execute in two instruction cycles.

One instruction cycle consists of four oscillator periods. Thus, for an oscillator frequency of 4 MHz, the normal instruction execution time is 1 μs. If a conditional test is true, or the program counter is changed as a result of an instruction, the instruction execution time is 2 μs. Two-word branch instructions (if true) would take 3 μs.

Figure 35-1 shows the general formats that the instructions can have. All examples use the convention ‘nnh’ to represent a hexadecimal number.

The Instruction Set Summary, shown in Table 35-2, lists the standard instructions recognized by the Microchip Assembler (MPASM™).

Section 35.1.1 “Standard Instruction Set” provides a description of each instruction.

COMF **Complement f**

Syntax: COMF f {,d {,a}}

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

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.
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 35.2.3 "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: $0 \le f \le 255$
$a \in [0,1]$

Operation: (f) – (W),
skip if (f) = (W)
(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 2-cycle instruction.
If 'a' is '0', the Access Bank is selected.
If 'a' is '1', the BSR is used to select the GPR bank.
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 35.2.3 "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)

DECFSZ **Decrement f, skip if 0**

Syntax: DECFSZ f {,d {,a}}

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

Operation: (f) – 1 → dest,
skip if result = 0

Status Affected: None

Encoding:

0010	11da	ffff	ffff

Description: The contents of register 'f' are decremented. If 'd' is '0', the result is placed in W. If 'd' is '1', the result is placed back in register 'f' (default).
If the result is '0', the next instruction, which is already fetched, is discarded and a NOP is executed instead, making it a 2-cycle instruction.
If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank.
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 35.2.3 "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	Write to destination

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    DECFSZ  CNT, 1, 1
        GOTO    LOOP
CONTINUE
```
Before Instruction
PC = Address (HERE)
After Instruction
CNT = CNT - 1
If CNT = 0;
PC = Address (CONTINUE)
If CNT ≠ 0;
PC = Address (HERE + 2)

DCFSNZ **Decrement f, skip if not 0**

Syntax: DCFSNZ f {,d {,a}}

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

Operation: (f) – 1 → dest,
skip if result ≠ 0

Status Affected: None

Encoding:

0100	11da	ffff	ffff

Description: The contents of register 'f' are decremented. If 'd' is '0', the result is placed in W. If 'd' is '1', the result is placed back in register 'f' (default).
If the result is not '0', the next instruction, which is already fetched, is discarded and a NOP is executed instead, making it a 2-cycle instruction.
If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank.
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 35.2.3 "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	Write to destination

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    DCFSNZ  TEMP, 1, 0
ZERO    :
NZERO   :
```
Before Instruction
TEMP = ?
After Instruction
TEMP = TEMP – 1,
If TEMP = 0;
PC = Address (ZERO)
If TEMP ≠ 0;
PC = Address (NZERO)

INCFSZ **Increment f, skip if 0**

Syntax: INCFSZ f {,d {,a}}

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

Operation: (f) + 1 → dest,
skip if result = 0

Status Affected: None

Encoding:

0011	11da	ffff	ffff

Description: The contents of register 'f' are incremented. If 'd' is '0', the result is placed in W. If 'd' is '1', the result is placed back in register 'f' (default).
If the result is '0', the next instruction, which is already fetched, is discarded and a NOP is executed instead, making it a 2-cycle instruction.
If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank.
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 35.2.3 "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	Write to destination

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    INCFSZ   CNT, 1, 0
NZERO   :
ZERO    :
```

Before Instruction
PC = Address (HERE)
After Instruction
CNT = CNT + 1
If CNT = 0;
PC = Address (ZERO)
If CNT ≠ 0;
PC = Address (NZERO)

INFSNZ **Increment f, skip if not 0**

Syntax: INFSNZ f {,d {,a}}

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

Operation: (f) + 1 → dest,
skip if result ≠ 0

Status Affected: None

Encoding:

0100	10da	ffff	ffff

Description: The contents of register 'f' are incremented. If 'd' is '0', the result is placed in W. If 'd' is '1', the result is placed back in register 'f' (default).
If the result is not '0', the next instruction, which is already fetched, is discarded and a NOP is executed instead, making it a 2-cycle instruction.
If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank.
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 35.2.3 "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	Write to destination

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    INFSNZ  REG, 1, 0
ZERO
NZERO
```

Before Instruction
PC = Address (HERE)
After Instruction
REG = REG + 1
If REG ≠ 0;
PC = Address (NZERO)
If REG = 0;
PC = Address (ZERO)

LFSR **Load FSR**

Syntax: LFSR f, k

Operands: $0 \le f \le 2$
$0 \le k \le 4095$

Operation: $k \rightarrow FSRf$

Status Affected: None

Encoding:

1110 1111	1110 0000	00ff $k_7$kkk	k_{11}kkk kkkk

Description: The 12-bit literal 'k' is loaded into the File Select Register pointed to by 'f'.

Words: 2

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k' MSB	Process Data	Write literal 'k' MSB to FSRfH
Decode	Read literal 'k' LSB	Process Data	Write literal 'k' to FSRfL

Example: `LFSR 2, 3ABh`

After Instruction
FSR2H = 03h
FSR2L = ABh

MOVF **Move f**

Syntax: MOVF f {,d {,a}}

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

Operation: $f \rightarrow dest$

Status Affected: N, Z

Encoding:

0101	00da	ffff	ffff

Description: The contents of register 'f' are moved to a destination dependent upon the status of 'd'. If 'd' is '0', the result is placed in W. If 'd' is '1', the result is placed back in register 'f' (default). Location 'f' can be anywhere in the 256-byte bank.
If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank.
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 35.2.3 "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 W

Example: `MOVF REG, 0, 0`

Before Instruction
REG = 22h
W = FFh
After Instruction
REG = 22h
W = 22h

SUBFSR **Subtract Literal from FSR**

Syntax: SUBFSR f, k

Operands: $0 \le k \le 63$

$f \in [0, 1, 2]$

Operation: FSR(f) – k → FSRf

Status Affected: None

Encoding:

1110	1001	ffkk	kkkk

Description: The 6-bit literal 'k' is subtracted from the contents of the FSR specified by 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: SUBFSR 2, 23h

Before Instruction
FSR2 = 03FFh

After Instruction
FSR2 = 03DCh

SUBULNK **Subtract Literal from FSR2 and Return**

Syntax: SUBULNK k

Operands: $0 \le k \le 63$

Operation: FSR2 – k → FSR2
(TOS) → PC

Status Affected: None

Encoding:

1110	1001	11kk	kkkk

Description: The 6-bit literal 'k' is subtracted from the contents of the FSR2. A RETURN is then executed by loading the PC with the TOS. The instruction takes two cycles to execute; a NOP is performed during the second cycle.
This may be thought of as a special case of the SUBFSR instruction, where f = 3 (binary '11'); it operates only on FSR2.

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination
No Operation	No Operation	No Operation	No Operation

Example: SUBULNK 23h

Before Instruction
FSR2 = 03FFh
PC = 0100h

After Instruction
FSR2 = 03DCh
PC = (TOS)

36.0 DEVELOPMENT SUPPORT

The PIC® microcontrollers (MCU) and dsPIC® digital signal controllers (DSC) are supported with a full range of software and hardware development tools:

- Integrated Development Environment
  - MPLAB® X IDE Software
- Compilers/Assemblers/Linkers
  - MPLAB XC Compiler
  - MPASM™ Assembler
  - MPLINK™ Object Linker/ MPLIB™ Object Librarian
  - MPLAB Assembler/Linker/Librarian for Various Device Families
- Simulators
  - MPLAB X SIM Software Simulator
- Emulators
  - MPLAB REAL ICE™ In-Circuit Emulator
- In-Circuit Debuggers/Programmers
  - MPLAB ICD 3
  - PICkit™ 3
- Device Programmers
  - MPLAB PM3 Device Programmer
- Low-Cost Demonstration/Development Boards, Evaluation Kits and Starter Kits
- Third-party development tools

36.1 MPLAB X Integrated Development Environment Software

The MPLAB X IDE is a single, unified graphical user interface for Microchip and third-party software, and hardware development tool that runs on Windows®, Linux and Mac OS® X. Based on the NetBeans IDE, MPLAB X IDE is an entirely new IDE with a host of free software components and plug-ins for high-performance application development and debugging. Moving between tools and upgrading from software simulators to hardware debugging and programming tools is simple with the seamless user interface.

With complete project management, visual call graphs, a configurable watch window and a feature-rich editor that includes code completion and context menus, MPLAB X IDE is flexible and friendly enough for new users. With the ability to support multiple tools on multiple projects with simultaneous debugging, MPLAB X IDE is also suitable for the needs of experienced users.

Feature-Rich Editor:

- Color syntax highlighting
- Smart code completion makes suggestions and provides hints as you type
- Automatic code formatting based on user-defined rules
- Live parsing

User-Friendly, Customizable Interface:

- Fully customizable interface: toolbars, toolbar buttons, windows, window placement, etc.
- Call graph window

Project-Based Workspaces:

- Multiple projects
- Multiple tools
- Multiple configurations
- Simultaneous debugging sessions

File History and Bug Tracking:

- Local file history feature
- Built-in support for Bugzilla issue tracker