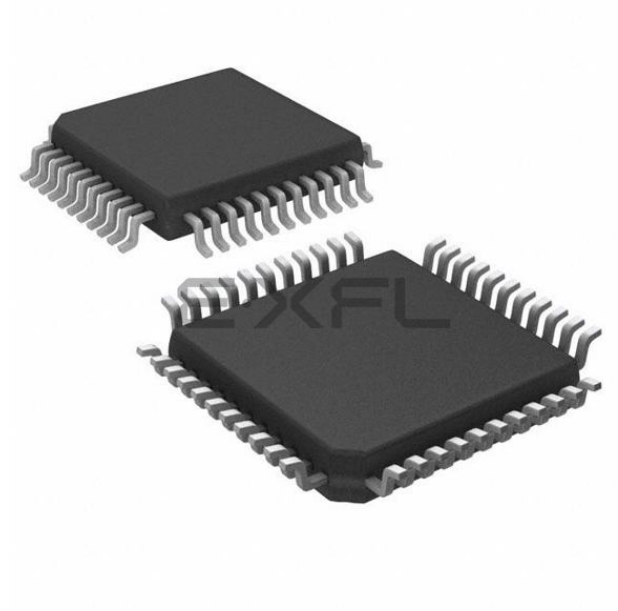


Welcome to E-XFL.COM

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	25MHz
Connectivity	UART/USART
Peripherals	POR, PWM, WDT
Number of I/O	33
Program Memory Size	16KB (8K 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	-40°C ~ 85°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/pic17c44t-25i-pq

3.0 ARCHITECTURAL OVERVIEW

The high performance of the PIC17C4X can be attributed to a number of architectural features commonly found in RISC microprocessors. To begin with, the PIC17C4X uses a modified Harvard architecture. This architecture has the program and data accessed from separate memories. So the device has a program memory bus and a data memory bus. This improves bandwidth over traditional von Neumann architecture, where program and data are fetched from the same memory (accesses over the same bus). Separating program and data memory further allows instructions to be sized differently than the 8-bit wide data word. PIC17C4X opcodes are 16-bits wide, enabling single word instructions. The full 16-bit wide program memory bus fetches a 16-bit instruction in a single cycle. A two-stage pipeline overlaps fetch and execution of instructions. Consequently, all instructions execute in a single cycle (121 ns @ 33 MHz), except for program branches and two special instructions that transfer data between program and data memory.

The PIC17C4X can address up to 64K x 16 of program memory space.

The **PIC17C42** and **PIC17C42A** integrate 2K x 16 of EPROM program memory on-chip, while the **PIC17CR42** has 2K x 16 of ROM program memory on-chip.

The **PIC17C43** integrates 4K x 16 of EPROM program memory, while the **PIC17CR43** has 4K x 16 of ROM program memory.

The **PIC17C44** integrates 8K x 16 EPROM program memory.

Program execution can be internal only (microcontroller or protected microcontroller mode), external only (microprocessor mode) or both (extended microcontroller mode). Extended microcontroller mode does not allow code protection.

The PIC17CXX can directly or indirectly address its register files or data memory. All special function registers, including the Program Counter (PC) and Working Register (WREG), are mapped in the data memory. The PIC17CXX has an orthogonal (symmetrical) instruction set that makes it possible to carry out any operation on any register using any addressing mode. This symmetrical nature and lack of 'special optimal situations' make programming with the PIC17CXX simple yet efficient. In addition, the learning curve is reduced significantly.

One of the PIC17CXX family architectural enhancements from the PIC16CXX family allows two file registers to be used in some two operand instructions. This allows data to be moved directly between two registers without going through the WREG register. This increases performance and decreases program memory usage.

The PIC17CXX devices contain an 8-bit ALU and working register. The ALU is a general purpose arithmetic unit. It performs arithmetic and Boolean functions between data in the working register and any register file.

The ALU is 8-bits wide and capable of addition, subtraction, shift, and logical operations. Unless otherwise mentioned, arithmetic operations are two's complement in nature.

The WREG register is an 8-bit working register used for ALU operations.

All PIC17C4X devices (except the PIC17C42) have an 8 x 8 hardware multiplier. This multiplier generates a 16-bit result in a single cycle.

Depending on the instruction executed, the ALU may affect the values of the Carry (C), Digit Carry (DC), and Zero (Z) bits in the STATUS register. The C and DC bits operate as a $\overline{\text{borrow}}$ and $\overline{\text{digit}}$ $\overline{\text{borrow}}$ out bit, respectively, in subtraction. See the SUBLW and SUBWF instructions for examples.

Although the ALU does not perform signed arithmetic, the Overflow bit (OV) can be used to implement signed math. Signed arithmetic is comprised of a magnitude and a sign bit. The overflow bit indicates if the magnitude overflows and causes the sign bit to change state. Signed math can have greater than 7-bit values (magnitude), if more than one byte is used. The use of the overflow bit only operates on bit6 (MSb of magnitude) and bit7 (sign bit) of the value in the ALU. That is, the overflow bit is not useful if trying to implement signed math where the magnitude, for example, is 11-bits. If the signed math values are greater than 7-bits (15-, 24- or 31-bit), the algorithm must ensure that the low order bytes ignore the overflow status bit.

Care should be taken when adding and subtracting signed numbers to ensure that the correct operation is executed. Example 3-1 shows an item that must be taken into account when doing signed arithmetic on an ALU which operates as an unsigned machine.

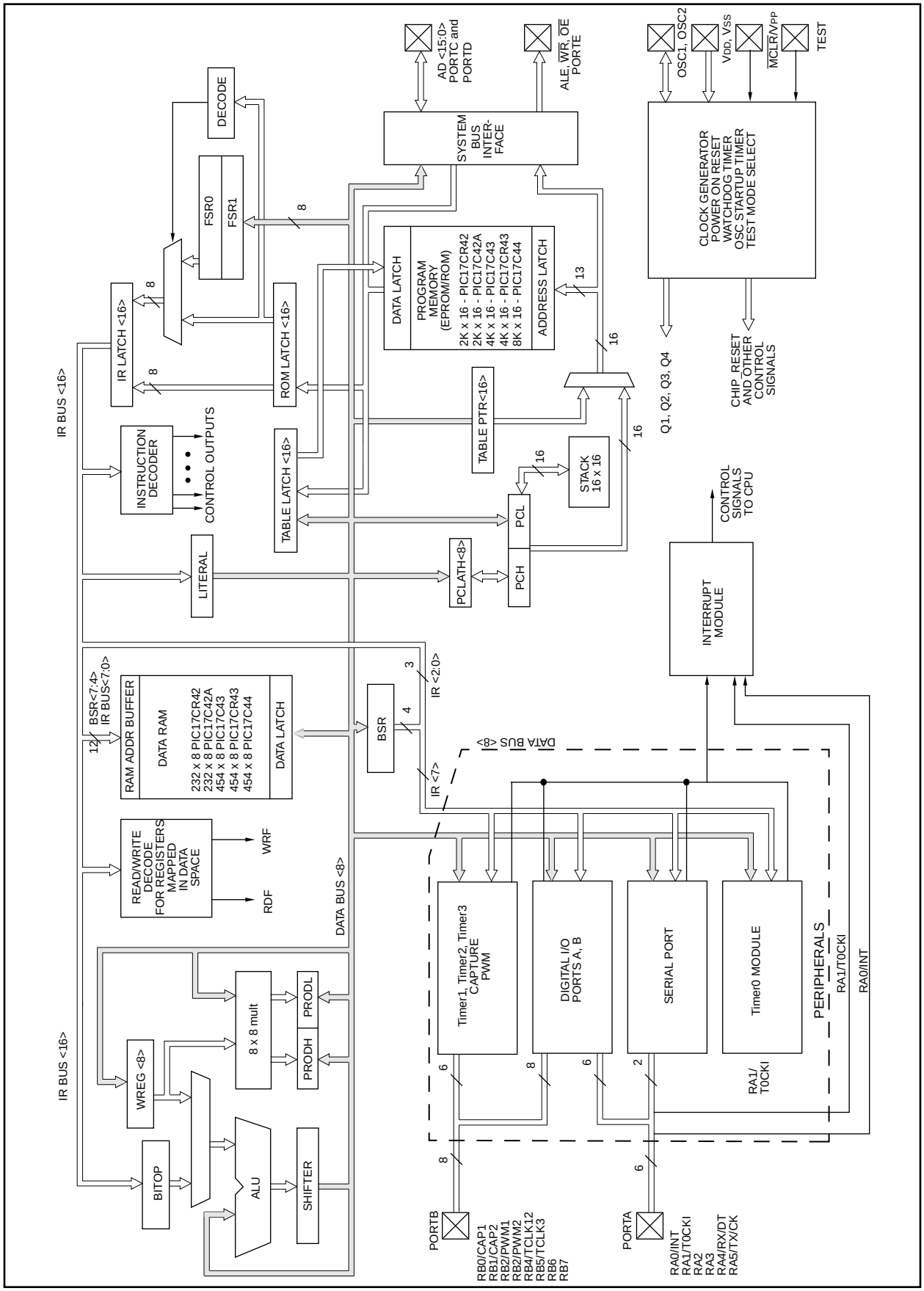
EXAMPLE 3-1: SIGNED MATH

Hex Value	Signed Value Math	Unsigned Value Math
FFh	-127	255
+ 01h	+ 1	+ 1
= ?	= -126 (FEh)	= 0 (00h); Carry bit = 1

Signed math requires the result in REG to be FEh (-126). This would be accomplished by subtracting one as opposed to adding one.

Simplified block diagrams are shown in Figure 3-1 and Figure 3-2. The descriptions of the device pins are listed in Table 3-1.

FIGURE 3-2: PIC17CR42/42A/43/R43/44 BLOCK DIAGRAM



6.2.2.2 CPU STATUS REGISTER (CPUSTA)

The CPUSTA register contains the status and control bits for the CPU. This register is used to globally enable/disable interrupts. If only a specific interrupt is desired to be enabled/disabled, please refer to the INTerrupt STAtus (INTSTA) register and the Peripheral Interrupt Enable (PIE) register. This register also indicates if the stack is available and contains the Power-down ($\overline{PD}$) and Time-out ($\overline{TO}$) bits. The $\overline{TO}$, $\overline{PD}$, and STKAV bits are not writable. These bits are set and cleared according to device logic. Therefore, the result of an instruction with the CPUSTA register as destination may be different than intended.

FIGURE 6-8: CPUSTA REGISTER (ADDRESS: 06h, UNBANKED)

U - 0	U - 0	R - 1	R/W - 1	R - 1	R - 1	U - 0	U - 0
—	—	STKAV	GLINTD	$\overline{TO}$	$\overline{PD}$	—	—
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, Read as '0'
- n = Value at POR reset

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

bit 5: **STKAV**: Stack Available bit
This bit indicates that the 4-bit stack pointer value is Fh, or has rolled over from Fh→ 0h (stack overflow).
1 = Stack is available
0 = Stack is full, or a stack overflow may have occurred (Once this bit has been cleared by a stack overflow, only a device reset will set this bit)

bit 4: **GLINTD**: Global Interrupt Disable bit
This bit disables all interrupts. When enabling interrupts, only the sources with their enable bits set can cause an interrupt.
1 = Disable all interrupts
0 = Enables all un-masked interrupts

bit 3: **$\overline{TO}$**: WDT Time-out Status bit
1 = After power-up or by a CLRWDT instruction
0 = A Watchdog Timer time-out occurred

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

bit 1-0: **Unimplemented**: Read as '0'

6.3 Stack Operation

The PIC17C4X devices have a 16 x 16-bit wide hardware stack (Figure 6-1). The stack is not part of either the program or data memory space, and the stack pointer is neither readable nor writable. The PC is "PUSHed" onto the stack when a CALL instruction is executed or an interrupt is acknowledged. The stack is "POPed" in the event of a RETURN, RETLW, or a RETFIE instruction execution. PCLATH is not affected by a "PUSH" or a "POP" operation.

The stack operates as a circular buffer, with the stack pointer initialized to '0' after all resets. There is a stack available bit (STKAV) to allow software to ensure that the stack has not overflowed. The STKAV bit is set after a device reset. When the stack pointer equals Fh, STKAV is cleared. When the stack pointer rolls over from Fh to 0h, the STKAV bit will be held clear until a device reset.

Note 1: There is not a status bit for stack underflow. The STKAV bit can be used to detect the underflow which results in the stack pointer being at the top of stack.

Note 2: There are no instruction mnemonics called PUSH or POP. These are actions that occur from the execution of the CALL, RETURN, RETLW, and RETFIE instructions, or the vectoring to an interrupt vector.

Note 3: After a reset, if a "POP" operation occurs before a "PUSH" operation, the STKAV bit will be cleared. This will appear as if the stack is full (underflow has occurred). If a "PUSH" operation occurs next (before another "POP"), the STKAV bit will be locked clear. Only a device reset will cause this bit to set.

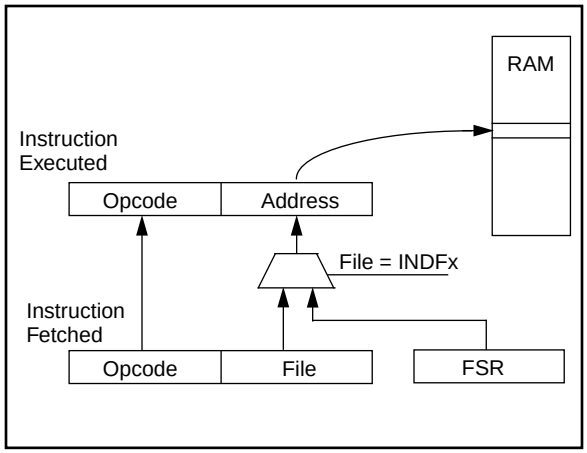
After the device is "PUSHed" sixteen times (without a "POP"), the seventeenth push overwrites the value from the first push. The eighteenth push overwrites the second push (and so on).

6.4 Indirect Addressing

Indirect addressing is a mode of addressing data memory where the data memory address in the instruction is not fixed. That is, the register that is to be read or written can be modified by the program. This can be useful for data tables in the data memory. Figure 6-10 shows the operation of indirect addressing. This shows the moving of the value to the data memory address specified by the value of the FSR register.

Example 6-1 shows the use of indirect addressing to clear RAM in a minimum number of instructions. A similar concept could be used to move a defined number of bytes (block) of data to the USART transmit register (TXREG). The starting address of the block of data to be transmitted could easily be modified by the program.

FIGURE 6-10: INDIRECT ADDRESSING



Example 8-4 shows the sequence to do an 16 x 16 signed multiply. Equation 8-2 shows the algorithm that used. The 32-bit result is stored in four registers RES3:RES0. To account for the sign bits of the arguments, each argument pairs most significant bit (MSb) is tested and the appropriate subtractions are done.

EQUATION 8-2: 16 x 16 SIGNED MULTIPLICATION ALGORITHM

RES3:RES0

= ARG1H:ARG1L * ARG2H:ARG2L

= (ARG1H * ARG2H * 2^{16}) +
(ARG1H * ARG2L * 2^{8}) +
(ARG1L * ARG2H * 2^{8}) +
(ARG1L * ARG2L) +
(-1 * ARG2H<7> * ARG1H:ARG1L * 2^{16}) +
(-1 * ARG1H<7> * ARG2H:ARG2L * 2^{16})

EXAMPLE 8-4: 16 x 16 SIGNED MULTIPLY ROUTINE

```
   MOVFP    ARG1L, WREG
   MULWF    ARG2L       ; ARG1L * ARG2L ->
                        ;   PRODH:PRODL
   MOVPF    PRODH, RES1 ;
   MOVPF    PRODL, RES0 ;
;
   MOVFP    ARG1H, WREG
   MULWF    ARG2H       ; ARG1H * ARG2H ->
                        ;   PRODH:PRODL
   MOVPF    PRODH, RES3 ;
   MOVPF    PRODL, RES2 ;
;
   MOVFP    ARG1L, WREG
   MULWF    ARG2H       ; ARG1L * ARG2H ->
                        ;   PRODH:PRODL
   MOVFP    PRODL, WREG ;
   ADDWF    RES1, F     ; Add cross
   MOVFP    PRODH, WREG ;    products
   ADDWFC   RES2, F     ;
   CLRF     WREG, F     ;
   ADDWFC   RES3, F     ;
;
   MOVFP    ARG1H, WREG ;
   MULWF    ARG2L       ; ARG1H * ARG2L ->
                        ;   PRODH:PRODL

   MOVFP    PRODL, WREG ;
   ADDWF    RES1, F     ; Add cross
   MOVFP    PRODH, WREG ;    products
   ADDWFC   RES2, F     ;
   CLRF     WREG, F     ;
   ADDWFC   RES3, F     ;
;
   BTFSS    ARG2H, 7    ; ARG2H:ARG2L neg?
   GOTO     SIGN_ARG1   ; no, check ARG1
   MOVFP    ARG1L, WREG ;
   SUBWF    RES2        ;
   MOVFP    ARG1H, WREG ;
   SUBWFB   RES3
;
SIGN_ARG1
   BTFSS    ARG1H, 7    ; ARG1H:ARG1L neg?
   GOTO     CONT_CODE   ; no, done
   MOVFP    ARG2L, WREG ;
   SUBWF    RES2        ;
   MOVFP    ARG2H, WREG ;
   SUBWFB   RES3
;
CONT_CODE
     :
```

12.1.3.3.1 MAX RESOLUTION/FREQUENCY FOR EXTERNAL CLOCK INPUT

The use of an external clock for the PWM time-base (Timer1 or Timer2) limits the PWM output to a maximum resolution of 8-bits. The PWxDCL<7:6> bits must be kept cleared. Use of any other value will distort the PWM output. All resolutions are supported when internal clock mode is selected. The maximum attainable frequency is also lower. This is a result of the timing requirements of an external clock input for a timer (see the Electrical Specification section). The maximum PWM frequency, when the timers clock source is the RB4/TCLK12 pin, is shown in Table 12-3 (standard resolution mode).

12.2 Timer3

Timer3 is a 16-bit timer consisting of the TMR3H and TMR3L registers. TMR3H is the high byte of the timer and TMR3L is the low byte. This timer has an associated 16-bit period register (PR3H/CA1H:PR3L/CA1L). This period register can be software configured to be a second 16-bit capture register.

When the TMR3CS bit (TCON1<2>) is clear, the timer increments every instruction cycle (Fosc/4). When TMR3CS is set, the timer increments on every falling edge of the RB5/TCLK3 pin. In either mode, the TMR3ON bit must be set for the timer to increment. When TMR3ON is clear, the timer will not increment or set the TMR3IF bit.

Timer3 has two modes of operation, depending on the CA1/$\overline{PR3}$ bit (TCON2<3>). These modes are:

- One capture and one period register mode
- Dual capture register mode

The PIC17C4X has up to two 16-bit capture registers that capture the 16-bit value of TMR3 when events are detected on capture pins. There are two capture pins (RB0/CAP1 and RB1/CAP2), one for each capture register. The capture pins are multiplexed with PORTB pins. An event can be:

- a rising edge
- a falling edge
- every 4th rising edge
- every 16th rising edge

Each 16-bit capture register has an interrupt flag associated with it. The flag is set when a capture is made. The capture module is truly part of the Timer3 block. Figure 12-7 and Figure 12-8 show the block diagrams for the two modes of operation.

TABLE 12-4: REGISTERS/BITS ASSOCIATED WITH PWM

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)
16h, Bank 3	TCON1	CA2ED1	CA2ED0	CA1ED1	CA1ED0	T16	TMR3CS	TMR2CS	TMR1CS	0000 0000	0000 0000
17h, Bank 3	TCON2	CA2OVF	CA1OVF	PWM2ON	PWM1ON	CA1/$\overline{PR3}$	TMR3ON	TMR2ON	TMR1ON	0000 0000	0000 0000
10h, Bank 2	TMR1	Timer1 register								xxxx xxxx	uuuu uuuu
11h, Bank 2	TMR2	Timer2 register								xxxx xxxx	uuuu uuuu
16h, Bank 1	PIR	RBIF	TMR3IF	TMR2IF	TMR1IF	CA2IF	CA1IF	TXIF	RCIF	0000 0010	0000 0010
17h, Bank 1	PIE	RBIE	TMR3IE	TMR2IE	TMR1IE	CA2IE	CA1IE	TXIE	RCIE	0000 0000	0000 0000
07h, Unbanked	INTSTA	PEIF	T0CKIF	T0IF	INTF	PEIE	T0CKIE	T0IE	INTE	0000 0000	0000 0000
06h, Unbanked	CPUSTA	—	—	STKAV	GLINTD	$\overline{TO}$	$\overline{PD}$	—	—	--11 11--	--11 qq--
10h, Bank 3	PW1DCL	DC1	DC0	—	—	—	—	—	—	xx-- ----	uu-- ----
11h, Bank 3	PW2DCL	DC1	DC0	TM2PW2	—	—	—	—	—	xx0- ----	uu0- ----
12h, Bank 3	PW1DCH	DC9	DC8	DC7	DC6	DC5	DC4	DC3	DC2	xxxx xxxx	uuuu uuuu
13h, Bank 3	PW2DCH	DC9	DC8	DC7	DC6	DC5	DC4	DC3	DC2	xxxx xxxx	uuuu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented read as '0', q = value depends on conditions, shaded cells are not used by PWM.

13.1 USART Baud Rate Generator (BRG)

The BRG supports both the Asynchronous and Synchronous modes of the USART. It is a dedicated 8-bit baud rate generator. The SPBRG register controls the period of a free running 8-bit timer. Table 13-1 shows the formula for computation of the baud rate for different USART modes. These only apply when the USART is in synchronous master mode (internal clock) and asynchronous mode.

Given the desired baud rate and Fosc, the nearest integer value between 0 and 255 can be calculated using the formula below. The error in baud rate can then be determined.

TABLE 13-1: BAUD RATE FORMULA

SYNC	Mode	Baud Rate
0	Asynchronous	FOSC/(64(X+1))
1	Synchronous	FOSC/(4(X+1))

X = value in SPBRG (0 to 255)

Example 13-1 shows the calculation of the baud rate error for the following conditions:

FOSC = 16 MHz
Desired Baud Rate = 9600
SYNC = 0

EXAMPLE 13-1: CALCULATING BAUD RATE ERROR

Desired Baud rate=Fosc / (64 (X + 1))

9600 = 16000000 /(64 (X + 1))

X = 25.042 = 25

Calculated Baud Rate=16000000 / (64 (25 + 1))

= 9615

$$\text{Error} = \frac{\text{(Calculated Baud Rate - Desired Baud Rate)}}{\text{Desired Baud Rate}}$$

= (9615 - 9600) / 9600

= 0.16%

Writing a new value to the SPBRG, causes the BRG timer to be reset (or cleared), this ensures that the BRG does not wait for a timer overflow before outputting the new baud rate.

TABLE 13-2: REGISTERS ASSOCIATED WITH BAUD RATE GENERATOR

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)
13h, Bank 0	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00u
15h, Bank 0	TXSTA	CSRC	TX9	TXEN	SYNC	—	—	TRMT	TX9D	0000 --1x	0000 --1u
17h, Bank 0	SPBRG	Baud rate generator register								xxxx xxxx	uuuu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented read as a '0', shaded cells are not used by the Baud Rate Generator.
Note 1: Other (non power-up) resets include: external reset through $\overline{\text{MCLR}}$ and Watchdog Timer Reset.

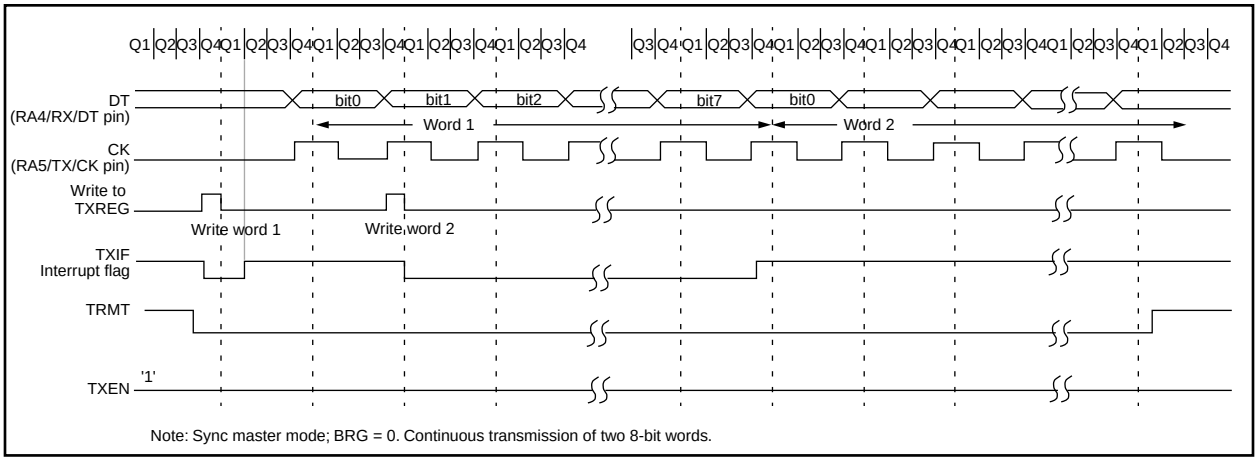
TABLE 13-7: REGISTERS ASSOCIATED WITH SYNCHRONOUS MASTER TRANSMISSION

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)
16h, Bank 1	PIR	RBIF	TMR3IF	TMR2IF	TMR1IF	CA2IF	CA1IF	TXIF	RCIF	0000 0010	0000 0010
13h, Bank 0	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00u
16h, Bank 0	TXREG	TX7	TX6	TX5	TX4	TX3	TX2	TX1	TX0	xxxx xxxx	uuuu uuuu
17h, Bank 1	PIE	RBIE	TMR3IE	TMR2IE	TMR1IE	CA2IE	CA1IE	TXIE	RCIE	0000 0000	0000 0000
15h, Bank 0	TXSTA	CSRC	TX9	TXEN	SYNC	—	—	TRMT	TX9D	0000 --1x	0000 --1u
17h, Bank 0	SPBRG	Baud rate generator register								xxxx xxxx	uuuu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented read as a '0', shaded cells are not used for synchronous master transmission.

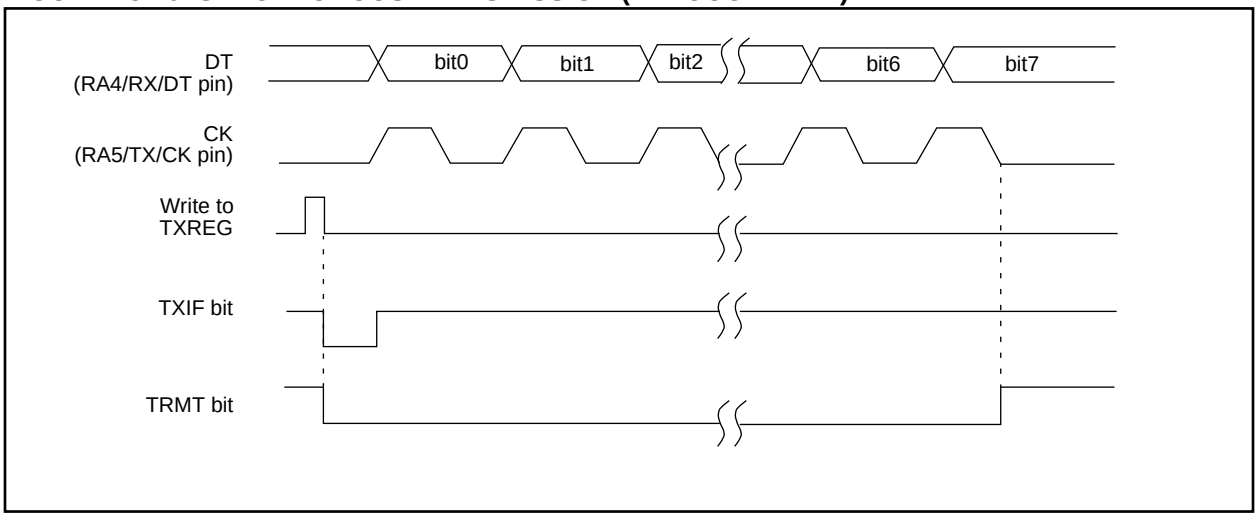
Note 1: Other (non power-up) resets include: external reset through $\overline{\text{MCLR}}$ and Watchdog Timer Reset.

FIGURE 13-9: SYNCHRONOUS TRANSMISSION



Note: Sync master mode; BRG = 0. Continuous transmission of two 8-bit words.

FIGURE 13-10: SYNCHRONOUS TRANSMISSION (THROUGH TXEN)



14.2.4 EXTERNAL CRYSTAL OSCILLATOR CIRCUIT

Either a prepackaged oscillator can be used or a simple oscillator circuit with TTL gates can be built. Prepackaged oscillators provide a wide operating range and better stability. A well-designed crystal oscillator will provide good performance with TTL gates. Two types of crystal oscillator circuits can be used: one with series resonance, or one with parallel resonance.

Figure 14-5 shows implementation of a parallel resonant oscillator circuit. The circuit is designed to use the fundamental frequency of the crystal. The 74AS04 inverter performs the 180-degree phase shift that a parallel oscillator requires. The 4.7 kΩ resistor provides the negative feedback for stability. The 10 kΩ potentiometer biases the 74AS04 in the linear region. This could be used for external oscillator designs.

FIGURE 14-5: EXTERNAL PARALLEL RESONANT CRYSTAL OSCILLATOR CIRCUIT

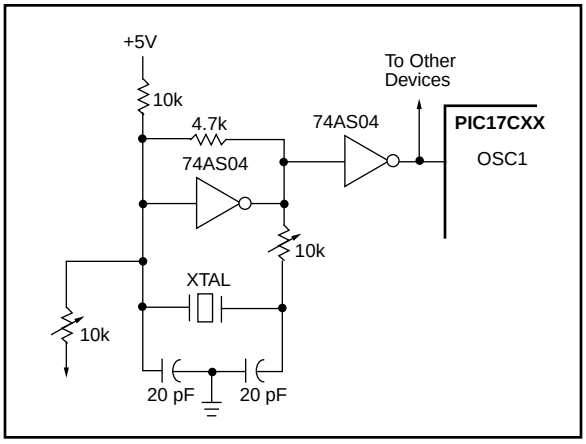
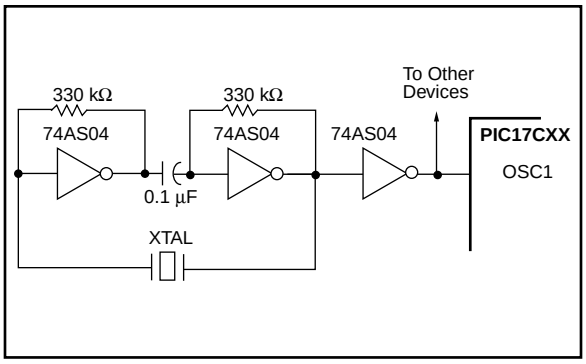


Figure 14-6 shows a series resonant oscillator circuit. This circuit is also designed to use the fundamental frequency of the crystal. The inverter performs a 180-degree phase shift in a series resonant oscillator circuit. The 330 kΩ resistors provide the negative feedback to bias the inverters in their linear region.

FIGURE 14-6: EXTERNAL SERIES RESONANT CRYSTAL OSCILLATOR CIRCUIT



14.2.5 RC OSCILLATOR

For timing insensitive applications, the RC device option offers additional cost savings. RC oscillator frequency is a function of the supply voltage, the resistor (Rext) and capacitor (Cext) values, and the operating temperature. In addition to this, oscillator frequency will vary from unit to unit due to normal process parameter variation. Furthermore, the difference in lead frame capacitance between package types will also affect oscillation frequency, especially for low Cext values. The user also needs to take into account variation due to tolerance of external R and C components used. Figure 14-6 shows how the R/C combination is connected to the PIC17CXX. For Rext values below 2.2 kΩ, the oscillator operation may become unstable, or stop completely. For very high Rext values (e.g. 1 MΩ), the oscillator becomes sensitive to noise, humidity and leakage. Thus, we recommend to keep Rext between 3 kΩ and 100 kΩ.

Although the oscillator will operate with no external capacitor (Cext = 0 pF), we recommend using values above 20 pF for noise and stability reasons. With little or no external capacitance, oscillation frequency can vary dramatically due to changes in external capacitances, such as PCB trace capacitance or package lead frame capacitance.

See Section 18.0 for RC frequency variation from part to part due to normal process variation. The variation is larger for larger R (since leakage current variation will affect RC frequency more for large R) and for smaller C (since variation of input capacitance will affect RC frequency more).

See Section 18.0 for variation of oscillator frequency due to VDD for given Rext/Cext values as well as frequency variation due to operating temperature for given R, C, and VDD values.

The oscillator frequency, divided by 4, is available on the OSC2/CLKOUT pin, and can be used for test purposes or to synchronize other logic (see Figure 3-2 for waveform).

FIGURE 14-7: RC OSCILLATOR MODE

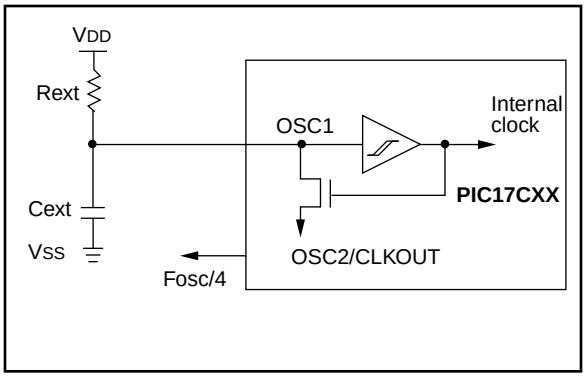


Table 15-2 lists the instructions recognized by the MPASM assembler.

Note 1: Any unused opcode is Reserved. Use of any reserved opcode may cause unexpected operation.

Note 2: The shaded instructions are not available in the PIC17C42

All instruction examples use the following format to represent a hexadecimal number:

0xhh

where h signifies a hexadecimal digit.

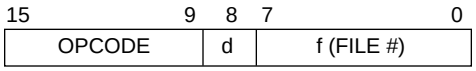
To represent a binary number:

`0000 0100b`

where b signifies a binary string.

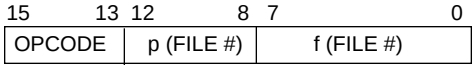
FIGURE 15-1: GENERAL FORMAT FOR INSTRUCTIONS

Byte-oriented file register operations



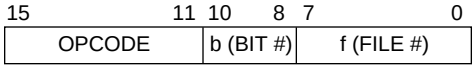
d = 0 for destination WREG
d = 1 for destination f
f = 8-bit file register address

Byte to Byte move operations



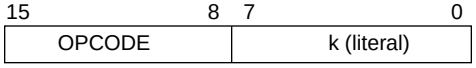
p = peripheral register file address
f = 8-bit file register address

Bit-oriented file register operations



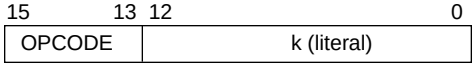
b = 3-bit address
f = 8-bit file register address

Literal and control operations



k = 8-bit immediate value

Call and GOTO operations



k = 13-bit immediate value

15.1 Special Function Registers as Source/Destination

The PIC17C4X's orthogonal instruction set allows read and write of all file registers, including special function registers. There are some special situations the user should be aware of:

15.1.1 ALUSTA AS DESTINATION

If an instruction writes to ALUSTA, the Z, C, DC and OV bits may be set or cleared as a result of the instruction and overwrite the original data bits written. For example, executing `CLRF ALUSTA` will clear register ALUSTA, and then set the Z bit leaving `0000 0100b` in the register.

15.1.2 PCL AS SOURCE OR DESTINATION

Read, write or read-modify-write on PCL may have the following results:

Read PC:	PCH → PCLATH; PCL → dest
Write PCL:	PCLATH → PCH; 8-bit destination value → PCL
Read-Modify-Write:	PCL→ ALU operand PCLATH → PCH; 8-bit result → PCL

Where PCH = program counter high byte (not an addressable register), PCLATH = Program counter high holding latch, dest = destination, WREG or f.

15.1.3 BIT MANIPULATION

All bit manipulation instructions are done by first reading the entire register, operating on the selected bit and writing the result back (read-modify-write). The user should keep this in mind when operating on special function registers, such as ports.

15.2 Q Cycle Activity

Each instruction cycle (Tcy) is comprised of four Q cycles (Q1-Q4). The Q cycles provide the timing/designation for the Decode, Read, Execute, Write etc., of each instruction cycle. The following diagram shows the relationship of the Q cycles to the instruction cycle.

The 4 Q cycles that make up an instruction cycle (Tcy) can be generalized as:

Q1: Instruction Decode Cycle or forced NOP

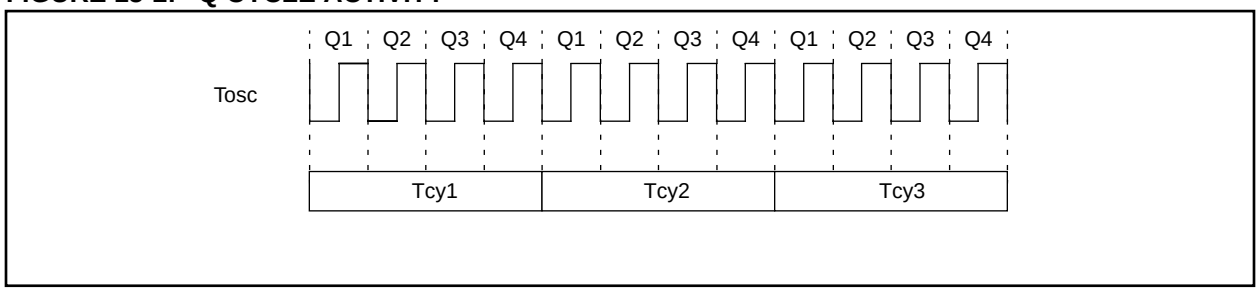
Q2: Instruction Read Cycle or NOP

Q3: Instruction Execute

Q4: Instruction Write Cycle or NOP

Each instruction will show the detailed Q cycle operation for the instruction.

FIGURE 15-2: Q CYCLE ACTIVITY



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

Syntax: [*label*] ADDWFC f,d

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

Operation: (WREG) + (f) + C → (dest)

Status Affected: OV, C, DC, Z

Encoding:

0001	000d	ffff	ffff

Description: Add WREG, the Carry Flag and data memory location 'f'. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed in data memory location 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `ADDWFC REG 0`

Before Instruction
Carry bit = 1
REG = 0x02
WREG = 0x4D

After Instruction
Carry bit = 0
REG = 0x02
WREG = 0x50

ANDLW **And Literal with WREG**

Syntax: [*label*] ANDLW k

Operands: $0 \le k \le 255$

Operation: (WREG) .AND. (k) → (WREG)

Status Affected: Z

Encoding:

1011	0101	kkkk	kkkk

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

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Execute	Write to WREG

Example: `ANDLW 0x5F`

Before Instruction
WREG = 0xA3

After Instruction
WREG = 0x03

MOVFP **Move f to p**

Syntax: [*label*] MOVFP f,p

Operands: $0 \le f \le 255$
$0 \le p \le 31$

Operation: (f) → (p)

Status Affected: None

Encoding:

011p	pppp	ffff	ffff

Description: Move data from data memory location 'f' to data memory location 'p'. Location 'f' can be anywhere in the 256 word data space (00h to FFh) while 'p' can be 00h to 1Fh.

Either 'p' or 'f' can be WREG (a useful special situation).

MOVFP is particularly useful for transferring a data memory location to a peripheral register (such as the transmit buffer or an I/O port). Both 'f' and 'p' can be indirectly addressed.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: MOVFP REG1, REG2

Before Instruction
REG1 = 0x33,
REG2 = 0x11

After Instruction
REG1 = 0x33,
REG2 = 0x33

MOVLB **Move Literal to low nibble in BSR**

Syntax: [*label*] MOVLB k

Operands: $0 \le k \le 15$

Operation: k → (BSR<3:0>)

Status Affected: None

Encoding:

1011	1000	uuuu	kkkk

Description: The four bit literal 'k' is loaded in the Bank Select Register (BSR). Only the low 4-bits of the Bank Select Register are affected. The upper half of the BSR is unchanged. The assembler will encode the “u” fields as '0'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'u:k'	Execute	Write literal 'k' to BSR<3:0>

Example: MOVLB 0x5

Before Instruction
BSR register = 0x22

After Instruction
BSR register = 0x25

Note: For the PIC17C42, only the low four bits of the BSR register are physically implemented. The upper nibble is read as '0'.

SUBWF **Subtract WREG from f**

Syntax: [*label*] SUBWF f,d

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

Operation: (f) – (W) → (dest)

Status Affected: OV, C, DC, Z

Encoding:

0000	010d	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'.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1:
```
SUBWF   REG1, 1
```
Before Instruction
```
REG1    =   3
WREG    =   2
C       =   ?
```
After Instruction
```
REG1    =   1
WREG    =   2
C       =   1     ; result is positive
Z       =   0
```
Example 2:

Before Instruction
```
REG1    =   2
WREG    =   2
C       =   ?
```
After Instruction
```
REG1    =   0
WREG    =   2
C       =   1     ; result is zero
Z       =   1
```
Example 3:

Before Instruction
```
REG1    =   1
WREG    =   2
C       =   ?
```
After Instruction
```
REG1    =   FF
WREG    =   2
C       =   0     ; result is negative
Z       =   0
```

SUBWFB **Subtract WREG from f with Borrow**

Syntax: [*label*] SUBWFB f,d

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

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

Status Affected: OV, C, DC, Z

Encoding:

0000	001d	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'.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1:
```
SUBWFB  REG1, 1
```
Before Instruction
```
REG1    =   0x19    (0001 1001)
WREG    =   0x0D    (0000 1101)
C       =   1
```
After Instruction
```
REG1    =   0x0C    (0000 1011)
WREG    =   0x0D    (0000 1101)
C       =   1       ; result is positive
Z       =   0
```
Example2:
```
SUBWFB  REG1,0
```
Before Instruction
```
REG1    =   0x1B    (0001 1011)
WREG    =   0x1A    (0001 1010)
C       =   0
```
After Instruction
```
REG1    =   0x1B    (0001 1011)
WREG    =   0x00
C       =   1       ; result is zero
Z       =   1
```
Example3:
```
SUBWFB  REG1,1
```
Before Instruction
```
REG1    =   0x03    (0000 0011)
WREG    =   0x0E    (0000 1101)
C       =   1
```
After Instruction
```
REG1    =   0xF5    (1111 0100) [2's comp]
WREG    =   0x0E    (0000 1101)
C       =   0       ; result is negative
Z       =   0
```

TABLRD **Table Read**

Example1: `TABLRD 1, 1, REG ;`

Before Instruction

REG	=	0x53
TBLATH	=	0xAA
TBLATL	=	0x55
TBLPTR	=	0xA356
MEMORY(TBLPTR)	=	0x1234

After Instruction (table write completion)

REG	=	0xAA
TBLATH	=	0x12
TBLATL	=	0x34
TBLPTR	=	0xA357
MEMORY(TBLPTR)	=	0x5678

Example2: `TABLRD 0, 0, REG ;`

Before Instruction

REG	=	0x53
TBLATH	=	0xAA
TBLATL	=	0x55
TBLPTR	=	0xA356
MEMORY(TBLPTR)	=	0x1234

After Instruction (table write completion)

REG	=	0x55
TBLATH	=	0x12
TBLATL	=	0x34
TBLPTR	=	0xA356
MEMORY(TBLPTR)	=	0x1234

TABLWT **Table Write**

Syntax: [*label*] TABLWT t,i,f

Operands: $0 \le f \le 255$
$i \in [0,1]$
$t \in [0,1]$

Operation: If t = 0,
f → TBLATL;
If t = 1,
f → TBLATH;
TBLAT → Prog Mem (TBLPTR);
If i = 1,
TBLPTR + 1 → TBLPTR

Status Affected: None

Encoding:

1010	11ti	ffff	ffff

Description:

1. Load value in 'f' into 16-bit table latch (TBLAT)
   If t = 0: load into low byte;
   If t = 1: load into high byte
2. The contents of TBLAT is written to the program memory location pointed to by TBLPTR
   If TBLPTR points to external program memory location, then the instruction takes two-cycle
   If TBLPTR points to an internal EPROM location, then the instruction is terminated when an interrupt is received.

Note: The $\overline{MCLR}/V_{PP}$ pin must be at the programming voltage for successful programming of internal memory.
If $\overline{MCLR}/V_{PP} = V_{DD}$
the programming sequence of internal memory will be executed, but will not be successful (although the internal memory location may be disturbed)

3. The TBLPTR can be automatically incremented
   If i = 0; TBLPTR is not incremented
   If i = 1; TBLPTR is incremented

Words: 1

Cycles: 2 (many if write is to on-chip EPROM program memory)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Execute	Write register TBLATH or TBLATL

MPASM allow full symbolic debugging from the Microchip Universal Emulator System (PICMASTER).

MPASM has the following features to assist in developing software for specific use applications.

- Provides translation of Assembler source code to object code for all Microchip microcontrollers.
- Macro assembly capability.
- Produces all the files (Object, Listing, Symbol, and special) required for symbolic debug with Microchip's emulator systems.
- Supports Hex (default), Decimal and Octal source and listing formats.

MPASM provides a rich directive language to support programming of the PIC16/17. Directives are helpful in making the development of your assemble source code shorter and more maintainable.

16.11 Software Simulator (MPLAB-SIM)

The MPLAB-SIM Software Simulator allows code development in a PC host environment. It allows the user to simulate the PIC16/17 series microcontrollers on an instruction level. On any given instruction, the user may examine or modify any of the data areas or provide external stimulus to any of the pins. The input/output radix can be set by the user and the execution can be performed in; single step, execute until break, or in a trace mode.

MPLAB-SIM fully supports symbolic debugging using MPLAB-C and MPASM. The Software Simulator offers the low cost flexibility to develop and debug code outside of the laboratory environment making it an excellent multi-project software development tool.

16.12 C Compiler (MPLAB-C)

The MPLAB-C Code Development System is a complete 'C' compiler and integrated development environment for Microchip's PIC16/17 family of microcontrollers. The compiler provides powerful integration capabilities and ease of use not found with other compilers.

For easier source level debugging, the compiler provides symbol information that is compatible with the MPLAB IDE memory display (PICMASTER emulator software versions 1.13 and later).

16.13 Fuzzy Logic Development System (*fuzzy*TECH-MP)

*fuzzy*TECH-MP fuzzy logic development tool is available in two versions - a low cost introductory version, MP Explorer, for designers to gain a comprehensive working knowledge of fuzzy logic system design; and a full-featured version, *fuzzy*TECH-MP, edition for implementing more complex systems.

Both versions include Microchip's *fuzzy*LAB™ demonstration board for hands-on experience with fuzzy logic systems implementation.

16.14 MP-DriveWay™ – Application Code Generator

MP-DriveWay is an easy-to-use Windows-based Application Code Generator. With MP-DriveWay you can visually configure all the peripherals in a PIC16/17 device and, with a click of the mouse, generate all the initialization and many functional code modules in C language. The output is fully compatible with Microchip's MPLAB-C C compiler. The code produced is highly modular and allows easy integration of your own code. MP-DriveWay is intelligent enough to maintain your code through subsequent code generation.

16.15 SEEVAL® Evaluation and Programming System

The SEEVAL SEEPROM Designer's Kit supports all Microchip 2-wire and 3-wire Serial EEPROMs. The kit includes everything necessary to read, write, erase or program special features of any Microchip SEEPROM product including Smart Serials™ and secure serials. The Total Endurance™ Disk is included to aid in trade-off analysis and reliability calculations. The total kit can significantly reduce time-to-market and result in an optimized system.

16.16 TrueGauge® Intelligent Battery Management

The TrueGauge development tool supports system development with the MTA11200B TrueGauge Intelligent Battery Management IC. System design verification can be accomplished before hardware prototypes are built. User interface is graphically-oriented and measured data can be saved in a file for exporting to Microsoft Excel.

16.17 KEELOQ® Evaluation and Programming Tools

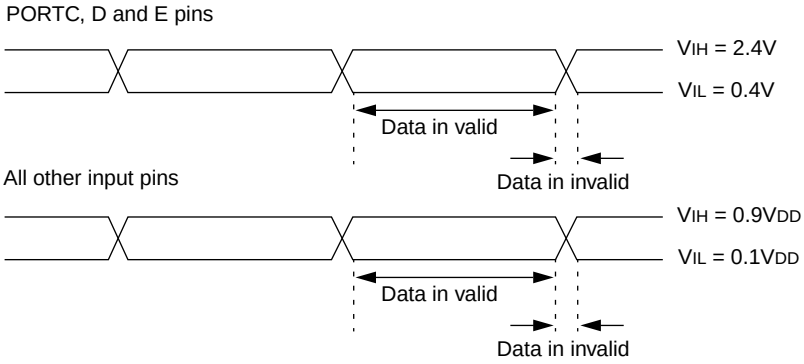
KEELOQ evaluation and programming tools support Microchips HCS Secure Data Products. The HCS evaluation kit includes an LCD display to show changing codes, a decoder to decode transmissions, and a programming interface to program test transmitters.

Applicable Devices	42	R42	42A	43	R43	44

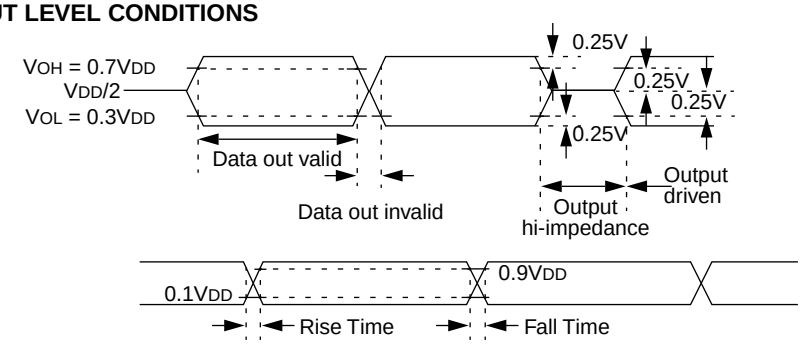
FIGURE 17-1: PARAMETER MEASUREMENT INFORMATION

All timings are measure between high and low measurement points as indicated in the figures below.

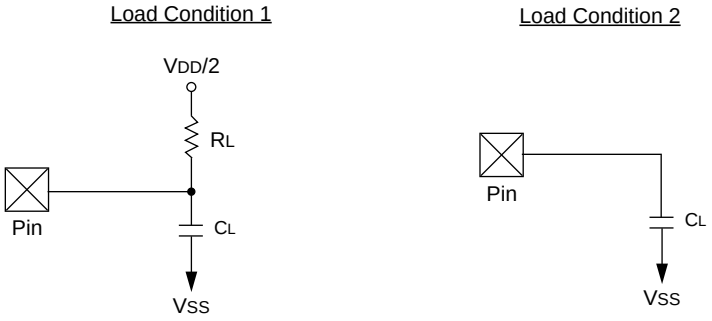
INPUT LEVEL CONDITIONS



OUTPUT LEVEL CONDITIONS



LOAD CONDITIONS



R_L = 464
$C_L \leq$ 50 pF

Applicable Devices	42	R42	42A	43	R43	44

FIGURE 19-9: USART MODULE: SYNCHRONOUS TRANSMISSION (MASTER/SLAVE) TIMING

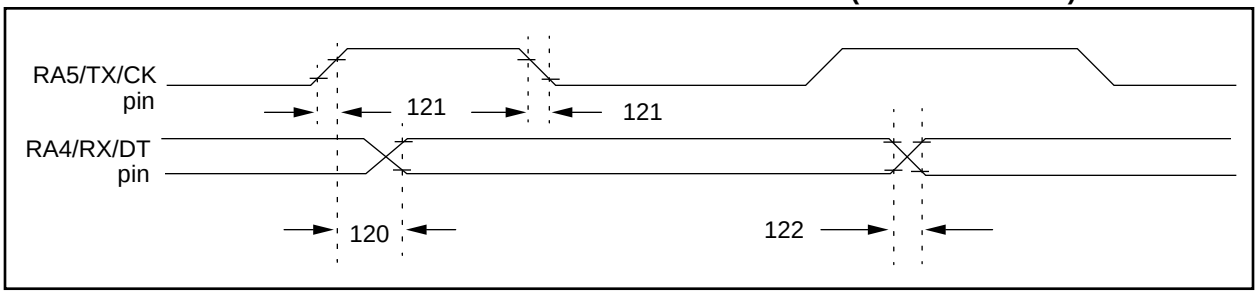


TABLE 19-9: SYNCHRONOUS TRANSMISSION REQUIREMENTS

Param No.	Sym	Characteristic		Min	Typ†	Max	Units	Conditions
120	TckH2dtV	SYNC XMIT (MASTER & SLAVE) Clock high to data out valid	PIC17CR42/42A/43/R43/44	—	—	50	ns	
			PIC17LCR42/42A/43/R43/44	—	—	75	ns	
121	TckRF	Clock out rise time and fall time (Master Mode)	PIC17CR42/42A/43/R43/44	—	—	25	ns	
			PIC17LCR42/42A/43/R43/44	—	—	40	ns	
122	TdtRF	Data out rise time and fall time	PIC17CR42/42A/43/R43/44	—	—	25	ns	
			PIC17LCR42/42A/43/R43/44	—	—	40	ns	

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

FIGURE 19-10: USART MODULE: SYNCHRONOUS RECEIVE (MASTER/SLAVE) TIMING

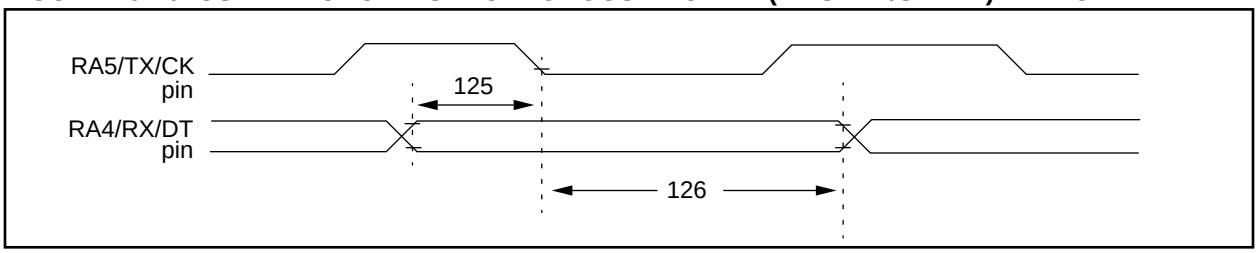
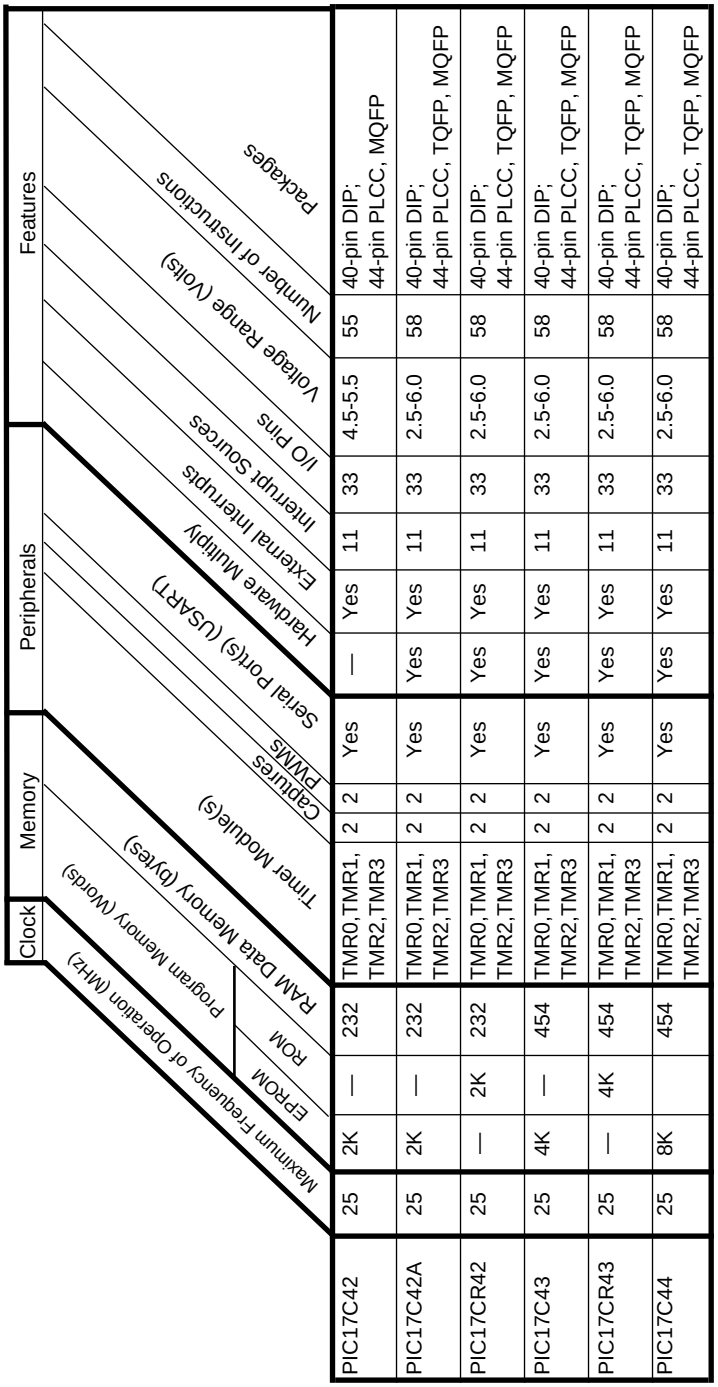


TABLE 19-10: SYNCHRONOUS RECEIVE REQUIREMENTS

Parameter No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions
125	TdtV2ckL	SYNC RCV (MASTER & SLAVE) Data hold before CK↓ (DT hold time)	15	—	—	ns	
126	TckL2dtl	Data hold after CK↓ (DT hold time)	15	—	—	ns	

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

E.8 PIC17CXX Family of Devices



	Clock	Memory					Peripherals					Features		
	Maximum Frequency of Operation (MHz)	EPROM	ROM	RAM Data Memory (bytes)	Program Memory (Words)	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
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, TQFP, MQFP

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

N

O

P

R