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, LVD, POR, PWM, WDT
Number of I/O	22
Program Memory Size	32KB (16K x 16)
Program Memory Type	EPROM, UV
EEPROM Size	-
RAM Size	1.5K x 8
Voltage - Supply (Vcc/Vdd)	4.2V ~ 5.5V
Data Converters	A/D 5x10b
Oscillator Type	External
Operating Temperature	0°C ~ 70°C (TA)
Mounting Type	Through Hole
Package / Case	28-CDIP (0.300", 7.62mm) Window
Supplier Device Package	28-CerDip
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18c252-jw

4.0 MEMORY ORGANIZATION

There are two memory blocks in Enhanced MCU devices. These memory blocks are:

- Program Memory
- Data Memory

Program and data memory use separate buses so that concurrent access can occur.

4.1 Program Memory Organization

A 21-bit program counter is capable of addressing the 2-Mbyte program memory space. Accessing a location between the physically implemented memory and the 2-Mbyte address will cause a read of all '0's (a NOP instruction).

PIC18C252 and PIC18C452 have 32 Kbytes of EPROM, while PIC18C242 and PIC18C442 have 16 Kbytes of EPROM. This means that PIC18CX52 devices can store up to 16K of single word instructions, and PIC18CX42 devices can store up to 8K of single word instructions.

The RESET vector address is at 0000h and the interrupt vector addresses are at 0008h and 0018h.

Figure 4-1 shows the Program Memory Map for PIC18C242/442 devices and Figure 4-2 shows the Program Memory Map for PIC18C252/452 devices.

4.9 Data Memory Organization

The data memory is implemented as static RAM. Each register in the data memory has a 12-bit address, allowing up to 4096 bytes of data memory. Figure 4-6 and Figure 4-7 show the data memory organization for the PIC18CXX2 devices.

The data memory map is divided into as many as 16 banks that contain 256 bytes each. The lower 4 bits of the Bank Select Register (BSR<3:0>) select which bank will be accessed. The upper 4 bits for the BSR are not implemented.

The data memory contains Special Function Registers (SFR) and General Purpose Registers (GPR). The SFRs are used for control and status of the controller and peripheral functions, while GPRs are used for data storage and scratch pad operations in the user's application. The SFRs start at the last location of Bank 15 (0xFFF) and extend downwards. Any remaining space beyond the SFRs in the Bank may be implemented as GPRs. GPRs start at the first location of Bank 0 and grow upwards. Any read of an unimplemented location will read as '0's.

The entire data memory may be accessed directly, or indirectly. Direct addressing may require the use of the BSR register. Indirect addressing requires the use of a File Select Register (FSRn) and corresponding Indirect File Operand (INDFn). Each FSR holds a 12-bit address value that can be used to access any location in the Data Memory map without banking.

The instruction set and architecture allow operations across all banks. This may be accomplished by indirect addressing or by the use of the `MOVFF` instruction. The `MOVFF` instruction is a two-word/two-cycle instruction that moves a value from one register to another.

To ensure that commonly used registers (SFRs and select GPRs) can be accessed in a single cycle, regardless of the current BSR values, an Access Bank is implemented. A segment of Bank 0 and a segment of Bank 15 comprise the Access RAM. Section 4.10 provides a detailed description of the Access RAM.

4.9.1 GENERAL PURPOSE REGISTER FILE

The register file can be accessed either directly, or indirectly. Indirect addressing operates using the File Select Registers (FSRn) and corresponding Indirect File Operand (INDFn). The operation of indirect addressing is shown in Section 4.12.

Enhanced MCU devices may have banked memory in the GPR area. GPRs are not initialized by a Power-on Reset and are unchanged on all other RESETS.

Data RAM is available for use as GPR registers by all instructions. The top half of bank 15 (0xF80 to 0xFFF) contains SFRs. All other banks of data memory contain GPR registers, starting with bank 0.

4.9.2 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. A list of these registers is given in Table 4-1 and Table 4-2.

The SFRs can be classified into two sets; those associated with the "core" function and those related to the peripheral functions. Those registers related to the "core" are described in this section, while those related to the operation of the peripheral features are described in the section of that peripheral feature.

The SFRs are typically distributed among the peripherals whose functions they control.

The unused SFR locations will be unimplemented and read as '0's. See Table 4-1 for addresses for the SFRs.

TABLE 4-2: REGISTER FILE SUMMARY (CONTINUED)

File Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Details on page:
IPR2	—	—	—	—	BCLIP	LVDIP	TMR3IP	CCP2IP	---- 1111	73
PIR2	—	—	—	—	BCLIF	LVDIF	TMR3IF	CCP2IF	---- 0000	69
PIE2	—	—	—	—	BCLIE	LVDIE	TMR3IE	CCP2IE	---- 0000	71
IPR1	PSPIP	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	1111 1111	72
PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	68
PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	70
TRISE	IBF	OBF	IBOV	PSPMODE	—	Data Direction bits for PORTE			0000 -111	88
TRISD	Data Direction Control Register for PORTD								1111 1111	85
TRISC	Data Direction Control Register for PORTC								1111 1111	83
TRISB	Data Direction Control Register for PORTB								1111 1111	80
TRISA	—	TRISA6(1)	Data Direction Control Register for PORTA						-111 1111	77
LATE	—	—	—	—	—	Read PORTE Data Latch, Write PORTE Data Latch			---- -xxx	87
LATD	Read PORTD Data Latch, Write PORTD Data Latch								xxxx xxxx	85
LATC	Read PORTC Data Latch, Write PORTC Data Latch								xxxx xxxx	83
LATB	Read PORTB Data Latch, Write PORTB Data Latch								xxxx xxxx	80
LATA	—	LATA6(1)	Read PORTA Data Latch, Write PORTA Data Latch(1)						-xxx xxxx	77
PORTE	Read PORTE pins, Write PORTE Data Latch								---- -000	87
PORTD	Read PORTD pins, Write PORTD Data Latch								xxxx xxxx	85
PORTC	Read PORTC pins, Write PORTC Data Latch								xxxx xxxx	83
PORTB	Read PORTB pins, Write PORTB Data Latch								xxxx xxxx	80
PORTA	—	RA6(1)	Read PORTA pins, Write PORTA Data Latch(1)						-x0x 0000	77

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

Note 1: RA6 and associated bits are configured as port pins in RCIO and ECIO oscillator mode only, and read '0' in all other oscillator modes.

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

7.6 INT0 Interrupt

External interrupts on the RB0/INT0, RB1/INT1 and RB2/INT2 pins are edge triggered: either rising, if the corresponding INTEDGx bit is set in the INTCON2 register, or falling, if the INTEDGx bit is clear. When a valid edge appears on the RBx/INTx pin, the corresponding flag bit INTxF is set. This interrupt can be disabled by clearing the corresponding enable bit INTxE. Flag bit INTxF must be cleared in software in the Interrupt Service Routine before re-enabling the interrupt. All external interrupts (INT0, INT1 and INT2) can wake-up the processor from SLEEP, if bit INTxE was set prior to going into SLEEP. If the global interrupt enable bit GIE set, the processor will branch to the interrupt vector following wake-up.

Interrupt priority for INT1 and INT2 is determined by the value contained in the interrupt priority bits, INT1IP (INTCON3<6>) and INT2IP (INTCON3<7>). There is no priority bit associated with INT0. It is always a high priority interrupt source.

7.7 TMR0 Interrupt

In 8-bit mode (which is the default), an overflow (FFh → 00h) in the TMR0 register will set flag bit TMR0IF. In 16-bit mode, an overflow (FFFFh → 0000h) in the TMR0H:TMR0L registers will set flag bit TMR0IF. The interrupt can be enabled/disabled by setting/clearing enable bit T0IE (INTCON<5>). Interrupt priority for Timer0 is determined by the value contained in the interrupt priority bit TMR0IP (INTCON2<2>). See Section 8.0 for further details on the Timer0 module.

7.8 PORTB Interrupt-on-Change

An input change on PORTB<7:4> sets flag bit RBIF (INTCON<0>). The interrupt can be enabled/disabled by setting/clearing enable bit, RBIE (INTCON<3>). Interrupt priority for PORTB Interrupt-on-change is determined by the value contained in the interrupt priority bit, RBIP (INTCON2<0>).

7.9 Context Saving During Interrupts

During an interrupt, the return PC value is saved on the stack. Additionally, the WREG, STATUS and BSR registers are saved on the fast return stack. If a fast return from interrupt is not used (see Section 4.3), the user may need to save the WREG, STATUS and BSR registers in software. Depending on the user's application, other registers may also need to be saved. Example 7-1 saves and restores the WREG, STATUS and BSR registers during an Interrupt Service Routine.

EXAMPLE 7-1: SAVING STATUS, WREG AND BSR REGISTERS IN RAM

```
MOVWF   W_TEMP                  ; W_TEMP is in virtual bank
MOVFF   STATUS, STATUS_TEMP     ; STATUS_TEMP located anywhere
MOVFF   BSR, BSR_TEMP           ; BSR located anywhere
;
; USER ISR CODE
;
MOVFF   BSR_TEMP, BSR           ; Restore BSR
MOVF    W_TEMP, W               ; Restore WREG
MOVFF   STATUS_TEMP, STATUS     ; Restore STATUS
```

TABLE 8-1: PORTA FUNCTIONS

Name	Bit#	Buffer	Function
RA0/AN0	bit0	TTL	Input/output or analog input.
RA1/AN1	bit1	TTL	Input/output or analog input.
RA2/AN2/VREF-	bit2	TTL	Input/output or analog input or VREF-.
RA3/AN3/VREF+	bit3	TTL	Input/output or analog input or VREF+.
RA4/T0CKI	bit4	ST	Input/output or external clock input for Timer0. Output is open drain type.
RA5/$\overline{SS}$/AN4/LVDIN	bit5	TTL	Input/output or slave select input for synchronous serial port or analog input, or low voltage detect input.
OSC2/CLKO/RA6	bit6	TTL	OSC2 or clock output or I/O pin.

Legend: TTL = TTL input, ST = Schmitt Trigger input

TABLE 8-2: SUMMARY OF REGISTERS ASSOCIATED WITH PORTA

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
PORTA	—	RA6	RA5	RA4	RA3	RA2	RA1	RA0	--0x 0000	--0u 0000
LATA	—	Latch A Data Output Register							--xx xxxx	--uu uuuu
TRISA	—	PORTA Data Direction Register							--11 1111	--11 1111
ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	--0- 0000	--0- 0000

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'.
Shaded cells are not used by PORTA.

NOTES:

14.3.5 SLAVE MODE

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

While in Slave mode, the external clock is supplied by the external clock source on the SCK pin. This external clock must meet the minimum high and low times as specified in the electrical specifications.

While in SLEEP mode, the slave can transmit/receive data. When a byte is received, the device will wake-up from SLEEP.

14.3.6 SLAVE SELECT SYNCHRONIZATION

The $\overline{SS}$ pin allows a Synchronous Slave mode. The SPI must be in Slave mode with $\overline{SS}$ pin control enabled (SSPCON1<3:0> = 04h). The pin must not be driven low for the $\overline{SS}$ pin to function as an input. The Data Latch must be high. 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 (SSPCON<3:0> = 0100), the SPI module will reset if the $\overline{SS}$ pin is set to VDD.
2:	If the SPI is used in Slave mode with CKE set, then the $\overline{SS}$ pin control must be enabled.

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.

To emulate two-wire communication, the SDO pin can be connected to the SDI pin. When the SPI needs to operate as a receiver, the SDO pin can be configured as an input. This disables transmissions from the SDO. The SDI can always be left as an input (SDI function), since it cannot create a bus conflict.

FIGURE 14-4: SLAVE SYNCHRONIZATION WAVEFORM

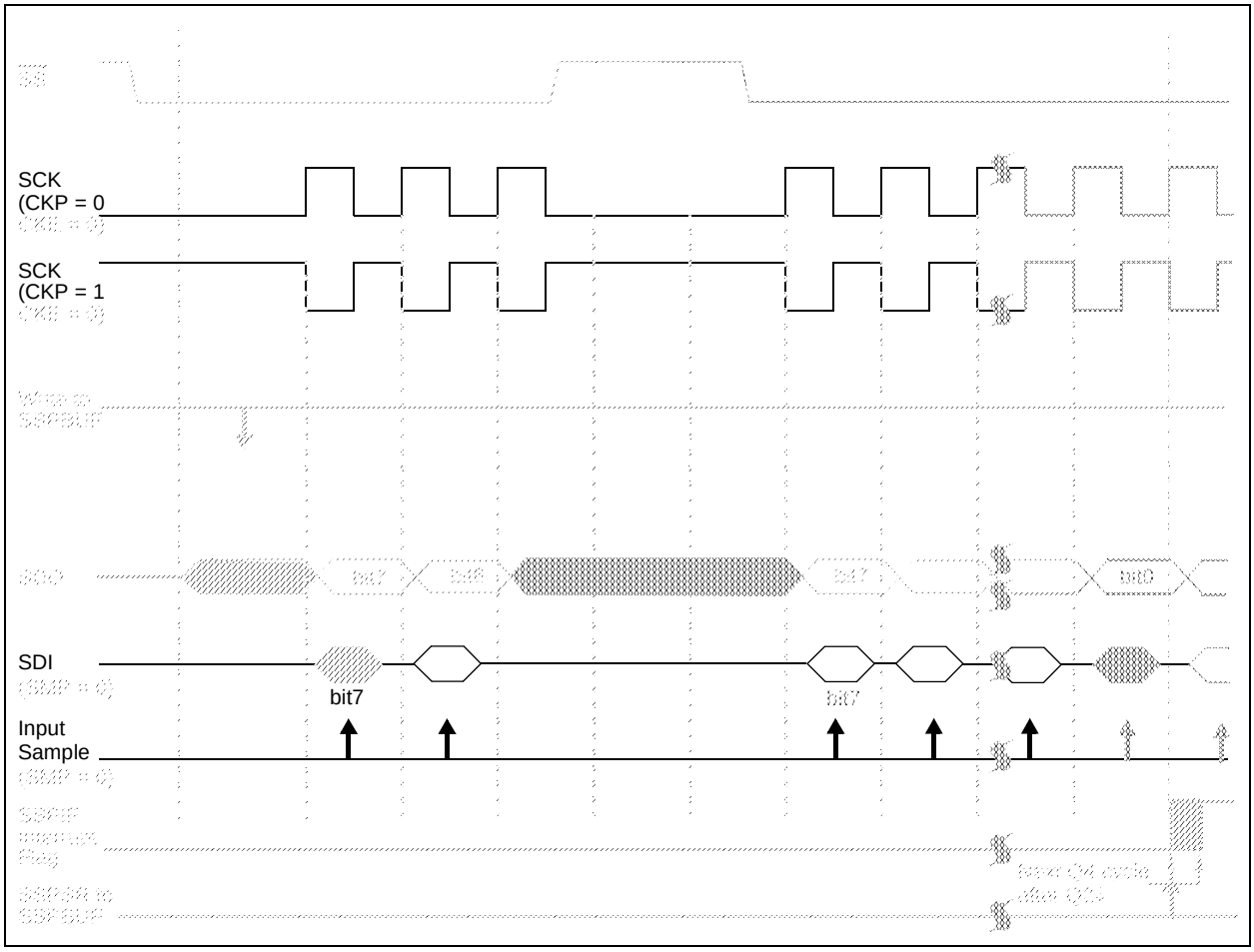
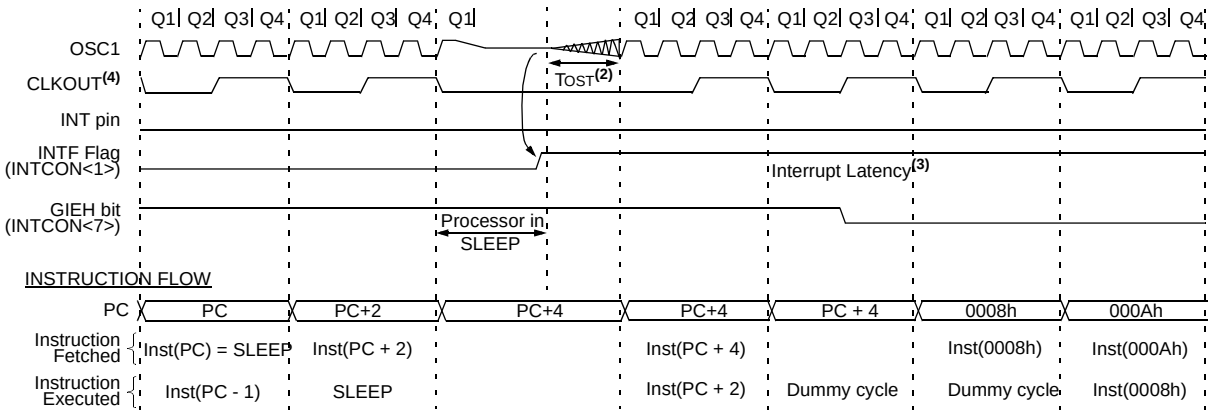


FIGURE 18-2: WAKE-UP FROM SLEEP THROUGH INTERRUPT(1,2)



Note 1: XT, HS or LP oscillator mode assumed.
2: GIE = '1' assumed. In this case, after wake- up, the processor jumps to the interrupt routine. If GIE = '0', 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 osc modes, but shown here for timing reference.

18.4 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.

18.5 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.

18.6 In-Circuit Serial Programming

PIC18CXXX 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.

19.0 INSTRUCTION SET SUMMARY

The PIC18CXXX instruction set adds many enhancements to the previous PIC 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 three 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 PIC18CXXX instruction set summary in Table 19-2 lists **byte-oriented**, **bit-oriented**, **literal** and **control** operations. Table 19-1 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 three double word instructions. These three instructions were made double word instructions so that all the required information is available in these 32-bits. In the second word, the 4 MSb’s 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 19-1 shows the general formats that the instructions can have.

All examples use the format `‘nnh’` to represent a hexadecimal number, where `‘h’` signifies a hexadecimal digit.

The Instruction Set Summary, shown in Table 19-2, lists the instructions recognized by the Microchip assembler (MPASM™).

Section 19.1 provides a description of each instruction.

TABLE 19-2: PIC18CXXX INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	16-bit Instruction Word				Status Affected	Notes
				MSb			LSb		
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d, a	Add WREG and f	1	0010	01da	ffff	ffff	C, DC, Z, OV, N	1, 2
ADDWFC	f, d, a	Add WREG and Carry bit to f	1	0010	00da	ffff	ffff	C, DC, Z, OV, N	1, 2
ANDWF	f, d, a	AND WREG with f	1	0001	01da	ffff	ffff	Z, N	1,2
CLRF	f, a	Clear f	1	0110	101a	ffff	ffff	Z	2
COMF	f, d, a	Complement f	1	0001	11da	ffff	ffff	Z, N	1, 2
CPFSEQ	f, a	Compare f with WREG, skip =	1 (2 or 3)	0110	001a	ffff	ffff	None	4
CPFSGT	f, a	Compare f with WREG, skip >	1 (2 or 3)	0110	010a	ffff	ffff	None	4
CPFSLT	f, a	Compare f with WREG, skip <	1 (2 or 3)	0110	000a	ffff	ffff	None	1, 2
DECF	f, d, a	Decrement f	1	0000	01da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
DECFSZ	f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010	11da	ffff	ffff	None	1, 2, 3, 4
DCFSNZ	f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100	11da	ffff	ffff	None	1, 2
INCF	f, d, a	Increment f	1	0010	10da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
INCFSZ	f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011	11da	ffff	ffff	None	4
INFSNZ	f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100	10da	ffff	ffff	None	1, 2
IORWF	f, d, a	Inclusive OR WREG with f	1	0001	00da	ffff	ffff	Z, N	1, 2
MOVF	f, d, a	Move f	1	0101	00da	ffff	ffff	Z, N	1
MOVFF	f_s, f_d	Move f_s (source) to 1st word f_d (destination)2nd word	2	1100 1111	ffff ffff	ffff ffff	ffff ffff	None	
MOVWF	f, a	Move WREG to f	1	0110	111a	ffff	ffff	None	
MULWF	f, a	Multiply WREG with f	1	0000	001a	ffff	ffff	None	
NEGF	f, a	Negate f	1	0110	110a	ffff	ffff	C, DC, Z, OV, N	1, 2
RLCF	f, d, a	Rotate Left f through Carry	1	0011	01da	ffff	ffff	C, Z, N	
RLNCF	f, d, a	Rotate Left f (No Carry)	1	0100	01da	ffff	ffff	Z, N	1, 2
RRCF	f, d, a	Rotate Right f through Carry	1	0011	00da	ffff	ffff	C, Z, N	
RRNCF	f, d, a	Rotate Right f (No Carry)	1	0100	00da	ffff	ffff	Z, N	
SETF	f, a	Set f	1	0110	100a	ffff	ffff	None	
SUBFWB	f, d, a	Subtract f from WREG with borrow	1	0101	01da	ffff	ffff	C, DC, Z, OV, N	1, 2
SUBWF	f, d, a	Subtract WREG from f	1	0101	11da	ffff	ffff	C, DC, Z, OV, N	
SUBWFB	f, d, a	Subtract WREG from f with borrow	1	0101	10da	ffff	ffff	C, DC, Z, OV, N	1, 2
SWAPF	f, d, a	Swap nibbles in f	1	0011	10da	ffff	ffff	None	4
TSTFSZ	f, a	Test f, skip if 0	1 (2 or 3)	0110	011a	ffff	ffff	None	1, 2
XORWF	f, d, a	Exclusive OR WREG with f	1	0001	10da	ffff	ffff	Z, N	
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b, a	Bit Clear f	1	1001	bbba	ffff	ffff	None	1, 2
BSF	f, b, a	Bit Set f	1	1000	bbba	ffff	ffff	None	1, 2
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011	bbba	ffff	ffff	None	3, 4
BTFSS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010	bbba	ffff	ffff	None	3, 4
BTG	f, d, a	Bit Toggle f	1	0111	bbba	ffff	ffff	None	1, 2

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.

19.1 Instruction Set

ADDLW **ADD literal to WREG**

Syntax: [*label*] ADDLW k

Operands: $0 \le k \le 255$

Operation: (WREG) + k → WREG

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

Encoding:

0000	1111	kkkk	kkkk

Description: The contents of WREG are added to the 8-bit literal 'k' and 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 WREG

Example: `ADDLW 0x15`

Before Instruction

`WREG = 0x10`

After Instruction

`WREG = 0x25`

ADDWF **ADD WREG to f**

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

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

Operation: (WREG) + (f) → dest

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

Encoding:

0010	01da	ffff	ffff

Description: Add WREG to register 'f'. If 'd' is 0, the result is stored 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. If ‘a’ is 1, the BSR is used.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `ADDWF REG, 0, 0`

Before Instruction

```
WREG = 0x17
REG  = 0xC2
```

After Instruction

```
WREG = 0xD9
REG  = 0xC2
```

IORLW **Inclusive OR literal with WREG**

Syntax: [*label*] IORLW k

Operands: $0 \le k \le 255$

Operation: (WREG) .OR. k → WREG

Status Affected: N,Z

Encoding:

0000	1001	kkkk	kkkk

Description: The contents of WREG are OR'ed with 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 WREG

Example: `IORLW 0x35`

Before Instruction

`WREG = 0x9A`

After Instruction

`WREG = 0xBF`

IORWF **Inclusive OR WREG with f**

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

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

Operation: (WREG) .OR. (f) → dest

Status Affected: N,Z

Encoding:

0001	00da	ffff	ffff

Description: Inclusive OR WREG with register 'f'. 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' = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `IORWF RESULT, 0, 1`

Before Instruction

```
RESULT = 0x13
WREG   = 0x91
```

After Instruction

```
RESULT = 0x13
WREG   = 0x93
```

RCALL **Relative Call**

Syntax: [*label*] RCALL n

Operands: $-1024 \le n \le 1023$

Operation: $(PC) + 2 \rightarrow TOS$,
$(PC) + 2 + 2n \rightarrow PC$

Status Affected: None

Encoding:

1101	1nnn	nnnn	nnnn

Description: Subroutine call with a jump up to 1K from the current location. First, return address (PC+2) is pushed onto the stack. Then, add the 2's complement number '2n' 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 a two-cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

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

Example: HERE RCALL Jump

Before Instruction
```
PC  =  Address(HERE)
```
After Instruction
```
PC  =  Address(Jump)
TOS =  Address (HERE+2)
```

RESET **Reset**

Syntax: [*label*] RESET

Operands: None

Operation: Reset all registers and flags that are affected by a MCLR reset.

Status Affected: All

Encoding:

0000	0000	1111	1111

Description: This instruction provides a way to execute a MCLR Reset in software.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Start reset	No operation	No operation

Example: RESET

After Instruction
```
Registers  =  Reset Value
Flags*     =  Reset Value
```

SLEEP **Enter SLEEP mode**

Syntax: [*label*] SLEEP

Operands: None

Operation: 00h → WDT,
0 → WDT postscaler,
1 → $\overline{TO}$,
0 → $\overline{PD}$

Status Affected: $\overline{TO}$, $\overline{PD}$

Encoding:

0000	0000	0000	0011

Description: The power-down status bit ($\overline{PD}$) is cleared. The time-out status bit ($\overline{TO}$) is set. Watchdog Timer and its postscaler are cleared.
The processor is put into SLEEP mode with the oscillator stopped.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	Process Data	Go to sleep

Example: SLEEP

Before Instruction
$\overline{TO}$ = ?
$\overline{PD}$ = ?

After Instruction
$\overline{TO}$ = 1 †
$\overline{PD}$ = 0

† If WDT causes wake-up, this bit is cleared.

SUBFWB **Subtract f from WREG with borrow**

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

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

Operation: (WREG) – (f) – ($\overline{C}$) → dest

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

Encoding:

0101	01da	ffff	ffff

Description: Subtract register 'f' and carry flag (borrow) from WREG (2's complement method). If 'd' is 0, the result is stored in WREG. If 'd' is 1, the result is stored in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1:
```
SUBFWB   REG, 1, 0
```
Before Instruction
```
REG  =  3
WREG =  2
C    =  1
```
After Instruction
```
REG  =  FF
WREG =  2
C    =  0
Z    =  0
N    =  1     ; result is negative
```

Example 2:
```
SUBFWB   REG, 0, 0
```
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, 1, 0
```
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 WREG

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    =   FF      ; (2's complement)
C       =   0       ; result is negative
Z       =   0
N       =   1
```

SUBWF **Subtract WREG from f**

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

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

Operation: (f) – (WREG) → dest

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

Encoding:

0101	11da	ffff	ffff

Description: Subtract WREG from register 'f' (2's complement method). If 'd' is 0, the result is stored 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, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1:
```
SUBWF   REG, 1, 0
```
Before Instruction
```
REG     =   3
WREG    =   2
C       =   ?
```
After Instruction
```
REG     =   1
WREG    =   2
C       =   1       ; result is positive
Z       =   0
N       =   0
```

Example 2:
```
SUBWF   REG, 0, 0
```
Before Instruction
```
REG     =   2
WREG    =   2
C       =   ?
```
After Instruction
```
REG     =   2
WREG    =   0
C       =   1       ; result is zero
Z       =   1
N       =   0
```

Example 3:
```
SUBWF   REG, 1, 0
```
Before Instruction
```
REG     =   1
WREG    =   2
C       =   ?
```
After Instruction
```
REG     =   FFh     ;(2's complement)
WREG    =   2
C       =   0       ; result is negative
Z       =   0
N       =   1
```

SUBWFB **Subtract WREG from f with Borrow**

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

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

Operation: (f) – (WREG) – ($\overline{C}$) → dest

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

Encoding:

0101	10da	ffff	ffff

Description: Subtract WREG and the carry flag (borrow) from register 'f' (2's complement method). If 'd' is 0, the result is stored 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, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1:
```
SUBWFB  REG, 1, 0
```
Before Instruction
```
REG     =   0x19   (0001 1001)
WREG    =   0x0D   (0000 1101)
C       =   1
```
After Instruction
```
REG     =   0x0C   (0000 1011)
WREG    =   0x0D   (0000 1101)
C       =   1
Z       =   0
N       =   0     ; result is positive
```

Example 2:
```
SUBWFB  REG, 0, 0
```
Before Instruction
```
REG     =   0x1B    (0001 1011)
WREG    =   0x1A    (0001 1010)
C       =   0
```
After Instruction
```
REG     =   0x1B    (0001 1011)
WREG    =   0x00
C       =   1
Z       =   1     ; result is zero
N       =   0
```

Example 3:
```
SUBWFB  REG, 1, 0
```
Before Instruction
```
REG     =   0x03    (0000 0011)
WREG    =   0x0E    (0000 1101)
C       =   1
```
After Instruction
```
REG     =   0xF5    (1111 0100)
                  ; [2's comp]
WREG    =   0x0E    (0000 1101)
C       =   0
Z       =   0
N       =   1     ; result is negative
```

SWAPF **Swap f**

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

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

Operation: (f<3:0>) → dest<7:4>,
(f<7:4>) → dest<3:0>

Status Affected: None

Encoding:

0011	10da	ffff	ffff

Description: The upper and lower nibbles of register 'f' are exchanged. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example:
```
SWAPF   REG, 1, 0
```
Before Instruction
```
REG     =   0x53
```
After Instruction
```
REG     =   0x35
```

TBLRD **Table Read**

Syntax: [*label*] TBLRD (*; *+; *-; +*)

Operands: None

Operation:
if TBLRD *,
(Prog Mem (TBLPTR)) → TABLAT;
TBLPTR - No Change;
if TBLRD *+,
(Prog Mem (TBLPTR)) → TABLAT;
(TBLPTR) +1 → TBLPTR;
if TBLRD *-,
(Prog Mem (TBLPTR)) → TABLAT;
(TBLPTR) -1 → TBLPTR;
if TBLRD +*,
(TBLPTR) +1 → TBLPTR;
(Prog Mem (TBLPTR)) → TABLAT;

Status Affected: None

Encoding:

0000	0000	0000	10nn nn=0 * =1 *+ =2 *- =3 +*

Description: This instruction is used to read the contents of Program Memory (P.M.). To address the program memory, a pointer called Table Pointer (TBLPTR) is used.
The TBLPTR (a 21-bit pointer) points to each byte in the program memory. TBLPTR has a 2 Mbyte address range.

TBLPTR[0] = 0: Least Significant Byte of Program Memory Word

TBLPTR[0] = 1: Most Significant Byte of Program Memory Word

The TBLRD instruction can modify the value of TBLPTR as follows:

- no change
- post-increment
- post-decrement
- pre-increment

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation
No operation	No operation (Read Program Memory)	No operation	No operation (Write TABLAT)

TBLRD **Table Read (cont'd)**

Example 1:
```
TBLRD  *+ ;
```
Before Instruction
```
TABLAT              =   0x55
TBLPTR              =   0x00A356
MEMORY(0x00A356)    =   0x34
```
After Instruction
```
TABLAT              =   0x34
TBLPTR              =   0x00A357
```

Example 2:
```
TBLRD  +* ;
```
Before Instruction
```
TABLAT              =   0xAA
TBLPTR              =   0x01A357
MEMORY(0x01A357)    =   0x12
MEMORY(0x01A358)    =   0x34
```
After Instruction
```
TABLAT              =   0x34
TBLPTR              =   0x01A358
```

21.2 DC Characteristics: PIC18CXX2 (Industrial, Extended) PIC18LCXX2 (Industrial)

DC CHARACTERISTICS			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial -40°C ≤ TA ≤ +125°C for extended			
Param No.	**Symbol**	**Characteristic**	**Min**	**Max**	**Units**	**Conditions**
	VIL	**Input Low Voltage**				
		I/O ports:				
D030		with TTL buffer	Vss	0.15VDD	V	VDD < 4.5V
D030A			—	0.8	V	4.5V ≤ VDD ≤ 5.5V
D031		with Schmitt Trigger buffer	Vss	0.2VDD	V	
		RC3 and RC4	Vss	0.3VDD	V	
D032		$\overline{MCLR}$	Vss	0.2VDD	V	
D032A		OSC1 (in XT, HS and LP modes) and T1OSI	Vss	0.3VDD	V	
D033		OSC1 (in RC and EC mode)(1)	Vss	0.2VDD	V	
	VIH	**Input High Voltage**				
		I/O ports:				
D040		with TTL buffer	0.25VDD + 0.8V	VDD	V	VDD < 4.5V
D040A			2.0	VDD	V	4.5V ≤ VDD ≤ 5.5V
D041		with Schmitt Trigger buffer	0.8VDD	VDD	V	
		RC3 and RC4	0.7VDD	VDD	V	
D042		$\overline{MCLR}$, OSC1 (EC mode)	0.8VDD	VDD	V	
D042A		OSC1 (in XT, HS and LP modes) and T1OSI	0.7VDD	VDD	V	
D043		OSC1 (RC mode)(1)	0.9VDD	VDD	V	
	IIL	**Input Leakage Current(2,3)**				
D060		I/O ports	—	±1	µA	VSS ≤ VPIN ≤ VDD, Pin at hi-impedance
D061		$\overline{MCLR}$	—	±5	µA	Vss ≤ VPIN ≤ VDD
D063		OSC1	—	±5	µA	Vss ≤ VPIN ≤ VDD
	IPU	**Weak Pull-up Current**				
D070	IPURB	PORTB weak pull-up current	50	400	µA	VDD = 5V, VPIN = VSS

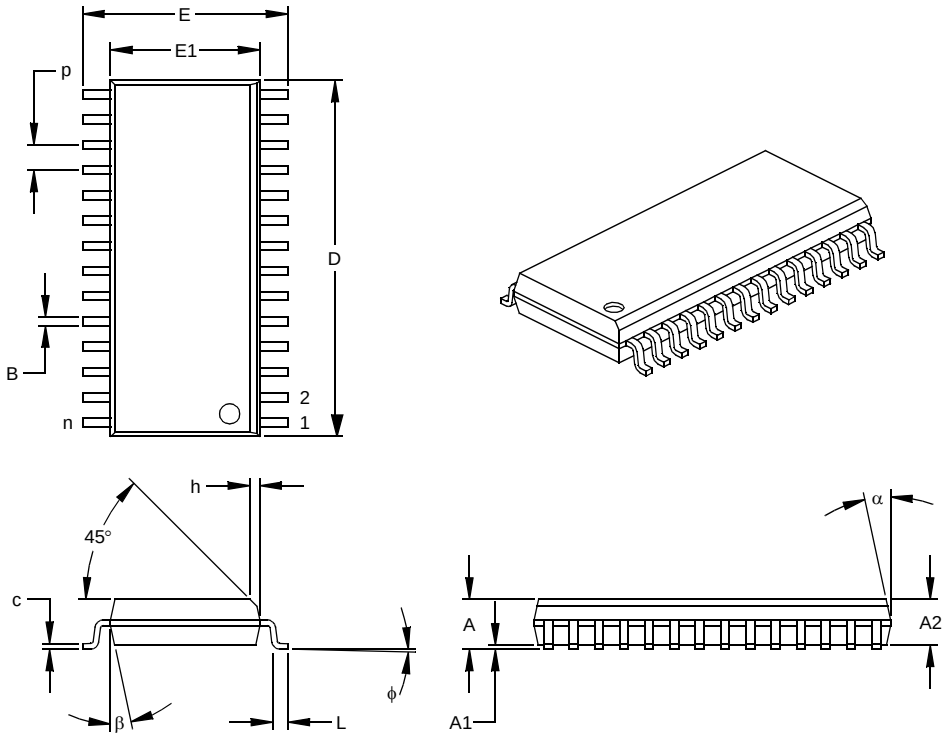
Note 1: In RC oscillator configuration, the OSC1/CLKIN pin is a Schmitt Trigger input. It is not recommended that the PIC MCU be driven with an external clock while in RC mode.

2: The leakage current on the $\overline{MCLR}$ pin is strongly dependent on the applied voltage level. The specified levels represent normal operating conditions. Higher leakage current may be measured at different input voltages.

3: Negative current is defined as current sourced by the pin.

28-Lead Plastic Small Outline (SO) – Wide, 300 mil (SOIC)

Note: For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		28			28	
Pitch	p		.050			1.27	
Overall Height	A	.093	.099	.104	2.36	2.50	2.64
Molded Package Thickness	A2	.088	.091	.094	2.24	2.31	2.39
Standoff §	A1	.004	.008	.012	0.10	0.20	0.30
Overall Width	E	.394	.407	.420	10.01	10.34	10.67
Molded Package Width	E1	.288	.295	.299	7.32	7.49	7.59
Overall Length	D	.695	.704	.712	17.65	17.87	18.08
Chamfer Distance	h	.010	.020	.029	0.25	0.50	0.74
Foot Length	L	.016	.033	.050	0.41	0.84	1.27
Foot Angle Top	ϕ	0	4	8	0	4	8
Lead Thickness	c	.009	.011	.013	0.23	0.28	0.33
Lead Width	B	.014	.017	.020	0.36	0.42	0.51
Mold Draft Angle Top	α	0	12	15	0	12	15
Mold Draft Angle Bottom	β	0	12	15	0	12	15

* Controlling Parameter
§ Significant Characteristic

Notes:
Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010” (0.254mm) per side.
JEDEC Equivalent: MS-013
Drawing No. C04-052

APPENDIX C: CONVERSION CONSIDERATIONS

This appendix discusses the considerations for converting from previous versions of a device to the ones listed in this data sheet. Typically, these changes are due to the differences in the process technology used. An example of this type of conversion is from a PIC16C74A to a PIC16C74B.

Not Applicable

APPENDIX D: MIGRATION FROM BASELINE TO ENHANCED DEVICES

This section discusses how to migrate from a Baseline device (i.e., PIC16C5X) to an Enhanced MCU device (i.e., PIC18CXXX).

The following are the list of modifications over the PIC16C5X microcontroller family:

Not Currently Available