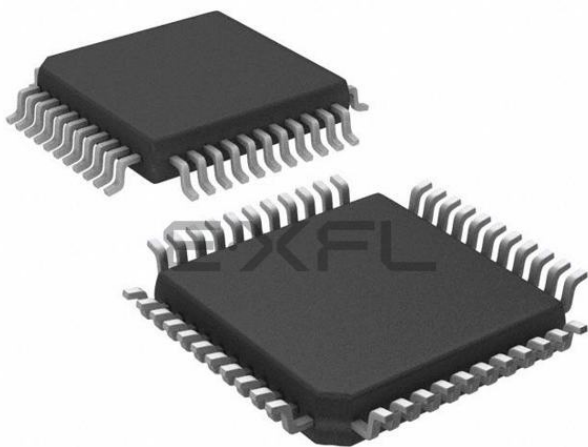




Welcome to [E-XFL.COM](#)

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	16MHz
Connectivity	UART/USART
Peripherals	POR, PWM, WDT
Number of I/O	33
Program Memory Size	8KB (4K x 16)
Program Memory Type	OTP
EEPROM Size	-
RAM Size	454 x 8
Voltage - Supply (Vcc/Vdd)	4.5V ~ 6V
Data Converters	-
Oscillator Type	External
Operating Temperature	0°C ~ 70°C (TA)
Mounting Type	Surface Mount
Package / Case	44-QFP
Supplier Device Package	44-MQFP (10x10)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic17c43t-16-pq

3.1 Clocking Scheme/Instruction Cycle

The clock input (from OSC1) is internally divided by four to generate four non-overlapping quadrature clocks, namely Q1, Q2, Q3, and Q4. Internally, the program counter (PC) is incremented every Q1, and the instruction is fetched from the program memory and latched into the instruction register in Q4. The instruction is decoded and executed during the following Q1 through Q4. The clocks and instruction execution flow are shown in Figure 3-3.

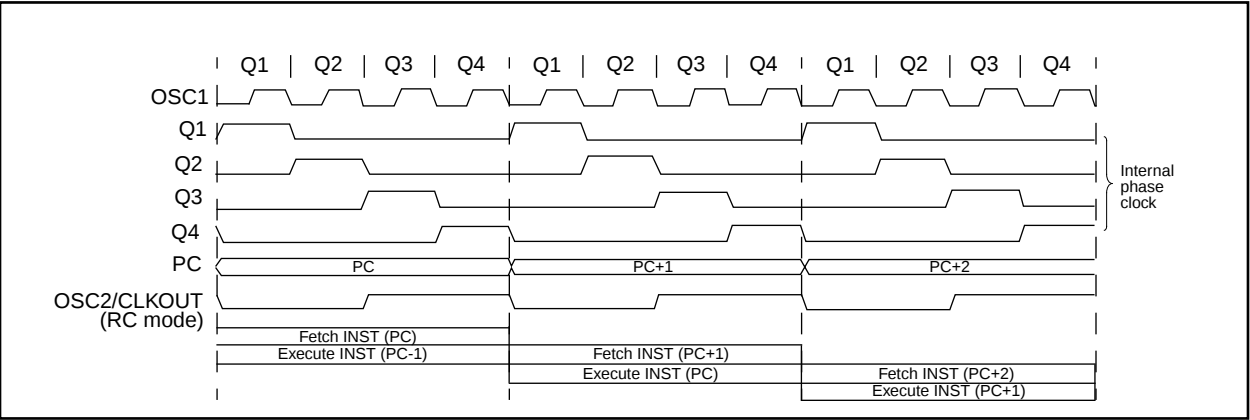
3.2 Instruction Flow/Pipelining

An "Instruction Cycle" consists of four Q cycles (Q1, Q2, Q3, and Q4). The instruction fetch and execute are pipelined such that fetch takes one instruction cycle while decode and execute takes another instruction cycle. However, due to the pipelining, each instruction effectively executes in one cycle. If an instruction causes the program counter to change (e.g. GOTO) then two cycles are required to complete the instruction (Example 3-2).

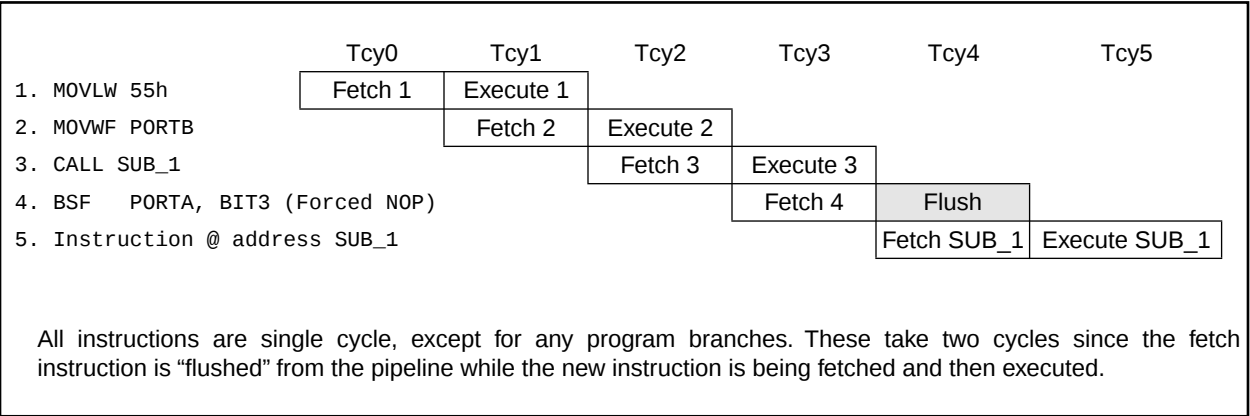
A fetch cycle begins with the program counter incrementing in Q1.

In the execution cycle, the fetched instruction is latched into the "Instruction Register (IR)" in cycle Q1. This instruction is then decoded and executed during the Q2, Q3, and Q4 cycles. Data memory is read during Q2 (operand read) and written during Q4 (destination write).

FIGURE 3-3: CLOCK/INSTRUCTION CYCLE



EXAMPLE 3-2: INSTRUCTION PIPELINE FLOW



6.4.1 INDIRECT ADDRESSING REGISTERS

The PIC17C4X has four registers for indirect addressing. These registers are:

- INDF0 and FSR0
- INDF1 and FSR1

Registers INDF0 and INDF1 are not physically implemented. Reading or writing to these registers activates indirect addressing, with the value in the corresponding FSR register being the address of the data. The FSR is an 8-bit register and allows addressing anywhere in the 256-byte data memory address range. For banked memory, the bank of memory accessed is specified by the value in the BSR.

If file INDF0 (or INDF1) itself is read indirectly via an FSR, all '0's are read (Zero bit is set). Similarly, if INDF0 (or INDF1) is written to indirectly, the operation will be equivalent to a NOP, and the status bits are not affected.

6.4.2 INDIRECT ADDRESSING OPERATION

The indirect addressing capability has been enhanced over that of the PIC16CXX family. There are two control bits associated with each FSR register. These two bits configure the FSR register to:

- Auto-decrement the value (address) in the FSR after an indirect access
- Auto-increment the value (address) in the FSR after an indirect access
- No change to the value (address) in the FSR after an indirect access

These control bits are located in the ALUSTA register. The FSR1 register is controlled by the FS3:FS2 bits and FSR0 is controlled by the FS1:FS0 bits.

When using the auto-increment or auto-decrement features, the effect on the FSR is not reflected in the ALUSTA register. For example, if the indirect address causes the FSR to equal '0', the Z bit will not be set.

If the FSR register contains a value of 0h, an indirect read will read 0h (Zero bit is set) while an indirect write will be equivalent to a NOP (status bits are not affected).

Indirect addressing allows single cycle data transfers within the entire data space. This is possible with the use of the MOVPF and MOVFP instructions, where either 'p' or 'f' is specified as INDF0 (or INDF1).

If the source or destination of the indirect address is in banked memory, the location accessed will be determined by the value in the BSR.

A simple program to clear RAM from 20h - FFh is shown in Example 6-1.

EXAMPLE 6-1: INDIRECT ADDRESSING

```
MOVLW    0x20      ;  
MOVWF    FSR0      ; FSR0 = 20h  
BCF      ALUSTA, FS1 ; Increment FSR  
BSF      ALUSTA, FS0 ; after access  
BCF      ALUSTA, C   ; C = 0  
MOVLW    END_RAM + 1 ;  
LP  CLRF    INDF0    ; Addr(FSR) = 0  
    CPFSEQ  FSR0      ; FSR0 = END_RAM+1?  
    GOTO    LP        ; NO, clear next  
    :                ; YES, All RAM is  
    :                ; cleared
```

6.5 Table Pointer (TBLPTRL and TBLPTRH)

File registers TBLPTRL and TBLPTRH form a 16-bit pointer to address the 64K program memory space. The table pointer is used by instructions TABLWT and TABLRD.

The TABLRD and the TABLWT instructions allow transfer of data between program and data space. The table pointer serves as the 16-bit address of the data word within the program memory. For a more complete description of these registers and the operation of Table Reads and Table Writes, see Section 7.0.

6.6 Table Latch (TBLATH, TBLATL)

The table latch (TBLAT) is a 16-bit register, with TBLATH and TBLATL referring to the high and low bytes of the register. It is not mapped into data or program memory. The table latch is used as a temporary holding latch during data transfer between program and data memory (see descriptions of instructions TABLRD, TABLWT, TLRD and TLWT). For a more complete description of these registers and the operation of Table Reads and Table Writes, see Section 7.0.

6.7 Program Counter Module

The Program Counter (PC) is a 16-bit register. PCL, the low byte of the PC, is mapped in the data memory. PCL is readable and writable just as is any other register. PCH is the high byte of the PC and is not directly addressable. Since PCH is not mapped in data or program memory, an 8-bit register PCLATH (PC high latch) is used as a holding latch for the high byte of the PC. PCLATH is mapped into data memory. The user can read or write PCH through PCLATH.

The 16-bit wide PC is incremented after each instruction fetch during Q1 unless:

- Modified by GOTO, CALL, LCALL, RETURN, RETLW, or RETFIE instruction
- Modified by an interrupt response
- Due to destination write to PCL by an instruction

“Skips” are equivalent to a forced NOP cycle at the skipped address.

Figure 6-11 and Figure 6-12 show the operation of the program counter for various situations.

FIGURE 6-11: PROGRAM COUNTER OPERATION

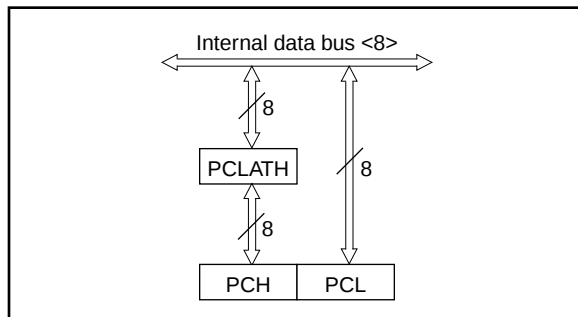
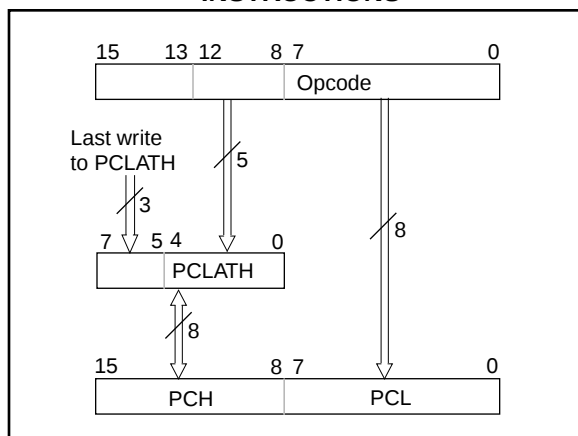


FIGURE 6-12: PROGRAM COUNTER USING THE CALL AND GOTO INSTRUCTIONS



Using Figure 6-11, the operations of the PC and PCLATH for different instructions are as follows:

- LCALL instructions:**
An 8-bit destination address is provided in the instruction (opcode). PCLATH is unchanged.
PCLATH → PCH
Opcode<7:0> → PCL
- Read instructions on PCL:**
Any instruction that reads PCL.
PCL → data bus → ALU or destination
PCH → PCLATH
- Write instructions on PCL:**
Any instruction that writes to PCL.
8-bit data → data bus → PCL
PCLATH → PCH
- Read-Modify-Write instructions on PCL:**
Any instruction that does a read-write-modify operation on PCL, such as ADDWF PCL.
Read: PCL → data bus → ALU
Write: 8-bit result → data bus → PCL
PCLATH → PCH
- RETURN instruction:**
PCH → PCLATH
Stack<MRU> → PC<15:0>

Using Figure 6-12, the operation of the PC and PCLATH for GOTO and CALL instructions is as follows:

CALL, GOTO instructions:

A 13-bit destination address is provided in the instruction (opcode).

Opcode<12:0> → PC <12:0>

PC<15:13> → PCLATH<7:5>

Opcode<12:8> → PCLATH <4:0>

The read-modify-write only affects the PCL with the result. PCH is loaded with the value in the PCLATH. For example, ADDWF PCL will result in a jump within the current page. If PC = 03F0h, WREG = 30h and PCLATH = 03h before instruction, PC = 0320h after the instruction. To accomplish a true 16-bit computed jump, the user needs to compute the 16-bit destination address, write the high byte to PCLATH and then write the low value to PCL.

The following PC related operations do not change PCLATH:

- LCALL, RETLW, and RETFIE instructions.
- Interrupt vector is forced onto the PC.
- Read-modify-write instructions on PCL (e.g. BSF PCL).

10.0 OVERVIEW OF TIMER RESOURCES

The PIC17C4X has four timer modules. Each module can generate an interrupt to indicate that an event has occurred. These timers are called:

- Timer0 - 16-bit timer with programmable 8-bit prescaler
- Timer1 - 8-bit timer
- Timer2 - 8-bit timer
- Timer3 - 16-bit timer

For enhanced time-base functionality, two input Captures and two Pulse Width Modulation (PWM) outputs are possible. The PWMs use the TMR1 and TMR2 resources and the input Captures use the TMR3 resource.

10.1 Timer0 Overview

The Timer0 module is a simple 16-bit overflow counter. The clock source can be either the internal system clock ($F_{osc}/4$) or an external clock.

The Timer0 module also has a programmable prescaler option. The PS3:PS0 bits (T0STA<4:1>) determine the prescaler value. TMR0 can increment at the following rates: 1:1, 1:2, 1:4, 1:8, 1:16, 1:32, 1:64, 1:128, 1:256.

When Timer0's clock source is an external clock, the Timer0 module can be selected to increment on either the rising or falling edge.

Synchronization of the external clock occurs after the prescaler. When the prescaler is used, the external clock frequency may be higher than the device's frequency. The maximum frequency is 50 MHz, given the high and low time requirements of the clock.

10.2 Timer1 Overview

The Timer1 module is an 8-bit timer/counter with an 8-bit period register (PR1). When the TMR1 value rolls over from the period match value to 0h, the TMR1IF flag is set, and an interrupt will be generated when enabled. In counter mode, the clock comes from the RB4/TCLK12 pin, which can also be selected to be the clock for the Timer2 module.

TMR1 can be concatenated to TMR2 to form a 16-bit timer. The TMR1 register is the LSB and TMR2 is the MSB. When in the 16-bit timer mode, there is a corresponding 16-bit period register (PR2:PR1). When the TMR2:TMR1 value rolls over from the period match value to 0h, the TMR1IF flag is set, and an interrupt will be generated when enabled.

10.3 Timer2 Overview

The TMR2 module is an 8-bit timer/counter with an 8-bit period register (PR2). When the TMR2 value rolls over from the period match value to 0h, the TMR2IF flag is set, and an interrupt will be generated when enabled. In counter mode, the clock comes from the RB4/TCLK12 pin, which can also be selected to be the clock for the TMR1 module.

TMR1 can be concatenated to TMR2 to form a 16-bit timer. The TMR2 register is the MSB and TMR1 is the LSB. When in the 16-bit timer mode, there is a corresponding 16-bit period register (PR2:PR1). When the TMR2:TMR1 value rolls over from the period match value to 0h, the TMR1IF flag is set, and an interrupt will be generated when enabled.

10.4 Timer3 Overview

The Timer3 module is a 16-bit timer/counter with a 16-bit period register. When the TMR3H:TMR3L value rolls over to 0h, the TMR3IF bit is set and an interrupt will be generated when enabled. In counter mode, the clock comes from the RB5/TCLK3 pin.

When operating in the dual capture mode, the period registers become the second 16-bit capture register.

10.5 Role of the Timer/Counters

The timer modules are general purpose, but have dedicated resources associated with them. Timer1 and Timer2 are the time-bases for the two Pulse Width Modulation (PWM) outputs, while Timer3 is the time-base for the two input captures.

11.1 Timer0 Operation

When the T0CS (T0STA<5>) bit is set, TMR0 increments on the internal clock. When T0CS is clear, TMR0 increments on the external clock (RA1/T0CKI pin). The external clock edge can be configured in software. When the T0SE (T0STA<6>) bit is set, the timer will increment on the rising edge of the RA1/T0CKI pin. When T0SE is clear, the timer will increment on the falling edge of the RA1/T0CKI pin. The prescaler can be programmed to introduce a prescale of 1:1 to 1:256. The timer increments from 0000h to FFFFh and rolls over to 0000h. On overflow, the TMR0 Interrupt Flag bit (T0IF) is set. The TMR0 interrupt can be masked by clearing the corresponding TMR0 Interrupt Enable bit (T0IE). The TMR0 Interrupt Flag bit (T0IF) is automatically cleared when vectoring to the TMR0 interrupt vector.

11.2 Using Timer0 with External Clock

When the external clock input is used for Timer0, it is synchronized with the internal phase clocks. Figure 11-3 shows the synchronization of the external clock. This synchronization is done after the prescaler. The output of the prescaler (PSOUT) is sampled twice in every instruction cycle to detect a rising or a falling edge. The timing requirements for the external clock are detailed in the electrical specification section for the desired device.

11.2.1 DELAY FROM EXTERNAL CLOCK EDGE

Since the prescaler output is synchronized with the internal clocks, there is a small delay from the time the external clock edge occurs to the time TMR0 is actually incremented. Figure 11-3 shows that this delay is between $3T_{osc}$ and $7T_{osc}$. Thus, for example, measuring the interval between two edges (e.g. period) will be accurate within $\pm 4T_{osc}$ (± 121 ns @ 33 MHz).

FIGURE 11-2: TIMER0 MODULE BLOCK DIAGRAM

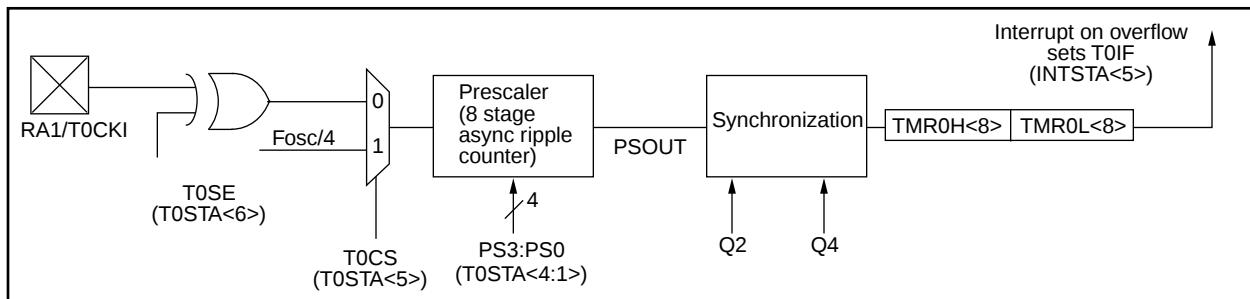


FIGURE 11-3: TMR0 TIMING WITH EXTERNAL CLOCK (INCREMENT ON FALLING EDGE)

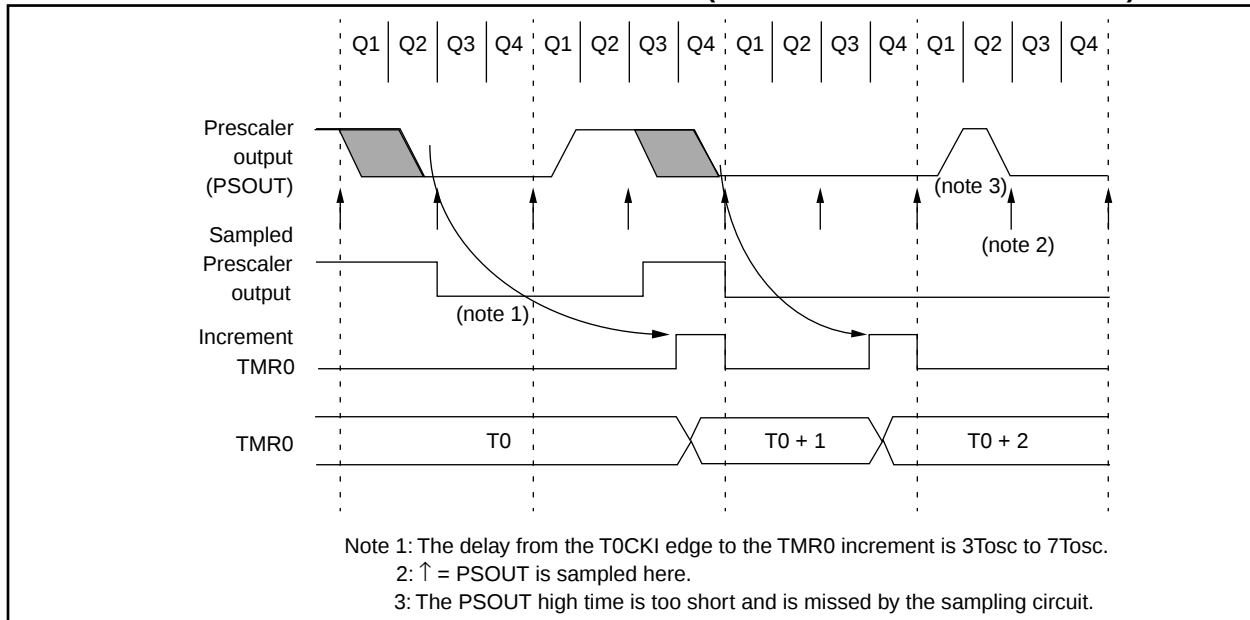


FIGURE 11-5: TMR0 READ/WRITE IN TIMER MODE

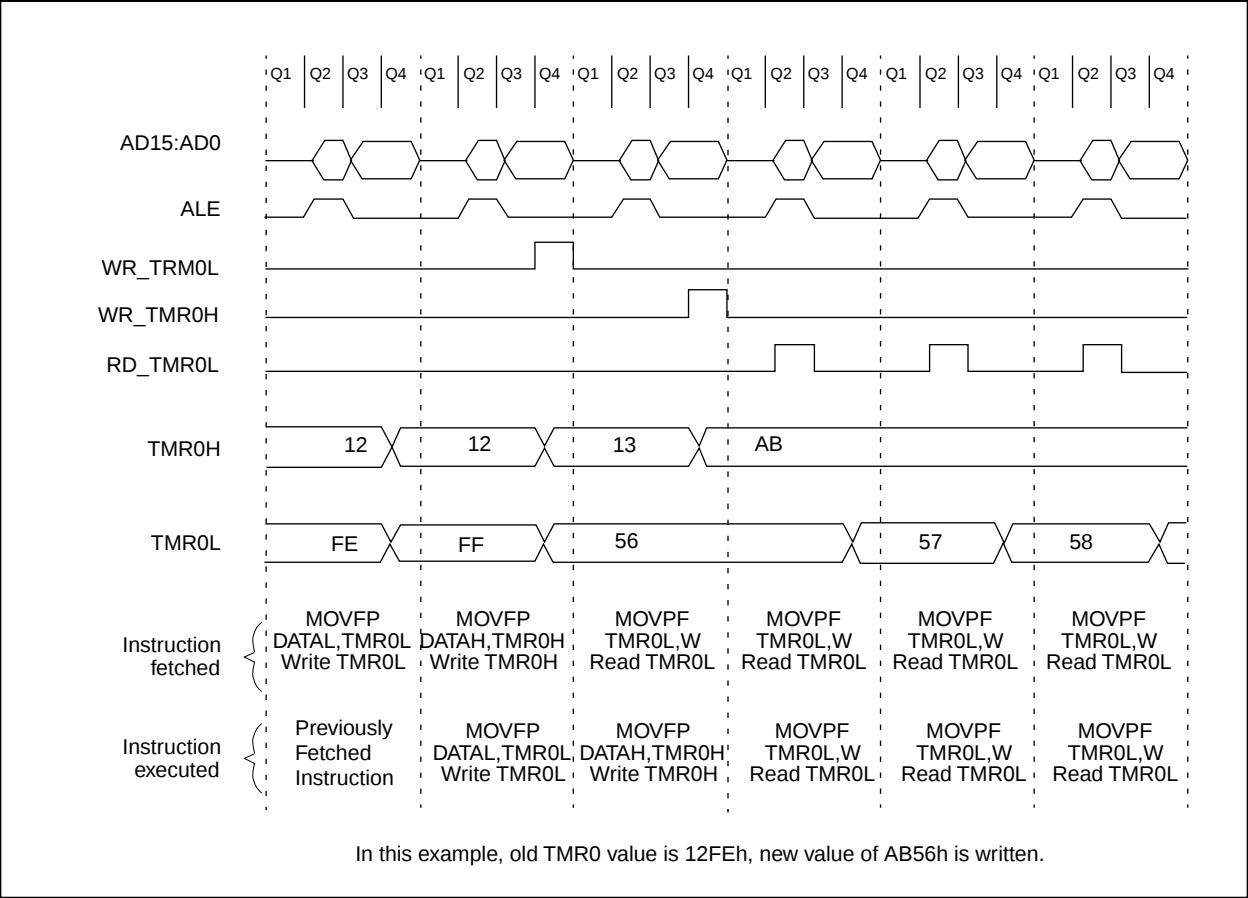


TABLE 11-1: REGISTERS/BITS ASSOCIATED WITH TIMER0

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets (Note1)
05h, Unbanked	T0STA	INTEDG	T0SE	T0CS	PS3	PS2	PS1	PS0	—	0000 000-	0000 000-
06h, Unbanked	CPUSTA	—	—	STKAV	GLINTD	$\overline{TO}$	$\overline{PD}$	—	—	--11 11--	--11 qq--
07h, Unbanked	INTSTA	PEIF	T0CKIF	T0IF	INTF	PEIE	T0CKIE	T0IE	INTE	0000 0000	0000 0000
0Bh, Unbanked	TMR0L	TMR0 register; low byte								xxxx xxxx	uuuu uuuu
0Ch, Unbanked	TMR0H	TMR0 register; high byte								xxxx xxxx	uuuu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented read as a '0', q - value depends on condition, Shaded cells are not used by Timer0.
Note 1: Other (non power-up) resets include: external reset through MCLR and the Watchdog Timer Reset.

FIGURE 13-2: RCSTA REGISTER (ADDRESS: 13h, BANK 0)

R/W - 0	R/W - 0	R/W - 0	R/W - 0	U - 0	R - 0	R - 0	R - x
SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D
bit 7							bit 0

R = Readable bit
W = Writable bit
-n = Value at POR reset
(x = unknown)

bit 7: **SPEN**: Serial Port Enable bit
1 = Configures RA5/RX/DT and RA4/TX/CK pins as serial port pins
0 = Serial port disabled

bit 6: **RX9**: 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5: **SREN**: Single Receive Enable bit
This bit enables the reception of a single byte. After receiving the byte, this bit is automatically cleared.
Synchronous mode:
1 = Enable reception
0 = Disable reception
Note: This bit is ignored in synchronous slave reception.
Asynchronous mode:
Don't care

bit 4: **CREN**: Continuous Receive Enable bit
This bit enables the continuous reception of serial data.
Asynchronous mode:
1 = Enable reception
0 = Disables reception
Synchronous mode:
1 = Enables continuous reception until CREN is cleared (CREN overrides SREN)
0 = Disables continuous reception

bit 3: **Unimplemented**: Read as '0'

bit 2: **FERR**: Framing Error bit
1 = Framing error (Updated by reading RCREG)
0 = No framing error

bit 1: **OERR**: Overrun Error bit
1 = Overrun (Cleared by clearing CREN)
0 = No overrun error

bit 0: **RX9D**: 9th bit of receive data (can be the software calculated parity bit)

13.4 USART Synchronous Slave Mode

The synchronous slave mode differs from the master mode in the fact that the shift clock is supplied externally at the RA5/TX/CK pin (instead of being supplied internally in the master mode). This allows the device to transfer or receive data in the SLEEP mode. The slave mode is entered by clearing the CSRC (TXSTA<7>) bit.

13.4.1 USART SYNCHRONOUS SLAVE TRANSMIT

The operation of the sync master and slave modes are identical except in the case of the SLEEP mode.

If two words are written to TXREG and then the SLEEP instruction executes, the following will occur. The first word will immediately transfer to the TSR and will transmit as the shift clock is supplied. The second word will remain in TXREG. TXIF will not be set. When the first word has been shifted out of TSR, TXREG will transfer the second word to the TSR and the TXIF flag will now be set. If TXIE is enabled, the interrupt will wake the chip from SLEEP and if the global interrupt is enabled, then the program will branch to interrupt vector (0020h).

Steps to follow when setting up a Synchronous Slave Transmission:

1. Enable the synchronous slave serial port by setting the SYNC and SPEN bits and clearing the CSRC bit.
2. Clear the CREN bit.
3. If interrupts are desired, then set the TXIE bit.
4. If 9-bit transmission is desired, then set the TX9 bit.
5. Start transmission by loading data to TXREG.
6. If 9-bit transmission is selected, the ninth bit should be loaded in TX9D.
7. Enable the transmission by setting TXEN.

Writing the transmit data to the TXREG, then enabling the transmit (setting TXEN) allows transmission to start sooner than doing these two events in the reverse order.

Note: To terminate a transmission, either clear the SPEN bit, or the TXEN bit. This will reset the transmit logic, so that it will be in the proper state when transmit is re-enabled.

13.4.2 USART SYNCHRONOUS SLAVE RECEPTION

Operation of the synchronous master and slave modes are identical except in the case of the SLEEP mode. Also, SREN is a don't care in slave mode.

If receive is enabled (CREN) prior to the SLEEP instruction, then a word may be received during SLEEP. On completely receiving the word, the RSR will transfer the data to RCREG (setting RCIF) and if the RCIE bit is set, the interrupt generated will wake the chip from SLEEP. If the global interrupt is enabled, the program will branch to the interrupt vector (0020h).

Steps to follow when setting up a Synchronous Slave Reception:

1. Enable the synchronous master serial port by setting the SYNC and SPEN bits and clearing the CSRC bit.
2. If interrupts are desired, then set the RCIE bit.
3. If 9-bit reception is desired, then set the RX9 bit.
4. To enable reception, set the CREN bit.
5. The RCIF bit will be set when reception is complete and an interrupt will be generated if the RCIE bit was set.
6. Read RCSTA to get the ninth bit (if enabled) and determine if any error occurred during reception.
7. Read the 8-bit received data by reading RCREG.
8. If any error occurred, clear the error by clearing the CREN bit.

Note: To abort reception, either clear the SPEN bit, the SREN bit (when in single receive mode), or the CREN bit (when in continuous receive mode). This will reset the receive logic, so that it will be in the proper state when receive is re-enabled.

TABLE 15-2: PIC17CXX INSTRUCTION SET

Mnemonic, Operands	Description	Cycles	16-bit Opcode		Status Affected	Notes
			MSb	LSb		
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f,d	ADD WREG to f	1	0000 111d ffff ffff	OV,C,DC,Z	
ADDWFC	f,d	ADD WREG and Carry bit to f	1	0001 000d ffff ffff	OV,C,DC,Z	
ANDWF	f,d	AND WREG with f	1	0000 101d ffff ffff	Z	
CLRF	f,s	Clear f, or Clear f and Clear WREG	1	0010 100s ffff ffff	None	3
COMF	f,d	Complement f	1	0001 001d ffff ffff	Z	
CPFSEQ	f	Compare f with WREG, skip if f = WREG	1 (2)	0011 0001 ffff ffff	None	6,8
CPFSGT	f	Compare f with WREG, skip if f > WREG	1 (2)	0011 0010 ffff ffff	None	2,6,8
CPFSLT	f	Compare f with WREG, skip if f < WREG	1 (2)	0011 0000 ffff ffff	None	2,6,8
DAW	f,s	Decimal Adjust WREG Register	1	0010 111s ffff ffff	C	3
DECF	f,d	Decrement f	1	0000 011d ffff ffff	OV,C,DC,Z	
DECFSZ	f,d	Decrement f, skip if 0	1 (2)	0001 011d ffff ffff	None	6,8
DCFSNZ	f,d	Decrement f, skip if not 0	1 (2)	0010 011d ffff ffff	None	6,8
INCF	f,d	Increment f	1	0001 010d ffff ffff	OV,C,DC,Z	
INCFSZ	f,d	Increment f, skip if 0	1 (2)	0001 111d ffff ffff	None	6,8
INFSNZ	f,d	Increment f, skip if not 0	1 (2)	0010 010d ffff ffff	None	6,8
IORWF	f,d	Inclusive OR WREG with f	1	0000 100d ffff ffff	Z	
MOVFP	f,p	Move f to p	1	011p pppp ffff ffff	None	
MOVPF	p,f	Move p to f	1	010p pppp ffff ffff	Z	
MOVWF	f	Move WREG to f	1	0000 0001 ffff ffff	None	
MULWF	f	Multiply WREG with f	1	0011 0100 ffff ffff	None	9
NEGW	f,s	Negate WREG	1	0010 110s ffff ffff	OV,C,DC,Z	1,3
NOP	—	No Operation	1	0000 0000 0000 0000	None	
RLCF	f,d	Rotate left f through Carry	1	0001 101d ffff ffff	C	
RLNCF	f,d	Rotate left f (no carry)	1	0010 001d ffff ffff	None	
RRCF	f,d	Rotate right f through Carry	1	0001 100d ffff ffff	C	
RRNCF	f,d	Rotate right f (no carry)	1	0010 000d ffff ffff	None	
SETF	f,s	Set f	1	0010 101s ffff ffff	None	3
SUBWF	f,d	Subtract WREG from f	1	0000 010d ffff ffff	OV,C,DC,Z	1
SUBWFB	f,d	Subtract WREG from f with Borrow	1	0000 001d ffff ffff	OV,C,DC,Z	1
SWAPF	f,d	Swap f	1	0001 110d ffff ffff	None	
TABLRD	t,i,f	Table Read	2 (3)	1010 10ti ffff ffff	None	7

Legend: Refer to Table 15-1 for opcode field descriptions.

Note 1: 2's Complement method.

2: Unsigned arithmetic.

3: If s = '1', only the file is affected: If s = '0', both the WREG register and the file are affected; If only the Working register (WREG) is required to be affected, then f = WREG must be specified.

4: During an LCALL, the contents of PCLATH are loaded into the MSB of the PC and kkkk kkkk is loaded into the LSB of the PC (PCL)

5: Multiple cycle instruction for EPROM programming when table pointer selects internal EPROM. The instruction is terminated by an interrupt event. When writing to external program memory, it is a two-cycle instruction.

6: Two-cycle instruction when condition is true, else single cycle instruction.

7: Two-cycle instruction except for TABLRD to PCL (program counter low byte) in which case it takes 3 cycles.

8: A "skip" means that instruction fetched during execution of current instruction is not executed, instead an NOP is executed.

9: These instructions are not available on the PIC17C42.

ANDWF
Syntax: [*label*] ANDWF f,d
Operands: $0 \leq f \leq 255$
 $d \in [0,1]$
Operation: (WREG) .AND. (f) → (dest)
Status Affected: Z
Encoding:

0000	101d	ffff	ffff
------	------	------	------

Description: The contents of WREG are AND'ed with register 'f'. If 'd' is 0 the result is stored in WREG. If 'd' is 1 the result is stored back in register 'f'.
Words: 1
Cycles: 1
Q Cycle Activity:

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

Example: ANDWF REG, 1

Before Instruction

WREG = 0x17

REG = 0xC2

After Instruction

WREG = 0x17

REG = 0x02

BCF
Syntax: [*label*] BCF f,b
Operands: $0 \leq f \leq 255$
 $0 \leq b \leq 7$
Operation: $0 \rightarrow (f)$
Status Affected: None
Encoding:

1000	1bbb	ffff	ffff
------	------	------	------

Description: Bit 'b' in register 'f' is cleared.
Words: 1
Cycles: 1
Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Execute	Write register 'f'

Example: BCF FLAG_REG, 7

Before Instruction

FLAG_REG = 0xC7

After Instruction

FLAG_REG = 0x47

IORWF		Inclusive OR WREG with f						
Syntax:	[<i>label</i>] IORWF f,d							
Operands:	$0 \leq f \leq 255$ $d \in [0,1]$							
Operation:	(WREG) .OR. (f) $\rightarrow$ (dest)							
Status Affected:	Z							
Encoding:	<table><tr><td>0000</td><td>100d</td><td>ffff</td><td>ffff</td></tr></table>				0000	100d	ffff	ffff
0000	100d	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'.							
Words:	1							
Cycles:	1							
Q Cycle Activity:								
	Q1	Q2	Q3	Q4				
	Decode	Read register 'f'	Execute	Write to destination				

Example: IORWF RESULT, 0

Before Instruction

RESULT = 0x13
WREG = 0x91

After Instruction

RESULT = 0x13
WREG = 0x93

LCALL		Long Call						
Syntax:	[<i>label</i>] LCALL k							
Operands:	$0 \leq k \leq 255$							
Operation:	PC + 1 $\rightarrow$ TOS; k $\rightarrow$ PCL, (PCLATH) $\rightarrow$ PCH							
Status Affected:	None							
Encoding:	<table><tr><td>1011</td><td>0111</td><td>kkkk</td><td>kkkk</td></tr></table>				1011	0111	kkkk	kkkk
1011	0111	kkkk	kkkk					
Description:	LCALL allows an unconditional subroutine call to anywhere within the 64k program memory space. First, the return address (PC + 1) is pushed onto the stack. A 16-bit destination address is then loaded into the program counter. The lower 8-bits of the destination address is embedded in the instruction. The upper 8-bits of PC is loaded from PC high holding latch, PCLATH.							
Words:	1							
Cycles:	2							
Q Cycle Activity:								
	Q1	Q2	Q3	Q4				
	Decode	Read literal 'k'	Execute	Write register PCL				
	Forced NOP	NOP	Execute	NOP				

Example: MOVLW HIGH(SUBROUTINE)
MOVWF WREG, PCLATH
LCALL LOW(SUBROUTINE)

Before Instruction

SUBROUTINE = 16-bit Address
PC = ?

After Instruction

PC = Address (SUBROUTINE)

RRNCF Rotate Right f (no carry)

Syntax: [label] RRNCF f,d

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

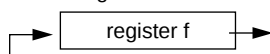
Operation: $f \langle n \rangle \rightarrow d \langle n-1 \rangle$;
 $f \langle 0 \rangle \rightarrow d \langle 7 \rangle$

Status Affected: None

Encoding:

0010	000d	ffff	ffff
------	------	------	------

Description: The contents of register 'f' are rotated one bit to the right. If 'd' is 0 the result is placed in WREG. If 'd' is 1 the result is placed back in register 'f'.



Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1: RRNCF REG, 1

Before Instruction

WREG = ?
 REG = 1101 0111

After Instruction

WREG = 0
 REG = 1110 1011

Example 2: RRNCF REG, 0

Before Instruction

WREG = ?
 REG = 1101 0111

After Instruction

WREG = 1110 1011
 REG = 1101 0111

SETF Set f

Syntax: [label] SETF f,s

Operands: $0 \leq f \leq 255$
 $s \in [0,1]$

Operation: FFh $\rightarrow$ f;
 FFh $\rightarrow$ d

Status Affected: None

Encoding:

0010	101s	ffff	ffff
------	------	------	------

Description: If 's' is 0, both the data memory location 'f' and WREG are set to FFh. If 's' is 1 only the data memory location 'f' is set to FFh.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Execute	Write register 'f' and other specified register

Example1: SETF REG, 0

Before Instruction

REG = 0xDA
 WREG = 0x05

After Instruction

REG = 0xFF
 WREG = 0xFF

Example2: SETF REG, 1

Before Instruction

REG = 0xDA
 WREG = 0x05

After Instruction

REG = 0xFF
 WREG = 0x05

TLWT		Table Latch Write						
Syntax:	[<i>label</i>] TLWT t,f							
Operands:	0 ≤ f ≤ 255 t ∈ [0,1]							
Operation:	If t = 0, f → TBLATL; If t = 1, f → TBLATH							
Status Affected:	None							
Encoding:	<table><tr><td>1010</td><td>01tx</td><td>ffff</td><td>ffff</td></tr></table>				1010	01tx	ffff	ffff
1010	01tx	ffff	ffff					
Description:	Data from file register 'f' is written into the 16-bit table latch (TBLAT). If t = 1; high byte is written If t = 0; low byte is written This instruction is used in conjunction with TABLWT to transfer data from data memory to program memory.							
Words:	1							
Cycles:	1							
Q Cycle Activity:								
Q1		Q2		Q3		Q4		
Decode		Read register 'f'		Execute		Write register TBLATH or TBLATL		

Example: TLWT t, RAM

Before Instruction

t = 0
RAM = 0xB7
TBLAT = 0x0000 (TBLATH = 0x00)
(TBLATL = 0x00)

After Instruction

RAM = 0xB7
TBLAT = 0x00B7 (TBLATH = 0x00)
(TBLATL = 0xB7)

Before Instruction

t = 1
RAM = 0xB7
TBLAT = 0x0000 (TBLATH = 0x00)
(TBLATL = 0x00)

After Instruction

RAM = 0xB7
TBLAT = 0xB700 (TBLATH = 0xB7)
(TBLATL = 0x00)

TSTFSZ		Test f, skip if 0						
Syntax:	[<i>label</i>] TSTFSZ f							
Operands:	0 ≤ f ≤ 255							
Operation:	skip if f = 0							
Status Affected:	None							
Encoding:	<table><tr><td>0011</td><td>0011</td><td>ffff</td><td>ffff</td></tr></table>				0011	0011	ffff	ffff
0011	0011	ffff	ffff					
Description:	If 'f' = 0, the next instruction, fetched during the current instruction execution, is discarded and an NOP is executed making this a two-cycle instruction.							
Words:	1							
Cycles:	1 (2)							
Q Cycle Activity:								
Q1		Q2		Q3		Q4		
Decode		Read register 'f'		Execute		NOP		
If skip:								
Q1		Q2		Q3		Q4		
Forced NOP		NOP		Execute		NOP		

Example: HERE TSTFSZ CNT
NZERO :
ZERO :

Before Instruction

PC = Address(HERE)

After Instruction

If CNT = 0x00,
PC = Address (ZERO)
If CNT ≠ 0x00,
PC = Address (NZERO)

XORLW		Exclusive OR Literal with WREG						
Syntax:	[<i>label</i>] XORLW k							
Operands:	0 ≤ k ≤ 255							
Operation:	(WREG) .XOR. k → (WREG)							
Status Affected:	Z							
Encoding:	<table><tr><td>1011</td><td>0100</td><td>kkkk</td><td>kkkk</td></tr></table>				1011	0100	kkkk	kkkk
1011	0100	kkkk	kkkk					
Description:	The contents of WREG are XOR'ed with the 8-bit literal 'k'. The result is placed in WREG.							
Words:	1							
Cycles:	1							
Q Cycle Activity:								
	Q1	Q2	Q3	Q4				
	Decode	Read literal 'k'	Execute	Write to WREG				

Example: XORLW 0xAF

Before Instruction
WREG = 0xB5

After Instruction
WREG = 0x1A

XORWF		Exclusive OR WREG with f						
Syntax:	[<i>label</i>] XORWF f,d							
Operands:	0 ≤ f ≤ 255 d ∈ [0,1]							
Operation:	(WREG) .XOR. (f) → (dest)							
Status Affected:	Z							
Encoding:	<table><tr><td>0000</td><td>110d</td><td>ffff</td><td>ffff</td></tr></table>				0000	110d	ffff	ffff
0000	110d	ffff	ffff					
Description:	Exclusive OR the contents of WREG with register 'f'. If 'd' is 0 the result is stored in WREG. If 'd' is 1 the result is stored back in the register 'f'.							
Words:	1							
Cycles:	1							
Q Cycle Activity:								
Q1		Q2		Q3				
Q4								
Decode		Read register 'f'		Execute				
				Write to destination				

Example: XORWF REG, 1

Before Instruction
REG = 0xAF
WREG = 0xB5

After Instruction
REG = 0x1A
WREG = 0xB5

16.6 PICDEM-1 Low-Cost PIC16/17 Demonstration Board

The PICDEM-1 is a simple board which demonstrates the capabilities of several of Microchip's microcontrollers. The microcontrollers supported are: PIC16C5X (PIC16C54 to PIC16C58A), PIC16C61, PIC16C62X, PIC16C71, PIC16C8X, PIC17C42, PIC17C43 and PIC17C44. All necessary hardware and software is included to run basic demo programs. The users can program the sample microcontrollers provided with the PICDEM-1 board, on a PRO MATE II or PICSTART-16B programmer, and easily test firmware. The user can also connect the PICDEM-1 board to the PICMASTER emulator and download the firmware to the emulator for testing. Additional prototype area is available for the user to build some additional hardware and connect it to the microcontroller socket(s). Some of the features include an RS-232 interface, a potentiometer for simulated analog input, push-button switches and eight LEDs connected to PORTB.

16.7 PICDEM-2 Low-Cost PIC16CXX Demonstration Board

The PICDEM-2 is a simple demonstration board that supports the PIC16C62, PIC16C64, PIC16C65, PIC16C73 and PIC16C74 microcontrollers. All the necessary hardware and software is included to run the basic demonstration programs. The user can program the sample microcontrollers provided with the PICDEM-2 board, on a PRO MATE II programmer or PICSTART-16C, and easily test firmware. The PICMASTER emulator may also be used with the PICDEM-2 board to test firmware. Additional prototype area has been provided to the user for adding additional hardware and connecting it to the microcontroller socket(s). Some of the features include a RS-232 interface, push-button switches, a potentiometer for simulated analog input, a Serial EEPROM to demonstrate usage of the I²C bus and separate headers for connection to an LCD module and a keypad.

16.8 PICDEM-3 Low-Cost PIC16CXXX Demonstration Board

The PICDEM-3 is a simple demonstration board that supports the PIC16C923 and PIC16C924 in the PLCC package. It will also support future 44-pin PLCC microcontrollers with a LCD Module. All the necessary hardware and software is included to run the basic demonstration programs. The user can program the sample microcontrollers provided with the PICDEM-3 board, on a PRO MATE II programmer or PICSTART Plus with an adapter socket, and easily test firmware. The PICMASTER emulator may also be used with the PICDEM-3 board to test firmware. Additional prototype area has been provided to the user for adding hardware and connecting it to the microcontroller socket(s). Some of the features

include an RS-232 interface, push-button switches, a potentiometer for simulated analog input, a thermistor and separate headers for connection to an external LCD module and a keypad. Also provided on the PICDEM-3 board is an LCD panel, with 4 commons and 12 segments, that is capable of displaying time, temperature and day of the week. The PICDEM-3 provides an additional RS-232 interface and Windows 3.1 software for showing the demultiplexed LCD signals on a PC. A simple serial interface allows the user to construct a hardware demultiplexer for the LCD signals. PICDEM-3 will be available in the 3rd quarter of 1996.

16.9 MPLAB Integrated Development Environment Software

The MPLAB IDE Software brings an ease of software development previously unseen in the 8-bit microcontroller market. MPLAB is a windows based application which contains:

- A full featured editor
- Three operating modes
 - editor
 - emulator
 - simulator
- A project manager
- Customizable tool bar and key mapping
- A status bar with project information
- Extensive on-line help

MPLAB allows you to:

- Edit your source files (either assembly or 'C')
- One touch assemble (or compile) and download to PIC16/17 tools (automatically updates all project information)
- Debug using:
 - source files
 - absolute listing file
- Transfer data dynamically via DDE (soon to be replaced by OLE)
- Run up to four emulators on the same PC

The ability to use MPLAB with Microchip's simulator allows a consistent platform and the ability to easily switch from the low cost simulator to the full featured emulator with minimal retraining due to development tools.

16.10 Assembler (MPASM)

The MPASM Universal Macro Assembler is a PC-hosted symbolic assembler. It supports all microcontroller series including the PIC12C5XX, PIC14000, PIC16C5X, PIC16CXXX, and PIC17CXX families.

MPASM offers full featured Macro capabilities, conditional assembly, and several source and listing formats. It generates various object code formats to support Microchip's development tools as well as third party programmers.

PIC17C4X

Applicable Devices 42 R42 42A 43 R43 44

FIGURE 19-4: RESET, WATCHDOG TIMER, OSCILLATOR START-UP TIMER, AND POWER-UP TIMER TIMING

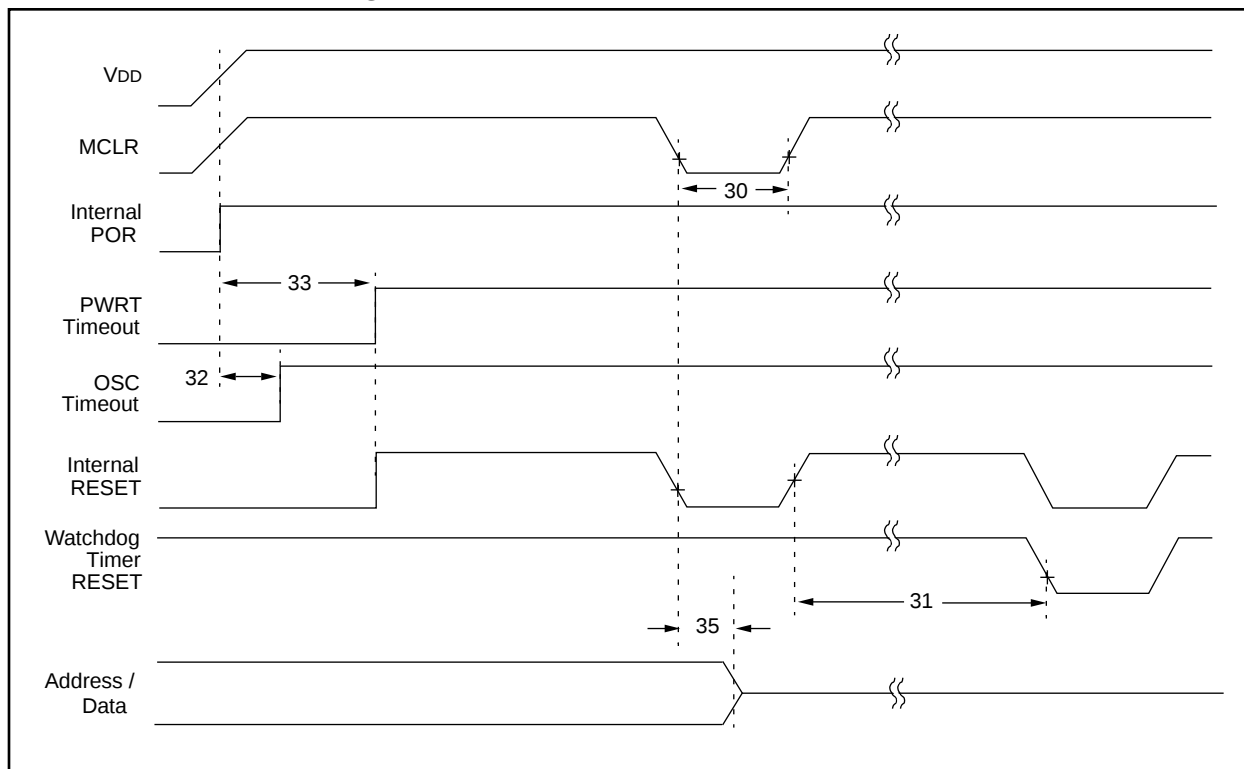


TABLE 19-4: RESET, WATCHDOG TIMER, OSCILLATOR START-UP TIMER AND POWER-UP TIMER REQUIREMENTS

Parameter No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions
30	Tmcl	$\overline{\text{MCLR}}$ Pulse Width (low)	100 *	—	—	ns	VDD = 5V
31	Twdt	Watchdog Timer Time-out Period (Prescale = 1)	5 *	12	25 *	ms	VDD = 5V
32	Tost	Oscillation Start-up Timer Period	—	1024Tosc‡	—	ms	Tosc = OSC1 period
33	Tpwrt	Power-up Timer Period	40 *	96	200 *	ms	VDD = 5V
35	Tmcl2adi	MCLR to System Interface bus (AD15:AD0>) invalid					
		PIC17CR42/42A/43/R43/44	—	—	100 *	ns	
		PIC17LCR42/42A/43/R43/44	—	—	120 *	ns	

* These parameters are characterized but not tested.

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

‡ These parameters are for design guidance only and are not tested, nor characterized.

§ This specification ensured by design.

PIC17C4X

NOTES:

PIC17C4X

Applicable Devices 42 R42 42A 43 R43 44

FIGURE 20-9: TYPICAL I_{PD} vs. V_{DD} WATCHDOG DISABLED 25°C

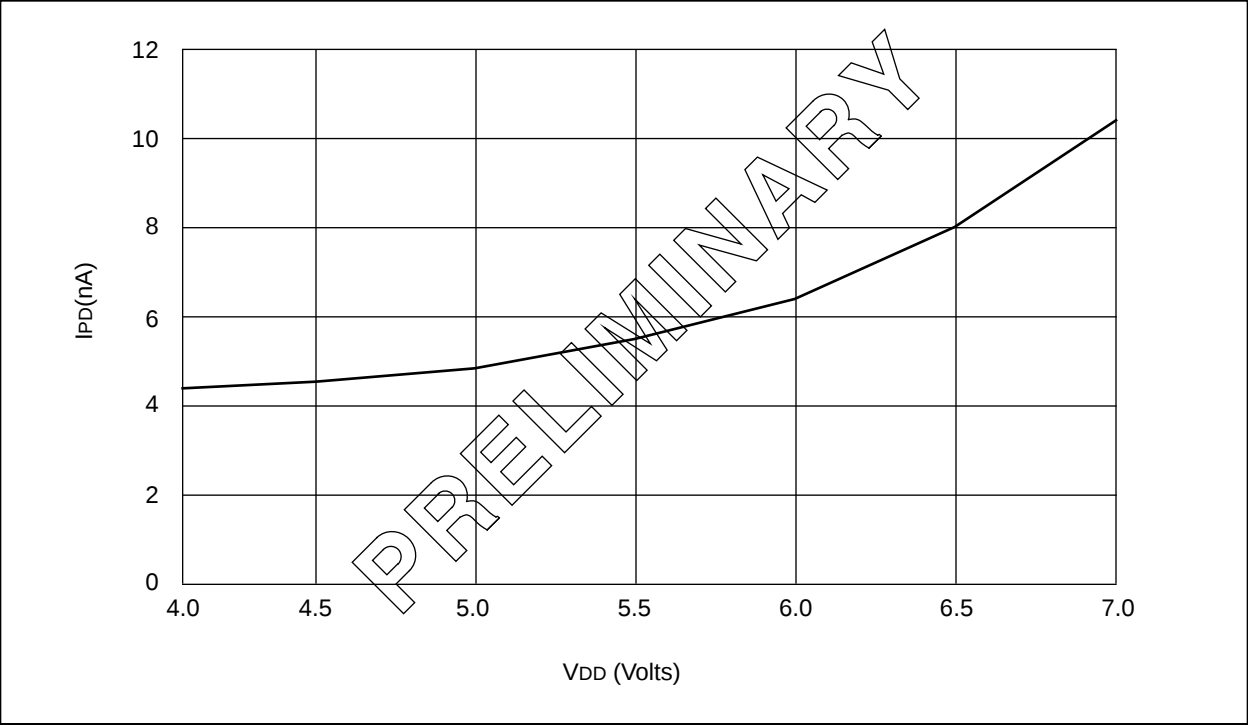
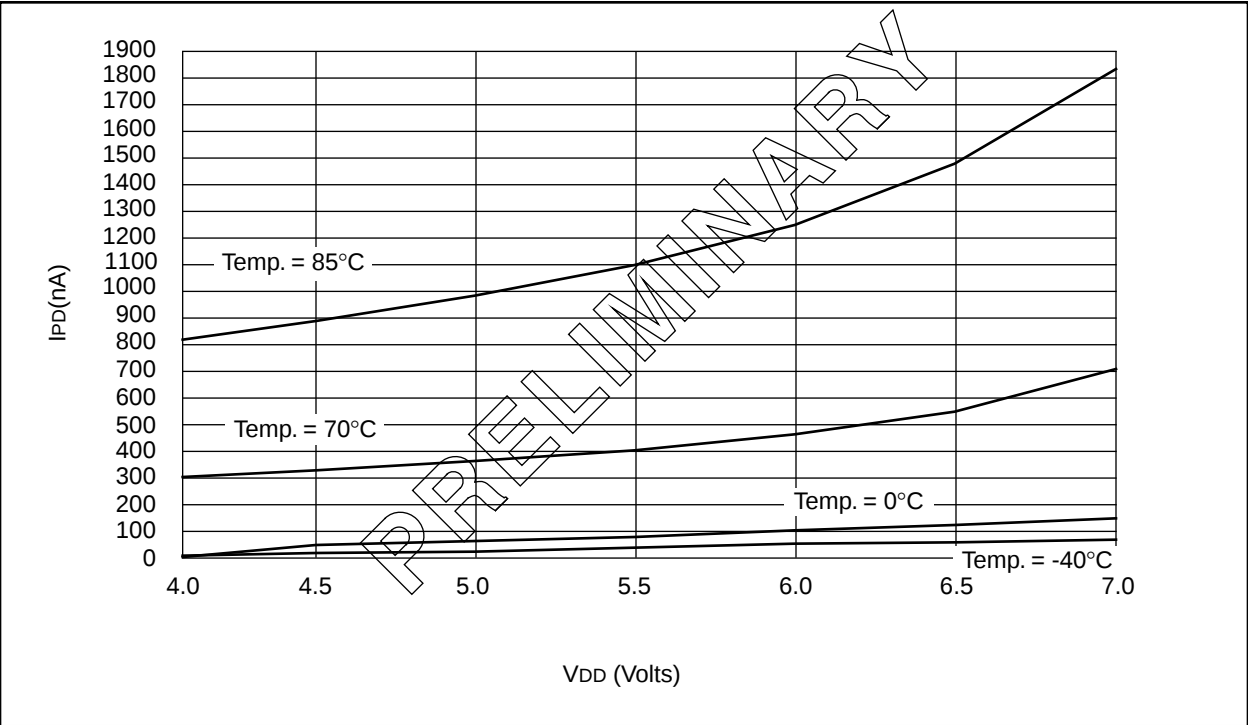


FIGURE 20-10: MAXIMUM I_{PD} vs. V_{DD} WATCHDOG DISABLED



PIC17C4X

E.8 PIC17CXX Family of Devices

	Clock		Memory		Peripherals				Features					
	Maximum Frequency of Operation (MHz)	Program Memory (Words)	RAM Data Memory (bytes)		Timer Module(s)	Captures PWMs	Serial Port(s) (USART)	Hardware Multiply	External Interrupts	Interrupt Sources	I/O Pins	Voltage Range (Volts)	Number of Instructions	Packages
			EPROM	Program										
PIC17C42	25	2K	—	232	TMR0,TMR1, TMR2,TMR3	2 2	Yes	—	Yes	11	33	4.5-5.5	55	40-pin DIP; 44-pin PLCC, MQFP
PIC17C42A	25	2K	—	232	TMR0,TMR1, TMR2,TMR3	2 2	Yes	Yes	Yes	11	33	2.5-6.0	58	40-pin DIP; 44-pin PLCC, TQFP, MQFP
PIC17CR42	25	—	2K	232	TMR0,TMR1, TMR2,TMR3	2 2	Yes	Yes	Yes	11	33	2.5-6.0	58	40-pin DIP; 44-pin PLCC, TQFP, MQFP
PIC17C43	25	4K	—	454	TMR0,TMR1, TMR2,TMR3	2 2	Yes	Yes	Yes	11	33	2.5-6.0	58	40-pin DIP; 44-pin PLCC, TQFP, MQFP
PIC17CR43	25	—	4K	454	TMR0,TMR1, TMR2,TMR3	2 2	Yes	Yes	Yes	11	33	2.5-6.0	58	40-pin DIP; 44-pin PLCC, TQFP, MQFP
PIC17C44	25	8K	—	454	TMR0,TMR1, TMR2,TMR3	2 2	Yes	Yes	Yes	11	33	2.5-6.0	58	40-pin DIP; 44-pin PLCC, TOFP, MQFP

All PIC16/17 Family devices have Power-on Reset, selectable Watchdog Timer, selectable code protect and high I/O current capability.