PIC18F010/020
Data Sheet

High Performance Microcontrollers

Trademarks

DNV Certification, Inc. USA

ANSI-RAB QMS ACCREDITED

DNV

DNV MSC The Netherlands Accredited by the RvA

ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.

PIC18F010/020

High Performance Microcontrollers

High Performance RISC CPU:

- C compiler optimized instruction set
- Linear program memory addressing
  - 4096 x 8 on-chip FLASH program memory
  - 2048 x 8 on-chip FLASH program memory (PIC18F010)
- Linear data memory addressing
  - 256 x 8 general purpose registers
  - 64 x 8 EEPROM
- Operating speed:
  - DC - 40MHz clock input
  - DC - 100 ns instruction cycle
  - Internal oscillator with 5 program selectable speeds (32kHz, 500kHz, 1MHz, 4MHz, 8MHz)
- 2.0V operation (4MHz)
- 16-bit wide instructions
- 8-bit wide data path
- 31 levels of hardware stack
- Software stack capability
- Multi-vector interrupt capability
- 8 x 8 multiply single cycle hardware

Special Microcontroller Features:

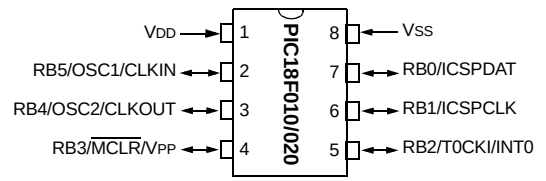
- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Brown-out Reset (BOR)
- Programmable Low Voltage Detection circuitry (PLVD)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving SLEEP mode with Wake-up on Pin Change
- In-Circuit Serial Programming (ICSP™) via two pins
- Low cost MPLAB® ICD available

Peripheral Features:

- High current sink/source 25mA/25mA
- Timer0: 8-bit/16-bit timer/counter with 8-bit programmable prescaler

Pinout Diagram:

PDIP, SOIC



CMOS Technology:

- Low power, high speed CMOS FLASH technology
- Fully static design
- Wide operating voltage range (2.0V to 5.5V)
- Commercial, Industrial and Extended temperature ranges
- Low power consumption

Table of Contents

TO OUR VALUED CUSTOMERS

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your Microchip products. To this end, we will continue to improve our publications to better suit your needs. Our publications will be refined and enhanced as new volumes and updates are introduced.

If you have any questions or comments regarding this publication, please contact the Marketing Communications Department via E-mail at **docerrors@mail.microchip.com** or fax the **Reader Response Form** in the back of this data sheet to (480) 792-4150. We welcome your feedback.

Most Current Data Sheet

To obtain the most up-to-date version of this data sheet, please register at our Worldwide Web site at:

http://www.microchip.com

You can determine the version of a data sheet by examining its literature number found on the bottom outside corner of any page. The last character of the literature number is the version number, (e.g., DS30000A is version A of document DS30000).

Errata

An errata sheet, describing minor operational differences from the data sheet and recommended workarounds, may exist for current devices. As device/documentation issues become known to us, we will publish an errata sheet. The errata will specify the revision of silicon and revision of document to which it applies.

To determine if an errata sheet exists for a particular device, please check with one of the following:

- Microchip's Worldwide Web site; http://www.microchip.com
- Your local Microchip sales office (see last page)
- The Microchip Corporate Literature Center; U.S. FAX: (480) 792-7277

When contacting a sales office or the literature center, please specify which device, revision of silicon and data sheet (include literature number) you are using.

Customer Notification System

Register on our web site at **www.microchip.com/cn** to receive the most current information on all of our products.

1.0 DEVICE OVERVIEW

This document contains device specific information for the PIC18F010/020 microcontrollers. These devices come in 8-pin packages. Table 1-1 is an overview of the features. Figure 1-1 presents the block diagram for the PIC18F010/020 devices and Table 1-2 gives the pin descriptions.

TABLE 1-1: DEVICE FEATURES

Features	PIC18F010	PIC18F020
Operating Frequency	DC - 40 MHz	DC - 40 MHz
Program Memory (Bytes)	2K	4K
Program Memory (Instructions)	1024	2048
Data Memory (SRAM)	256	256
Data Memory (EEPROM)	64	64
Interrupt Sources	5	5
I/O Ports	PORTB (6-bit)	PORTB (6-bit)
Timers	1 (8/16-bit)	1 (8/16-bit)
RESETS (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)
Programmable Low Voltage Detect	Yes	Yes
Programmable Brown-out Reset	Yes	Yes
Instruction Set	75	75
Packages	8-pin PDIP 8-pin SOIC	8-pin PDIP 8-pin SOIC

FIGURE 1-1: PIC18F010/020 BLOCK DIAGRAM

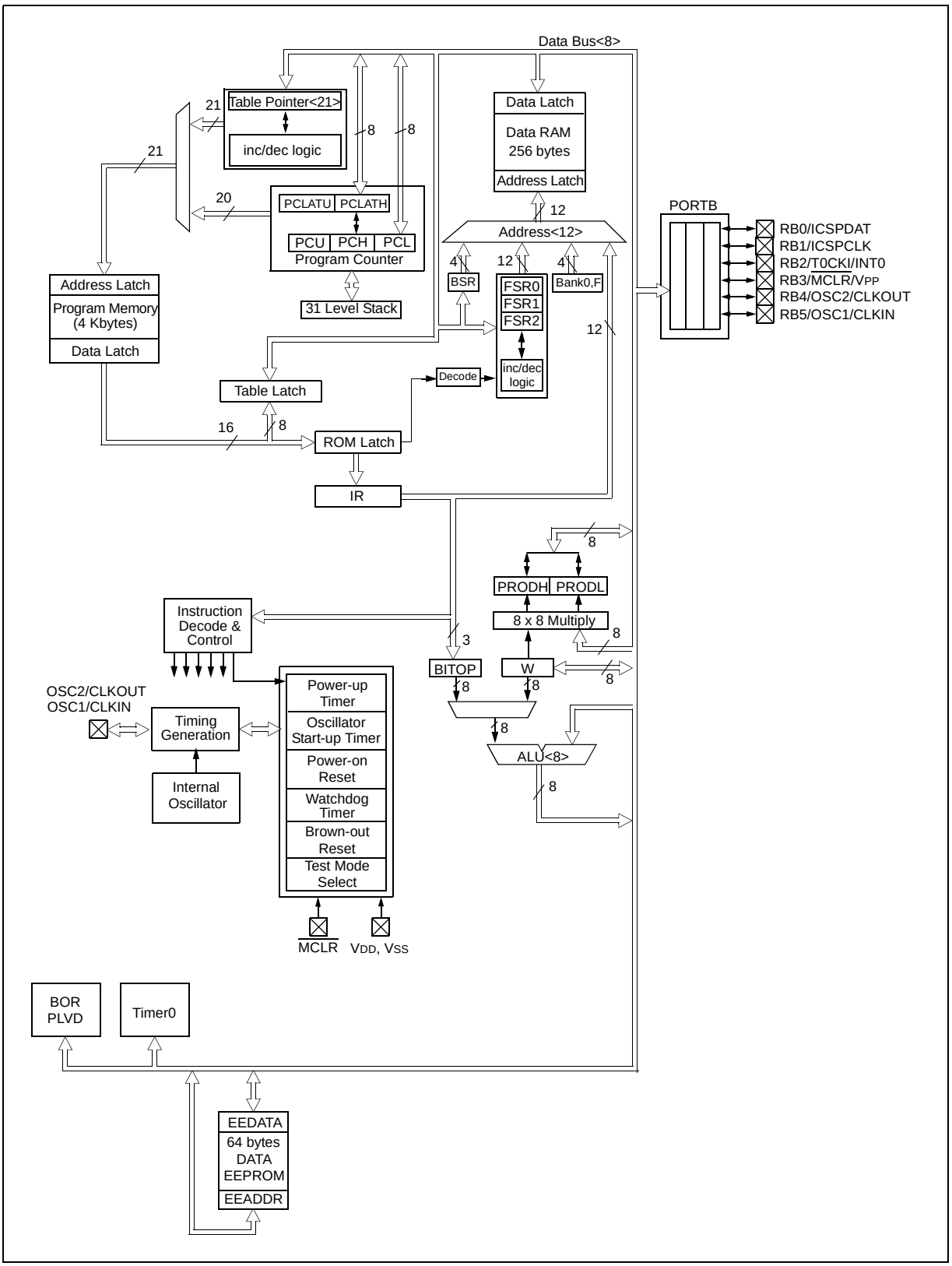


TABLE 1-2: PIC18F010/020 PRODUCT PINOUT OVERVIEW

Bondpad Name	Devices		Function/Description
	8-Pin PDIP	8-Pin SOIC	
V_{DD}	1	1	Power
V_{SS}	8	8	Ground
RB5/OSC1/CLKIN	2	2	Bi-directional I/O pin (TTL) with optional interrupt-on-change, clock input, or oscillator input
RB4/OSC2/CLKOUT	3	3	Bi-directional I/O pin (TTL) with optional interrupt-on-change, oscillator output, or CLKOUT output
RB3/$\overline{MCLR}$/V_{PP}	4	4	Bi-directional I/O pin (TTL), open drain, with optional interrupt-on-change, or Master Clear External Reset input (ST)
RB2/T0CKI/INT0	5	5	Bi-directional I/O pin (TTL) with optional interrupt-on-change, TMR0 clock input (ST), or interrupt input (ST)
RB1	6	6	Bi-directional I/O pin (TTL) with optional interrupt-on-change
RB0	7	7	Bi-directional I/O pin (TTL) with optional interrupt-on-change

NOTES:

2.0 OSCILLATOR CONFIGURATIONS

2.1 Oscillator Types

The PIC18F010/020 can be operated in eight different oscillator modes. Programming these modes is done via the CONFIG1H register (FOSC2, FOSC1, and FOSC0).

1. LP Low Power Crystal
2. XT Crystal/Resonator
3. HS High Speed Crystal/Resonator
4. EC External Clock
5. RC External Resistor/Capacitor
6. RCIO External Resistor/Capacitor with I/O pin enabled
7. INTOSC Precision Internal Oscillator
8. INTOSCIO Precision Internal Oscillator with I/O pin enabled

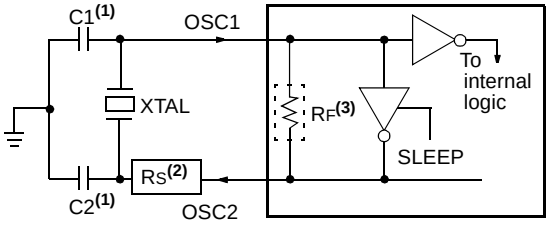
2.2 Crystal Oscillator/Ceramic Resonators

In XT, LP, or HS oscillator modes, a crystal or ceramic resonator is connected to the RB5/OSC1 and RB4/OSC2 pins to establish oscillation. Figure 2-1 shows the pin connections. An external clock source may also be connected to the OSC1 pin in these modes, as shown in Figure 2-2.

The PIC18F010/020 oscillator design requires the use of a parallel cut crystal.

Note:	Use of a series cut crystal may give a frequency out of the crystal manufacturers specifications.

FIGURE 2-1: CRYSTAL/CERAMIC RESONATOR OPERATION (HS, XT OR LP OSC CONFIGURATION)



Note 1: See Table 2-1 and Table 2-2 for recommended values of C1 and C2.

2: A series resistor (RS) may be required for AT strip cut crystals.

3: RF varies with the crystal chosen.

FIGURE 2-2: EXTERNAL CLOCK INPUT OPERATION (HS, XT OR LP OSC CONFIGURATION)

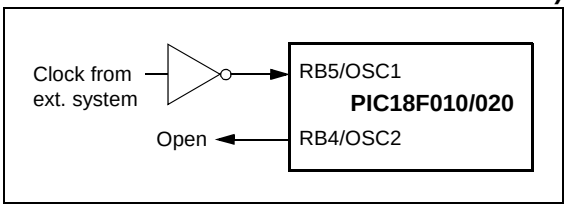


TABLE 2-1: CERAMIC RESONATORS

Ranges Tested:			
Mode	**Freq.**	**OSC1**	**OSC2**
XT	455 kHz	68 - 100 pF	68 - 100 pF
	2.0 MHz	15 - 68 pF	15 - 68 pF
	4.0 MHz	15 - 68 pF	15 - 68 pF
HS	8.0 MHz	10 - 68 pF	10 - 68 pF
	16.0 MHz	10 - 22 pF	10 - 22 pF
These values are for design guidance only. See notes at bottom of page.			
Resonators Used:			
455 kHz	Panasonic EFO-A455K04B	± 0.3%	
2.0 MHz	Murata Erie CSA2.00MG	± 0.5%	
4.0 MHz	Murata Erie CSA4.00MG	± 0.5%	
8.0 MHz	Murata Erie CSA8.00MT	± 0.5%	
16.0 MHz	Murata Erie CSA16.00MX	± 0.5%	
All resonators used did not have built-in capacitors.			

TABLE 2-2: CAPACITOR SELECTION FOR CRYSTAL OSCILLATOR

Osc Type	Crystal Freq.	Cap. Range C1	Cap. Range C2
LP	32.0 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	47-68 pF	47-68 pF
	1.0 MHz	15 pF	15 pF
	4.0 MHz	15 pF	15 pF
HS	4.0 MHz	15 pF	15 pF
	8.0 MHz	15-33 pF	15-33 pF
	20.0 MHz	15-33 pF	15-33 pF
	25.0 MHz	TBD	TBD
These values are for design guidance only. See notes at bottom of page.			
Crystals Used			
32.0 kHz	Epson C-001R32.768K-A	± 20 PPM	
200 kHz	STD XTL 200.000KHz	± 20 PPM	
1.0 MHz	ECS ECS-10-13-1	± 50 PPM	
4.0 MHz	ECS ECS-40-20-1	± 50 PPM	
8.0 MHz	EPSON CA-301 8.000M-C	± 30 PPM	
20.0 MHz	EPSON CA-301 20.000M-C	± 30 PPM	

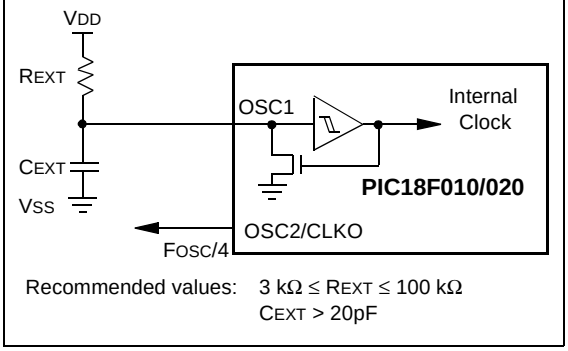
Note 1: Recommended values of C1 and C2 are identical to the ranges tested (Table 2-1).
2: Higher capacitance increases the stability of the oscillator, but also increases the start-up time.
3: Since each resonator/crystal has its own characteristics, the user should consult the resonator/crystal manufacturer for appropriate values of external components.
4: Rs may be required in HS mode, as well as XT mode, to avoid overdriving crystals with low drive level specification.

2.3 RC Oscillator

For applications where precise timing is not a requirement, the RC and RCIO oscillator options are available. The operation and functionality of the RC oscillator is dependent on a number of variables. The RC oscillator is a function of the supply voltage, the resistor (REXT) and capacitor (CEXT) values, and the operating temperature. The oscillator frequency will vary from unit to unit due to normal process parameter variation. Plus, the difference in lead frame capacitance between package types will also affect the oscillation frequency, especially for low CEXT values. The user also needs to account for the tolerance of the external R and C components. Figure 2-3 shows how the R/C combination is connected.

Note: The RC oscillator is not recommended for applications that require precise timing.

FIGURE 2-3: RC OSCILLATOR MODE



In the RC mode, the oscillator frequency divided by 4 is available on the OSC2 pin. This signal may be used for test purposes, or to synchronize other logic. In the RCIO mode, the OSC2 pin becomes a general purpose I/O pin. This pin is RB4 of PORTB.

2.4 The Internal Oscillator

The INTOSC and INTOSCIO device options are available to minimize part count and cost, while maximizing the number of I/O pins. There are five different frequencies of which the user has the option to select. They are 32 kHz, 500 kHz, 1 MHz, 4 MHz, and 8 MHz. The 1 MHz, 4 MHz, and 8 MHz internal clock selections are all derived from one 8 MHz clock source, and the other two are produced independently. Tuning is available for the 1 MHz, 4 MHz, and 8 MHz options; refer to Section 2.10.

2.5 External Clock Input

The EC oscillator mode requires an external clock source to be connected to the OSC1 pin. The feedback device between OSC1 and OSC2 is turned off in this mode to save current. There is no oscillator start-up time required after a Power-on Reset or after a recovery from SLEEP mode.

In the EC oscillator mode, the oscillator frequency divided by 4 is available on the OSC2 pin. This signal may be used for test purposes or to synchronize other logic. Figure 2-4 shows the pin connections for the EC oscillator mode.

FIGURE 2-4: EXTERNAL CLOCK INPUT OPERATION (EC OSC CONFIGURATION)

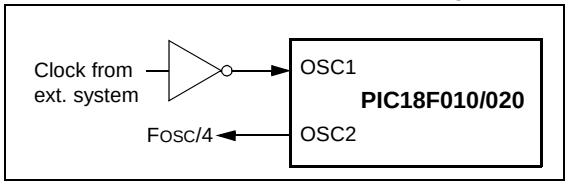
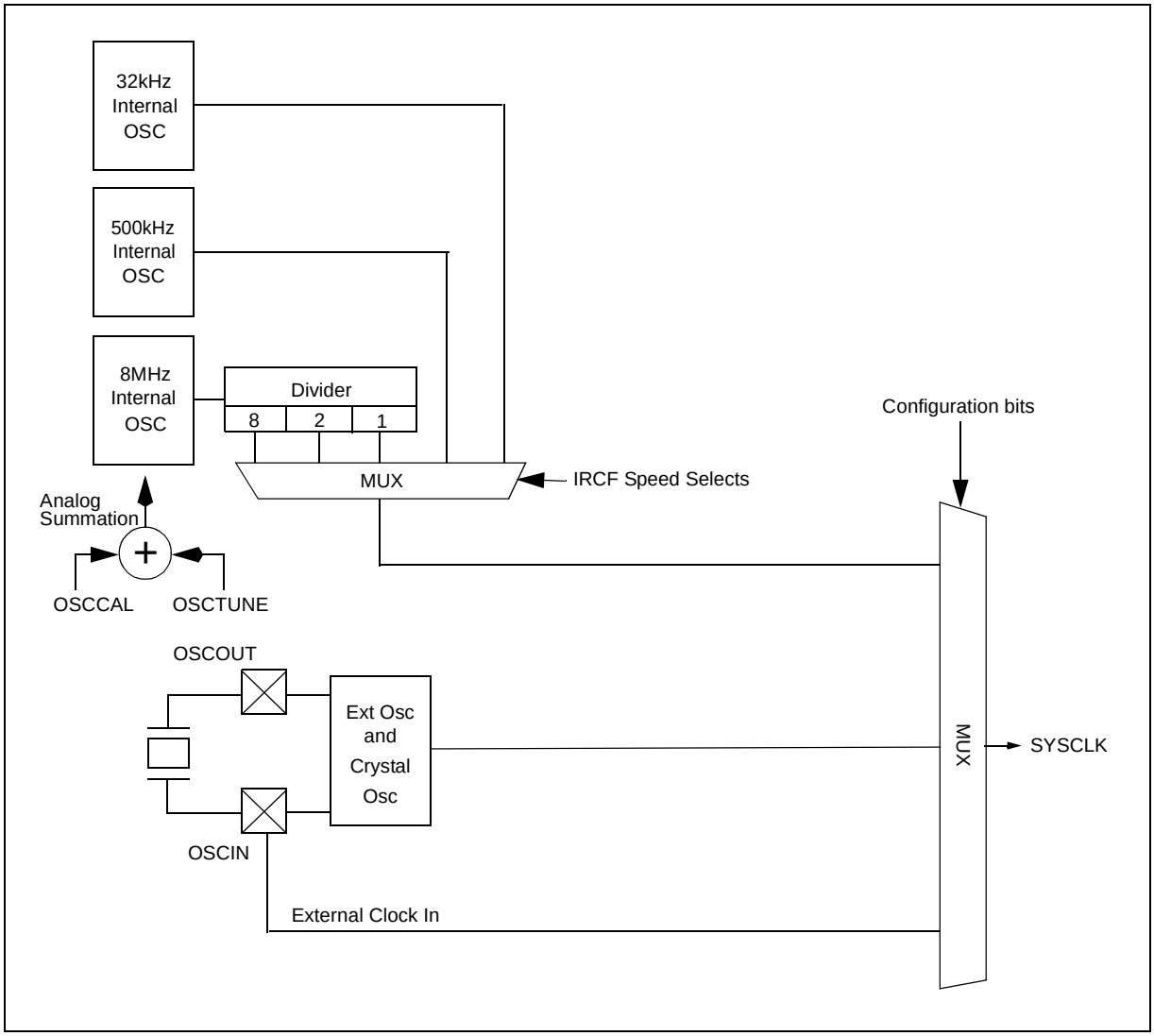


FIGURE 2-5: PIC18F010/020 OSCILLATOR CONFIGURATION



2.6 Two-Speed Clock Start-up Mode

In order to minimize the latency between oscillator start-up and code execution, a mode which allows the system clock to initially use the internal clock, may be selected with IESO (Internal-External Switchover) bit. In this mode and upon RESET, the system will begin execution with the internal oscillator at the frequency selected by the IRCFx bits of the OSCCON register.

Note:	Only on Power-on Reset, the register contents are zeroed by the POR circuitry and the frequency selection is forced to 32 kHz. The register is not effected by any other forms of RESET.

After the OST has timed out, a glitchless switchover will be made to the oscillator mode selected by FOSCx in the CONFIG1H register. The software may read the OSTO bit to determine when the switchover takes place, so that any software timing delays may be adjusted.

Wake-up from SLEEP causes a unique start-up procedure. The power supply is assumed to be stable, since neither the POR nor the BOR Resets have been invoked. This assumption allows the Power-on Timer (PWRT) time-out to be bypassed, and only the OST time-out to be used. This results in almost immediate code execution with the minimum of delay. The internal oscillator frequency can be selected to be close to final crystal frequency to reduce timing differences, or a lower frequency can be chosen to reduce power consumption.

REGISTER 2-1: OSCCON REGISTER (ADDRESS FD3h)

U-0	R/W-0	R/W-0	R/W-0	R-0	R/W-0	U-0	R/W-0
—	IRCF2	IRCF1	IRCF0	OSTO	IESO	—	SCS
bit 7							bit 0

bit 7 **Unimplemented:** Read as ‘0’

bit 6-4 **IRCF<2:0>:** Internal Oscillator Frequency Select bits
000 = 32 kHz
001 = Reserved
010 = Reserved
011 = 500 kHz
100 = 1 MHz
101 = Reserved
110 = 4 MHz
111 = 8 MHz

bit 3 **OSTO:** Oscillator Start-up Time-out Status bit
1 = Oscillator Start-up Timer has timed out
0 = Oscillator Start-up Timer running

bit 2 **IESO:** Internal-External Switchover bit
1 = Start with internal oscillator, then switch over to selected oscillator mode after OST
0 = No switch from internal oscillator from RESET

bit 1 **Unimplemented:** Read as ‘0’

bit 0 **SCS:** System Clock Switch bit
1 = Clock source comes from internal oscillator input
0 = Clock source comes from external clock source on OSC1

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

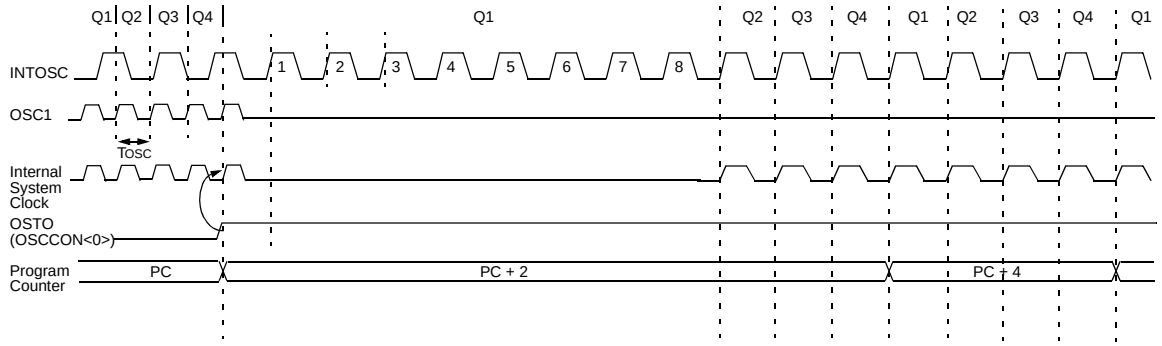
Note:	This register must be unlocked to modify, see Section 12.4.

2.6.1 OSCILLATOR TRANSITIONS

The PIC18F010/020 devices contain circuitry to prevent "glitches" when switching between oscillator sources. Essentially, the circuitry waits for eight rising edges of the clock source that the processor is switching to. This ensures that the new clock source is stable and that its pulse width will not be less than the shortest pulse width of the two clock sources.

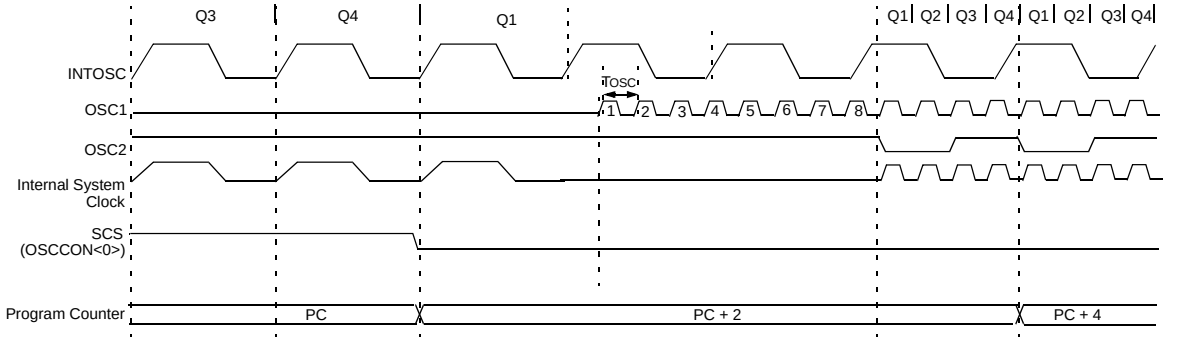
A timing diagram, indicating the transition from the internal oscillator to the external crystal is shown in Figure 2-6. The internal oscillator is assumed to be running all the time. After the OST bit is set, the processor is frozen at the next occurring Q1 cycle. After eight synchronization cycles are counted from the external oscillator, operation resumes. No additional delays are required after the synchronization cycles.

FIGURE 2-6: TIMING DIAGRAM FOR TRANSITION FROM EXTERNAL OSCILLATOR TO INTERNAL OSCILLATOR



Note 1: Delay on internal system clock is eight oscillator cycles for synchronization.

FIGURE 2-7: TIMING FOR TRANSITION BETWEEN INTERNAL OSCILLATOR AND OSC1 (EC)



Note 1: Internal oscillator mode assumed.

2.7 Effects of SLEEP Mode on the On-chip Oscillator

When the device executes a SLEEP instruction, the on-chip clocks and oscillator are turned off and the device is held at the beginning of an instruction cycle (Q1 state). With the oscillator off, the OSC1 and OSC2 signals will stop oscillating. Since all the transistor switching currents have been removed, SLEEP mode achieves the lowest current consumption of the device (only leakage currents). Enabling any on-chip feature that will operate during SLEEP will increase the current consumed during SLEEP. The user can wake from SLEEP through external RESET, Watchdog Timer Reset or through an interrupt.

TABLE 2-3: OSC1 AND OSC2 PIN STATES IN SLEEP MODE

OSC Mode	OSC1 Pin	OSC2 Pin
Internal Oscillator	Floating, external resistor should pull high	At logic low
RCIO	Floating, external resistor should pull high	Configured as PORTB, RB4
EC	Floating	At logic low
LP, XT, and HS	Feedback inverter disabled, at quiescent voltage level	Feedback inverter disabled, at quiescent voltage level

Note: See Table 3-1 in the RESET Section, for time-outs due to SLEEP and $\overline{\text{MCLR}}$ Reset.

2.8 Power-up Delays

Power-up delays are controlled by two timers, so that no external RESET circuitry is required for most applications. The delays ensure that the device is kept in RESET until the device power supply and clock are stable. For additional information on RESET operation, see the “RESET” section.

The first timer is the Power-up Timer (PWRT), which optionally provides a fixed delay of 72 ms (nominal) on power-up only (POR and BOR). The second timer is the Oscillator Start-up Timer OST, intended to keep the chip in RESET until the crystal oscillator is stable.

2.9 Frequency Calibrations

The 8 MHz frequency is calibrated at the factory. Since the 4 MHz and 1 MHz clock outputs are derived digitally from the 8 MHz, the accuracy specifications of the 4 MHz and 1 MHz clocks are the same as the 8 MHz.

The 500 kHz and 32 kHz frequencies are not calibrated. The 500 kHz and 32 kHz are nominal frequencies. Their accuracy specifications are shown in the Specifications section.

2.10 Frequency Tuning in User Mode

In addition to the factory calibration, 8 MHz frequency can be tuned in the user’s application. This frequency tuning capability allows user to deviate from the factory calibrated frequency. The user can tune the frequency by writing to the OSCTUNE register. See Register 2-2 for details of the OSCTUNE register. The tuning range of the 8 MHz oscillator is ±1 MHz, or ±12.5% nominal. See the Specifications section for further specification details.

Since the 4 MHz and 1 MHz are derived from the 8 MHz, the tuning range of the 4 MHz is ±500 kHz nominal, and the tuning range of the 1 MHz is ±125 kHz nominal. The tuning sensitivity (%FINTOSC/bit) is constant throughout the frequency selections and tuning range.

Note:	Frequency tuning is not available in the 500 kHz and 32 kHz frequencies.

REGISTER 2-2: OSCTUNE REGISTER (ADDRESS 0F9Bh)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	TUN5	TUN4	TUN3	TUN2	TUN1	TUN0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as ‘0’

bit 5-0 **TUN<5:0>:** 6-bit Frequency Tuning

011111 = Maximum frequency

011110

•

•

•

000001

000000 = Center frequency. Oscillator module is running at the calibrated frequency.

111111

•

•

•

100000 = Minimum frequency

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

2.11 Base Frequency Change

There are two methods to change frequency during normal program operation. One option is to switch frequencies using the internal oscillator only; IRCF<2:0> in the OSCCON register selects the internal oscillator frequency. Refer to Register 2-1.

Switching for an external clock to an internal oscillator and vice versa is also possible. Use the SCS bit in the OSCCON register to select an external or internal clock source.

Note:	The OSCEN bit in the CONFIG1H configuration byte must be set to allow clock switching.

2.12 Oscillator Delay Upon Start-up and Base Frequency Change

When the INTOSC Oscillator Module starts up, an 8-cycle delay of the base frequency is invoked. During this delay, the FINTOSC output signal is held at '0'.

The INTOSC Oscillator Module also allows user to change frequency during run time. For example, the frequency can be changed from 8 MHz to 32 kHz, while the device is operating. When the application requires a base frequency change, a delay of 8 cycles of the new base frequency is invoked.

Writing to the OSCTUNE register will not cause any delay. In applications where the OSCTUNE register is used to shift the FINTOSC frequency, the application should not expect the FINTOSC frequency to stabilize immediately. In this case, the frequency may shift gradually toward the new value. The time for this frequency shift is less than 8 cycles of the base frequency.

Table 2-4 below, shows examples of when the oscillator delay is invoked.

TABLE 2-4: OSCILLATOR DELAY EXAMPLES

Old Frequency	New Frequency	New Base Frequency	Oscillator Delay	Comments
8 MHz	4 MHz or 1 MHz	No	None	The 8 MHz, 4 MHz, and 1 MHz are all running from the same 8 MHz base frequency.
500 kHz	32 kHz	32 kHz	250µS nominal	Base frequency changes from 500 kHz to 32 kHz.
4 MHz	32 kHz	32 kHz	250µS nominal	Base frequency changes from 8 MHz to 32 kHz.
500 kHz	8 MHz	8 MHz	1µS nominal	Base frequency changes from 500 kHz to 8 MHz.
Off or SLEEP mode	1 MHz	8 MHz	1µS nominal	Upon power-up and wake-up from SLEEP, there is always oscillator delay.
Off or SLEEP mode	500 kHz	500 kHz	16µS nominal	Upon power-up and wake-up from SLEEP, there is always oscillator delay.

3.0 RESET

The PIC18F010/020 differentiates between various kinds of RESET:

a) Power-on Reset (POR)
b) $\overline{\text{MCLR}}$ Reset during normal operation
c) $\overline{\text{MCLR}}$ Reset during SLEEP
d) Watchdog Timer (WDT) Reset (during normal operation)
e) Programmable Brown-out Reset (BOR)
f) RESET Instruction
g) Stack Full Reset
h) Stack Underflow Reset

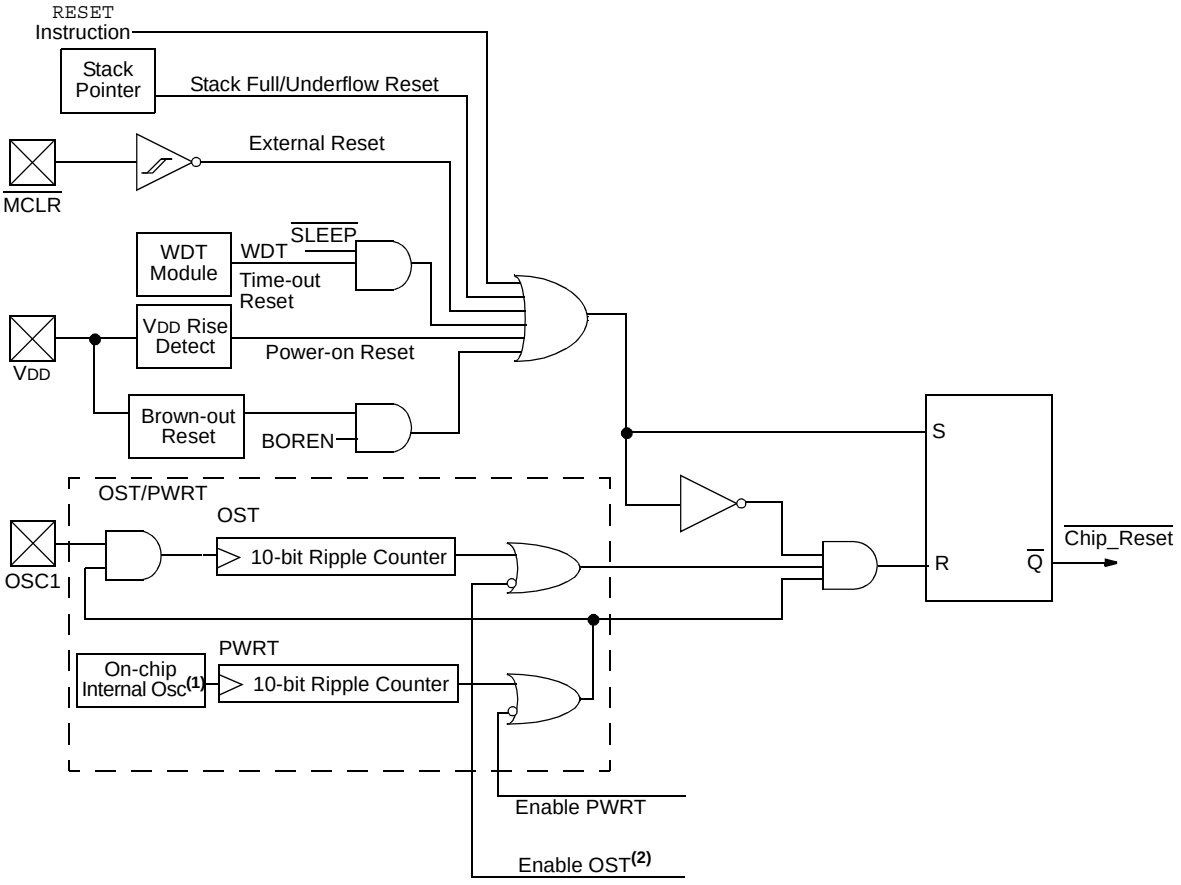
Most registers are unaffected by a RESET. Their status is unknown on POR and unchanged by all other RESETS. The other registers are forced to a "RESET state" on Power-on Reset, $\overline{\text{MCLR}}$, WDT Reset, Brown-out Reset, $\overline{\text{MCLR}}$ Reset during SLEEP and by the RESET instruction.

Most registers are not affected by a WDT wake-up, since this is viewed as the resumption of normal operation. Status bits from the RCON register, $\overline{\text{RI}}$, $\overline{\text{TO}}$, $\overline{\text{PD}}$, $\overline{\text{POR}}$ and $\overline{\text{BOR}}$, are set or cleared differently in different RESET situations, as indicated in Table 3-2. These bits are used in software to determine the nature of the RESET. See Table 3-3 for a full description of the RESET states of all registers.

A simplified block diagram of the on-chip RESET circuit is shown in Figure 3-1.

The Enhanced MCU devices have a $\overline{\text{MCLR}}$ noise filter in the $\overline{\text{MCLR}}$ Reset path. The filter will detect and ignore small pulses.

FIGURE 3-1: SIMPLIFIED BLOCK DIAGRAM OF ON-CHIP RESET CIRCUIT



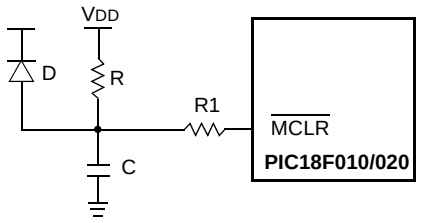
Note 1: This is a separate oscillator from the internal oscillator of the CLKIN pin.
2: See Table 3-1 for time-out situations.

3.1 Power-on Reset (POR)

A Power-on Reset pulse is generated on-chip when VDD rise is detected. To take advantage of the POR circuitry, tie the MCLR pin directly (or through a resistor) to VDD, or disable MCLR. This will eliminate external oscillator components usually needed to create a Power-on Reset delay. A maximum rise time for VDD is specified (parameter D004). For a slow rise time, see Figure 3-2.

When the device starts normal operation (exits the RESET condition), device operating parameters (voltage, frequency, temperature,...) must be met to ensure operation. If these conditions are not met, the device must be held in RESET until the operating conditions are met. Brown-out Reset may be used to meet the voltage start-up condition.

FIGURE 3-2: EXTERNAL POWER-ON RESET CIRCUIT (FOR SLOW VDD POWER-UP)



Note 1: External Power-on Reset circuit is required only if the VDD power-up slope is too slow. The diode D helps discharge the capacitor quickly when VDD powers down.

2: R < 40kΩ is recommended to make sure that the voltage drop across R does not violate the device's electrical specification.

3: R1 = 100Ω to 1kΩ will limit any current flowing into MCLR from external capacitor C, in the event of MCLR/VPP pin breakdown due to Electrostatic Discharge (ESD), or Electrical Overstress (EOS).

3.2 Power-up Timer (PWRT)

The Power-up Timer provides a fixed nominal time-out (parameter #33) only on power-up from the POR or BOR, if enabled. The Power-up Timer operates on an internal oscillator. The chip is kept in RESET as long as the PWRT is active. The PWRT's time delay allows VDD to rise to an acceptable level. A configuration bit is provided to enable/disable the PWRT.

The power-up time delay will vary from chip-to-chip due to VDD, temperature and process variation. See DC parameter #33 for details.

3.3 Oscillator Start-up Timer (OST)

The Oscillator Start-up Timer (OST) provides 1024 oscillator cycle (from OSC1 input) delay after the PWRT delay is over (parameter #32). This ensures that the crystal oscillator or resonator has started and stabilized.

The OST time-out is invoked only for XT, LP and HS modes and only on Power-on Reset or wake-up from SLEEP.

3.4 Brown-out Reset (BOR)

A configuration bit, BOREN, can disable (if clear/programmed), or enable (if set) the Brown-out Reset circuitry. If VDD falls below parameter D005 for greater than parameter #35, the brown-out situation will reset the chip. A RESET may not occur if VDD falls below parameter D005 for less than parameter #35. The chip will remain in Brown-out Reset until VDD rises above BVDD. The Power-up Timer will then be invoked and will keep the chip in RESET an additional time delay (parameter #33). If VDD drops below BVDD while the Power-up Timer is running, the chip will go back into a Brown-out Reset and the Power-up Timer will be initialized. Once VDD rises above BVDD, the Power-up Timer will execute the additional time delay.

3.5 Time-out Sequence

On power-up, the time-out sequence is as follows: first, PWRT time-out is invoked after the POR time delay has expired; then, OST is activated. The total time-out will vary based on oscillator configuration and the status of the PWRT. For example, in Internal Oscillator mode with the PWRT disabled, there will be no time-out at all. Figure 3-3, Figure 3-4, Figure 3-5 and Figure 3-6 depict time-out sequences on power-up.

Since the time-outs occur from the POR pulse, if $\overline{MCLR}$ is kept low long enough, the time-outs will expire. Bringing $\overline{MCLR}$ high will begin execution immediately (Figure 3-5). This is useful for testing purposes or to synchronize more than one PIC18F010/020 device operating in parallel.

Table 3-2 shows the RESET conditions for some Special Function Registers, while Table 3-3 shows the RESET conditions for all the registers.

TABLE 3-1: TIME-OUT IN VARIOUS SITUATIONS

Oscillator Configuration	Power-up(1)		Brown-out(1)	Wake-up from SLEEP or Oscillator Switch
	$\overline{PWRTE}$ = 0	$\overline{PWRTE}$ = 1		
HS, XT, LP	72 ms + 1024Tosc	1024Tosc	72 ms + 1024Tosc	1024Tosc
EC	72 ms	—	72 ms	—
External Oscillator	72 ms	—	72 ms	—
Internal Oscillator(2)	72 ms	—	72 ms	—

Note 1: 72 ms is the nominal power-up timer delay.
2: 8-cycle delay.

REGISTER 3-1: RCON REGISTER BITS AND POSITIONS

R/W-0	U-0	U-0	R/W-1	R-1	R-1	R/W-1	R/W-1
IPEN	—	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

TABLE 3-2: STATUS BITS, THEIR SIGNIFICANCE AND THE INITIALIZATION CONDITION FOR RCON REGISTER

Condition	Program Counter	RCON Register	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$	STKFUL	STKUNF
Power-on Reset	0000h	00-1 1100	1	1	1	0	0	u	u
$\overline{MCLR}$ Reset during normal operation	0000h	00-u uuuu	u	u	u	u	u	u	u
Software Reset during normal operation	0000h	0u-0 uuuu	0	u	u	u	u	u	u
Stack Full Reset during normal operation	0000h	0u-u uu11	u	u	u	u	u	1	u
Stack Underflow Reset during normal operation	0000h	0u-u uu11	u	u	u	u	u	u	1
$\overline{MCLR}$ Reset during SLEEP	0000h	00-u 10uu	u	1	0	u	u	u	u
WDT Reset	0000h	0u-u 01uu	1	0	1	u	u	u	u
WDT Wake-up	PC + 2	uu-u 00uu	u	0	0	u	u	u	u
Brown-out Reset	0000h	0u-1 11u0	1	1	1	1	0	u	u
Interrupt Wake-up from SLEEP	PC + 2(1)	uu-u 00uu	u	1	0	u	u	u	u

Legend: u = unchanged, x = unknown, - = unimplemented bit, read as '0'.

Note 1: When the wake-up is due to an interrupt and the GIEH or GIEL bits are set, the PC is loaded with the interrupt vector (0x000008h or 0x000018h).

TABLE 3-3: INITIALIZATION CONDITIONS FOR ALL REGISTERS

Register	Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Reset WDT Reset Reset Instruction Stack Reset	Wake-up via WDT or Interrupt
TOSH	0000 0000	0000 0000	uuuu uuuu(3)
TOSL	0000 0000	0000 0000	uuuu uuuu(3)
STKPTR	00-0 0000	00-0 0000	uu-u uuuu(3)
PCLATU	---0 0000	---0 0000	---u uuuu
PCLATH	0000 0000	0000 0000	uuuu uuuu
PCL	0000 0000	0000 0000	PC + 2(2)
TBLPTRU	---0 00--	---0 00--	---u uu--
TBLPTRH	---- 0000	---- 0000	---- uuuu
TBLPTRL	0000 0000	0000 0000	uuuu uuuu
TABLAT	0000 0000	0000 0000	uuuu uuuu
PRODH	xxxx xxxx	uuuu uuuu	uuuu uuuu
PRODL	xxxx xxxx	uuuu uuuu	uuuu uuuu
INTCON	0000 000x	0000 000u	uuuu uuuu(1)
INTCON2	11-- -1-1	11-- -1-1	uu-- -u-u(1)
INDF0	N/A	N/A	N/A
POSTINC0	N/A	N/A	N/A
POSTDEC0	N/A	N/A	N/A
PREINC0	N/A	N/A	N/A
PLUSW0	N/A	N/A	N/A
FSR0H	---- 0000	---- 0000	---- uuuu
FSR0L	xxxx xxxx	uuuu uuuu	uuuu uuuu
WREG	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF1	N/A	N/A	N/A
POSTINC1	N/A	N/A	N/A
POSTDEC1	N/A	N/A	N/A
PREINC1	N/A	N/A	N/A
PLUSW1	N/A	N/A	N/A
FSR1H	---- 0000	---- 0000	---- uuuu
FSR1L	xxxx xxxx	uuuu uuuu	uuuu uuuu
BSR	---- 0000	---- 0000	---- uuuu
INDF2	N/A	N/A	N/A
POSTINC2	N/A	N/A	N/A
POSTDEC2	N/A	N/A	N/A
PREINC2	N/A	N/A	N/A
PLUSW2	N/A	N/A	N/A

Legend: u = unchanged, x = unknown, - = unimplemented bit, read as '0', q = value depends on condition

Note 1: One or more bits in the INTCONx or PIRx registers will be affected (to cause wake-up).
2: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the PC is loaded with the interrupt vector (0008h or 0018h).
3: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the TOSU, TOSH and TOSL are updated with the current value of the PC. The STKPTR is modified to point to the next location in the hardware stack.
4: See Table 3-2 for RESET value for specific condition.
5: The long write enable is only reset on a POR or $\overline{\text{MCLR}}$ Reset.

TABLE 3-3: INITIALIZATION CONDITIONS FOR ALL REGISTERS (CONTINUED)

Register	Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Reset WDT Reset Reset Instruction Stack Reset	Wake-up via WDT or Interrupt
FSR2H	---- 0000	---- 0000	---- uuuu
FSR2L	xxxx xxxx	uuuu uuuu	uuuu uuuu
STATUS	---x xxxx	---u uuuu	---u uuuu
TMR0H	0000 0000	0000 0000	uuuu uuuu
TMR0L	xxxx xxxx	uuuu uuuu	uuuu uuuu
T0CON	1111 1111	1111 1111	uuuu uuuu
OSCCON	-000 00-0	-uuu uu-u	-uuu uu-u
LVDCON	--00 0101	--00 0101	--uu uuuu
WDTCON	---- ---0	---- ---0	---- ---u
RCON(4,5)	0--1 11qq	0--q qquu	u--u qquu
IPR2	---- 1111	---- 1111	---- uuuu
PIR2	---- 0000	---- 0000	---- uuuu(1)
PIE2	---- 0000	---- 0000	---- uuuu
TRISB	--11 1111	--11 1111	--uu uuuu
LATB	--xx xxxx	--uu uuuu	--uu uuuu
PORTB	--xx xxxx	--uu uuuu	--uu uuuu
PSPCON	---- --00	---- --00	---- --uu
EEADR	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEDATA	xxxx xxxx	uuuu uuuu	uuuu uuuu
EECON2	---- ----	---- ----	---- ----
EECON1	x--0 x000	u--0 u000	u--u uuuu
OSCTUNE	--00 0000	--qq qqqq	--uu uuuu
WPUB	--11 1111	--11 1111	--uu uuuu
IOCB	--00 0000	--00 0000	--uu uuuu

Legend: u = unchanged, x = unknown, - = unimplemented bit, read as '0', q = value depends on condition

Note 1: One or more bits in the INTCONx or PIRx registers will be affected (to cause wake-up).

2: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the PC is loaded with the interrupt vector (0008h or 0018h).

3: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the TOSU, TOSH and TOSL are updated with the current value of the PC. The STKPTR is modified to point to the next location in the hardware stack.

4: See Table 3-2 for RESET value for specific condition.

5: The long write enable is only reset on a POR or $\overline{\text{MCLR}}$ Reset.

FIGURE 3-3: TIME-OUT SEQUENCE ON POWER-UP ($\overline{MCLR}$ TIED TO V_{DD})

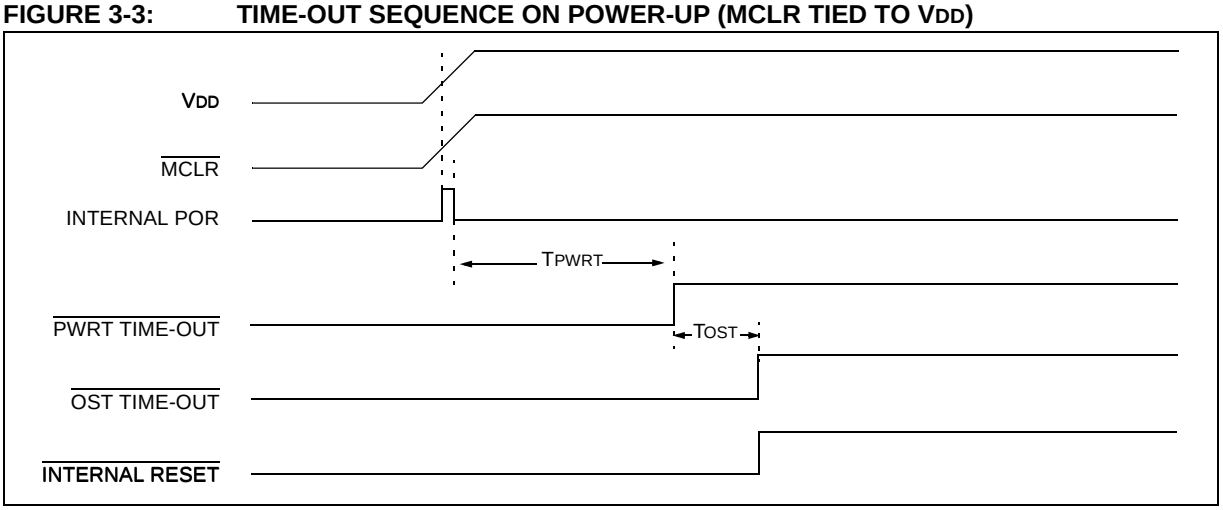


FIGURE 3-4: TIME-OUT SEQUENCE ON POWER-UP ($\overline{MCLR}$ NOT TIED TO V_{DD}): CASE 1

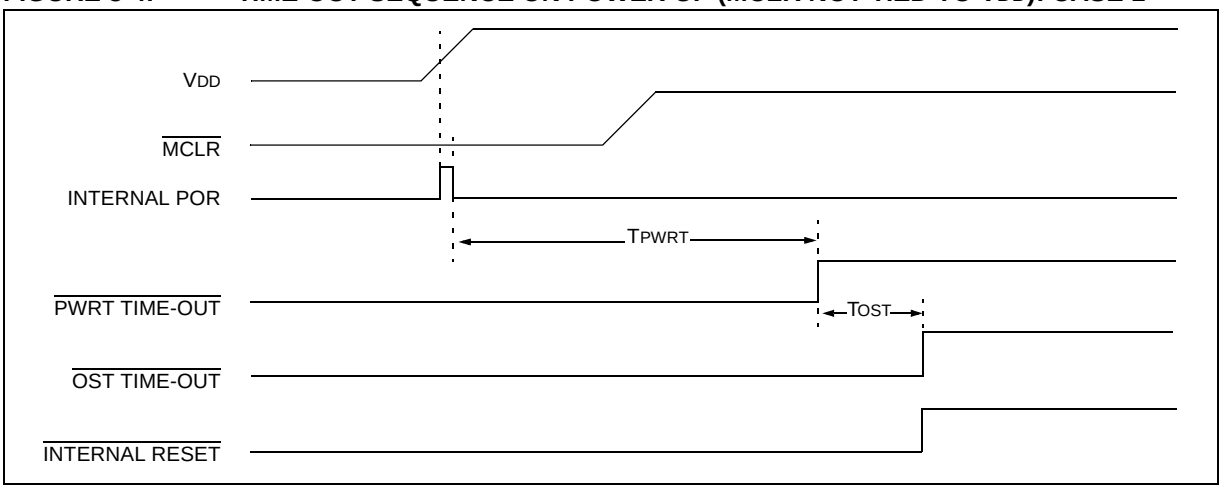


FIGURE 3-5: TIME-OUT SEQUENCE ON POWER-UP ($\overline{MCLR}$ NOT TIED TO V_{DD}): CASE 2

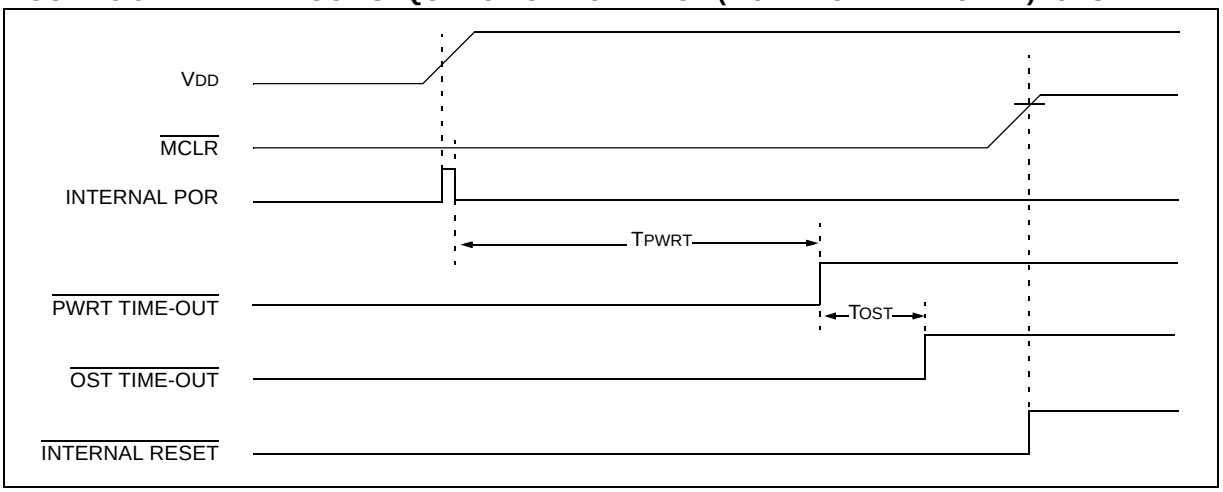
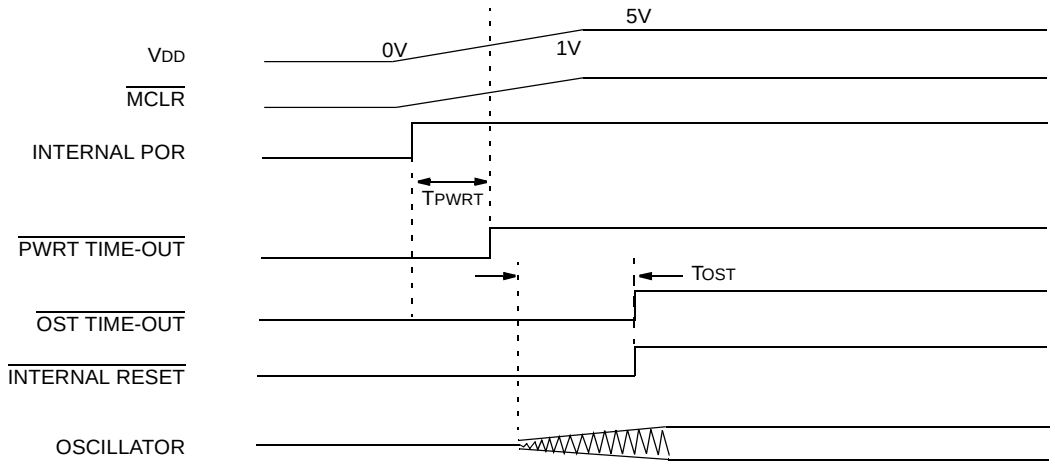


FIGURE 3-6: SLOW RISE TIME ($\overline{\text{MCLR}}$ TIED TO V_{DD})



Note: For slow starting crystals, OST can start beyond PWRT.

NOTES:

4.0 MEMORY ORGANIZATION

There are three memory blocks in PIC18F010/020 Enhanced MCU devices. These memory blocks are:

- Program Memory
- Data Memory
- EEPROM Data Memory

The EEPROM Data Memory is described in detail in Section 5.0.

4.1 Program Memory Organization

The PIC18F010/020 devices have a 21-bit program counter. Bits 12 through 16 are implemented as ‘0’ internally; therefore, accessing locations 0x01000 through 0x1FFFF actually mirror what is present in program memory from 0x0000 through 0x0FFF. The PIC18F010 device reads all zeros (`NOP`) from 0x0800 through 0x0FFF.

PIC18F020 has 4 Kbytes of FLASH program memory, while PIC18F010 has 2 Kbytes of FLASH program memory. This means the PIC18F020 can store up to 2K of single word instructions, and the PICF18010 can store up to 1K of single word instructions.

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

Figure 4-1 shows the Program Memory Map for PIC18F010 and Figure 4-2 shows the Program Memory Map for PIC18F020 devices.

FIGURE 4-1: PIC18F010 MEMORY

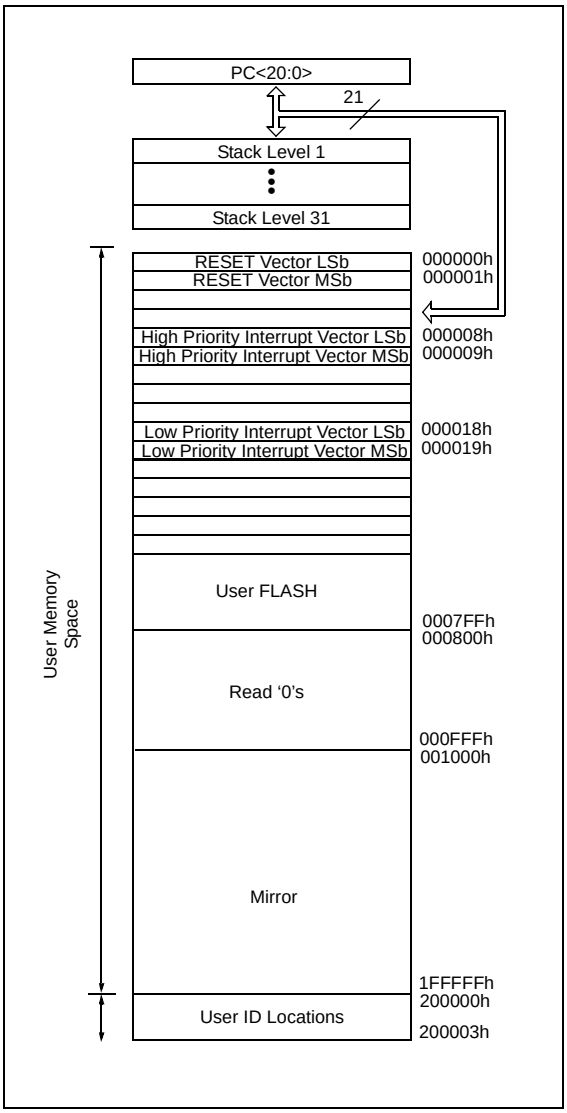
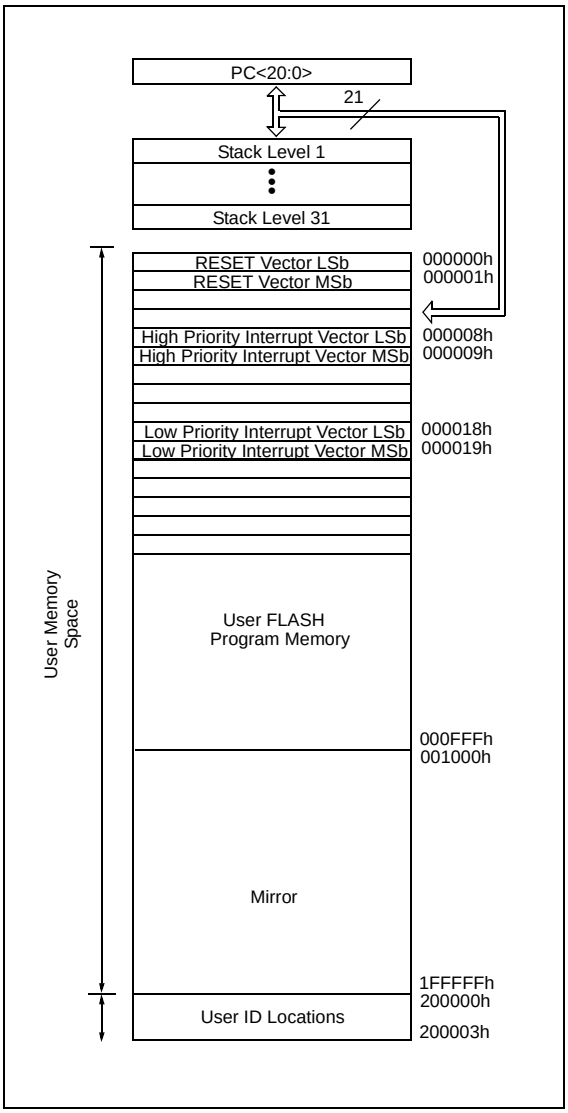


FIGURE 4-2: PIC18F020 MEMORY



4.2 Return Address Stack

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

The stack operates as a 31-word by 21-bit RAM with a 5-bit stack pointer. Although there are 21 bits in the TOS latch, bits 12 through 16 are not physically implemented in the stack and are read as zeros. The stack pointer initializes to 0x00 after all RESETS, and there is no RAM associated with stack pointer 0x00. This is only a RESET value. During a CALL type instruction causing a push onto the stack, the stack pointer is first incremented and the RAM location pointed to by the stack pointer is written with the contents of the PC. During a RETURN type instruction causing a pop from the stack, the contents of the RAM location pointed to by the STKPTR is transferred to the PC and then, the stack pointer is decremented.

The stack space is not part of either program or data space. The stack pointer is readable and writable, and the address on the top of the stack is readable and writable through SFR registers. Data can also be pushed to, or popped from the stack, using the top-of-stack SFRs. Status bits indicate if the stack pointer is at, or beyond, the 31 levels provided.

Note:	Do not push data onto the stack in bits 12 through 16. This data will be lost. Bits 12 through 16 are always read as '0'.

4.2.1 TOP-OF-STACK ACCESS

The top of the stack is readable and writable. Three register locations, TOSH and TOSL hold the contents of the stack location pointed to by the STKPTR register. This allows users to implement a software stack, if necessary. After a CALL, RCALL or interrupt, the software can read the pushed value by reading the TOSH and TOSL registers. These values can be placed on a user defined software stack. At return time, the software can replace the TOSH and TOSL and do a return.

The user must disable the global interrupt enable bits during this time to prevent inadvertent stack operations.

4.2.2 RETURN STACK POINTER (STKPTR)

The STKPTR register contains the stack pointer value, the STKFUL (stack full) status bit, and the STKUNF (stack underflow) status bits. Register 4-1 shows the STKPTR register. The value of the stack pointer can be 0 through 31. The stack pointer increments when values are pushed onto the stack and decrements when values are popped off the stack. At RESET, the stack pointer value will be 0. The user may read and write the stack pointer value. This feature can be used by a Real Time Operating System for return stack maintenance.

After the PC is pushed onto the stack 31 times (without popping any values off the stack), the STKFUL bit is set. The STKFUL bit can only be cleared in software or by a POR.

The action that takes place when the stack becomes full depends on the state of the STVREN (Stack Overflow Reset Enable) configuration bit. Refer to Section 12.0 for a description of the device configuration bits. If STVREN is set (default), the 31st push will push the (PC + 2) value onto the stack, set the STKFUL bit, and reset the device. The STKFUL bit will remain set and the stack pointer will be set to 0.

If STVREN is cleared, the STKFUL bit will be set on the 31st push and the stack pointer will increment to 31. The 32nd push and beyond will be lost while STKPTR remains at 31, and the 31st push is maintained.

When the stack has been popped enough times to unload the stack, the next pop will return a value of zero to the PC and sets the STKUNF bit, while the stack pointer remains at 0. The STKUNF bit will remain set until cleared in software or a POR occurs.

Note:	Returning a value of zero to the PC on an underflow, has the effect of vectoring the program to the RESET vector, where the stack conditions can be verified and appropriate actions can be taken.

REGISTER 4-1: STKPTR - STACK POINTER REGISTER

R/C-0	R/C-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
STKFUL	STKUNF	—	SP4	SP3	SP2	SP1	SP0
bit7							bit0

bit 7(1) **STKFUL:** Stack Full Flag bit
1 = Stack became full or overflowed
0 = Stack has not become full or overflowed

bit 6(1) **STKUNF:** Stack Underflow Flag bit
1 = Stack underflow occurred
0 = Stack underflow did not occur

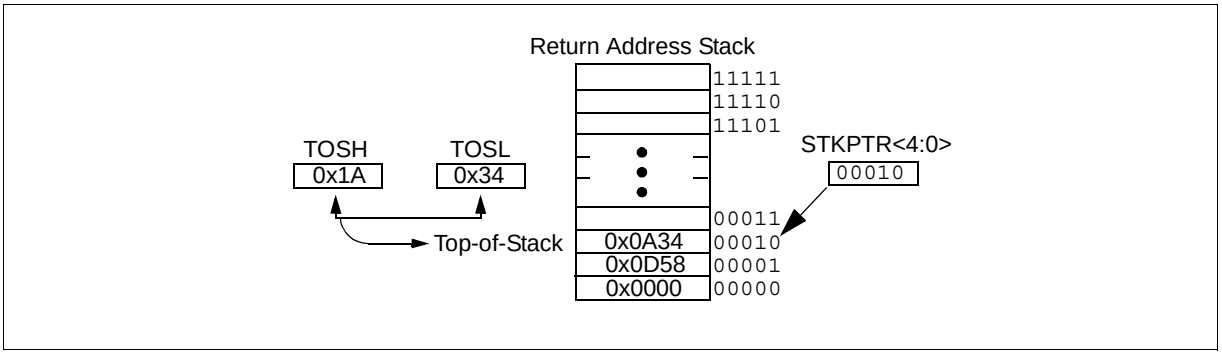
bit 5 **Unimplemented:** Read as '0'

bit 4-0 **SP4:SP0**: Stack Pointer Location bits

Note 1: Bit 7 and bit 6 can only be cleared in user software, or by a POR.

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

FIGURE 4-3: RETURN ADDRESS STACK AND ASSOCIATED REGISTERS



4.2.3 PUSH AND POP INSTRUCTIONS

Since the Top-of-Stack (TOS) is readable and writable, the ability to push values onto the stack and pull values off the stack, without disturbing normal program execution, is a desirable option. To push the current PC value onto the stack, a PUSH instruction can be executed. This will increment the stack pointer and load the current PC value onto the stack. TOSU, TOSH and TOSL can then be modified to place a return address on the stack.

The ability to pull the TOS value off of the stack and replace it with the value that was previously pushed onto the stack, without disturbing normal execution, is achieved by using the POP instruction. The POP instruction discards the current TOS by decrementing the stack pointer. The previous value pushed onto the stack then becomes the TOS value.

4.2.4 STACK FULL/UNDERFLOW RESETS

These RESETS are enabled by programming the STVREN configuration bit. When the STVREN bit is disabled, a full or underflow condition will set the appropriate STKFUL or STKUNF bit, but not cause a device RESET. When the STVREN bit is enabled, a full or underflow condition will set the appropriate STKFUL or STKUNF bit and then cause a device RESET. The STKFUL or STKUNF bits are only cleared by the user software or a POR Reset.

4.3 Fast Register Stack

A "fast interrupt return" option is available for interrupts. A Fast Register Stack is provided for the STATUS, WREG and BSR registers and are only one in depth. The stack is not readable or writable and is loaded with the current value of the corresponding register when the processor vectors for an interrupt. The values in the registers are then loaded back into the working registers, if the `fast return` instruction is used to return from the interrupt.

A low or high priority interrupt source will push values into the stack registers. If both low and high priority interrupts are enabled, the stack registers cannot be used reliably for low priority interrupts. If a high priority interrupt occurs while servicing a low priority interrupt, the stack register values stored by the low priority interrupt will be overwritten.

If high priority interrupts are not disabled during low priority interrupts, users must save the key registers in software during a low priority interrupt.

If no interrupts are used, the fast register stack can be used to restore the STATUS, WREG and BSR registers at the end of a subroutine call. To use the fast register stack for a subroutine call, a `fast call` instruction must be executed.

Example 4-1 shows a source code example that uses the fast register stack.

EXAMPLE 4-1: FAST REGISTER STACK CODE EXAMPLE

```
CALL SUB1, FAST    ;STATUS, WREG, BSR
                   ;SAVED IN FAST REGISTER
                   ;STACK
        •
        •

SUB1    •
        •
        •
   RETURN FAST     ;RESTORE VALUES SAVED
                   ;IN FAST REGISTER STACK
```

4.4 PCL, PCLATH and PCLATU

The program counter (PC) specifies the address of the instruction to fetch for execution. The PC is 21-bits wide. The low byte is called the PCL register. This register is readable and writable. The high byte is called the PCH register. This register contains the PC<11:8> bits and is not directly readable or writable. Updates to the PCH register may be performed through the PCLATH register. The upper byte is called PCU. This register contains the PC<20:17> bits and is not directly readable or writable. Updates to the PCU register may be performed through the PCLATU register.

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

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

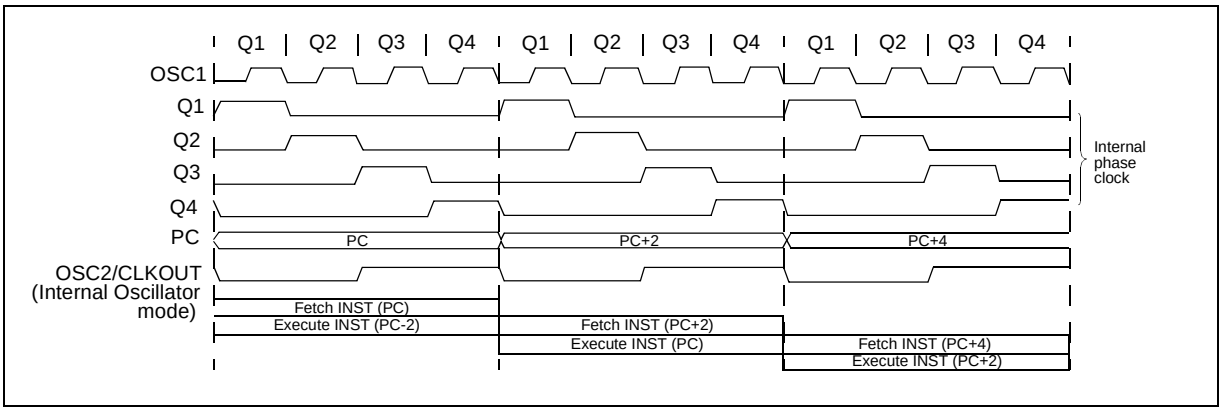
The contents of PCLATH and PCLATU will be transferred to the program counter by an operation that writes PCL. Similarly, the upper two bytes of the program counter will be transferred to PCLATH and PCLATU, by an operation that reads PCL. This is useful for computed offsets to the PC (see Section 4.8.1).

Note:	Bits 12 through 16 are not implemented in the PC and PCLAT.

4.5 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, 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 4-4.

FIGURE 4-4: CLOCK/INSTRUCTION CYCLE



4.6 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 4-2).

A fetch cycle begins with the program counter (PC) 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).

EXAMPLE 4-2: INSTRUCTION PIPELINE FLOW

	TCY0	TCY1	TCY2	TCY3	TCY4	TCY5
1. MOVLW 55h	Fetch 1	Execute 1				
2. MOVWF PORTB		Fetch 2	Execute 2			
3. BRA SUB_1			Fetch 3	Execute 3		
4. BSF PORTA, BIT3 (Forced NOP)				Fetch 4	Flush	
5. Instruction @ address SUB_1					Fetch SUB_1	Execute SUB_1

All instructions are single cycle, except for any program branches. These take two cycles, since the fetch instruction is “flushed” from the pipeline, while the new instruction is being fetched and then executed.

4.7 Instructions in Program Memory

The program memory is addressed in bytes. Instructions are stored as two bytes or four bytes in program memory. The least significant byte of an instruction word is always stored in a program memory location with an even address (LSB = '0'). Figure 4-5 shows an example of how instruction words are stored in the program memory. To maintain alignment with instruction boundaries, the PC increments in steps of 2 and the LSB will always read '0' (see Section 4.4).

The CALL and GOTO instructions have an absolute program memory address embedded into the instruction. Since instructions are always stored on word boundaries, the data contained in the instruction is a word address. The word address is written to PC<20:1>, which accesses the desired byte address in program memory. Instruction #2 in Figure 4-5 shows how the instruction "GOTO 000006h' is encoded in the program memory. Program branch instructions, which encode a relative address offset, operate in the same manner. The offset value stored in a branch instruction represents the number of single word instructions that the PC will be offset by. Section 13.0 provides further details of the instruction set.

FIGURE 4-5: INSTRUCTIONS IN PROGRAM MEMORY

			LSB = 1	LSB = 0	Word Address ↓
	Program Memory Byte Locations →				000000h
					000002h
					000004h
					000006h
Instruction 1:	MOVLW	055h	0Fh	55h	000008h
Instruction 2:	GOTO	000006h	EFh	03h	00000Ah
			F0h	00h	00000Ch
Instruction 3:	MOVFF	123h, 456h	C1h	23h	00000Eh
			F4h	56h	000010h
					000012h
					000014h

4.7.1 TWO-WORD INSTRUCTIONS

The PIC18F010/020 devices have 4 two-word instructions: MOVFF, CALL, GOTO and LFSR. The second word of these instructions has the 4 MSB's set to 1's and is a special kind of NOP instruction. The lower 12 bits of the second word contain data to be used by the instruction. If the first word of the instruction is executed, the data in the second word is accessed. If the second word of the instruction is executed by itself (first word was skipped), it will execute as a NOP. This action is necessary when the two-word instruction is preceded by a conditional instruction that changes the PC. A program example that demonstrates this concept is shown in Example 4-3. Refer to Section 13.0 for further details of the instruction set.

EXAMPLE 4-3: TWO-WORD INSTRUCTIONS

CASE 1:			
Object Code	**Source Code**		
0110 0110 0000 0000	TSTFSZ	REG1	; is RAM location 0?
1100 0001 0010 0011	MOVFF	REG1, REG2	; No, execute 2-word instruction
1111 0100 0101 0110			; 2nd operand holds address of REG2
0010 0100 0000 0000	ADDWF	REG3	; continue code
CASE 2:			
Object Code	**Source Code**		
0110 0110 0000 0000	TSTFSZ	REG1	; is RAM location 0?
1100 0001 0010 0011	MOVFF	REG1, REG2	; Yes
1111 0100 0101 0110			; 2nd operand becomes NOP
0010 0100 0000 0000	ADDWF	REG3	; continue code

4.8 Lookup Tables

Lookup tables are implemented two ways. These are:

- Computed GOTO
- Table Reads

4.8.1 COMPUTED GOTO

A computed GOTO is accomplished by adding an offset to the program counter (ADDWF PCL).

A lookup table can be formed with an ADDWF PCL instruction and a group of RETLW 0xnn instructions. WREG is loaded with an offset into the table, before executing a call to that table. The first instruction of the called routine is the ADDWF PCL instruction. The next instruction executed will be one of the RETLW 0xnn instructions, that returns the value 0xnn to the calling function.

The offset value (value in WREG) specifies the number of bytes that the program counter should advance.

In this method, only one data byte may be stored in each instruction location and room on the return address stack is required.

4.8.2 TABLE READS/TABLE WRITES

A better method of storing data in program memory allows 2 bytes of data to be stored in each instruction location.

Lookup table data may be stored 2 bytes per program word by using table reads and writes. The table pointer (TBLPTR) specifies the byte address and the table latch (TABLAT) contains the data that is read from, or written to, program memory. Data is transferred to/from program memory one byte at a time.

A description of the Table Read/Table Write operation is shown in Section 6.0.

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 PIC18F010/020 devices.

Banking is required to allow more than 256 bytes to be accessed. The data memory map is divided into 2 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 grow downwards. 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 the File Select Register (FSR). 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.

Note:	Only 2 banks are implemented, Bank 0 and Bank 15.

4.9.1 GENERAL PURPOSE REGISTER FILE

The register file can be accessed either directly or indirectly. Indirect addressing operates through the File Select Registers (FSR). 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. Bank 15 (0xF80 to 0xFFF) contains SFRs. Bank 0 contains GPR registers.

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 Figure 4-7 and Figure 4-8.

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 Figure 4-7 for addresses for the SFRs.

FIGURE 4-6: DATA MEMORY MAP PIC18F010/020

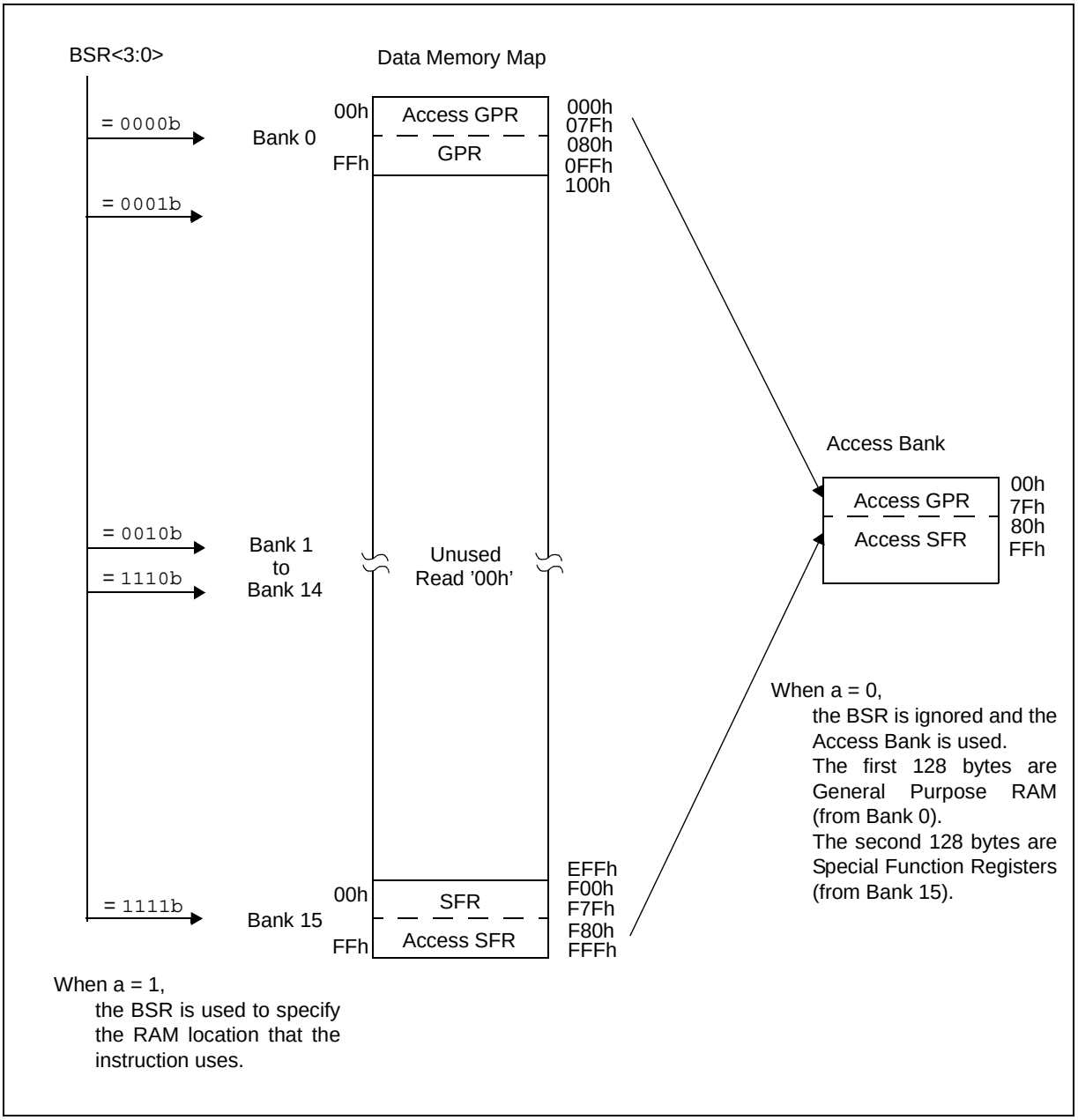


FIGURE 4-7: SPECIAL FUNCTION REGISTER MAP (F80h-FFFh)

FFFh		FDFh	INDF2	FBFh		F9Fh	
FFEh	TOSH	FDEh	POSTINC2	FBEh		F9Eh	
FFDh	TOSL	FDDh	POSTDEC2	FBDh		F9Dh	
FFCh	STKPTR	FDCh	PREINC2	FBCh		F9Ch	reserved
FFBh	PCLATU	FDBh	PLUSW2	FBBh		F9Bh	OSCTUNE
FFAh	PCLATH	FDAh	FSR2H	FBAh		F9Ah	
FF9h	PCL	FD9h	FSR2L	FB9h	reserved	F99h	
FF8h	TBLPTRU	FD8h	STATUS	FB8h	reserved	F98h	
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	reserved	F97h	
FF6h	TBLPTRL	FD6h	TMR0L	FB6h		F96h	
FF5h	TABLAT	FD5h	T0CON	FB5h		F95h	
FF4h	PRODH	FD4h	reserved	FB4h		F94h	
FF3h	PRODL	FD3h	OSCCON	FB3h		F93h	TRISB
FF2h	INTCON	FD2h	LVDCON	FB2h		F92h	
FF1h	INTCON2	FD1h	WDTCON	FB1h		F91h	
FF0h	INTCON3	FD0h	RCON	FB0h		F90h	
FEFh	INDF0	FCFh		FAFh		F8Fh	
FEEh	POSTINC0	FCEh		FAEh		F8Eh	
FEDh	POSTDEC0	FCDh		FADh		F8Dh	
FECh	PREINC0	FCCh		FACh		F8Ch	
FEBh	PLUSW0	FCBh		FABh		F8Bh	
FEAh	FSR0H	FCAh		FAAh	EEADRH	F8Ah	LATB
FE9h	FSR0L	FC9h		FA9h	EEADR	F89h	
FE8h	WREG	FC8h		FA8h	EEDATA	F88h	
FE7h	INDF1	FC7h		FA7h	EECON2	F87h	
FE6h	POSTINC1	FC6h		FA6h	EECON1	F86h	
FE5h	POSTDEC1	FC5h		FA5h		F85h	
FE4h	PREINC1	FC4h		FA4h		F84h	
FE3h	PLUSW1	FC3h		FA3h		F83h	
FE2h	FSR1H	FC2h		FA2h	IPR2	F82h	
FE1h	FSR1L	FC1h		FA1h	PIR2	F81h	PORTB
FE0h	BSR	FC0h		FA0h	PIE2	F80h	

Note: Shading indicates addresses within Access Bank. Blank areas indicate reserved register space that may or may not be implemented in this device.

FIGURE 4-8: SPECIAL FUNCTION REGISTER MAP (F00h-F7Fh)

F7Fh		F5Fh		F3Fh		F1Fh	
F7Eh		F5Eh		F3Eh		F1Eh	
F7Dh		F5Dh		F3Dh		F1Dh	
F7Ch		F5Ch		F3Ch		F1Ch	
F7Bh		F5Bh		F3Bh		F1Bh	
F7Ah		F5Ah		F3Ah		F1Ah	
F79h	WPUB	F59h		F39h		F19h	
F78h	IOCB	F58h		F38h		F18h	
F77h		F57h		F37h		F17h	
F76h		F56h		F36h		F16h	
F75h		F55h		F35h		F15h	
F74h		F54h		F34h		F14h	
F73h		F53h		F33h		F13h	
F72h		F52h		F32h		F12h	
F71h		F51h		F31h		F11h	
F70h		F50h		F30h		F10h	
F6Fh		F4Fh		F2Fh		F0Fh	
F6Eh		F4Eh		F2Eh		F0Eh	
F6Dh		F4Dh		F2Dh		F0Dh	
F6Ch		F4Ch		F2Ch		F0Ch	
F6Bh		F4Bh		F2Bh		F0Bh	
F6Ah		F4Ah		F2Ah		F0Ah	
F69h		F49h		F29h		F09h	
F68h		F48h		F28h		F08h	
F67h		F47h		F27h		F07h	
F66h		F46h		F26h		F06h	
F65h		F45h		F25h		F05h	
F64h		F44h		F24h		F04h	
F63h		F43h		F23h		F03h	
F62h		F42h		F22h		F02h	
F61h		F41h		F21h		F01h	
F60h		F40h		F20h		F00h	

Note: Shading indicates addresses within Access Bank. Blank areas indicate reserved register space that may or may not be implemented in this device.

TABLE 4-1: REGISTER FILE SUMMARY (PIC18F010/020)

File 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 (Note 1)
FFEh	TOSH	Top-of-Stack High Byte (TOS<11:8>)								---- 0000	---- 0000
FFDh	TOSL	Top-of-Stack Low Byte (TOS<7:0>)								0000 0000	0000 0000
FFCh	STKPTR	STKOVF	STKUNF	—	Return Stack Pointer					00-0 0000	00-0 0000
FFBh	PCLATU	—	—	bit21(3)	Holding Register for PC<20:18>			—	—	--00 00--	--00 00--
FFAh	PCLATH	—	—	—	—	Holding Register for PC<11:8>				---- 0000	---- 0000
FF9h	PCL	PC Low Byte (PC<7:0>)								0000 0000	0000 0000
FF8h	TBLPTRU	—	—	bit21(2)	Program Memory Table Pointer Upper Byte (TBLPTR<20:18>)			—	—	---0 0000	---0 0000
FF7h	TBLPTRH	—	—	—	—	Program Memory Table Pointer High Byte (TBLPTR<11:8>)				0000 0000	0000 0000
FF6h	TBLPTRL	Program Memory Table Pointer Low Byte (TBLPTR<7:0>)								0000 0000	0000 0000
FF5h	TABLAT	Program Memory Table Latch								0000 0000	0000 0000
FF4h	PRODH	Product Register High Byte								xxxx xxxx	uuuu uuuu
FF3h	PRODL	Product Register Low Byte								xxxx xxxx	uuuu uuuu
FF2h	INTCON	GIE/GIEH	PEIE/GIEL	T0IE	INT0E	RBIE	T0IF	INT0F	RBIF	0000 000x	0000 000u
FF1h	INTCON2	RBPU	INTEDG0	—	—	—	T0IP	—	RBIP	11-- -1-1	11-- -1-1
FEFh	INDF0	Uses contents of FSR0 to address data memory - value of FSR0 not changed (not a physical register)								N/A	N/A
FEEh	POSTINC0	Uses contents of FSR0 to address data memory - value of FSR0 post-incremented (not a physical register)								N/A	N/A
FEDh	POSTDEC0	Uses contents of FSR0 to address data memory - value of FSR0 post-decremented (not a physical register)								N/A	N/A
FECh	PREINC0	Uses contents of FSR0 to address data memory - value of FSR0 pre-incremented (not a physical register)								N/A	N/A
FEBh	PLUSW0	Uses contents of FSR0 to address data memory - value of FSR0 pre-incremented (not a physical register) - value of FSR0 offset by W								N/A	N/A
FEAh	FSR0H	—	—	—	—	Indirect Data Memory Address Pointer 0 High				---- 0000	---- 0000
FE9h	FSR0L	Indirect Data Memory Address Pointer 0 Low Byte								xxxx xxxx	uuuu uuuu
FE8h	WREG	Working Register								xxxx xxxx	uuuu uuuu
FE7h	INDF1	Uses contents of FSR1 to address data memory - value of FSR1 not changed (not a physical register)								N/A	N/A
FE6h	POSTINC1	Uses contents of FSR1 to address data memory - value of FSR1 post-incremented (not a physical register)								N/A	N/A
FE5h	POSTDEC1	Uses contents of FSR1 to address data memory - value of FSR1 post-decremented (not a physical register)								N/A	N/A
FE4h	PREINC1	Uses contents of FSR1 to address data memory - value of FSR1 pre-incremented (not a physical register)								N/A	N/A
FE3h	PLUSW1	Uses contents of FSR1 to address data memory - value of FSR1 pre-incremented (not a physical register) - value of FSR1 offset by W								N/A	N/A
FE2h	FSR1H	—	—	—	—	Indirect Data Memory Address Pointer 1 High				---- 0000	---- 0000
FE1h	FSR1L	Indirect Data Memory Address Pointer 1 Low Byte								xxxx xxxx	uuuu uuuu
FE0h	BSR	—	—	—	—	Bank Select Register				---- 0000	---- 0000
FDFh	INDF2	Uses contents of FSR2 to address data memory - value of FSR2 not changed (not a physical register)								N/A	N/A
FDEh	POSTINC2	Uses contents of FSR2 to address data memory - value of FSR2 post-incremented (not a physical register)								N/A	N/A
FDDh	POSTDEC2	Uses contents of FSR2 to address data memory - value of FSR2 post-decremented (not a physical register)								N/A	N/A
FDCh	PREINC2	Uses contents of FSR2 to address data memory - value of FSR2 pre-incremented (not a physical register)								N/A	N/A
FDBh	PLUSW2	Uses contents of FSR2 to address data memory - value of FSR2 pre-incremented (not a physical register) - value of FSR2 offset by W								N/A	N/A
FDAh	FSR2H	—	—	—	—	Indirect Data Memory Address Pointer 2 High				---- 0000	---- 0000
FD9h	FSR2L	Indirect Data Memory Address Pointer 2 Low Byte								xxxx xxxx	uuuu uuuu
FD8h	STATUS	—	—	—	N	OV	Z	DC	C	---x xxxx	---u uuuu
FD7h	TMR0H	Timer0 Register High Byte								0000 0000	0000 0000
FD6h	TMR0L	Timer0 Register Low Byte								xxxx xxxx	uuuu uuuu
FD5h	T0CON	TMR0ON	T08BIT	T0CS	T0SE	T0PS3	T0PS2	T0PS1	T0PS0	1111 1111	1111 1111

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

Note 1: These registers can only be modified when the combination lock is open.

TABLE 4-1: REGISTER FILE SUMMARY (PIC18F010/020) (CONTINUED)

File 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 (Note 1)
FD3h	OSCCON	—	IRCF2	IRCF1	IRCF0	OSTO	IESO	—	SCS	-000 00-0	-qqq qq-q
FD2h	LVDCON	—	—	BGST	LVDEN	LVV3	LVV2	LVV1	LVV0	--00 0101	--00 0101
FD1h	WDTCON	—	—	—	—	SWP2	SWP1	SWP0	SWDTE	---- 0000	---- 0000
FD0h	RCON	$\overline{IPE}$	—	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$	0--1 11qq	0--q qquu
FB0h	PSPCON	—	—	—	—	—	—	CMLK1	CMLK0	---- --00	---- --00
FA9h	EEADR	EEPROM Address Register								xxxx xxxx	uuuu uuuu
FA8h	EEDATA	EEPROM Data Register								xxxx xxxx	uuuu uuuu
FA7h	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	---- ----
FA6h	EECON1	EEPGD	—	—	FREE	WRERR	WREN	WR	RD	x--0 x000	u--0 u000
FA2h	IPR2	—	—	—	EEIP	—	LVDIP	—	—	---1 -1--	---1 -1--
FA1h	PIR2	—	—	—	EEIF	—	LVDIF	—	—	---0 -0--	---0 -0--
FA0h	PIE2	—	—	—	EEIE	—	LVDIE	—	—	---0 -0--	---0 -0--
F9Bh	OSCTUNE	—	—	TUN5	TUN4	TUN3	TUN2	TUN1	TUN0	--00 0000	--qq qqqq
F93h	TRISB	—	—	Data Direction Control Register for PORTB						--11 1111	1111 1111
F8Ah	LATB	—	—	Read PORTB Data Latch, Write PORTB Data Latch						--xx xxxx	uuuu uuuu
F81h	PORTB	—	—	Read PORTB pins, Write PORTB Data Latch						--xx xxxx	uuuu uuuu
F79h	WPUB	—	—	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0	--11 1111	0011 1111
F78h	IOCB	—	—	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0	--00 0000	0000 0000

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

Note 1: These registers can only be modified when the combination lock is open.

4.10 Access Bank

The Access Bank is an architectural enhancement which is very useful for C compiler code optimization. The techniques used by the C compiler may also be useful for programs written in assembly.

This data memory region can be used for:

- Intermediate computational values
- Local variables of subroutines
- Faster context saving/switching of variables
- Common variables
- Faster evaluation/control of SFRs (no banking)

The Access Bank is comprised of the upper 128 bytes in Bank 15 (SFRs) and the lower 128 bytes in Bank 0. These two sections will be referred to as Access RAM High and Access RAM Low, respectively. Figure 4-6 and Figure 4-7 indicate the Access RAM areas.

A bit in the instruction word specifies if the operation is to occur in the bank specified by the BSR register, or in the Access Bank. This bit is denoted by the 'a' bit (for access bit).

When forced in the Access Bank (a = '0'), the last address in Access RAM Low is followed by the first address in Access RAM High. Access RAM High maps the Special Function registers, so that these registers can be accessed without any software overhead. This is useful for testing status flags and modifying control bits.

4.11 Bank Select Register (BSR)

The need for a large general purpose memory space dictates a RAM banking scheme. The data memory is partitioned into sixteen banks. When using direct addressing, the BSR should be configured for the desired bank.

BSR<3:0> holds the upper 4 bits of the 12-bit RAM address. The BSR<7:4> bits will always read '0's, and writes will have no effect.

A `MOVLB` instruction has been provided in the instruction set to assist in selecting banks.

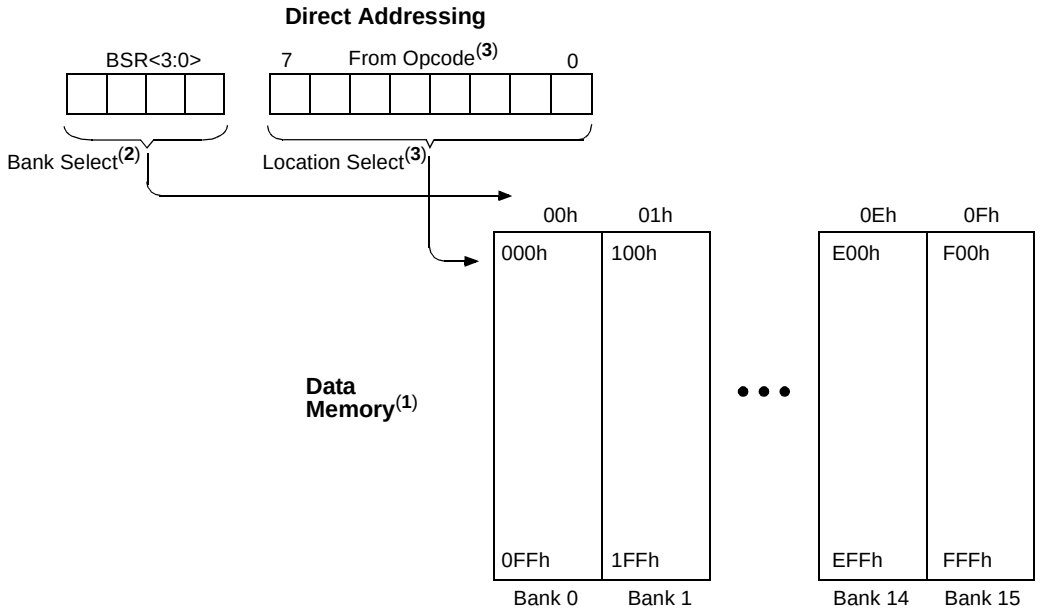
If the currently selected bank is not implemented, any read will return all '0's and all writes are ignored. The STATUS register bits will be set/cleared as appropriate for the instruction performed.

Each Bank extends up to FFh (256 bytes). All data memory is implemented as static RAM.

A `MOVFF` instruction ignores the BSR, since the 12-bit addresses are embedded into the instruction word.

Section 4.12 provides a description of indirect addressing, which allows linear addressing of the entire RAM space.

FIGURE 4-9: DIRECT ADDRESSING



Note 1: For register file map detail, see Table 4-7.

2: The access bit of the instruction can be used to force an override of the selected bank (BSR<3:0>) to the registers of the Access Bank.

3: The `MOVFF` instruction embeds the entire 12-bit address in the instruction.

4.12 Indirect Addressing, INDF and FSR Registers

Indirect addressing is a mode of addressing data memory, where the data memory address in the instruction is not fixed. An SFR register is used as a pointer to the data memory location that is to be read or written. Since this pointer is in RAM, the contents can be modified by the program. This can be useful for data tables in the data memory and for software stacks. Figure 4-10 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.

Indirect addressing is possible by using one of the INDF registers. Any instruction using the INDF register actually accesses the register pointed to by the File Select Register, FSR. Reading the INDF register itself indirectly (FSR = '0') will read 00h. Writing to the INDF register indirectly results in a no-operation. The FSR register contains a 12-bit address, which is shown in Figure 4-10.

The INDFn register is not a physical register. Addressing INDFn actually addresses the register whose address is contained in the FSRn register (FSRn is a pointer). This is indirect addressing.

There are three indirect addressing registers. To address the entire data memory space (4096 bytes), these registers are 12-bit wide. To store the 12-bits of addressing information, two 8-bit registers are required. These indirect addressing registers are:

1. FSR0: composed of FSR0H:FSR0L
2. FSR1: composed of FSR1H:FSR1L
3. FSR2: composed of FSR2H:FSR2L

In addition, there are registers INDF0, INDF1 and INDF2, which 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.

If an instruction writes a value to INDF0, the value will be written to the address pointed to by FSR0H:FSR0L. A read from INDF1, reads the data from the address pointed to by FSR1H:FSR1L. INDFn can be used in code anywhere an operand can be used.

If INDF0, INDF1, or INDF2 are read indirectly via an FSR, all '0's are read (zero bit is set). Similarly, if INDF0, INDF1, or INDF2 are written to indirectly, the operation will be equivalent to a `NOP` instruction and the STATUS bits are not affected.

4.12.1 INDIRECT ADDRESSING OPERATION

Each FSR register has an INDF register associated with it, plus four additional register addresses. Performing an operation on one of these five registers determines how the FSR will be modified during indirect addressing.

When data access is done to one of the five INDFn locations, the address selected will configure the FSRn register to:

- Do nothing to FSRn after an indirect access (no change) - INDFn
- Auto-decrement FSRn after an indirect access (post-decrement) - POSTDECn
- Auto-increment FSRn after an indirect access (post-increment) - POSTINCn
- Auto-increment FSRn before an indirect access (pre-increment) - PREINCn
- Use the signed value of WREG as an offset to FSRn. Do not modify the value of the WREG or the FSRn register after an indirect access (no change) - PLUSWn

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

Incrementing or decrementing an FSR affects all 12 bits. That is, when FSRnL overflows from an increment, FSRnH will be incremented automatically.

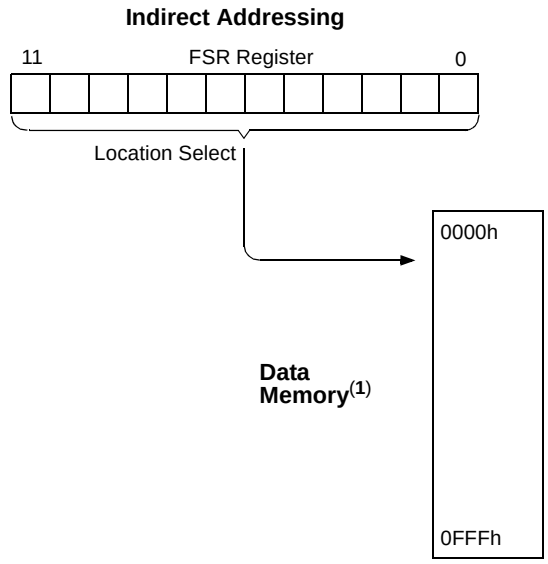
Adding these features allows the FSRn to be used as a stack pointer, in addition to its uses for table operations in data memory.

Each FSR has an address associated with it that performs an indexed indirect access. When a data access to this INDFn location (PLUSWn) occurs, the FSRn is configured to add the signed value in the WREG register and the value in FSR to form the address before an indirect access. The FSR value is not changed.

If an FSR register contains a value that points to one of the INDFn, an indirect read will read 00h (zero bit is set), while an indirect write will be equivalent to a `NOP` (STATUS bits are not affected).

If an indirect addressing operation is done where the target address is an FSRnH or FSRnL register, the write operation will dominate over the pre- or post-increment/decrement functions.

FIGURE 4-10: INDIRECT ADDRESSING



Note 1: For register file map detail, see Table 4-7.

4.13 STATUS Register

The STATUS register, shown in Register 4-2, contains the arithmetic status of the ALU. The STATUS register can be the destination for any instruction, as with any other register. If the STATUS register is the destination for an instruction that affects the Z, DC, C, OV, or N bits, then the write to these five bits is disabled. These bits are set or cleared according to the device logic. Therefore, the result of an instruction with the STATUS register as destination may be different than intended.

For example, `CLRF STATUS` will clear the upper three bits and set the Z bit. This leaves the STATUS register as `000u uluu` (where `u` = unchanged).

It is recommended, therefore, that only `BCF, BSF, SWAPF, MOVFF` and `MOVWF` instructions are used to alter the STATUS register, because these instructions do not affect the `Z, C, DC, OV` or `N` bits from the STATUS register. For other instructions not affecting any status bits, see Table 13-2.

Note:	The C and DC bits operate as a $\overline{\text{borrow}}$ and digit $\overline{\text{borrow}}$ bit respectively, in subtraction.

REGISTER 4-2: STATUS REGISTER

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC	C
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **N:** Negative bit
This bit is used for signed arithmetic (2's complement). It indicates whether the result was negative, (ALU MSB = 1).
`1` = Result was negative
`0` = Result was positive

bit 3 **OV:** Overflow bit
This bit is used for signed arithmetic (2's complement). It indicates an overflow of the 7-bit magnitude, which causes the sign bit (bit 7) to change state.
`1` = Overflow occurred for signed arithmetic (in this arithmetic operation)
`0` = No overflow occurred

bit 2 **Z:** Zero bit
`1` = The result of an arithmetic or logic operation is zero
`0` = The result of an arithmetic or logic operation is not zero

bit 1 **DC:** Digit carry/$\overline{\text{borrow}}$ bit
For `ADDWF, ADDLW, SUBLW`, and `SUBWF` instructions
`1` = A carry-out from the 4th low order bit of the result occurred
`0` = No carry-out from the 4th low order bit of the result

Note: For $\overline{\text{borrow}}$, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the bit 4 or bit 3 of the source register.

bit 0 **C:** Carry/$\overline{\text{borrow}}$ bit
For `ADDWF, ADDLW, SUBLW`, and `SUBWF` instructions
`1` = A carry-out from the most significant bit of the result occurred
`0` = No carry-out from the most significant bit of the result occurred

Note: For $\overline{\text{borrow}}$, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF, RLF`) instructions, this bit is loaded with either the high or low order bit of the source register.

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

4.14 RCON Register

The RESET Control (RCON) register contains flag bits, that allow differentiation between the sources of a device RESET. These flags include the $\overline{TO}$, $\overline{PD}$, $\overline{POR}$, $\overline{BOR}$ and $\overline{RI}$ bits. This register is readable and writable.

Note 1: If the BOREN configuration bit is set, $\overline{BOR}$ is '1' on Power-on Reset. If the BOREN configuration bit is clear, $\overline{BOR}$ is unknown on Power-on Reset.
The $\overline{BOR}$ status bit is a "don't care" and is not necessarily predictable if the brown-out circuit is disabled (the BOREN configuration bit is clear). $\overline{BOR}$ must then be set by the user and checked on subsequent RESETS to see if it is clear, indicating a brown-out has occurred.

2: It is recommended that the $\overline{POR}$ bit be set after a Power-on Reset has been detected, so that subsequent Power-on Resets may be detected.

REGISTER 4-3: RCON REGISTER

R/W-0	U-0	U-0	R/W-1	R-1	R-1	R/W-0	R/W-0
IPEN	—	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

bit 7 **IPEN:** Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (16CXXX compatibility mode)

bit 6-5 **Unimplemented:** Read as '0'

bit 4 **$\overline{RI}$:** RESET Instruction Flag bit
1 = The RESET instruction was not executed
0 = The RESET instruction was executed causing a device RESET
(must be set in software after a Brown-out Reset occurs)

bit 3 **$\overline{TO}$:** Watchdog Time-out Flag bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

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

bit 1 **$\overline{POR}$:** Power-on Reset Status bit
1 = A Power-on Reset has not occurred
0 = A Power-on Reset occurred
(must be set in software after a Power-on Reset occurs)

bit 0 **$\overline{BOR}$:** Brown-out Reset Status bit
1 = A Brown-out Reset has not occurred
0 = A Brown-out Reset occurred
(must be set in software after a Brown-out Reset occurs)

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

NOTES:

5.0 DATA EEPROM MEMORY

The Data EEPROM is readable and writable during normal operation (full VDD range). This memory is not directly mapped in the register file space. Instead, it is indirectly addressed through the Special Function Registers. There are four SFRs used to read and write this memory. These registers are:

- EECON1 (0FA6h)
- EECON2 (0FA7h)
- EEDATA (0FA8h)
- EEADR (0FA9h)

When interfacing the data memory block, EEDATA holds the 8-bit data for read/write, and EEADR holds the address of the EEPROM location being accessed. These devices have 64 bytes of data EEPROM with an address range from 0h to 03Fh.

The EEPROM data memory allows byte read and write. A byte write automatically erases the location and writes the new data (erase before write).

The EEPROM data memory is rated for high erase/write cycles. The write time is controlled by an on-chip timer. The write time will vary with voltage and temperature, as well as from chip-to-chip. Please refer to the specifications for exact limits.

When the device is code protected, the CPU may continue to read and write the data EEPROM memory.

5.1 EEADR

The EEADR register can address up to a maximum of 256 bytes of data.

When the device contains less memory than the full address reach of the EEADR register, the MSb's of the register must be set to '0'. For example, this device has 64 bytes of data EE, the Most Significant 2 bits of the register must be '0'.

5.2 EECON1 and EECON2 Registers

EECON1 is the control register for memory accesses.

EECON2 is not a physical register. Reading EECON2 will read all '0's. The EECON2 register is used exclusively in the memory write sequence.

Control bit EEPGD determines if the access will be a program or a data memory access. When clear, any subsequent operations will operate on the data memory. When set, any subsequent operations will operate on the program memory.

Control bits RD and WR initiate read and write operations, respectively. These bits cannot be cleared, only set, in software. They are cleared in hardware at the completion of the read or write operation. The inability to clear the WR bit in software prevents the accidental or premature termination of a write operation.

The WREN bit, when set, will allow a write operation. On power-up, the WREN bit is clear. The WRERR bit is set when a write operation is interrupted by a $\overline{\text{MCLR}}$ Reset, or a WDT Time-out Reset, during normal operation. In these situations, following RESET, the user can check the WRERR bit and rewrite the location. The value of the data and address registers and the EEPGD bit remains unchanged.

Interrupt flag bit EEIF in the PIR2 register, is set when a write is complete. It must be cleared in software.

REGISTER 5-1: EECON1 REGISTER (ADDRESS 18Ch)

R/W-U	U-0	U-0	R/W-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	—	—	FREE	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7 **EEPGD:** FLASH Program or Data EEPROM Memory Select bit
1 = Access Program FLASH memory
0 = Access Data EEPROM memory

bit 6-5 **Unimplemented:** Read as '0'

bit 4 **FREE:** FLASH Row Erase Enable bit
1 = Erase the row addressed by TBLPTR on the next WR command (reset by hardware)
0 = Perform write only

bit 3 **WRERR**: EEPROM Error Flag bit
1 = A write operation is prematurely terminated
(any $\overline{\text{MCLR}}$ Reset or any WDT Reset during normal operation)
0 = The write operation completed

bit 2 **WREN**: EEPROM Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1 **WR**: Write Control bit
1 = Initiates a write cycle. (The bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.)
0 = Write cycle to the EEPROM is complete

bit 0 **RD**: Read Control bit
1 = Initiates an EEPROM read. (Read takes one cycle. RD is cleared in hardware. The RD bit can only be set (not cleared) in software.)
0 = Does not initiate an EEPROM read

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

5.3 Reading the Data EEPROM Memory

To read a data memory location, the user must write the address to the EEADR register, clear the EEPGD control bit (EECON1<7>), and then set control bit RD (EECON1<0>). The data is available in the very next instruction cycle of the EEDATA register, therefore, it can be read by the next instruction. EEDATA will hold this value until another read operation or until it is written to by the user (during a write operation).

EXAMPLE 5-1: DATA EEPROM READ

```
MOVLW DATA_EE_ADDR  ;
MOVWF EEADR         ;Data Memory Address to read
BCF   EECON1, EEPGD ;Point to DATA memory
BSF   EECON1, RD    ;EEPROM Read
MOVF  EEDATA, W     ;W = EEDATA
```

5.4 Writing to the Data EEPROM Memory

To write an EEPROM data location, the address must first be written to the EEADR register and the data written to the EEDATA register. Then the sequence in Example 5-2 must be followed to initiate the write cycle.

Note:	Do not write to program memory or EECON1 while writing to EEDATA.

EXAMPLE 5-2: DATA EEPROM WRITE

```
            MOVLW  DATA_EE_ADDR  ;
            MOVWF  EEADR         ; Data Memory
                                   Address to write
            MOVLW  DATA_EE_DATA  ;
            MOVWF  EEDATA        ; Data Memory
                                   Value to write
            BCF    EECON1, EEPGD ; Point to DATA
                                   memory
            BSF    EECON1, WREN  ; Enable writes

            BCF    INTCON, GIE   ; Disable
                                   Interrupts
            MOVLW  55h           ;
Required    MOVWF  EECON2        ; Write 55h
Sequence    MOVLW  AAh           ;
            MOVWF  EECON2        ; Write AAh
            BSF    EECON1, WR    ; Set WR bit to
                                   begin write
            BSF    INTCON, GIE   ; Enable
                                   Interrupts

            SLEEP                ; Wait for
                                   interrupt to
                                   signal write
                                   complete
            BCF    EECON1, WREN  ; Disable writes
```

The write will not initiate if the above required sequence is not exactly followed (write 55h to EECON2, write AAh to EECON2, then set WR bit) for each byte. It is strongly recommended that interrupts be disabled during this code segment.

Additionally, the WREN bit in EECON1 must be set to enable writes. This mechanism prevents accidental writes to data EEPROM due to unexpected code execution (i.e., runaway programs). The WREN bit should be kept clear at all times, except when updating the EEPROM. The WREN bit is not cleared by hardware

After a write sequence has been initiated, clearing the WREN bit will not affect the current write cycle. The WR bit will be inhibited from being set unless the WREN bit is set. The WREN bit must be set on a previous instruction. Both WR and WREN cannot be set with the same instruction.

At the completion of the write cycle, the WR bit is cleared in hardware and the EEPROM Write Complete Interrupt Flag bit (EEIF) is set. EEIF must be cleared by software.

5.5 Protection Against Spurious Write

5.5.1 EEPROM DATA MEMORY

There are conditions when the device may not want to write to the data EEPROM memory. To protect against spurious EEPROM writes, various mechanisms have been built-in. On power-up, the WREN bit is cleared. Also, the Power-up Timer (72 ms duration) prevents EEPROM write.

The write initiate sequence and the WREN bit together help prevent an accidental write during brown-out, power glitch, or software malfunction.

5.6 Operation During Code Protect

Each reprogrammable memory block has its own code protect mechanism. External Read and Write operations are disabled if either of these mechanisms are enabled.

5.6.1 DATA EEPROM MEMORY

The microcontroller itself can both read and write to the internal Data EEPROM, regardless of the state of the code protect configuration bit.

TABLE 5-1: REGISTERS ASSOCIATED WITH DATA EEPROM/PROGRAM FLASH

Address	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
FA9h	EEADR	EEPROM Address Register								xxxx xxxx	uuuu uuuu
FA8h	EEDATA	EEPROM Data Register								xxxx xxxx	uuuu uuuu
FA7h	EECON2	EEPROM Control Register2 (not a physical register)								---- ----	---- ----
FA6h	EECON1	EEPGD	—	—	FREE	WRERR	WREN	WR	RD	x--0 x000	u--0 u000
FA2h	IPR2	—	—	—	EEIP	—	LVDIP	—	—	---1 1---	---1 1---
FA1h	PIR2	—	—	—	EEIF	—	LVDIF	—	—	---0 0---	---0 0---
FA0h	PIE2	—	—	—	EEIE	—	LVDIE	—	—	---0 0---	---0 0---
FF2h	INTCON	GIE/GIEH	PEIE/GIEL	T0IE	INT0IE	RBIE	T0IF	INT0F	RBIF	0000 000x	0000 000u

Legend: x = unknown, u = unchanged, r = reserved, - = unimplemented, read as '0'.
Shaded cells are not used during FLASH/EEPROM access.

Note 1: These bits are reserved; always maintain these bits clear.

6.0 TABLE READ/WRITE INSTRUCTIONS

The PIC18F010/020 has eight instructions that allow the processor to move data from the data memory space to the program memory space, and vice versa. These eight instructions manipulate the Table Pointer in a manner similar to the FSR's.

The TBLRD instructions are used to read data from the program memory space to the data memory space. The TBLWT instructions are used to write data from the data memory space to the program memory space.

6.1 Control Registers

A few control registers are used in conjunction with the TBLRD and TBLWT instructions. These include the:

- EECON1 register
- TABLAT register
- TBLPTR registers

6.1.1 EECON1 REGISTER

The EECON1 register holds bits to control erase and write operations in FLASH memory. The EEPGD bit selects data EEPROM, if clear, or program FLASH memory, if set. The FREE bit is used to select erasing versus writing to FLASH. The WREN bit enables writing. Finally, the WRERR bit indicates any errors. Refer to Register 5-1 for details.

6.2 Table Reads from FLASH Program Memory

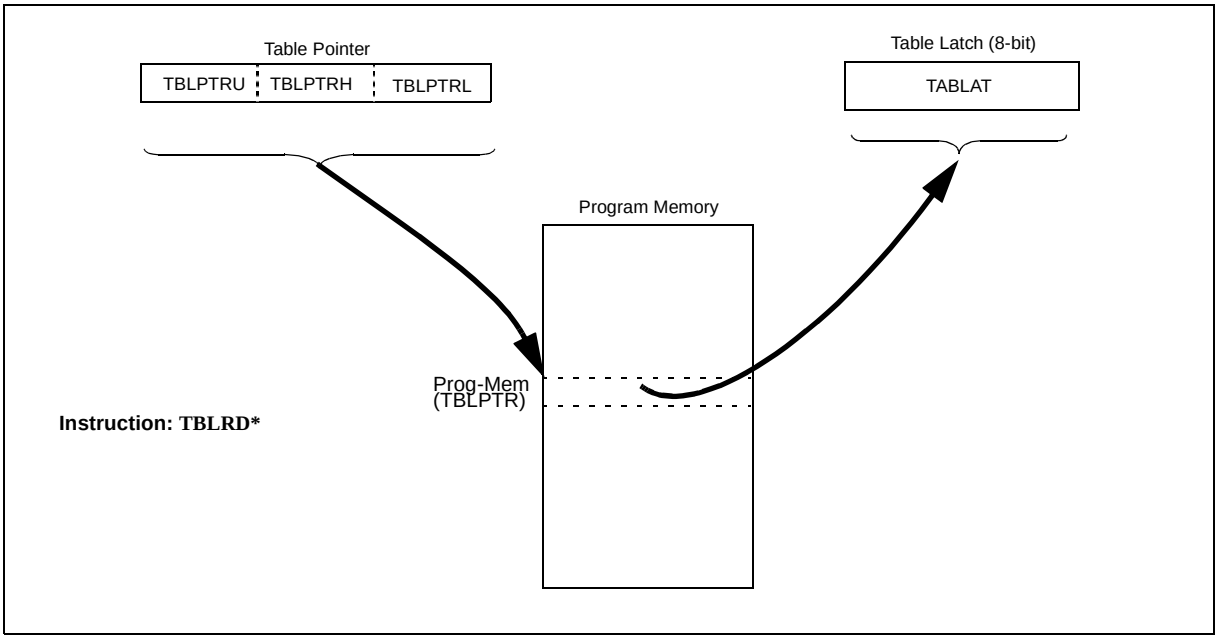
Table Reads from program memory are performed one byte at a time. The instruction will access one byte from the program memory pointed to by the TBLPTR and transfer that byte to the TABLAT. Figure 6-1 diagrams the Table Read operation.

The TBLPTR can be updated in one of four ways, based on the Table Read instructions:

- TBLRD* no-change
- TBLRD*+ post-increment
- TBLRD*- post-decrement
- TBLRD+* pre-increment

The internal program memory is normally word wide. The Least Significant bit of the address selects between the high and low bytes of the word. Figure 6-2 shows the typical interface between the internal program memory and the TABLAT.

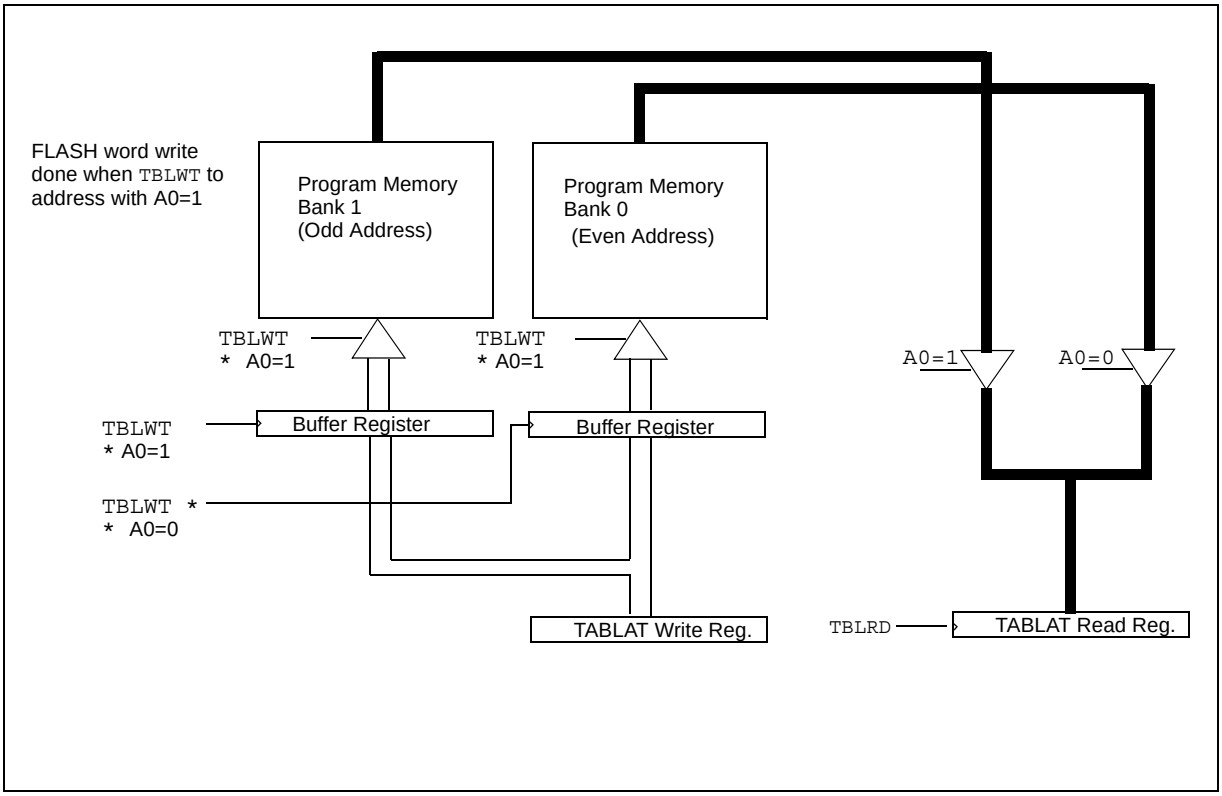
FIGURE 6-1: TBLRD* INSTRUCTION OPERATION



EXAMPLE 6-1: PROGRAM MEMORY READ

```
MOVLW   CODE_ADDR_UPPER; Load TBLPTR
                       ; Register
MOVWF   TBLPTRU        ; with Address to
                       ; Read
MOVLW   CODE_ADDR_HIGH ;
MOVWF   TBLPTRH        ;
MOVLW   CODE_ADDR_LOW  ;
MOVWF   TBLPTRL        ;
TBLRD*                 ; Read Memory
MOVF    TABLAT,W       ; W = Data
```

FIGURE 6-2: TABLE READS / WRITES TO INTERNAL PROGRAM MEMORY



6.3 Erasing FLASH Program Memory

Word erase in the FLASH array is not supported. The minimum erase block is one row of a panel, which is equivalent to 16 words or 32 bytes.

Erase operations may be commanded from one of two sources. Under user program control, the minimum one row of memory is erased. Under programmer or ICSP™ control, larger blocks of program memory may be bulk erased.

6.3.1 ERASING FLASH PROGRAM MEMORY IN OPERATIONAL MODE

In normal mode, a block of 32 bytes of program memory is erased. The Most Significant 16 bits of the TBLPTR<21:6> points to the block being erased. TBLPTR<4:0> are ignored.

The EECON1 register commands the erase operation. The EEPGD bit must be set to point to the FLASH program memory. The WREN bit must be set to enable write operations. The FREE bit is set to select an erase operation.

For protection, the write initiate sequence for EECON2 must be used. When the WR bit is set, a long write is necessary for erasing the internal FLASH. Instruction execution is halted while in a long write cycle. The long write will be terminated by the internal programming timer. Instruction execution will resume with no lost instructions.

The sequence of events for erasing a block of internal program memory location is:

1. Load Table Pointer with address of row being erased.
2. Set FREE bit to enable row erase; set WREN bit to enable writes and set EEPGD bit to point to program memory.
3. Disable interrupts.
4. Write '55' to EECON2.
5. Write 'AA' to EECON2.
6. Set the WR bit. This will begin the row erase cycle.
7. CPU will stall for duration of the erase (about 2ms using internal timer).

6.4 FLASH Array Programming Operations

Word or byte programming is not supported. The minimum programming block is 32-bits or 2 words.

6.4.1 PROGRAMMING FLASH PROGRAM MEMORY IN OPERATIONAL MODE (TABLE LONG WRITES)

Conceptually, Table Writes are performed one byte at a time. The instruction will write one byte contained in the TABLAT register to the internal memory, pointed to by the TBLPTR, as shown in Figure 6-3.

The TBLPTR can be updated in one of four ways, based on the Table Write instructions:

- `TBLWT*` no-change
- `TBLWT*+` post-increment
- `TBLWT*-` post-decrement
- `TBLWT+*` pre-increment

The program memory FLASH uses a similar mechanism to the data EEPROM. Table Writes are used internally to load the Write registers used to program the FLASH memory. The EECON1 register is used to actually command a write or erase event.

Each FLASH panel is programmed with 32 of 256 columns at a time. This translates into 32 write bit latches. These write latches are accessed using Table Write instructions, which can write a byte at a time. There are then 4 Table Writes required to write the latches for one panel.

Since the table latch is only a single byte, the `TBLWT` instruction has to be executed 4 times for each programming operation. All of the Table Write operations will essentially be short writes, because only the table latches are written. At the end of updating 4 latches, the EECON1 register must be written to start the programming operation with a long write.

The long write is necessary for programming the internal FLASH. Instruction execution is halted while in a long write cycle. The long write will be terminated by the internal programming timer. Instruction execution will resume with two lost instructions.

The write time is controlled by the EEPROM on-chip timer. The write/erase voltages are generated by an on-chip charge pump, rated to operate over the voltage range of the device for byte or word operations. When doing block operations, the device must be operating in the 5V ±10% range.

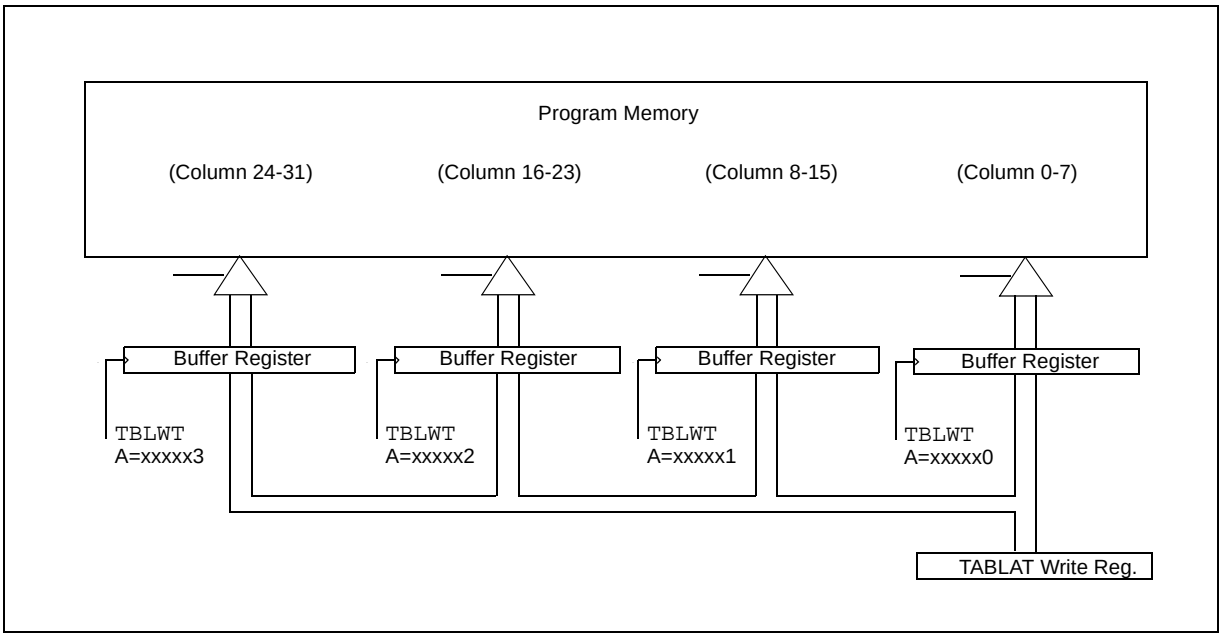
Note:	When writing a block, insure the table pointer is pointing to the desired block after the last short write. The first and second instruction following the `TBLWT` must be `NOP`s.

The sequence of events for programming an internal program memory location should be:

1. Read 32 bytes of row into RAM.
2. Update data values in RAM, as necessary.
3. Load Table Pointer with address of row being erased.
4. Perform the row erase procedure.
5. CPU will stall for duration of the erase (about 2ms using internal timer).
6. Load Table Pointer with address first byte of row being written.
7. Set WREN bit to enable writes and set EEPGD bit to point to program memory.
8. Write first 3 bytes into table latches with auto-increment. Write the last byte without auto-increment.
9. Disable interrupts.
10. Write ’55’ to EECON2.
11. Write 'AA’ to EECON2.
12. Set the WR bit. This will begin the write cycle.
13. CPU will stall for duration of the write (about 2ms using internal timer).
14. Repeat steps 7-13, 8 times total to write 32 bytes.
15. Verify the memory row (Table Read).

This procedure will require about 18msec to update 1 row of 32 bytes of memory.

FIGURE 6-3: TABLE WRITES TO INTERNAL PROGRAM MEMORY



EXAMPLE 6-2: PROGRAM MEMORY WRITE

This example will buffer a segment of memory, modify one word in the buffer, erase the segment row, and write the buffer back to memory.

```
    MOVLW   32              ; number of bytes in row
    MOVWF   COUNTER
    MOVLW   BUFFER_ADDR_HIGH; point to buffer
    MOVWF   FSR0H
    MOVLW   BUFFER_ADDR_LOW ;
    MOVWF   FSR0L
    MOVLW   CODE_ADDR_UPPER ; Load TBLPTR with the base
    MOVWF   TBLPTRU         ; address of the memory row
    MOVLW   CODE_ADDR_HIGH  ;
    MOVWF   TBLPTRH         ;
    MOVLW   CODE_ADDR_LOW   ;
    MOVWF   TBLPTRL         ;
READ_ROW
    TBLRD*+                 ; read into TABLAT, and inc
    MOVF    TABLAT, W       ; get data
    MOVWF   POSTINC0        ; store data
    DECFSZ  COUNTER         ; done?
    GOTO    READ_ROW        ; repeat
MODIFY_WORD
    MOVLW   DATA_ADDR_HIGH  ; point to buffer
    MOVWF   FSR0H
    MOVLW   DATA_ADDR_LOW   ;
    MOVWF   FSR0L
    MOVLW   NEW_DATA_LOW    ; update buffer word
    MOVWF   POSTINC0
    MOVLW   NEW_DATA_HIGH
    MOVWF   INDF0
ERASE_ROW
    MOVLW   CODE_ADDR_UPPER ; Load TBLPTR with the base
    MOVWF   TBLPTRU         ; address of the memory row
    MOVLW   CODE_ADDR_HIGH  ;
    MOVWF   TBLPTRH         ;
    MOVLW   CODE_ADDR_LOW   ;
    MOVWF   TBLPTRL         ;
    BSF     EECON1,WREN     ; enable write to memory
    BSF     EECON1,FREE     ; Enable Row Erase operation
    BSF     EECON1,EEPGD    ; Point to FLASH program memory
    MOVLW   55h
    MOVWF   EECON2          ; write 55H
    MOVLW   AAh
    MOVWF   EECON2          ; write AAH
    BSF     EECON1,WR       ; start erase (CPU stall)
WRITE_BUFFER_BACK
    MOVLW   8               ; number of write buffer groups of 4 bytes
    MOVWF   COUNTER_HI
    MOVLW   BUFFER_ADDR_HIGH; point to buffer
    MOVWF   FSR0H
    MOVLW   BUFFER_ADDR_LOW ;
    MOVWF   FSR0L
    TBLRD*-                 ; back the TBLPTR up one
PROGRAM_LOOP
    MOVLW   4               ; number of bytes in write buffer
    MOVWF   COUNTER
```

EXAMPLE 6-2: PROGRAM MEMORY WRITE (CONTINUED)

```
WRITE_WORD_TO_BUFFERS
    MOVF    POSTINC0, W     ; get low byte of buffer data
    MOVWF   TABLAT          ; present data to table latch
    TBLWT+*                 ; write data, perform a short write to pre-increment and load data to
    NOP                     ; internal TBLWT holding register.
    NOP                     ; loop until buffers are full
    DECFSZ  COUNTER
    GOTO    WRITE_WORD_TO_BUFFERS
PROGRAM_MEMORY
    BSF     EECON1,WREN     ; enable write to memory
    BSF     EECON1,EEPGD    ; Point to FLASH program memory
    MOVLW   55h
    MOVWF   EECON2          ; write 55H
    MOVLW   AAh
    MOVWF   EECON2          ; write AAH
    BSF     EECON1,WR       ; start program (CPU stall)
    DECFSZ  COUNTER_HI      ; loop until done
    GOTO    PROGRAM_LOOP
    BCF     EECON1,WREN     ; disable write to memory
```


6.4.2 TABLAT - TABLE LATCH REGISTER

The Table Latch (TABLAT) is an 8-bit register mapped into the SFR space. The Table Latch is used to hold 8-bit data during data transfers between program memory and data memory.

6.4.3 TBLPTR - TABLE POINTER REGISTER

The Table Pointer (TBLPTR) addresses a byte within the program memory. The TBLPTR is comprised of three SFR registers (Table Pointer Upper byte, High byte and Low byte). These three registers (TBLPTRU:TBLPTRH:TBLPTRL) join to form a 22-bit wide pointer. The low order 21-bits allow the device to address up to 2 Mbytes of program memory space. The 22nd bit allows access to the Device ID, the User ID and the Configuration bits.

The table pointer TBLPTR is used by the `TBLRD` and `TBLWT` instructions. These instructions can update the TBLPTR in one of four ways, based on the table operation. These operations are shown in Table 6-1. These operations on the TBLPTR only affect the low order 21-bits.

TABLE 6-1: TABLE POINTER OPERATIONS WITH `TBLRD` AND `TBLWT` INSTRUCTIONS

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

NOTES:

7.0 8 X 8 HARDWARE MULTIPLIER

7.1 Introduction

An 8 x 8 hardware multiplier is included in the ALU of the PIC18F010/020 devices. By making the multiply a hardware operation, it completes in a single instruction cycle. This is an unsigned multiply that gives a 16-bit result. The result is stored into the 16-bit product register pair (PRODH:PRODL). The multiplier does not affect any flags in the ALUSTA register.

Making the 8 x 8 multiplier execute in a single cycle gives the following advantages:

- Higher computational throughput
- Reduces code size requirements for multiply algorithms

The performance increase allows the device to be used in applications previously reserved for Digital Signal Processors.

Table 7-1 shows a performance comparison between enhanced devices using the single cycle hardware multiply, and performing the same function without the hardware multiply.

TABLE 7-1: PERFORMANCE COMPARISON

Routine	Multiply Method	Program Memory (Words)	Cycles (Max)	Time		
				@ 40 MHz	@ 10 MHz	@ 4 MHz
8 x 8 unsigned	Without hardware multiply	13	69	6.9 µs	27.6 µs	69 µs
	Hardware multiply	1	1	100 ns	400 ns	1 µs
8 x 8 signed	Without hardware multiply	33	91	9.1 µs	36.4 µs	91 µs
	Hardware multiply	6	6	600 ns	2.4 µs	6 µs
16 x 16 unsigned	Without hardware multiply	21	242	24.2 µs	96.8 µs	242 µs
	Hardware multiply	24	24	2.4 µs	9.6 µs	24 µs
16 x 16 signed	Without hardware multiply	52	254	25.4 µs	102.6 µs	254 µs
	Hardware multiply	36	36	3.6 µs	14.4 µs	36 µs

7.2 Operation

Example 7-1 shows the sequence to do an 8 x 8 unsigned multiply. Only one instruction is required, when one argument of the multiply is already loaded in the WREG register.

Example 7-2 shows the sequence to do an 8 x 8 signed multiply. To account for the sign bits of the arguments, each argument's most significant bit (MSb) is tested and the appropriate subtractions are done.

EXAMPLE 7-1: 8 x 8 UNSIGNED MULTIPLY ROUTINE

```
MOVFF    ARG1, WREG  ;
MULWF    ARG2        ; ARG1 * ARG2 ->
                     ;   PRODH:PRODL
```

EXAMPLE 7-2: 8 x 8 SIGNED MULTIPLY ROUTINE

```
MOVFF    ARG1, WREG
MULWF    ARG2        ; ARG1 * ARG2 ->
                     ; PRODH:PRODL
BTFSC    ARG2, SB    ; Test Sign Bit
SUBWF    PRODH       ; PRODH = PRODH
                     ;         - ARG1
MOVFF    ARG2, WREG
BTFSC    ARG1, SB    ; Test Sign Bit
SUBWF    PRODH       ; PRODH = PRODH
                     ;         - ARG2
```

Example 7-3 shows the sequence to do a 16 x 16 unsigned multiply. Equation 7-1 shows the algorithm that is used. The 32-bit result is stored in 4 registers RES3:RES0.

EQUATION 7-1: 16 x 16 UNSIGNED MULTIPLICATION ALGORITHM

$$
\begin{aligned}
RES3:RES0 &= ARG1H:ARG1L \bullet ARG2H:ARG2L \\
&= (ARG1H \bullet ARG2H \bullet 2^{16}) + \\
&\quad (ARG1H \bullet ARG2L \bullet 2^{8}) + \\
&\quad (ARG1L \bullet ARG2H \bullet 2^{8}) + \\
&\quad (ARG1L \bullet ARG2L)
\end{aligned}
$$

EXAMPLE 7-3: 16 x 16 UNSIGNED MULTIPLY ROUTINE

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

Example 7-4 shows the sequence to do a 16 x 16 signed multiply. Equation 7-2 shows the algorithm 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 7-2: 16 x 16 SIGNED MULTIPLICATION ALGORITHM

$$
\begin{aligned}
RES3:RES0 &= ARG1H:ARG1L \bullet ARG2H:ARG2L \\
&= (ARG1H \bullet ARG2H \bullet 2^{16}) + \\
&\quad (ARG1H \bullet ARG2L \bullet 2^{8}) + \\
&\quad (ARG1L \bullet ARG2H \bullet 2^{8}) + \\
&\quad (ARG1L \bullet ARG2L) + \\
&\quad (-1 \bullet ARG2H\langle 7\rangle \bullet ARG1H:ARG1L \bullet 2^{16}) + \\
&\quad (-1 \bullet ARG1H\langle 7\rangle \bullet ARG2H:ARG2L \bullet 2^{16})
\end{aligned}
$$

EXAMPLE 7-4: 16 x 16 SIGNED MULTIPLY ROUTINE

```
    MOVFF    ARG1L, WREG
    MULWF    ARG2L       ; ARG1L * ARG2L ->
                         ;   PRODH:PRODL
    MOVFF    PRODH, RES1 ;
    MOVFF    PRODL, RES0 ;
;
    MOVFF    ARG1H, WREG
    MULWF    ARG2H       ; ARG1H * ARG2H ->
                         ;   PRODH:PRODL
    MOVFF    PRODH, RES3 ;
    MOVFF    PRODL, RES2 ;
;
    MOVFF    ARG1L, WREG
    MULWF    ARG2H       ; ARG1L * ARG2H ->
                         ;   PRODH:PRODL
    MOVFF    PRODL, WREG ;
    ADDWF    RES1        ; Add cross
    MOVFF    PRODH, WREG ;    products
    ADDWFC   RES2        ;
    CLRF     WREG        ;
    ADDWFC   RES3        ;
;
    MOVFF    ARG1H, WREG ;
    MULWF    ARG2L       ; ARG1H * ARG2L ->
                         ;   PRODH:PRODL
    MOVFF    PRODL, WREG ;
    ADDWF    RES1        ; Add cross
    MOVFF    PRODH, WREG ;    products
    ADDWFC   RES2        ;
    CLRF     WREG        ;
    ADDWFC   RES3        ;
;
    BTFSS    ARG2H, 7    ; ARG2H:ARG2L neg?
    GOTO     SIGN_ARG1   ; no, check ARG1
    MOVFF    ARG1L, WREG ;
    SUBWF    RES2        ;
    MOVFF    ARG1H, WREG ;
    SUBWFB   RES3
;
SIGN_ARG1
    BTFSS    ARG1H, 7    ; ARG1H:ARG1L neg?
    GOTO     CONT_CODE   ; no, done
    MOVFF    ARG2L, WREG ;
    SUBWF    RES2        ;
    MOVFF    ARG2H, WREG ;
    SUBWFB   RES3
;
CONT_CODE
     :
```

NOTES:

8.0 INTERRUPTS

The PIC18F010/020 devices have multiple interrupt sources and an interrupt priority feature that allows each interrupt source to be assigned a high priority level, or a low priority level. The high priority interrupt vector is at 000008h and the low priority interrupt vector is at 000018h. High priority interrupt events will override any low priority interrupts that may be in progress.

There are six registers which are used to control interrupt operation. These registers are:

- RCON
- INTCON
- INTCON2
- PIR2
- PIE2
- IPR2

It is recommended that the Microchip header files supplied with MPLAB® IDE be used for the symbolic bit names in these registers. This allows the assembler/compiler to automatically take care of the placement of these bits within the specified register.

Each interrupt source has three bits to control its operation. The functions of these bits are:

- Flag bit to indicate that an interrupt event occurred
- Enable bit that allows program execution to branch to the interrupt vector address when the flag bit is set
- Priority bit to select high priority or low priority

The interrupt priority feature is enabled by setting the IPEN bit (RCON<7>). When interrupt priority is enabled, there are two bits which enable interrupts globally. Setting the GIEH bit (INTCON<7>), enables all interrupts that have the priority bit set. Setting the GIEL bit (INTCON<6>), enables all interrupts that have the priority bit cleared. When the interrupt flag, enable bit and appropriate global interrupt enable bit are set, the interrupt will vector immediately to address 000008h or 000018h, depending on the priority level. Individual interrupts can be disabled through their corresponding enable bits.

When the IPEN bit is cleared (default state), the interrupt priority feature is disabled and interrupts are compatible with PICmicro® mid-range devices. In compatibility mode, the interrupt priority bits for each source have no effect. INTCON<6> is the PEIE bit, which enables/disables all peripheral interrupt sources. INTCON<7> is the GIE bit, which enables/disables all interrupt sources. All interrupts branch to address 000008h in compatibility mode.

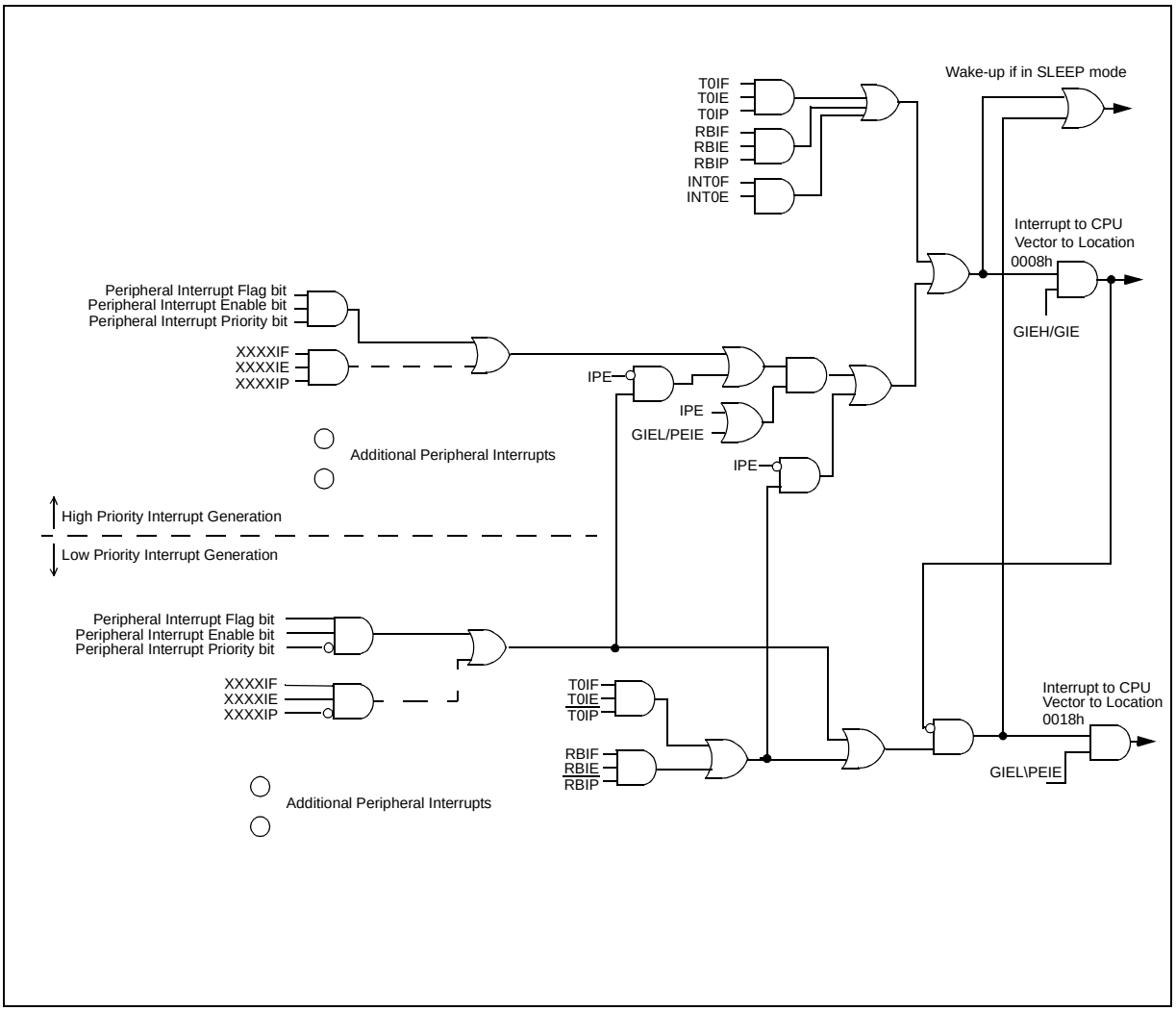
When an interrupt is responded to, the Global Interrupt Enable bit is cleared to disable further interrupts. If the IPEN bit is cleared, this is the GIE bit. If interrupt priority levels are used, this will be either the GIEH or GIEL bit. High priority interrupt sources can interrupt a low priority interrupt.

The return address is pushed onto the stack and the PC is loaded with the interrupt vector address (000008h or 000018h). Once in the Interrupt Service Routine, the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bits must be cleared in software before re-enabling interrupts to avoid recursive interrupts.

The "return from interrupt" instruction, `RETFIE`, exits the interrupt routine and sets the GIE bit (GIEH or GIEL if priority levels are used), which re-enables interrupts.

For external interrupt events, such as the INT pins or the PORTB input change interrupt, the interrupt latency will be three to four instruction cycles. The exact latency is the same for one or two cycle instructions. Individual interrupt flag bits are set, regardless of the status of their corresponding enable bit, or the GIE bit.

FIGURE 8-1: INTERRUPT LOGIC



8.1 INTCON Registers

The INTCON Registers are readable and writable registers, which contain various enable, priority and flag bits.

REGISTER 8-1: INTCON REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

bit 7 **GIE/GIEH:** Global Interrupt Enable bit

When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts

When IPEN = 1:
1 = Enables all interrupts
0 = Disables all interrupts

bit 6 **PEIE/GEIL:** Peripheral Interrupt Enable bit

When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE:** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF:** RB Port Change Interrupt Flag bit
1 = At least one of the RB5:RB0 pins changed state (must be cleared in software)
0 = None of the RB5:RB0 pins have changed state

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

Note: Interrupt flag bits get set when an interrupt condition occurs, regardless of the state of its corresponding enable bit, or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

REGISTER 8-2: INTCON2 REGISTER

R/W-1	R/W-1	U-0	U-0	U-0	R/W-1	U-0	R/W-1
$\overline{\text{RBPU}}$	INTEDG0	—	—	—	TMR0IP	—	RBIP
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$**: PORTB Pull-up Enable bit
1 = All PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG0**:External Interrupt 0 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 5-3 **Unimplemented:** Read as '0'

bit 2 **TMR0IP**: TMR0 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

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

bit 0 **RBIP**: RB Port Change Interrupt Priority bit
1 = High priority
0 = Low priority

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

Note: Interrupt flag bits get set when an interrupt condition occurs, regardless of the state of its corresponding enable bit, or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

8.2 PIR Registers

The PIR2 register contains the individual flag bits for the peripheral interrupts.

Note 1: Interrupt flag bits get set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>).

2: User software should ensure the appropriate interrupt flag bits are cleared prior to enabling an interrupt, and after servicing that interrupt.

8.3 PIE Registers

The PIE2 register contains the individual enable bits for the peripheral interrupts. When IPEN = 0, the PEIE bit must be set to enable any of these peripheral interrupts.

8.4 IPR Registers

The IPR2 register contains the individual priority bits for the peripheral interrupts. The operation of the priority bits requires that the Interrupt Priority Enable (IPEN) bit be set.

8.5 RCON Register

The RCON register contains the bit which is used to enable prioritized interrupts (IPEN).

REGISTER 8-3: RCON REGISTER

R/W-0	U-0	U-0	R/W-1	R-1	R-1	R/W-0	R/W-0
IPEN	—	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

bit 7 **IPEN:** Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (16CXXX compatibility mode)

bit 6-5 **Unimplemented:** Read as '0'

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

bit 3 **$\overline{TO}$:** Watchdog Time-out Flag bit
For details of bit operation see Register 4-1

bit 2 **$\overline{PD}$:** Power-down Detection Flag bit
For details of bit operation see Register 4-1

bit 1 **$\overline{POR}$:** Power-on Reset Status bit
For details of bit operation see Register 4-1

bit 0 **$\overline{BOR}$:** Brown-out Reset Status bit
For details of bit operation see Register 4-1

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

REGISTER 8-4: PIR2: PERIPHERAL INTERRUPT FLAG REGISTER2 (FA1h)

U-0	U-0	U-0	R/W-0	U-0	R/W-0	U-0	U-0
—	—	—	EEIF	—	LVDIF	—	—
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as ‘0’

bit 4 **EEIF:** EEPROM Write Timer Interrupt Flag bit
1 = Write complete

bit 3 **Unimplemented:** Read as ‘0’

bit 2 **LVDIF:** Low Voltage Detect Interrupt Flag bit
1 = The supply voltage has fallen below the specified LVD voltage (must be cleared in software)
0 = The supply voltage is greater than the specified LVD voltage

bit 1-0 **Unimplemented:** Read as ‘0’

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

REGISTER 8-5: PIE2: PERIPHERAL INTERRUPT ENABLE REGISTER2 (FA0h)

U-0	U-0	U-0	R/W-0	U-0	R/W-0	U-0	U-0
—	—	—	EEIE	—	LVDIE	—	—
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as ‘0’

bit 4 **EEIE**: EEPROM Write Timer Interrupt Enable bit
1 = Enables the EEPROM Write Timer interrupt
0 = Disables the EEPROM Write Timer interrupt

bit 3 **Unimplemented:** Read as ‘0’

bit 2 **LVDIE**: Low Voltage Detect Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 1-0 **Unimplemented:** Read as ‘0’

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

REGISTER 8-6: IPR2: PERIPHERAL INTERRUPT PRIORITY REGISTER2 (FA2h)

U-0	U-0	U-0	R/W-1	U-0	R/W-1	U-0	U-0
—	—	—	EEIP	—	LVDIP	—	—
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as ‘0’

bit 4 **EEIP**: EEPROM Write Timer Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **Unimplemented:** Read as ‘0’

bit 2 **LVDIP**: Low Voltage Detect Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1-0 **Unimplemented:** Read as ‘0’

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

8.5.1 INT0 INTERRUPT

The external interrupt on the RB2/INT0 pin is edge triggered: either rising if the INTEDG0 bit is set in the INTCON2 register, or falling if the INTEDG0 bit is clear. When a valid edge appears on the RB0/INT0 pin, the flag bit INT0F is set. Clearing the enable bit INT0E will disable this interrupt. Flag bit INT0F must be cleared in software in the Interrupt Service Routine before re-enabling the interrupt. The external interrupt can wake-up the processor from SLEEP. If the global interrupt enable bit GIE is set, the processor will branch to the interrupt vector following wake-up.

Note:	There is no priority bit associated with INT0. It is always a high priority interrupt source.

8.5.2 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.

8.5.3 PORTB INTERRUPT-ON-CHANGE

An interrupt change on any pin in PORTB sets flag bit RBIF in INTCON. The interrupt can be enabled/disabled by setting clearing the enable bit RBIE in INTCON. The bit RBIP in INTCON2 determines the priority of the interrupt.

Each of the PORTB pins is individually configurable as an interrupt-on-change pin. Control bits IOCBx in the IOCB register, Register 9-2, enable or disable the interrupt function for each pin. The interrupt-on-change is disabled on a Power-on Reset.

8.6 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 6-1 saves and restores the WREG, STATUS and BSR registers during an Interrupt Service Routine.

EXAMPLE 8-1: 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
```


9.0 I/O PORT

Depending on the device options enabled, there are as many as six general purpose I/O pins available. Some of the pins are multiplexed with alternative functions from the peripheral features on the device. Thus, when a peripheral is enabled, the associated pin may not be used as a general purpose I/O pin. On a Power-on Reset, all pins configured as general I/O are set as inputs.

9.1 PORTB, TRISB, and LATB Registers

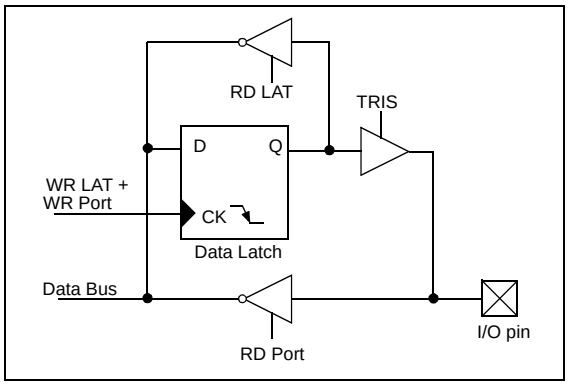
PORTB is a 6-bit wide, bi-directional port. The corresponding data direction register is TRISB. Setting a TRISB bit (= 1) will make the corresponding PORTB pin an input (i.e., put the corresponding output driver in a Hi-Impedance mode). Clearing a TRISB bit (= 0) will make the corresponding PORTB pin an output (i.e., put the contents of the output latch on the selected pin). On a Power-on Reset, these pins are configured as inputs. Example 9-1 demonstrates PORTB configuration.

EXAMPLE 9-1: INITIALIZING PORTB

```
CLRF    PORTB    ; Initialize PORTB by
                 ; clearing output
                 ; data latches
CLRF    LATB     ; Alternate method
                 ; to clear output
                 ; data latches
MOVLW   0x03     ; Value used to
                 ; initialize data
                 ; direction
MOVWF   TRISB    ; Set RB1:RB0 as inputs
                 ; RB5:RB2 as outputs
```

Read-modify-write operations on the LATB register, read and write the latched output value for PORTB. Figure 9-1 shows a simplified block diagram of the PORTB/LATB/TRISB operation.

FIGURE 9-1: SIMPLIFIED BLOCK DIAGRAM OF PORT/LAT/ TRIS OPERATION



9.2 Additional Functions

Each pin is multiplexed with other functions. Refer to Table 9-1 for information about individual pin functions.

9.2.1 WEAK PULL-UP

Each of the PORTB pins has an individually configurable weak internal pull-up. Control bits WPUBx enable or disable each pull-up (see Register 9-1). Each weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.

9.2.2 INTERRUPT-ON-CHANGE

Each of the PORTB pins is individually configurable as an interrupt-on-change pin. Control bits IOCBx enable or disable the interrupt function for each pin (see Register 9-2). The interrupt-on-change is disabled on a Power-on Reset.

For enabled interrupt-on-change pins, the values are compared with the old value latched on the last read of PORTB. The "mismatch" outputs of the last read are OR'd together to set, or clear the RB Port Change Interrupt flag bit RBIF, in the INTCON register.

This interrupt can wake the device from SLEEP. The user, in the Interrupt Service Routine, can clear the interrupt in the following manner:

a) Any read or write of PORTB (except with `MOVFF` instruction). This will end the mismatch condition.

b) Clear the flag bit RBIF.

9.2.3 RB2/T0CLK/INT0

The RB2 pin is configurable to function as a general I/O, the clock input for TIMER0, or as an external edge triggered interrupt. Figure 9-2 shows the block diagram of this I/O pin. Refer to Section 8.0 for details about interrupts and Section 10.0 for details about TIMER0.

9.2.4 RB3/$\overline{\text{MCLR}}$/V_{PP}

The RB3 pin is configurable to function as general I/O or as the RESET pin, $\overline{\text{MCLR}}$. This pin is open drain when configured as an output. Refer to Figure 9-3 for a block diagram of the I/O pin.

Note:	The voltage on RB3 open drain output must not exceed V_{DD}.

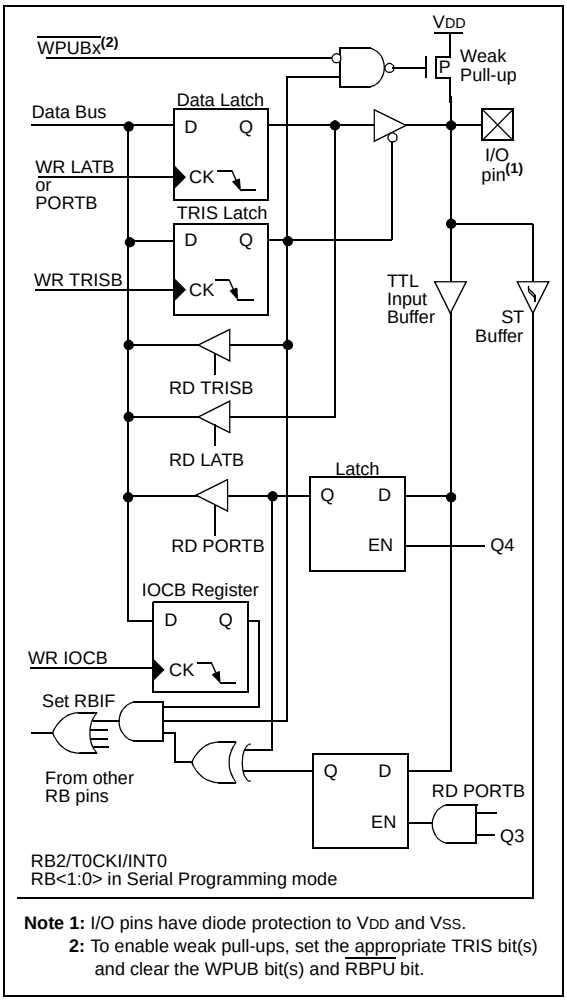
9.2.5 RB4/OSC2/CLKOUT

The RB4 pin is configurable to function as a general I/O pin, oscillator connection, or as a clock output. Figure 9-4 shows the block diagram of this I/O pin. Refer to Section 2.0 for clock/oscillator information.

9.2.6 RB5/OSC1/CLKIN

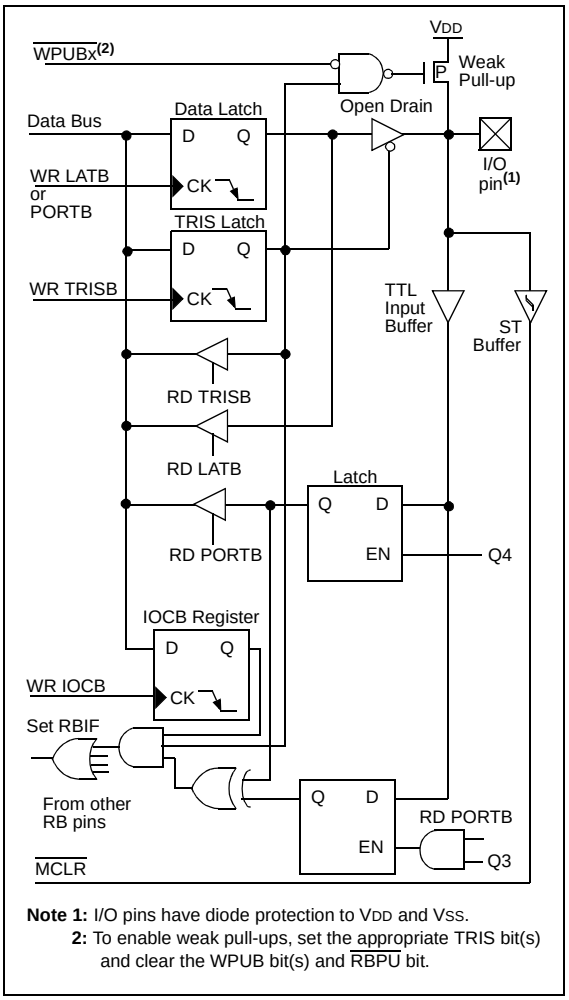
The RB5 pin is configurable to function as a general I/O pin, oscillator connection, or a clock input pin. Figure 9-5 shows a block diagram of this I/O pin. Refer to Section 2.0 for clock /oscillator information.

FIGURE 9-2: BLOCK DIAGRAM OF RB<2:0> PINS



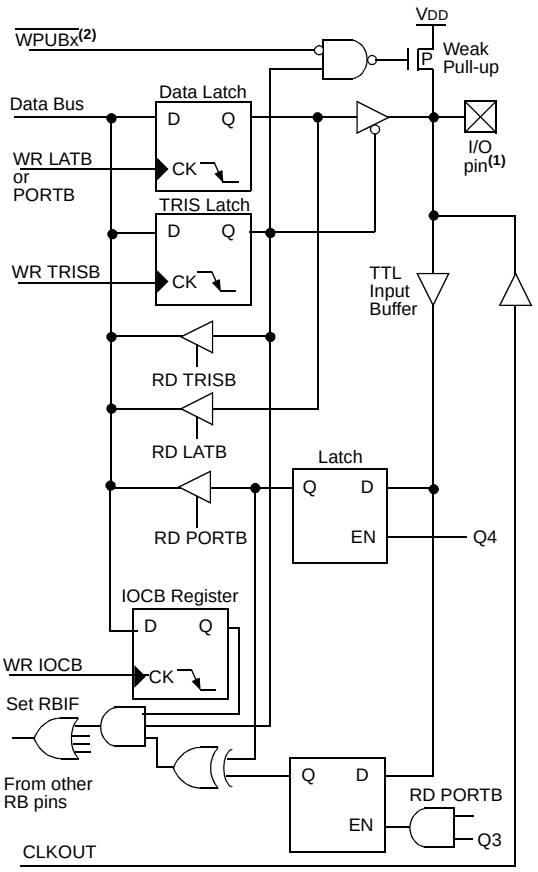
Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the WPUB bit(s) and $\overline{\text{RBPU}}$ bit.

FIGURE 9-3: BLOCK DIAGRAM OF RB3 PIN



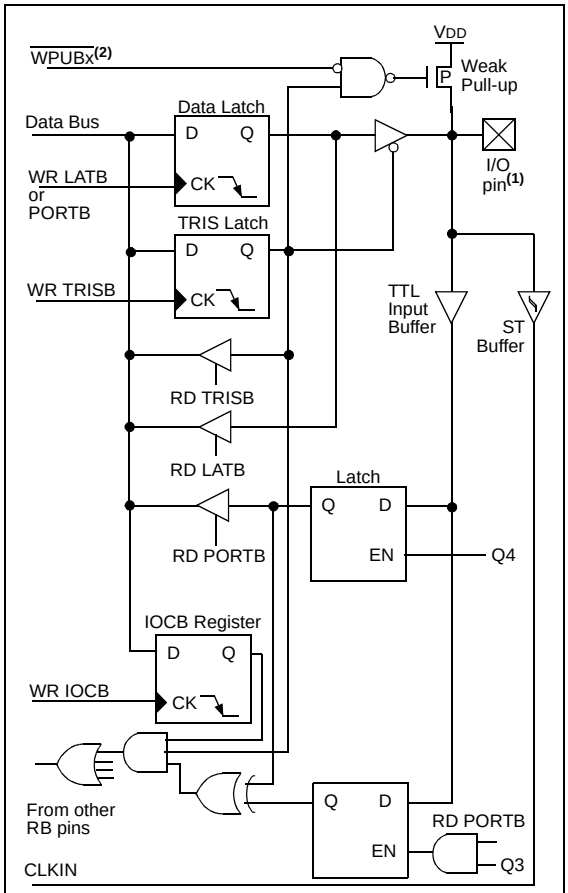
Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the WPUB bit(s) and $\overline{\text{RBPU}}$ bit.

FIGURE 9-4: BLOCK DIAGRAM OF RB4 PIN



Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the WPUB bit(s) and RBPU bit.

FIGURE 9-5: BLOCK DIAGRAM OF RB5 PIN



Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the WPUB bit(s) and RBPU bit.

REGISTER 9-1: WPUB: WEAK PULL-UP REGISTER (ADDRESS 0XF79h)

U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
—	—	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0
bit 7							bit 0

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

bit 5-0 **WPUB<5:0>:** Weak Pull-up Register bit
1 = Pull-up disabled
0 = Pull-up enabled

Note 1: Global RBPU must be enabled for individual pull-ups to be enabled.
2: The weak pull-up device is automatically disabled if the pin is in output mode (TRIS = 0).

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

REGISTER 9-2: IOCB: INTERRUPT-ON-CHANGE PORTB REGISTER (ADDRESS 0XF78h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0
bit 7							bit 0

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

bit 5-0 **IOCB<5:0>:** Interrupt-on-Change PORTB Control bit
1 = Interrupt-on-change enabled
0 = Interrupt-on-change disabled

Note 1: Global interrupt enables (GIE and RBIE) must be enabled for individual interrupts to be recognized.

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

TABLE 9-1: PORTB FUNCTIONS

Name	Bit#	Buffer	Function
RB0	bit0	TTL/ST(1)	Input/output port pin (with interrupt-on-change). Internal software programmable weak pull-up. In-circuit serial programming data.
RB1	bit1	TTL/ST(1)	Input/output port pin (with interrupt-on-change). Internal software programmable weak pull-up. In-circuit serial programming clock.
RB2/T0CKI/ INT0	bit2	TTL/ST(1)	Input/output port pin (with interrupt-on-change) or TMR0 clock input or Interrupt 0 input. Internal software programmable weak pull-up.
RB3/$\overline{\text{MCLR}}$/ V_{PP}	bit3	TTL/ST(1)	Input/output (open drain) port pin (with interrupt-on-change) or Master Clear External Reset input. Internal software programmable weak pull-up.
RB4/OSC2/ CLKOUT	bit4	TTL/ST(1)	Input/output port pin (with interrupt-on-change) or oscillator connection, or CLKOUT output. Internal software programmable weak pull-up.
RB5/OSC1/ CLKIN	bit5	TTL/ST(1)	Input/output port pin (with interrupt-on-change) or clock input, or oscillator connection. Internal software programmable weak pull-up.

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

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.

TABLE 9-2: SUMMARY OF REGISTERS ASSOCIATED WITH PORTB

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
TRISB	—	—	RB5	RB4	RB3	RB2	RB1	RB0	--11 1111	--11 1111
PORTB	—	—	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	--xx xxxx	--uu uuuu
LATB	—	—	LATB5	LATB4	LATB3	LATB2	LATB1	LATB0	--xx xxxx	--uu uuuu
INTCON	GIE/GIEH	PEIE/GIEL	T0IE	INT0E	RBIE	T0IF	INT0F	RBIF	0000 000x	0000 000u
INTCON2	RBPU	INTEG0	—	—	—	T0IP	—	RBIP	11-- -1-1	11-- -1-1
WPUB	—	—	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0	--11 1111	--11 1111
IOCB	—	—	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0	--00 0000	--00 0000

Legend: x = unknown, u = unchanged, - = unimplemented. Shaded cells are not used by PORTB.

NOTES:

10.0 TIMER0 MODULE

The Timer0 module has the following features:

- Software selectable as an 8-bit or 16-bit timer/ counter
- Readable and writable
- Dedicated 8-bit software programmable prescaler
- Clock source selectable to be external or internal
- Interrupt on overflow from FFh to 00h in 8-bit mode and FFFFh to 0000h in 16-bit mode
- Edge select for external clock

Figure 10-1 shows a simplified block diagram of the Timer0 module in 8-bit mode and Figure 10-1 shows a simplified block diagram of the Timer0 module in 16-bit mode.

The T0CON register is a readable and writable register that controls all the aspects of Timer0, including the prescale selection.

REGISTER 10-1: T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-bit/16-bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

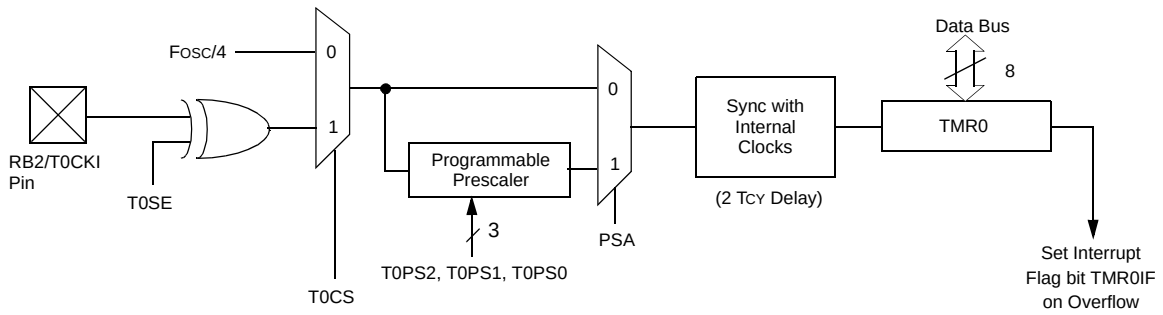
bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits
111 = 1:256 prescale value
110 = 1:128 prescale value
101 = 1:64 prescale value
100 = 1:32 prescale value
011 = 1:16 prescale value
010 = 1:8 prescale value
001 = 1:4 prescale value
000 = 1:2 prescale value

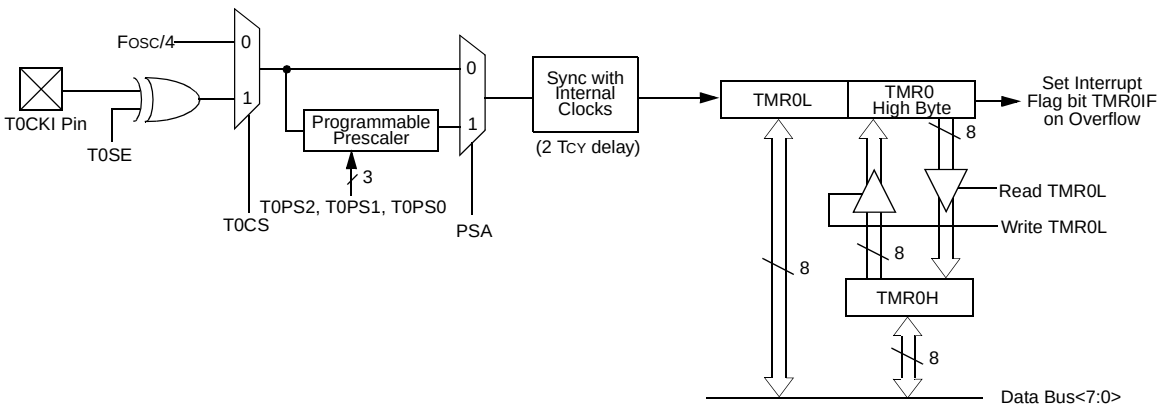
Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

FIGURE 10-1: TIMER0 BLOCK DIAGRAM IN 8-BIT MODE



Note: Upon RESET, Timer0 is enabled in 8-bit mode with clock input from T0CKI max. prescale.

FIGURE 10-2: TIMER0 BLOCK DIAGRAM IN 16-BIT MODE



Note: Upon RESET, Timer0 is enabled in 8-bit mode with clock input from T0CKI max. prescale.

10.1 Timer0 Operation

Timer0 can operate as a timer or as a counter.

Timer mode is selected by clearing the T0CS bit. In Timer mode, the Timer0 module will increment every instruction cycle (without prescaler). If the TMR0 register is written, the increment is inhibited for the following two instruction cycles. The user can work around this by writing an adjusted value to the TMR0 register.

Counter mode is selected by setting the T0CS bit. In Counter mode, Timer0 will increment either on every rising, or falling edge, of pin RB2/T0CKI. The incrementing edge is determined by the Timer0 Source Edge Select bit (T0SE). Clearing the T0SE bit selects the rising edge. Restrictions on the external clock input are discussed below.

When an external clock input is used for Timer0, it must meet certain requirements. The requirements ensure the external clock can be synchronized with the internal phase clock (TOSC). Also, there is a delay in the actual incrementing of Timer0 after synchronization.

10.2 Prescaler

An 8-bit counter is available as a prescaler for the Timer0 module. The prescaler is not readable or writable.

The PSA and T0PS2:T0PS0 bits determine the prescaler assignment and prescale ratio.

Clearing bit PSA will assign the prescaler to the Timer0 module. When the prescaler is assigned to the Timer0 module, prescale values of 1:2, 1:4, ..., 1:256 are selectable.

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

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

10.2.1 SWITCHING PRESCALER ASSIGNMENT

The prescaler assignment is fully under software control, (i.e., it can be changed “on-the-fly” during program execution).

10.3 Timer0 Interrupt

The TMR0 interrupt is generated when the TMR0 register overflows from FFh to 00h in 8-bit mode, or FFFFh to 0000h in 16-bit mode. This overflow sets the TMR0IF bit. The interrupt can be masked by clearing the TMR0IE bit. The TMR0IE bit must be cleared in software by the Timer0 module Interrupt Service Routine before re-enabling this interrupt. The TMR0 interrupt cannot awaken the processor from SLEEP, since the timer is shut off during SLEEP.

10.4 16-bit Mode Timer Reads and Writes

TMR0H is not the high byte of the timer/counter in 16-bit mode, but is actually a buffered version of the high byte of Timer0 (refer to Figure 10-1). The high byte of the Timer0 counter/timer is not directly readable nor writable. TMR0H is updated with the contents of the high byte of Timer0 during a read of TMR0L. This provides the ability to read all 16 bits of Timer0 without having to verify that the read of the high and low byte were valid, due to a rollover between successive reads of the high and low byte.

A write to the high byte of Timer0 must also take place through the TMR0H buffer register. Timer0 high byte is updated with the contents of TMR0H when a write occurs to TMR0L. This allows all 16 bits of Timer0 to be updated at once.

TABLE 10-1: REGISTERS ASSOCIATED WITH TIMER0

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
TMR0L	Timer0 Module’s Low Byte Register								xxxx xxxx	uuuu uuuu
TMR0H	Timer0 Module’s High Byte Register								0000 0000	0000 0000
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	1111 1111	1111 1111
TRISB	—	—	PORTB Data Direction Register						--11 1111	--11 1111

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'. Shaded cells are not used by Timer0.

NOTES:

11.0 LOW VOLTAGE DETECT

In many applications, the ability to determine if the device voltage (V_{DD}) is below a specified voltage level is a desirable feature. A window of operation for the application can be created, where the application software can do "housekeeping tasks" before the device voltage exits the valid operating range. This can be done using the Low Voltage Detect module.

This module is a software programmable circuitry, where a device voltage trip point can be specified. When the voltage of the device becomes lower then the specified point, an interrupt flag is set. If the interrupt is enabled, the program execution will branch to the interrupt vector address and the software can then respond to that interrupt source.

The Low Voltage Detect circuitry is completely under software control. This allows the circuitry to be "turned off" by the software, which minimizes the current consumption for the device.

Figure 11-1 shows a possible application voltage curve (typically for batteries). Over time, the device voltage decreases. When the device voltage equals voltage V_A, the LVD logic generates an interrupt. This occurs at time T_A. The application software then has the time, until the device voltage is no longer in valid operating range, to shut down the system. Voltage point V_B is the minimum valid operating voltage specification. This occurs at time T_B. T_B - T_A is the total time for shut down.

FIGURE 11-1: TYPICAL LOW VOLTAGE DETECT APPLICATION

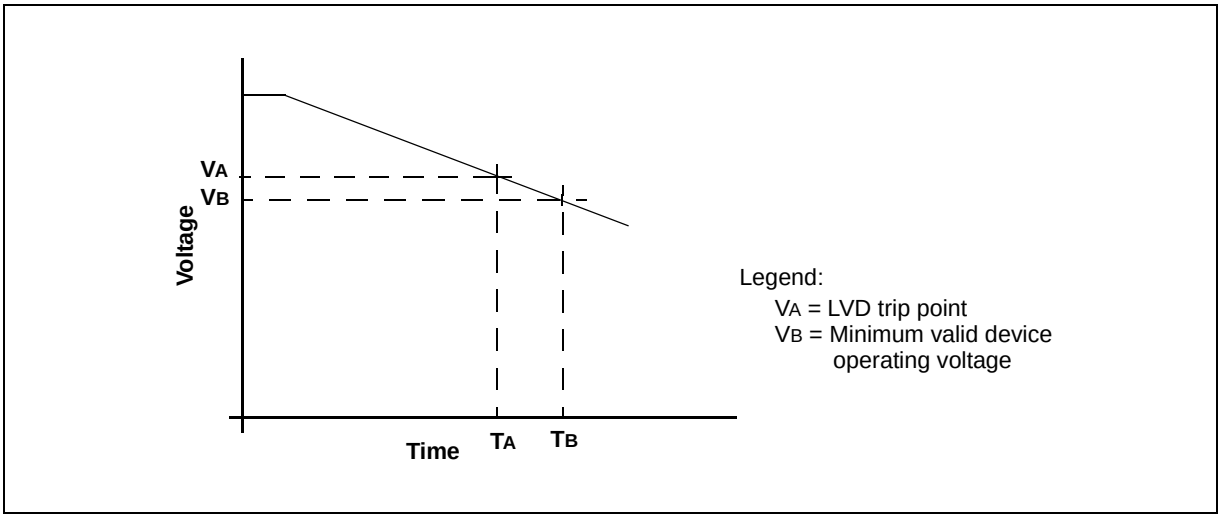
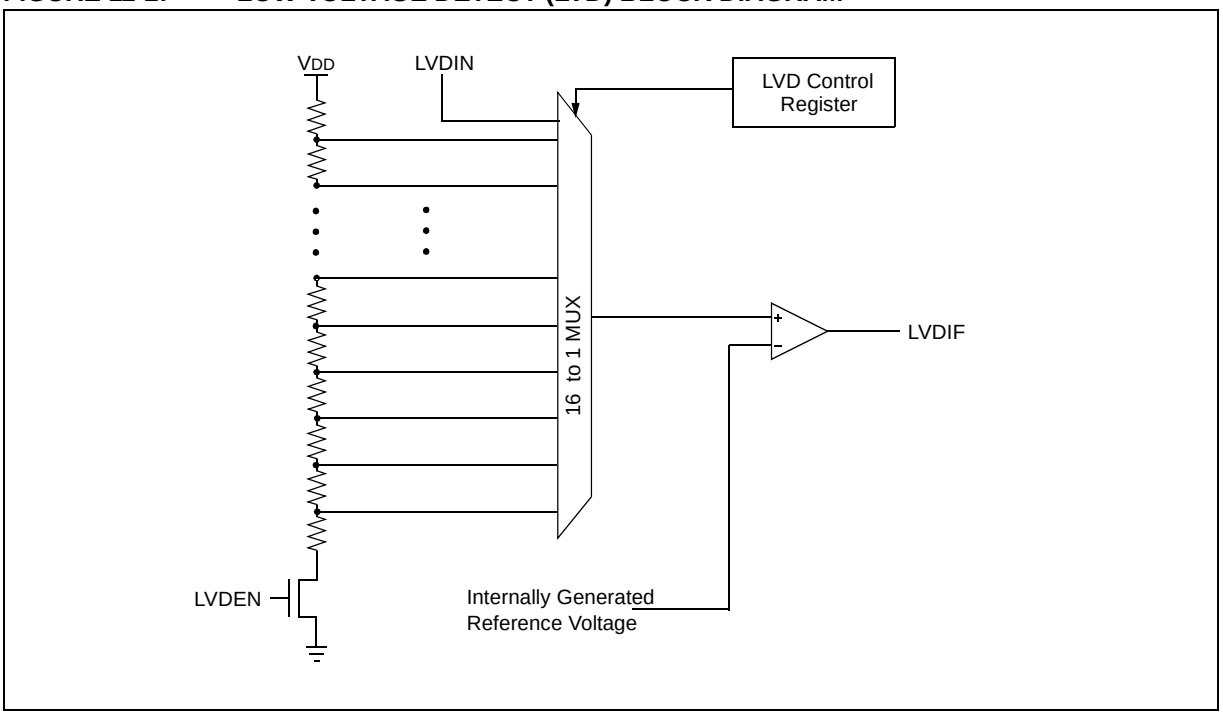


Figure 11-2 shows the block diagram for the LVD module. A comparator uses an internally generated reference voltage as the set point. When the selected tap output of the device voltage crosses the set point (is lower than), the LVDIF bit is set.

Each node in the resister divider represents a “trip point” voltage. The “trip point” voltage is the minimum supply voltage level at which the device can operate before the LVD module asserts an interrupt. When the supply voltage is equal to the trip point, the voltage tapped off of the resistor array is equal to the voltage generated by the internal voltage reference module. The comparator then generates an interrupt signal, setting the LVDIF bit. This voltage is software programmable to any one of 16 values (see Figure 11-2). The trip point is selected by programming the LVDL3:LVDL0 bits (LVDCON<3:0>).

FIGURE 11-2: LOW VOLTAGE DETECT (LVD) BLOCK DIAGRAM



11.1 Control Register

The Low Voltage Detect Control register controls the operation of the Low Voltage Detect circuitry.

REGISTER 11-1: LVDCON REGISTER

U-0	U-0	R-0	R/W-0	R/W-0	R/W-1	R/W-0	R/W-1
—	—	BGST	LVDEN	LVV3	LVV2	LVV1	LVV0
bit 7							bit 0

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

bit 5 **BGST:** Bandgap Stable Status Flag bit
1 = Indicates that the bandgap voltage is stable and LVD interrupt is reliable
0 = Indicates that the bandgap voltage is not stable and LVD interrupt should not be enabled

bit 4 **LVDEN:** Low Voltage Detect Power Enable bit
1 = Enables LVD, powers up LVD circuit and bandgap reference generator
0 = Disables LVD, powers down LVD and bandgap circuits

bit 3-0 **LVV3:LVV0:** Low Voltage Detection Limit bits
1111 = Reserved
1110 = Reserved
1101 = 4.0V
1100 = 3.5V
1011 = 3.0V
1010 = 2.9V
1001 = 2.8V
1000 = 2.7V
0111 = 2.6V
0110 = 2.5V
0101 = 2.4V
0100 = 2.3V
0011 = 2.2V
0010 = 2.1V
0001 = 2.0V
0000 = 1.9V

Legend:	
R = Readable bit	W = Writable bit
U = Unimplemented bit, read as ‘0’	- n = Value at POR Reset

Note: This register must be unlocked to modify, see Section 12.4.

11.2 Operation

Depending on the power source for the device voltage, the voltage normally decreases relatively slowly. This means that the LVD module does not need to be constantly operating. To decrease the current requirements, the LVD circuitry only needs to be enabled for short periods, where the voltage is checked. After doing the check, the LVD module may be disabled.

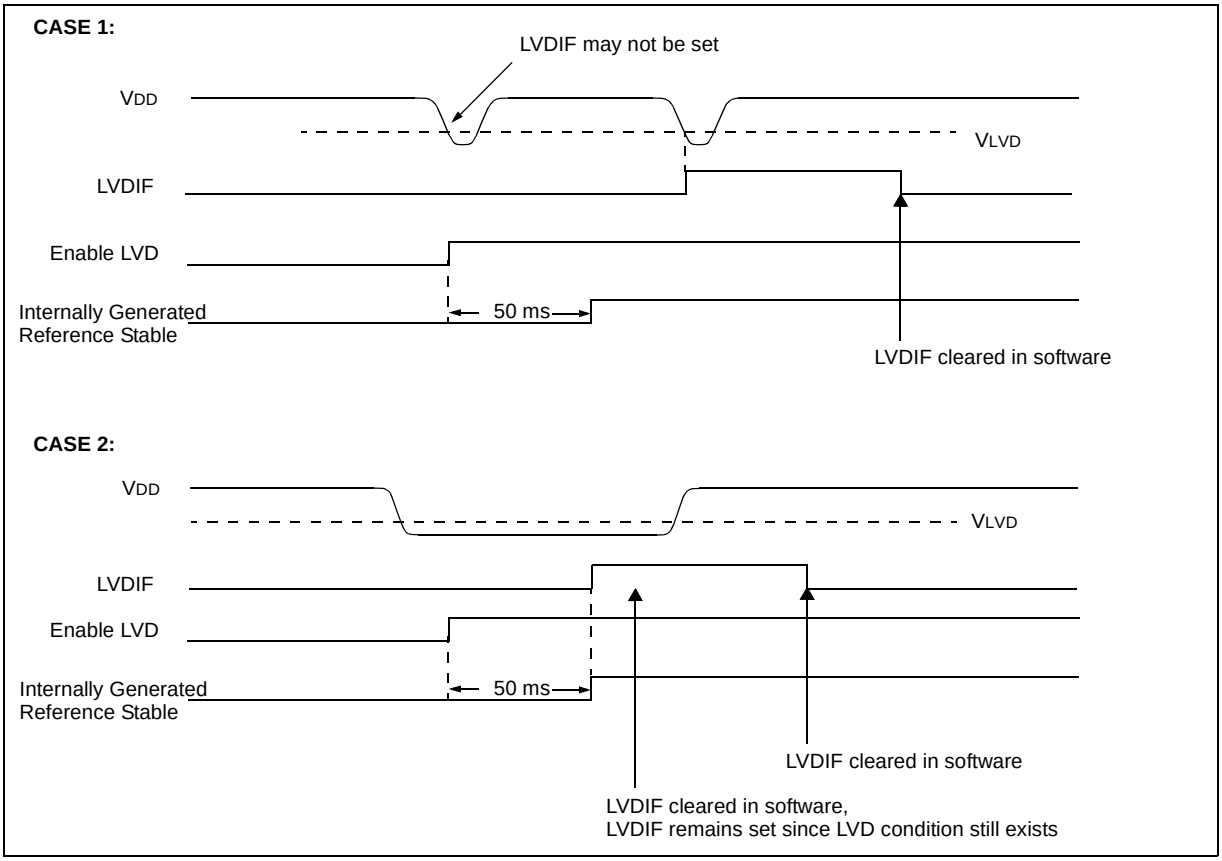
Each time that the LVD module is enabled, the circuitry requires some time to stabilize. After the circuitry has stabilized, all status flags may be cleared. The module will then indicate the proper state of the system.

The following steps are needed to set up the LVD module:

1. Unlock the LVDCON register using the unlock sequence described in Section 12.4.
2. Write the value to the LVDL3:LVDL0 bits (LVDCON register), which selects the desired LVD Trip Point.
3. Ensure that LVD interrupts are disabled (the LVDIE bit is cleared or the GIE bit is cleared).
4. Enable the LVD module (set the LVDEN bit in the LVDCON register).
5. Wait for the LVD module to stabilize (the IRVST bit to become set).
6. Clear the LVD interrupt flag, which may have falsely become set until the LVD module has stabilized (clear the LVDIF bit).
7. Enable the LVD interrupt (set the LVDIE and the GIE bits).

Figure 11-3 shows typical waveforms that the LVD module may be used to detect.

FIGURE 11-3: LOW VOLTAGE DETECT WAVEFORMS



11.2.1 CURRENT CONSUMPTION

When the module is enabled, the LVD comparator and voltage divider are enabled and will consume static current. The voltage divider can be tapped from multiple places in the resistor array. Total current consumption, when enabled, is specified in electrical specification parameter #D423 on page 147.

11.3 Operation During SLEEP

When enabled, the LVD circuitry continues to operate during SLEEP. If the device voltage crosses the trip point, the LVDIF bit will be set and the device will wake-up from SLEEP. Device execution will continue from the interrupt vector address, if interrupts have been globally enabled.

11.4 Effects of a RESET

A device RESET forces all registers to their RESET state. This forces the LVD module to be turned off.

NOTES:

12.0 SPECIAL FEATURES OF THE CPU

There are several features intended to maximize system reliability, minimize cost through elimination of external components, provide power saving operating modes and offer code protection. These are:

- OSC Selection
- RESET
  - Power-on Reset (POR)
  - Power-up Timer (PWRT)
  - Oscillator Start-up Timer (OST)
  - Brown-out Reset (BOR)
- Interrupts
- Watchdog Timer (WDT)
- SLEEP
- Code Protection
- ID Locations
- In-Circuit Serial Programming™

These devices have a Watchdog Timer, which is permanently enabled via the configuration bits or software-controlled. It runs off its own internal oscillator for added reliability. There are two timers that offer necessary delays on power-up. One is the Oscillator Start-up Timer (OST), intended to keep the chip in RESET until the crystal oscillator is stable. The other is the Power-up Timer (PWRT), which provides a fixed delay on power-up only, designed to keep the part in RESET while the power supply stabilizes. With these two timers on-chip, most applications need no external RESET circuitry.

SLEEP mode is designed to offer a very low current power-down mode. The user can wake-up from SLEEP through external RESET, Watchdog Timer Wake-up, or through an interrupt. Several oscillator options are also made available to allow the part to fit the application. The internal oscillator option saves system cost, while the LP crystal option saves power. A set of configuration bits are used to select various options.

12.1 Configuration Bits

The configuration bits can be programmed (read as '0'), or left unprogrammed (read as '1'), to select various device configurations. These bits are mapped starting at program memory location 300000h.

The user will note that address 300000h is beyond the user program memory space. In fact, it belongs to the configuration memory space (300000h - 3FFFFFh), which can only be accessed using table reads and table writes.

TABLE 12-1: CONFIGURATION BITS AND DEVICE IDS

File Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Factory/ Programmed Value
300000h	CONFIG1L	—	TR1	TW1	CP1	DP	TR0	TW0	CP0	-111 1111
300001h	CONFIG1H	—	—	OSCEN	MCLRE	—	FOSC2	FOSC1	FOSC0	--01 -100
300002h	CONFIG2L	—	—	—	—	—	—	BOREN	$\overline{\text{PWRTE}}$	---- --11
300003h	CONFIG2H	reserved	—	STVRE	$\overline{\text{WDTLE}}$	WDPS2	WDPS1	WDPS0	WDTE	1-11 1111
300104h	FOSCCAL	—	—	FCAL5	FCAL4	FCAL3	FCAL2	FCAL1	FCAL0	--uu uuuu
300105h	Unused. Always reads ‘0’s.									0000 0000
3FFFFEh	DEVID1	DEV2	DEV1	DEV0	REV4	REV3	REV2	REV1	REV0	01dr rrrr
3FFFFFh	DEVID2	DEV10	DEV9	DEV8	DEV7	DEV6	DEV5	DEV4	DEV3	0000 0011

Legend: x = unknown, u = unchanged, - = unimplemented, q = value depends on condition, grayed cells are unimplemented, read as ‘0’

REGISTER 12-1: CONFIG1H: CONFIGURATION BYTE (ADDRESS 300001h)

U-0	U-0	U-0	R/P-1	U-0	R/P-1	R/P-0	R/P-0
—	—	OSCEN	MCLRE	—	FOSC2	FOSC1	FOSC0
bit 7							bit 0

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

bit 5 **OSCEN**: Oscillator Enable bit
1 = Switching to the internal oscillator is enabled
0 = Switching to the internal oscillator is disabled

bit 4 **MCLRE**: RB3/$\overline{\text{MCLR}}$ Pin Function Select bit
1 = RB3/$\overline{\text{MCLR}}$ pin function is $\overline{\text{MCLR}}$
0 = RB3/$\overline{\text{MCLR}}$ pin function is digital I/O, $\overline{\text{MCLR}}$ internally tied to V_{DD}

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

bit 2-0 **FOSC2:FOSC0**: Oscillator Selection bits
111 = External RC oscillator/CLKOUT function on RB4/OSC2/CLKOUT pin
110 = EC external clock/CLKOUT function on RB4/OSC2/CLKOUT pin
101 = Internal oscillator/CLKOUT function on RB4/OSC2/CLKOUT pin, RB5 function on RB5/OSC1/CLKIN pin
100 = Internal oscillator/RB4 function on RB4/OSC2/CLKOUT pin, RB5 function on RB5/OSC1/CLKIN pin
011 = External RC oscillator/RB4 function on RB4/OSC2/CLKOUT pin
010 = HS oscillator
001 = XT oscillator
000 = LP oscillator

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

REGISTER 12-2: CONFIG1L: CONFIGURATION BYTE (ADDRESS 300000h)

U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
—	TR1	TW1	CP1	DP	TR0	TW0	CP0
bit 7							bit 0

bit 7 **Unimplemented:** Read as ‘0’

bit 6 **TR1:** Table Read Protection bit (memory area > 0400h byte address)
1 = Table reads are enabled
0 = Table reads are disabled from access outside of this block

bit 5 **TW1:** Table Write Protection bit (memory area > 0400h byte address)
1 = Table writes are enabled
0 = Table writes are disabled from access outside of this block

bit 4 **CP1:** Code Protection bit (memory area > 0400h byte address)
1 = Program memory code protection off
0 = Program memory code protected

bit 3 **DP:** Data Protection bit for EEDATA Memory
1 = External reads and writes are enabled
0 = External reads and writes are disabled

bit 2 **TR0:** Table Read Protection bit (memory area > 0000h - 03FFh byte address)
1 = Table reads are enabled
0 = Table reads are disabled from access outside of this block

bit 1 **TW0:** Table Write Protection bit (memory area > 0000h - 03FFh byte address)
1 = Table writes are enabled
0 = Table writes are disabled from access outside of this block

bit 0 **CP0:** Code Protection bit (memory area > 0000h - 03FFh byte address)
1 = Program memory code protection off
0 = Program memory code protected

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

REGISTER 12-3: CONFIG2H: CONFIGURATION REGISTER 2H (ADDRESS 300003h)

R/P-1	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
reserved	—	STVRE	$\overline{\text{WDTLE}}$	WDPS2	WDPS1	WDPS0	WDTE
bit 7							bit 0

bit 7 **Reserved**

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

bit 5 STVRE: Stack Full/Underflow Reset Enable bit
1 = Reset on stack full/underflow enabled
0 = Disabled

bit 4 **$\overline{\text{WDTLE}}$:** Watchdog Timer Long Delay Enable bit
1 = Use WDPS<2:0> bits to set delay
0 = Enable long postscaler divider; 16 x WDPS<2:0> bits

bit 3-1 **WDPS2:WDPS0:** Watchdog Timer Postscale Select bits
111 = 1:128
110 = 1:64
101 = 1:32
100 = 1:16
011 = 1:8
010 = 1:4
001 = 1:2
000 = 1:1

bit 0 **WDTE:** Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled (control is placed on the SWDTE bit)

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

REGISTER 12-4: CONFIG2L: CONFIGURATION REGISTER 2L (ADDRESS 300002h)

U-0	U-0	U-0	U-0	U-0	U-0	R/P-1	R/P-1
—	—	—	—	—	—	BOREN	$\overline{\text{PWRTE}}$
bit 7							bit 0

bit 7-2 **Unimplemented:** Read as '0'

bit 1 **BOREN:** Brown-out Reset Enable bit(1)
1 = Brown-out Reset enabled
0 = Brown-out Reset disabled

bit 0 **$\overline{\text{PWRTE}}$:** Power-up Timer Enable bit(1)
1 = PWRT disabled
0 = PWRT enabled

Note 1: Enabling Brown-out Reset automatically enables the Power-up Timer (PWRT), regardless of the value of bit $\overline{\text{PWRTE}}$. Ensure the Power-up Timer is enabled any time Brown-out Reset is enabled.

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

12.2 Watchdog Timer (WDT)

The Watchdog Timer is a free running, on-chip RC oscillator, which does not require any external components. This RC oscillator is separate from the internal oscillator of the OSC1/CLKI pin. That means that the WDT will run, even if the clock on the OSC1/CLKI and OSC2/CLKO/RA6 pins of the device has been stopped, for example, by execution of a SLEEP instruction.

During normal operation, a WDT time-out generates a device RESET (Watchdog Timer Reset). If the device is in SLEEP mode, a WDT time-out causes the device to wake-up and continue with normal operation (Watchdog Timer Wake-up). The $\overline{TO}$ bit in the RCON register will be cleared upon a WDT time-out.

The Watchdog Timer is enabled/disabled by a device configuration bit. If the WDT is enabled, software execution may not disable this function. When the WDTEN configuration bit is cleared, the SWDTEN bit enables/disables the operation of the WDT.

The WDT time-out period values may be found in the Electrical Specifications section under parameter #31. Values for the WDT postscaler may be assigned using the configuration bits or in software.

Note:	The CLRWDT and SLEEP instructions clear the WDT and the postscaler, if assigned to the WDT and prevent it from timing out and generating a device RESET condition.

Note:	When a CLRWDT instruction is executed and the prescaler is assigned to the WDT, the prescaler count will be cleared, but the prescaler assignment is not changed.

12.2.1 CONTROL REGISTER

Register 12-5 shows the WDTCON register. This is a readable and writable register, which contains a control bit that allows software to override the WDT enable configuration bit, only when the configuration bit has disabled the WDT.

REGISTER 12-5: WDTCON REGISTER

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0
—	—	—	—	—	—	—	SWDTEN
bit 7							bit 0

bit 7-1 **Unimplemented**: Read as ’0’

bit 0 **SWDTEN:** Software Controlled Watchdog Timer Enable bit
1 = Watchdog Timer is on
0 = Watchdog Timer is turned off

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

Note:	This register must be unlocked to modify, see Section 12.4.

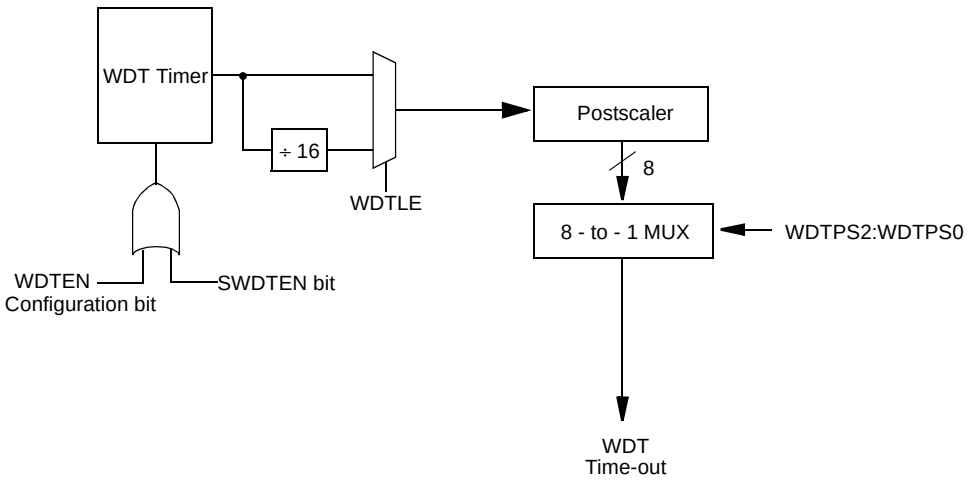
12.2.2 WDT POSTSCALER

The WDT has a postscaler that can extend the WDT Reset period. The postscaler is selected at the time of the device programming, by the value written to the CONFIG2H configuration register. An extended WDT is also available, multiplying the standard settings by 16.

The standard settings are also available in software when not setup in the CONFIG2H configuration. The WDTCON register allows enabling the WDT and setting the standard postscaler options.

Note:	The WDTCON register must be unlocked before it can be modified (see Section 12.4.1).

FIGURE 12-1: WATCHDOG TIMER BLOCK DIAGRAM



Note: WDPS2:WDPS0 are bits in a configuration register.

TABLE 12-2: SUMMARY OF WATCHDOG TIMER REGISTERS

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CONFIG2H	reserved	—	STVRE	$\overline{\text{WDTLE}}$	WDTPS2	WDTPS2	WDTPS0	WDTEN
RCON	IPEN	—	—	$\overline{\text{RI}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$
WDTCON	—	—	—	—	—	—	—	SWDTEN

Legend: Shaded cells are not used by the Watchdog Timer.

12.3 Power-down Mode (SLEEP)

Power-down mode is entered by executing a SLEEP instruction.

If enabled, the Watchdog Timer will be cleared, but keeps running, the $\overline{PD}$ bit (RCON<3>) is cleared, the $\overline{TO}$ (RCON<4>) bit is set, and the oscillator driver is turned off. The I/O ports maintain the status they had before the SLEEP instruction was executed (driving high, low or hi-impedance).

For lowest current consumption in this mode, place all I/O pins at either V_{DD} or V_{SS}, ensure no external circuitry is drawing current from the I/O pin, and disable external clocks. Pull all I/O pins that are hi-impedance inputs, high or low externally, to avoid switching currents caused by floating inputs. The T0CKI input should also be at V_{DD} or V_{SS} for lowest current consumption. The contribution from on-chip pull-ups should be considered.

The $\overline{MCLR}$ pin must be at a logic high level (V_{IHMC}), if enabled.

12.3.1 WAKE-UP FROM SLEEP

The device can wake-up from SLEEP through one of the following events:

1. External RESET input on $\overline{MCLR}$ pin.
2. Watchdog Timer Wake-up (if WDT was enabled).
3. Interrupt from INT pin, RB port change or a Peripheral Interrupt.

Other peripherals cannot generate interrupts, since during SLEEP, no on-chip clocks are present.

External $\overline{MCLR}$ Reset will cause a device RESET. All other events are considered a continuation of program execution and will cause a "wake-up". The $\overline{TO}$ and $\overline{PD}$ bits in the RCON register can be used to determine the cause of the device RESET. The $\overline{PD}$ bit, which is set on power-up, is cleared when SLEEP is invoked. The $\overline{TO}$ bit is cleared, if a WDT time-out occurred (and caused wake-up).

When the SLEEP instruction is being executed, the next instruction (PC + 2) is pre-fetched. For the device to wake-up through an interrupt event, the corresponding interrupt enable bit must be set (enabled). Wake-up is regardless of the state of the GIE bit. If the GIE bit is clear (disabled), the device continues execution at the instruction after the SLEEP instruction. If the GIE bit is set (enabled), the device executes the instruction after the SLEEP instruction and then branches to the interrupt address. In cases where the execution of the instruction following SLEEP is not desirable, the user should have a NOP after the SLEEP instruction.

12.3.2 WAKE-UP USING INTERRUPTS

When global interrupts are disabled (GIE cleared) and any interrupt source has both its interrupt enable bit and interrupt flag bit set, one of the following will occur:

- If an interrupt condition (interrupt flag bit and interrupt enable bits are set) occurs **before** the execution of a SLEEP instruction, the SLEEP instruction will complete as a NOP. Therefore, the WDT and WDT postscaler will not be cleared, the $\overline{TO}$ bit will not be set and $\overline{PD}$ bits will not be cleared.
- If the interrupt condition occurs **during or after** the execution of a SLEEP instruction, the device will immediately wake-up from SLEEP. The SLEEP instruction will be completely executed before the wake-up. Therefore, the WDT and WDT postscaler will be cleared, the $\overline{TO}$ bit will be set and the $\overline{PD}$ bit will be cleared.

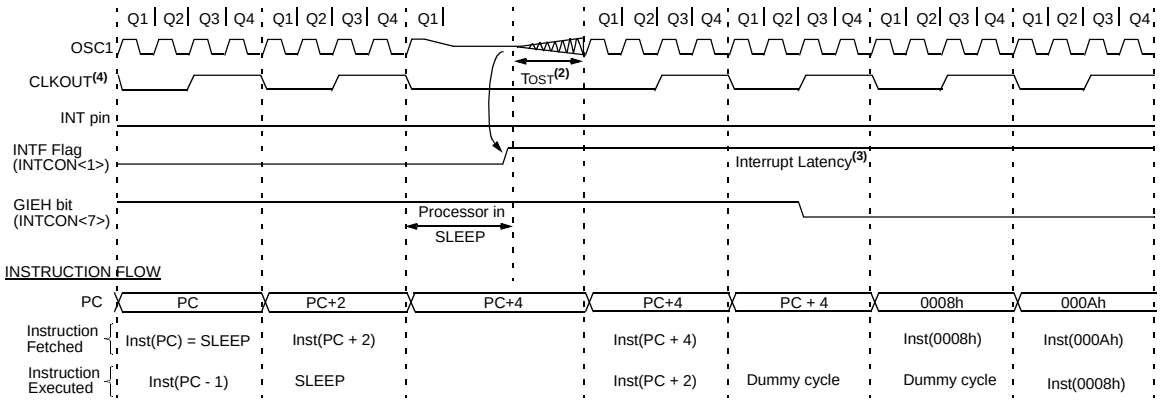
Even if the flag bits were checked before executing a SLEEP instruction, it may be possible for flag bits to become set before the SLEEP instruction completes. To determine whether a SLEEP instruction executed, test the $\overline{PD}$ bit. If the $\overline{PD}$ bit is set, the SLEEP instruction was executed as a NOP.

To ensure that the WDT is cleared, a CLRWDT instruction should be executed before a SLEEP instruction.

12.3.3 TWO-SPEED CLOCK START-UP

When using an external clock source, wake-up from SLEEP causes a unique start-up procedure. The internal oscillator starts immediately upon wake-up, while the external source is stabilizing. Once the Oscillator Start-up Time-out (OST) is complete, the clock source is switched to the external clock. The result is nearly immediate code execution upon wake-up. Refer to Section 2.6.

FIGURE 12-2: 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 external RC oscillator, EC osc, and INTOSC modes.
4: CLKOUT is not available in these osc modes, but shown here for timing reference.

12.4 Secured Access Registers

This device contains programming options for safety critical peripherals. Because these safety critical peripherals can be programmed in software, the registers used to control these peripherals should be given limited access by the user's code. This way, errant code won't accidentally change settings in peripherals that could cause catastrophic results.

The registers that are considered safety critical are the Watchdog Timer Control register (WDTCON), the Low Voltage Detect register (LVDTCON), and the Oscillator Control register (OSCCON).

12.4.1 COMBINATION LOCK MODULE

Access is limited to using the Combination Lock module.

Two bits called Combination Lock (CMLK) bits are located in the lower two bits of the PSPCON register. These two bits, and only these two bits, must be set in sequence by the user's code.

The Combination Lock bits must be set sequentially, meaning that as soon as Combination Lock bit 1 is set, the second Combination Lock bit must be set on the following instruction cycle. If the user waits more than one machine cycle to set the second bit after setting the first, both bits will automatically be cleared in hardware, and the lock will remain closed.

Each instruction must only modify one combination lock bit at a time. This means that the first write to the register will write the CMLK1 to a '1', but CMLK0 will equal '0'. The second write will only modify CMLK0. This means that the data written to the PSPCON register will have CMLK1 set to a '1' and CMLK0 set to a '1'. This leaves CMLK1 unmodified. This will restrict at least one of the instructions used to modify this register to a `BSF` of the PSPCON register. This will restrict the combination of instructions that will allow the lock to be opened, so that random code execution in the event of a software fault, will not cause the lock to be accidentally opened. The `BSF` instruction limitation will also prevent random code from setting both bits at the same time via a `MOVWF` instruction, since they are located in the same register.

Note:	The Combination lock bits are write only bits. These bits will always return '0' when read.

When each bit is set and the combination lock is opened, the user will have three instruction cycles to modify the safety critical register of his choice. After three cycles have expired, the CMLK bits are cleared, the lock will close, and the user will have to set the CMLK bits in sequence again, in order to open the lock. Thus, for each attempt to modify a safety critical register, the combination lock must be opened before the register can be written to. The reason that three instruction cycles were chosen for the unlock time was to allow the user to put the "unlock" code in a subroutine call. This way, the user's code will only have one instance of the code that is used to unlock the module. The user would first set up the WREG register with the desired data to load into a secured register, then call a subroutine that contains the two `BSF` instructions, return from the routine, and modify the secured register.

```
;Setup WREG with data to be stored
; in a safety critical register
MAIN
        MOVLW   0x5A
        CALL    UNLOCK
;Write must take place on next
;instruction cycle
        MOVWF   OSCCON, 0
        .
        .
        .
UNLOCK
        BSF     PSPCON, CMLK1, 0
        BSF     PSPCON, CMLK0, 0
        RETURN
```


12.5 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.

12.6 ID Locations

Five memory locations (200000h - 200007h) 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.

12.7 In-Circuit Serial Programming

PIC18F010/020 microcontrollers can be serially programmed while in the end application circuit. This is simply done with two lines for clock and data, and two other lines for power and ground. 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.

NOTES:

13.0 INSTRUCTION SET SUMMARY

The PIC18F010/020 instruction set adds many enhancements to the previous PICmicro® instruction sets, while maintaining an easy migration from these PICmicro instruction sets.

Most instructions are a single program memory word (16-bits), but there are four instructions that require two program memory locations.

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

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

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

The PIC18F010/020 instruction set summary in Table 13-2 lists **byte-oriented**, **bit-oriented**, **literal** and **control** operations. Table 13-1 shows the opcode field descriptions.

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

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

'f' represents a file register designator and 'd' represents a destination designator. The file register designator specifies which file register is to be used by the instruction.

The destination designator 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 the value of 'f')
2. The bit in the file register (specified by the value of 'b')
3. The accessed memory (specified by the value of 'a')

'b' represents a bit field designator which selects the number of the bit affected by the operation, while '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 the value of 'k')
- The desired FSR register to load the literal value into (specified by the value of 'f')
- No operand required (specified by the value of '—')

The **control** instructions may use some of the following operands:

- A program memory address (specified by the value of 'n')
- The mode of the Call or Return instructions (specified by the value of 's')
- The mode of the Table Read and Table Write instructions (specified by the value of 'm')
- No operand required (specified by the value of '—')

All instructions are a single word, except for four double word instructions. These four 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 13-1 shows the general formats that the instructions can have.

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

```
0xhh
```

where h signifies a hexadecimal digit.

The Instruction Set Summary, shown in Table 13-2, lists the instructions recognized by the Microchip assembler (MPASM™).

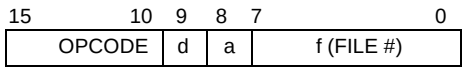
Section 13.1 provides a description of each instruction.

TABLE 13-1: OPCODE FIELD DESCRIPTIONS

Field	Description
a	RAM access bit a = 0: RAM location in Access RAM (BSR register is ignored) a = 1: RAM bank is specified by BSR register
ACCESS	ACCESS = 0: RAM access bit symbol
BANKED	BANKED = 1: RAM access bit symbol
bbb	Bit address within an 8-bit file register (0 to 7)
BSR	Bank Select Register. Used to select the current RAM bank.
d	Destination select bit; d = 0: store result in WREG, d = 1: store result in file register f.
dest	Destination either the WREG register or the specified register file location
f	8-bit Register file address (0x00 to 0xFF)
f_s	12-bit Register file address (0x000 to 0xFFF). This is the source address.
f_d	12-bit Register file address (0x000 to 0xFFF). This is the destination address.
k	Literal field, constant data or label (may be either an 8-bit, 12-bit or a 20-bit value)
label	Label name
mm	The mode of the TBLPTR register for the Table Read and Table Write instructions Only used with Table Read and Table Write instructions:
*	No Change to register (such as TBLPTR with Table reads and writes)
*+	Post-Increment register (such as TBLPTR with Table reads and writes)
*-	Post-Decrement register (such as TBLPTR with Table reads and writes)
+*	Pre-Increment register (such as TBLPTR with Table reads and writes)
n	The relative address (2's complement number) for relative branch instructions, or the direct address for Call/Branch and Return instructions
PRODH	Product of Multiply high byte (Register at address 0xFF4)
PRODL	Product of Multiply low byte (Register at address 0xFF3)
s	Fast Call / Return mode select bit. s = 0: do not update into/from shadow registers s = 1: certain registers loaded into/from shadow registers (Fast mode)
u	Unused or Unchanged (Register at address 0xFE8)
W	W = 0: Destination select bit symbol
WREG	Working register (accumulator) (Register at address 0xFE8)
x	Don't care (0 or 1) The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
TBLPTR	21-bit Table Pointer (points to a Program Memory location) (Register at address 0xFF6)
TABLAT	8-bit Table Latch (Register at address 0xFF5)
TOS	Top-of-Stack
PC	Program Counter
PCL	Program Counter Low Byte (Register at address 0xFF9)
PCH	Program Counter High Byte
PCLATH	Program Counter High Byte Latch (Register at address 0xFFA)
PCLATU	Program Counter Upper Byte Latch (Register at address 0xFFB)
GIE	Global Interrupt Enable bit
WDT	Watchdog Timer
$\overline{TO}$	Time-out bit
$\overline{PD}$	Power-down bit
C, DC, Z, OV, N	ALU status bits Carry, Digit Carry, Zero, Overflow, Negative
[ ]	Optional
( )	Contents
→	Assigned to
< >	Register bit field
∈	In the set of
italics	User defined term (font is courier)

FIGURE 13-1: GENERAL FORMAT FOR INSTRUCTIONS

Byte-oriented file register operations

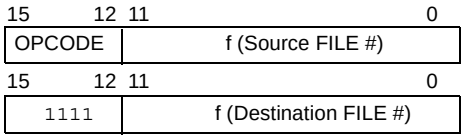


d = 0 for result destination to be WREG register
d = 1 for result destination to be file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select Bank
f = 8-bit file register address

Example Instruction

ADDWF MYREG, W, B

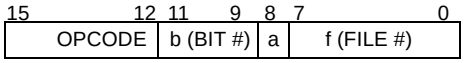
Byte to Byte move operations (2-word)



f = 12-bit file register address

MOVFF MYREG1, MYREG2

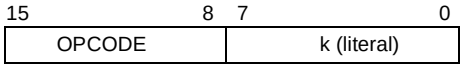
Bit-oriented file register operations



b = 3-bit position of bit in file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select Bank
f = 8-bit file register address

BSF MYREG, bit, B

Literal operations

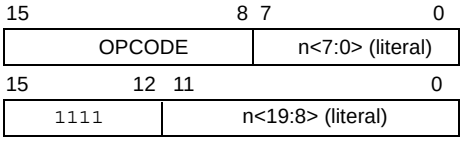


k = 8-bit immediate value

MOVLW 0x7F

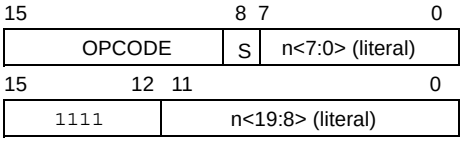
Control operations

CALL, GOTO and Branch operations



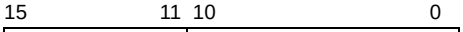
n = 20-bit immediate value

GOTO Label

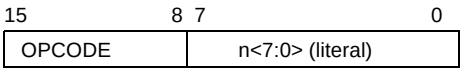


S = Fast bit

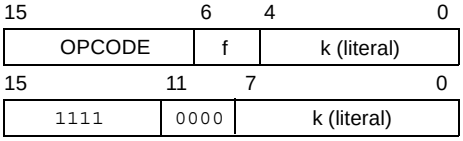
CALL MYFUNC



BRA MYFUNC



BC MYFUNC



LFSR FSR0, 0x100

TABLE 13-2: PIC18F010/020 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, 6
ADDWFC	f [,d] [,a]	Add WREG and Carry bit to f	1	0010	00da	ffff	ffff	C, DC, Z, OV, N	1, 2, 6
ANDWF	f [,d] [,a]	AND WREG with f	1	0001	01da	ffff	ffff	Z, N	1,2, 6
CLRF	f [,a]	Clear f	1	0110	101a	ffff	ffff	Z	2, 6
COMF	f [,d] [,a]	Complement f	1	0001	11da	ffff	ffff	Z, N	1, 2, 6
CPFSEQ	f [,a]	Compare f with WREG, skip =	1 (2 or 3)	0110	001a	ffff	ffff	None	4, 6
CPFSGT	f [,a]	Compare f with WREG, skip >	1 (2 or 3)	0110	010a	ffff	ffff	None	4, 6
CPFSLT	f [,a]	Compare f with WREG, skip <	1 (2 or 3)	0110	000a	ffff	ffff	None	1, 2, 6
DECF	f [,d] [,a]	Decrement f	1	0000	01da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4, 6
DECFSZ	f [,d] [,a]	Decrement f, Skip if 0	1 (2 or 3)	0010	11da	ffff	ffff	None	1, 2, 3, 4, 6
DCFSNZ	f [,d] [,a]	Decrement f, Skip if Not 0	1 (2 or 3)	0100	11da	ffff	ffff	None	1, 2, 6
INCF	f [,d] [,a]	Increment f	1	0010	10da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4, 6
INCFSZ	f [,d] [,a]	Increment f, Skip if 0	1 (2 or 3)	0011	11da	ffff	ffff	None	4, 6
INFSNZ	f [,d] [,a]	Increment f, Skip if Not 0	1 (2 or 3)	0100	10da	ffff	ffff	None	1, 2, 6
IORWF	f [,d] [,a]	Inclusive OR WREG with f	1	0001	00da	ffff	ffff	Z, N	1, 2, 6
MOVF	f [,d] [,a]	Move f	1	0101	00da	ffff	ffff	Z, N	1, 6
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	6
MULWF	f [,a]	Multiply WREG with f	1	0000	001a	ffff	ffff	None	6
NEGF	f [,a]	Negate f	1	0110	110a	ffff	ffff	C, DC, Z, OV, N	1, 2, 6
RLCF	f [,d] [,a]	Rotate Left f through Carry	1	0011	01da	ffff	ffff	C, Z, N	6
RLNCF	f [,d] [,a]	Rotate Left f (No Carry)	1	0100	01da	ffff	ffff	Z, N	1, 2, 6
RRCF	f [,d] [,a]	Rotate Right f through Carry	1	0011	00da	ffff	ffff	C, Z, N	6
RRNCF	f [,d] [,a]	Rotate Right f (No Carry)	1	0100	00da	ffff	ffff	Z, N	6
SETF	f [,a]	Set f	1	0110	100a	ffff	ffff	None	6
SUBFWB	f [,d] [,a]	Subtract f from WREG with borrow	1	0101	01da	ffff	ffff	C, DC, Z, OV, N	1, 2, 6
SUBWF	f [,d] [,a]	Subtract WREG from f	1	0101	11da	ffff	ffff	C, DC, Z, OV, N	6
SUBWFB	f [,d] [,a]	Subtract WREG from f with borrow	1	0101	10da	ffff	ffff	C, DC, Z, OV, N	1, 2, 6
SWAPF	f [,d] [,a]	Swap nibbles in f	1	0011	10da	ffff	ffff	None	4, 6
TSTFSZ	f [,a]	Test f, skip if 0	1 (2 or 3)	0110	011a	ffff	ffff	None	1, 2, 6
XORWF	f [,d] [,a]	Exclusive OR WREG with f	1	0001	10da	ffff	ffff	Z, N	6
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b [,a]	Bit Clear f	1	1001	bbba	ffff	ffff	None	1, 2, 6
BSF	f, b [,a]	Bit Set f	1	1000	bbba	ffff	ffff	None	1, 2, 6
BTFSC	f, b [,a]	Bit Test f, Skip if Clear	1 (2 or 3)	1011	bbba	ffff	ffff	None	3, 4, 6
BTFSS	f, b [,a]	Bit Test f, Skip if Set	1 (2 or 3)	1010	bbba	ffff	ffff	None	3, 4, 6
BTG	f [,d] [,a]	Bit Toggle f	1	0111	bbba	ffff	ffff	None	1, 2, 6

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.

6: Microchip Assembler MASM automatically defaults destination bit 'd' to '1', while access bit 'a' defaults to '1' or '0' according to address of register being used.

TABLE 13-2: PIC18F010/020 INSTRUCTION SET (CONTINUED)

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None	
BN	n	Branch if Negative	1 (2)	1110	0110	nnnn	nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110	0011	nnnn	nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110	0111	nnnn	nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110	0101	nnnn	nnnn	None	
BNZ	n	Branch if Not Zero	2	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	1 (2)	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	n, s	Call subroutine1st word	2	1110	110s	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
CLRWDT	—	Clear Watchdog Timer	1	0000	0000	0000	0100	$\overline{TO}$, $\overline{PD}$	
DAW	—	Decimal Adjust WREG	1	0000	0000	0000	0111	C	
GOTO	n	Go to address1st word	2	1110	1111	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
NOP	—	No Operation	1	0000	0000	0000	0000	None	
NOP	—	No Operation **(Note 4)**	1	1111	xxxx	xxxx	xxxx	None	
POP	—	Pop top of return stack (TOS)	1	0000	0000	0000	0110	None	
PUSH	—	Push top of return stack (TOS)	1	0000	0000	0000	0101	None	
RCALL	n	Relative Call	2	1101	1nnn	nnnn	nnnn	None	
RESET		Software device RESET	1	0000	0000	1111	1111	All	
RETFIE	s	Return from interrupt enable	2	0000	0000	0001	000s	GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
RETURN	s	Return from Subroutine	2	0000	0000	0001	001s	None	
SLEEP	—	Go into Standby mode	1	0000	0000	0000	0011	$\overline{TO}$, $\overline{PD}$	

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.

6: Microchip Assembler MASM automatically defaults destination bit 'd' to '1', while access bit 'a' defaults to '1' or '0' according to address of register being used.

TABLE 13-2: PIC18F010/020 INSTRUCTION SET (CONTINUED)

Mnemonic, Operands	Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
			MSb			LSb		
LITERAL OPERATIONS								
ADDLW k	Add literal and WREG	1	0000	1111	kkkk	kkkk	C, DC, Z, OV, N	
ANDLW k	AND literal with WREG	1	0000	1011	kkkk	kkkk	Z, N	
IORLW k	Inclusive OR literal with WREG	1	0000	1001	kkkk	kkkk	Z, N	
LFSR f, k	Load FSR(f) with a 12-bit	2	1110	1110	00ff	kkkk	None	
	literal (k)		1111	0000	kkkk	kkkk		
MOVLB k	Move literal to BSR<3:0>	1	0000	0001	0000	kkkk	None	
MOVLW k	Move literal to WREG	1	0000	1110	kkkk	kkkk	None	
MULLW k	Multiply literal with WREG	1	0000	1101	kkkk	kkkk	None	
RETLW k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
SUBLW k	Subtract WREG from literal	1	0000	1000	kkkk	kkkk	C, DC, Z, OV, N	
XORLW k	Exclusive OR literal with WREG	1	0000	1010	kkkk	kkkk	Z, N	
DATA MEMORY ↔ PROGRAM MEMORY OPERATIONS								
TBLRD*	Table Read	2	0000	0000	0000	1000	None	
TBLRD*+	Table Read with post-increment		0000	0000	0000	1001	None	
TBLRD*-	Table Read with post-decrement		0000	0000	0000	1010	None	
TBLRD+*	Table Read with pre-increment		0000	0000	0000	1011	None	
TBLWT*	Table Write	2 (5)	0000	0000	0000	1100	None	
TBLWT*+	Table Write with post-increment		0000	0000	0000	1101	None	
TBLWT*-	Table Write with post-decrement		0000	0000	0000	1110	None	
TBLWT+*	Table Write with pre-increment		0000	0000	0000	1111	None	

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.

6: Microchip Assembler MASM automatically defaults destination bit 'd' to '1', while access bit 'a' defaults to '1' or '0' according to address of register being used.

13.1 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
- N = ?
- OV = ?
- C = ?
- DC = ?
- Z = ?

After Instruction
- WREG = 0x25
- N = 0
- OV = 0
- C = 0
- DC = 0
- Z = 0

ADDWF **ADD WREG to f**

Syntax: [*label*] ADDWF 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 Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `ADDWF REG, W`

Before Instruction
- WREG = 0x17
- REG = 0xC2
- N = ?
- OV = ?
- C = ?
- DC = ?
- Z = ?

After Instruction
- WREG = 0xD9
- REG = 0xC2
- N = 1
- OV = 0
- C = 0
- DC = 0
- Z = 0

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

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

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

Operation: (WREG) + (f) + (C) → dest

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

Encoding:

0010	00da	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'. If 'a' is 0, the Access Bank will be selected. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `ADDWFC REG, W`

Before Instruction
C = 1
REG = 0x02
WREG = 0x4D
N = ?
OV = ?
DC = ?
Z = ?

After Instruction
C = 0
REG = 0x02
WREG = 0x50
N = 0
OV = 0
DC = 0
Z = 0

ANDLW **AND literal with WREG**

Syntax: [*label*] ANDLW k

Operands: $0 \le k \le 255$

Operation: (WREG) .AND. k → WREG

Status Affected: N,Z

Encoding:

0000	1011	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'	Process Data	Write to WREG

Example: `ANDLW 0x5F`

Before Instruction
WREG = 0xA3
N = ?
Z = ?

After Instruction
WREG = 0x03
N = 0
Z = 0

ANDWF **AND WREG with f**

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

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

Operation: (WREG) .AND. (f) → dest

Status Affected: N,Z

Encoding:

0001	01da	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' (default). If 'a' is 0, the Access Bank will be selected. If 'a' is 1, the bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `ANDWF REG, W`

Before Instruction
WREG = 0x17
REG = 0xC2
N = ?
Z = ?

After Instruction
WREG = 0x02
REG = 0xC2
N = 0
Z = 0

BC **Branch if Carry**

Syntax: [*label*] BC n

Operands: $-128 \le n \le 127$

Operation: if carry bit is '1'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1110	0010	nnnn	nnnn

Description: If the Carry bit is '1', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

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

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BC 5`

Before Instruction
PC = address (HERE)

After Instruction
If Carry = 1;
PC = address (HERE+12)
If Carry = 0;
PC = address (HERE+2)

BCF **Bit Clear f**

Syntax: [*label*] BCF f, b [,a]

Operands: $0 \le f \le 255$
$0 \le b \le 7$
$a \in [0,1]$

Operation: $0 \rightarrow f<b>$

Status Affected: None

Encoding:

1001	bbba	ffff	ffff

Description: Bit 'b' in register 'f' is cleared. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `BCF FLAG_REG, 7`

Before Instruction
FLAG_REG = 0xC7

After Instruction
FLAG_REG = 0x47

BN **Branch if Negative**

Syntax: [*label*] BN n

Operands: $-128 \le n \le 127$

Operation: if negative bit is '1'
$(PC) + 2 + 2n \rightarrow PC$

Status Affected: None

Encoding:

1110	0110	nnnn	nnnn

Description: If the Negative bit is '1', then the program will branch.

The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

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

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example:
```
HERE    BN   Jump
```

Before Instruction
PC = address (HERE)

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

BNC **Branch if Not Carry**

Syntax: [*label*] BNC n

Operands: -128 ≤ n ≤ 127

Operation: if carry bit is '0'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1110	0011	nnnn	nnnn

Description: If the Carry bit is '0', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

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

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BNC Jump`

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

BNN **Branch if Not Negative**

Syntax: [*label*] BNN n

Operands: -128 ≤ n ≤ 127

Operation: if negative bit is '0'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1110	0111	nnnn	nnnn

Description: If the Negative bit is '0', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

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

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BNN Jump`

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

BNOV **Branch if Not Overflow**

Syntax: [*label*] BNOV n

Operands: -128 ≤ n ≤ 127

Operation: if overflow bit is '0'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1110	0101	nnnn	nnnn

Description: If the Overflow bit is '0', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

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

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BNOV Jump`

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

BNZ **Branch if Not Zero**

Syntax: [*label*] BNZ n

Operands: -128 ≤ n ≤ 127

Operation: if zero bit is '0'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1110	0001	nnnn	nnnn

Description: If the Zero bit is '0', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

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

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BNZ Jump`

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

BRA **Unconditional Branch**

Syntax: [*label*] BRA n

Operands: -1024 ≤ n ≤ 1023

Operation: (PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1101	0nnn	nnnn	nnnn

Description: 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'	Process Data	Write to PC
No operation	No operation	No operation	No operation

Example: `HERE BRA Jump`

Before Instruction
PC = address (HERE)

After Instruction
PC = address (Jump)

BSF **Bit Set f**

Syntax: [*label*] BSF f, b [,a]

Operands: 0 ≤ f ≤ 255
0 ≤ b ≤ 7
a ∈ [0,1]

Operation: 1 → f<b>

Status Affected: None

Encoding:

1000	bbba	ffff	ffff

Description: Bit 'b' in register 'f' is set. If 'a' is 0 Access Bank will be selected, overriding the BSR value. If 'a' is 1, 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 register 'f'

Example: `BSF FLAG_REG, 7, 1`

Before Instruction
FLAG_REG = 0x0A

After Instruction
FLAG_REG = 0x8A

BTFSC **Bit Test File, Skip if Clear**

Syntax: [*label*] BTFSC f, b [,a]

Operands: $0 \le f \le 255$
$0 \le b \le 7$
$a \in [0,1]$

Operation: skip if (f<b>) = 0

Status Affected: None

Encoding:

1011	bbba	ffff	ffff

Description: If bit 'b' in register 'f' is 0, then the next instruction is skipped.
If bit 'b' is 0, then the next instruction fetched during the current instruction execution is discarded, and a NOP is executed instead, making this a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

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

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:
```
HERE    BTFSC   FLAG, 1, ACCESS
FALSE   :
TRUE    :
```

Before Instruction
PC = address (HERE)

After Instruction
If FLAG<1> = 0;
PC = address (TRUE)
If FLAG<1> = 1;
PC = address (FALSE)

BTFSS **Bit Test File, Skip if Set**

Syntax: [*label*] BTFSS f, b [,a]

Operands: $0 \le f \le 255$
$0 \le b < 7$
$a \in [0,1]$

Operation: skip if (f<b>) = 1

Status Affected: None

Encoding:

1010	bbba	ffff	ffff

Description: If bit 'b' in register 'f' is 1 then the next instruction is skipped.
If bit 'b' is 1, then the next instruction fetched during the current instruction execution, is discarded and an NOP is executed instead, making this a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

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

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:
```
HERE    BTFSS   FLAG, 1, ACCESS
FALSE   :
TRUE    :
```

Before Instruction
PC = address (HERE)

After Instruction
If FLAG<1> = 0;
PC = address (FALSE)
If FLAG<1> = 1;
PC = address (TRUE)

BTG **Bit Toggle f**

Syntax: [*label*] BTG f, b [,a]

Operands: $0 \le f \le 255$
$0 \le b < 7$
$a \in [0,1]$

Operation: $(\overline{f<b>}) \rightarrow f<b>$

Status Affected: None

Encoding:

0111	bbba	ffff	ffff

Description: Bit 'b' in data memory location 'f' is inverted. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `BTG PORTB, 4`

Before Instruction:
PORTB = 0111 0101 [0x35]

After Instruction:
PORTB = 0110 0101 [0x25]

BOV **Branch if Overflow**

Syntax: [*label*] BOV n

Operands: $-128 \le n \le 127$

Operation: if overflow bit is '1'
$(PC) + 2 + 2n \rightarrow PC$

Status Affected: None

Encoding:

1110	0100	nnnn	nnnn

Description: If the Overflow bit is '1', then the program will branch.

The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

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

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BOV Jump`

Before Instruction
PC = address (HERE)

After Instruction
If Overflow = 1;
PC = address (Jump)
If Overflow = 0;
PC = address (HERE+2)

BZ **Branch if Zero**

Syntax: [*label*] BZ n

Operands: -128 ≤ n ≤ 127

Operation: if Zero bit is '1'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1110	0000	nnnn	nnnn

Description: If the Zero bit is '1', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

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

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BZ Jump`

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

CALL **Subroutine Call**

Syntax: [*label*] CALL k [,s]

Operands: 0 ≤ k ≤ 1048575
s ∈ [0,1]

Operation: (PC) + 4 → TOS,
k → PC<20:1>,
if s = 1
(WREG) → WS,
(STATUS) → STATUSS,
(BSR) → BSRS

Status Affected: None

Encoding:

1st word (k<7:0>)	1110	110s	$k_7$kkk	$kkkk_0$
2nd word(k<19:8>)	1111	k_{19}kkk	kkkk	$kkkk_8$

Description: Subroutine call of entire 2M byte memory range. First, return address (PC+ 4) is pushed onto the return stack. If 's' = 1, the WREG, STATUS and BSR registers are also pushed into their respective shadow registers, WS, STATUSS and BSRS. If 's' = 0, no update occurs (default). Then the 20-bit value 'k' is loaded into PC<20:1>. CALL is a two-cycle instruction.

Words: 2

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>,	Push PC to stack	Read literal 'k'<19:8>, Write to PC
No operation	No operation	No operation	No operation

Example: `HERE CALL THERE, FAST`

Before Instruction
PC = Address (HERE)
After Instruction
PC = Address (THERE)
TOS = Address (HERE + 4)
WS = WREGREG
BSRS = BSR
STATUSS = STATUS

CLRF **Clear f**

Syntax: [*label*] CLRF f [,a]

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

Operation: 000h → f
1 → Z

Status Affected: Z

Encoding:

0110	101a	ffff	ffff

Description: Clears the contents of the specified register. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: CLRF FLAG_REG

Before Instruction
FLAG_REG = 0x5A
Z = ?

After Instruction
FLAG_REG = 0x00
Z = 0

CLRWDT **Clear Watchdog Timer**

Syntax: [*label*] CLRWDT

Operands: None

Operation: 000h → WDT,
000h → WDT postscaler,
1 → $\overline{TO}$,
1 → $\overline{PD}$

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

Encoding:

0000	0000	0000	0100

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

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	Process Data	No operation

Example: CLRWDT

Before Instruction
WDT counter = ?
WDT postscaler = ?
$\overline{TO}$ = ?
$\overline{PD}$ = ?

After Instruction
WDT counter = 0x00
WDT postscaler = 0
$\overline{TO}$ = 1
$\overline{PD}$ = 1

COMF **Complement f**

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

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

Operation: $(\overline{f}) \rightarrow$ dest

Status Affected: N,Z

Encoding:

0001	11da	ffff	ffff

Description: The contents of register 'f' are complemented. If 'd' is 0 the result is stored in 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, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `COMF REG`

Before Instruction
REG = 0x13
N = ?
Z = ?

After Instruction
REG = 0x13
WREG = 0xEC
N = 1
Z = 0

CPFSEQ **Compare f with WREG, skip if f = WREG**

Syntax: [*label*] CPFSEQ f [,a]

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

Operation: (f) – (WREG), skip if (f) = (WREG) (unsigned comparison)

Status Affected: None

Encoding:

0110	001a	ffff	ffff

Description: Compares the contents of data memory location 'f' to the contents of WREG by performing an unsigned subtraction.
If 'f' = WREG, then the fetched instruction is discarded and an `NOP` is executed instead making this a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

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

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

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

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

CPFSGT — **Compare f with WREG, skip if f > WREG**

Syntax: [*label*] CPFSGT f [,a]

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

Operation: (f) – (WREG),
skip if (f) > (WREG)
(unsigned comparison)

Status Affected: None

Encoding:

0110	010a	ffff	ffff

Description: Compares the contents of data memory location 'f' to the contents of the WREG by performing an unsigned subtraction.
If the contents of 'f' are greater than the contents of , then the fetched instruction is discarded and a NOP is executed instead making this a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

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

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:
```
HERE      CPFSGT REG
NGREATER  :
GREATER   :
```

Before Instruction
PC = Address (HERE)
WREG = ?
After Instruction
If REG > WREG;
PC = Address (GREATER)
If REG ≤ WREG;
PC = Address (NGREATER)

CPFSLT — **Compare f with WREG, skip if f < WREG**

Syntax: [*label*] CPFSLT f [,a]

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

Operation: (f) – (WREG),
skip if (f) < (WREG)
(unsigned comparison)

Status Affected: None

Encoding:

0110	000a	ffff	ffff

Description: Compares the contents of data memory location 'f' to the contents of WREG by performing an unsigned subtraction.
If the contents of 'f' are less than the contents of WREG, then the fetched instruction is discarded and a NOP is executed instead making this a two-cycle instruction. If 'a' is 0, the Access Bank will be selected. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

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

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:
```
HERE    CPFSLT REG
NLESS   :
LESS    :
```

Before Instruction
PC = Address (HERE)
WREG = ?
After Instruction
If REG < WREG;
PC = Address (LESS)
If REG ≥ WREG;
PC = Address (NLESS)

DAW **Decimal Adjust WREG Register**

Syntax: [*label*] DAW

Operands: None

Operation: If [WREG<3:0> >9] or [DC = 1] then
(WREG<3:0>) + 6 → W<3:0>;
else
(WREG<3:0>) → W<3:0>;

If [WREG<7:4> >9] or [C = 1] then
(WREG<7:4>) + 6 → WREG<7:4>;
else
(WREG<7:4>) → WREG<7:4>;

Status Affected: C

Encoding:

0000	0000	0000	0111

Description: DAW adjusts the eight bit value in WREG resulting from the earlier addition of two variables (each in packed BCD format) and produces a correct packed BCD result.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register WREG	Process Data	Write WREG

Example1:

```
DAW
```

Before Instruction
WREG = 0xA5
C = 0
DC = 0

After Instruction
WREG = 0x05
C = 1
DC = 0

Example 2:

Before Instruction
WREG = 0xCE
C = 0
DC = 0

After Instruction
WREG = 0x34
C = 1
DC = 0

DECF **Decrement f**

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

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

Operation: (f) – 1 → dest

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

Encoding:

0000	01da	ffff	ffff

Description: Decrement 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, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example:

```
DECF    CNT
```

Before Instruction
CNT = 0x01
Z = 0

After Instruction
CNT = 0x00
Z = 1

DECFSZ **Decrement f, skip if 0**

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

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

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

Status Affected: None

Encoding:

0010	11da	ffff	ffff

Description: The contents of register 'f' are decremented. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed back in register 'f' (default).
If the result is 0, the next instruction, which is already fetched, is discarded, and a NOP is executed instead making it a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

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

Q Cycle Activity:

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

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:
```
HERE    DECFSZ  CNT
        GOTO    LOOP
CONTINUE
```

Before Instruction
PC = Address (HERE)

After Instruction
CNT = CNT - 1
If CNT = 0;
PC = Address (CONTINUE)
If CNT ≠ 0;
PC = Address (HERE+2)

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

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

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

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

Status Affected: None

Encoding:

0100	11da	ffff	ffff

Description: The contents of register 'f' are decremented. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed back in register 'f' (default).
If the result is not 0, the next instruction, which is already fetched, is discarded, and a NOP is executed instead making it a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

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

Q Cycle Activity:

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

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:
```
HERE    DCFSNZ  TEMP
ZERO    :
NZERO   :
```

Before Instruction
TEMP = ?

After Instruction
TEMP = TEMP - 1,
If TEMP = 0;
PC = Address (ZERO)
If TEMP ≠ 0;
PC = Address (NZERO)

GOTO **Unconditional Branch**

Syntax: [*label*] GOTO k

Operands: $0 \le k \le 1048575$

Operation: $k \rightarrow PC<20:1>$

Status Affected: None

Encoding:

1st word (k<7:0>)	1110	1111	$k_7$kkk	kkkk$_0$
2nd word(k<19:8>)	1111	k_{19}kkk	kkkk	kkkk$_8$

Description: GOTO allows an unconditional branch anywhere within entire 2M byte memory range. The 20-bit value 'k' is loaded into PC<20:1>. GOTO is always a two-cycle instruction.

Words: 2

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>,	No operation	Read literal 'k'<19:8>, Write to PC
No operation	No operation	No operation	No operation

Example: `GOTO THERE`

After Instruction
PC = Address (THERE)

INCF **Increment f**

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

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

Operation: (f) + 1 → dest

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

Encoding:

0010	10da	ffff	ffff

Description: The contents of register 'f' are incremented. 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' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `INCF CNT`

Before Instruction
CNT = 0xFF
Z = 0
C = ?
DC = ?

After Instruction
CNT = 0x00
Z = 1
C = 1
DC = 1

INCFSZ **Increment f, skip if 0**

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

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

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

Status Affected: None

Encoding:

0011	11da	ffff	ffff

Description: The contents of register 'f' are incremented. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed back in register 'f' (default).
If the result is 0, the next instruction, which is already fetched, is discarded, and a NOP is executed instead making it a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

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

Q Cycle Activity:

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

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:
```
HERE    INCFSZ   CNT
NZERO   :
ZERO    :
```

Before Instruction
PC = Address (HERE)

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

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

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

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

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

Status Affected: None

Encoding:

0100	10da	ffff	ffff

Description: The contents of register 'f' are incremented. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed back in register 'f' (default).
If the result is not 0, the next instruction, which is already fetched, is discarded, and a NOP is executed instead making it a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

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

Q Cycle Activity:

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

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:
```
HERE    INFSNZ   REG
ZERO
NZERO
```

Before Instruction
PC = Address (HERE)

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

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
N = ?
Z = ?

After Instruction
WREG = 0xBF
N = 1
Z = 0

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' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `IORWF RESULT, W`

Before Instruction
RESULT = 0x13
WREG = 0x91
N = ?
Z = ?

After Instruction
RESULT = 0x13
WREG = 0x93
N = 1
Z = 0

LFSR **Load FSR**

Syntax: [*label*] LFSR f,k

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

Operation: $k \rightarrow FSRf$

Status Affected: None

Encoding:

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

Description: The 12-bit literal 'k' is loaded into the file select register pointed to by 'f'

Words: 2

Cycles: 2

Q Cycle Activity:

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

Example: `LFSR FSR2, 0x3AB`

After Instruction
FSR2H = 0x03
FSR2L = 0xAB

MOVF **Move f**

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

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

Operation: $f \rightarrow dest$

Status Affected: N,Z

Encoding:

0101	00da	ffff	ffff

Description: The contents of register 'f' is moved to a destination dependent upon the status of 'd'. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed back in register 'f' (default). Location 'f' can be anywhere in the 256 byte Bank. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `MOVF REG, W`

Before Instruction
REG = 0x22
WREG = 0xFF
N = ?
Z = ?

After Instruction
REG = 0x22
WREG = 0x22
N = 0
Z = 0

MOVFF **Move f to f**

Syntax: [*label*] MOVFF f_s,f_d

Operands: $0 \le f_s \le 4095$
$0 \le f_d \le 4095$

Operation: $(f_s) \rightarrow f_d$

Status Affected: None

Encoding:

1st word (source)	1100	ffff	ffff	$ffff_s$
2nd word (destin.)	1111	ffff	ffff	$ffff_d$

Description: The contents of source register 'f_s' are moved to destination register 'f_d'. Location of source 'f_s' can be anywhere in the 4096 byte data space (000h to FFFh), and location of destination 'f_d' can also be anywhere from 000h to FFFh.

Either source or destination can be WREG (a useful special situation).

MOVFF is particularly useful for transferring a data memory location to a peripheral register (such as the transmit buffer or an I/O port).

The MOVFF instruction cannot use the PCL, TOSU, TOSH or TOSL as the destination register.

Words: 2

Cycles: 2 (3)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f' (src)	Process Data	No operation
Decode	No operation No dummy read	No operation	Write register 'f' (dest)

Example: MOVFF 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: $0 \le k \le 255$

Operation: $k \rightarrow BSR$

Status Affected: None

Encoding:

0000	0001	kkkk	kkkk

Description: The 8-bit literal 'k' is loaded into the Bank Select Register (BSR).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write literal 'k' to BSR

Example: MOVLB 0x01

Before Instruction
BSR register = 0x0F

After Instruction
BSR register = 0x01

MOVLW **Move literal to WREG**

Syntax: [*label*] MOVLW k

Operands: $0 \le k \le 255$

Operation: $k \rightarrow$ WREG

Status Affected: None

Encoding:

0000	1110	kkkk	kkkk

Description: The eight bit literal 'k' is loaded into WREG.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `MOVLW 0x5A`

After Instruction
WREG = 0x5A

MOVWF **Move WREG to f**

Syntax: [*label*] MOVWF f [,a]

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

Operation: (WREG) $\rightarrow$ f

Status Affected: None

Encoding:

0110	111a	ffff	ffff

Description: Move data from WREG to register 'f'. Location 'f' can be anywhere in the 256 byte Bank. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `MOVWF REG`

Before Instruction
WREG = 0x4F
REG = 0xFF

After Instruction
WREG = 0x4F
REG = 0x4F

MULLW **Multiply Literal with WREG**

Syntax: [*label*] MULLW k

Operands: $0 \le k \le 255$

Operation: (WREG) x k → PRODH:PRODL

Status Affected: None

Encoding:

0000	1101	kkkk	kkkk

Description: An unsigned multiplication is carried out between the contents of WREG and the 8-bit literal 'k'. The 16-bit result is placed in PRODH:PRODL register pair. PRODH contains the high byte.

WREG is unchanged.

None of the status flags are affected.

Note that neither overflow nor carry is possible in this operation. A zero result is possible but not detected.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write registers PRODH: PRODL

Example: `MULLW 0xC4`

Before Instruction
- WREG = 0xE2
- PRODH = ?
- PRODL = ?

After Instruction
- WREG = 0xE2
- PRODH = 0xAD
- PRODL = 0x08

MULWF **Multiply WREG with f**

Syntax: [*label*] MULWF f [,a]

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

Operation: (WREG) x (f) → PRODH:PRODL

Status Affected: None

Encoding:

0000	001a	ffff	ffff

Description: An unsigned multiplication is carried out between the contents of WREG and the register file location 'f'. The 16-bit result is stored in the PRODH:PRODL register pair. PRODH contains the high byte.

Both WREG and 'f' are unchanged.

None of the status flags are affected.

Note that neither overflow nor carry is possible in this operation. A zero result is possible but not detected. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write registers PRODH: PRODL

Example: `MULWF REG`

Before Instruction
- WREG = 0xC4
- REG = 0xB5
- PRODH = ?
- PRODL = ?

After Instruction
- WREG = 0xC4
- REG = 0xB5
- PRODH = 0x8A
- PRODL = 0x94

NEGF **Negate f**

Syntax: [*label*] NEGF f [,a]

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

Operation: $(\overline{f}) + 1 \rightarrow f$

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

Encoding:

0110	110a	ffff	ffff

Description: Location 'f' is negated using two's complement. The result is placed in the data memory location 'f'. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: NEGF REG

Before Instruction
REG = 0011 1010 [0x3A]
N = ?
OV = ?
C = ?
DC = ?
Z = ?

After Instruction
REG = 1100 0110 [0xC6]
N = 1
OV = 0
C = 0
DC = 0
Z = 0

NOP **No Operation**

Syntax: [*label*] NOP

Operands: None

Operation: No operation

Status Affected: None

Encoding:

0000 1111	0000 xxxx	0000 xxxx	0000 xxxx

Description: No operation.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation

Example:

None.

POP **Pop Top of Return Stack**

Syntax: [*label*] POP

Operands: None

Operation: (TOS) → bit bucket

Status Affected: None

Encoding:

0000	0000	0000	0110

Description: The TOS value is pulled off the return stack and is discarded. The TOS value then becomes the previous value that was pushed onto the return stack.

This instruction is provided to enable the user to properly manage the return stack to incorporate a software stack.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	Pop TOS value	No operation

Example:

```
POP
GOTO    NEW
```

Before Instruction
TOS = 0031A2h
Stack (1 level down) = 014332h

After Instruction
TOS = 014332h
PC = NEW

PUSH **Push Top of Return Stack**

Syntax: [*label*] PUSH

Operands: None

Operation: (PC+2) → TOS

Status Affected: None

Encoding:

0000	0000	0000	0101

Description: The PC+2 is pushed onto the top of the return stack. The previous TOS value is pushed down on the stack.

This instruction allows implementing a software stack by modifying TOS, and then push it onto the return stack.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Push PC+2 onto return stack	No operation	No operation

Example:

```
PUSH
```

Before Instruction
TOS = 00345Ah
PC = 000124h

After Instruction
PC = 000126h
TOS = 000126h
Stack (1 level down) = 00345Ah

RCALL **Relative Call**

Syntax: [*label*] RCALL n

Operands: -1024 ≤ n ≤ 1023

Operation: (PC) + 2 → TOS,
(PC) + 2 + 2n → 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 $\overline{\text{MCLR}}$ Reset.

Status Affected: All

Encoding:

0000	0000	1111	1111

Description: This instruction provides a way to execute a $\overline{\text{MCLR}}$ 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

RETFIE **Return from Interrupt**

Syntax: [*label*] RETFIE [s]

Operands: s ∈ [0,1]

Operation: (TOS) → PC,
1 → GIE/GIEH or PEIE/GIEL,
if s = 1
(WS) → WREG,
(STATUSS) → STATUS,
(BSRS) → BSR,
PCLATU, PCLATH are unchanged.

Status Affected: None

Encoding:

0000	0000	0001	000s

Description: Return from Interrupt. Stack is popped and Top-of-Stack (TOS) is loaded into the PC. Interrupts are enabled by setting the either the high or low priority global interrupt enable bit. If 's' = 1, the contents of the shadow registers WS, STATUSS and BSRS are loaded into their corresponding registers, WREG, STATUS and BSR. If 's' = 0, no update of these registers occurs (default).

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	Pop PC from stack Set GIEH or GIEL
No operation	No operation	No operation	No operation

Example: `RETFIE 1`

After Interrupt
PC = TOS
WREG = WS
BSR = BSRS
STATUS = STATUSS
GIE/GIEH, PEIE/GIEL = 1

RETLW **Return Literal to WREG**

Syntax: [*label*] RETLW k

Operands: 0 ≤ k ≤ 255

Operation: k → W,
(TOS) → PC,
PCLATU, PCLATH are unchanged

Status Affected: None

Encoding:

0000	1100	kkkk	kkkk

Description: W is loaded with the eight bit literal 'k'. The program counter is loaded from the top of the stack (the return address). The high address latch (PCLATH) remains unchanged.

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Pop PC from stack, write to WREG
No operation	No operation	No operation	No operation

Example:

```
   CALL TABLE  ; WREG contains table
               ;  offset value
               ;  WREG now has
               ;  table value
   :
TABLE
   ADDWF PCL   ; WREG = offset
   RETLW k0    ; Begin table
   RETLW k1    ;
   :
   :
   RETLW kn    ; End of table
```

Before Instruction
WREG = 0x07
After Instruction
WREG = value of kn

RETURN **Return from Subroutine**

Syntax: [*label*] RETURN [s]

Operands: s ∈ [0,1]

Operation: (TOS) → PC,
if s = 1
(WS) → W,
(STATUSS) → STATUS,
(BSRS) → BSR,
PCLATU, PCLATH are unchanged

Status Affected: None

Encoding:

0000	0000	0001	001s

Description: Return from subroutine. The stack is popped and the top of the stack (TOS) is loaded into the program counter. If 's' = 1, the contents of the shadow registers WS, STATUSS and BSRS are loaded into their corresponding registers, WREG, STATUS and BSR. If 's' = 0, no update of these registers occurs (default).

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	Process Data	Pop PC from stack
No operation	No operation	No operation	No operation

Example:
```
RETURN
```
After Call
PC = TOS

```
RETURN FAST
```
Before Instruction
WRG = 0x04
STATUS = 0x00
BSR = 0x00

After Instruction
WREG = 0x04
STATUS = 0x00
BSR = 0x00
PC = TOS

RLCF **Rotate Left f through Carry**

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

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

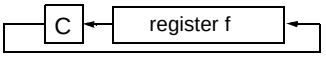
Operation: (f<n>) → dest<n+1>,
(f<7>) → C,
(C) → dest<0>

Status Affected: C,N,Z

Encoding:

0011	01da	ffff	ffff

Description: The contents of register 'f' are rotated one bit to the left through the Carry Flag. If 'd' is 0 the result is placed 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, the Bank will be selected as per the BSR value.



Words: 1

Cycles: 1

Q Cycle Activity:

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

Example:
```
RLCF    REG, W
```
Before Instruction
REG = 1110 0110
C = 0
N = ?
Z = ?

After Instruction
REG = 1110 0110
WREG = 1100 1100
C = 1
N = 1
Z = 0

RLNCF **Rotate Left f (no carry)**

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

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

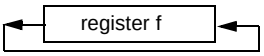
Operation: (f<n>) → dest<n+1>,
(f<7>) → dest<0>

Status Affected: N,Z

Encoding:

0100	01da	ffff	ffff

Description: The contents of register 'f' are rotated one bit to the left. If 'd' is 0 the result is placed 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, the Bank will be selected as per the BSR value.



Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: RLNCF REG

Before Instruction
REG = 1010 1011
N = ?
Z = ?

After Instruction
REG = 0101 0111
N = 0
Z = 0

RRCF **Rotate Right f through Carry**

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

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

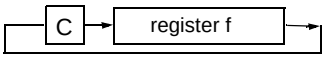
Operation: (f<n>) → dest<n-1>,
(f<0>) → C,
(C) → dest<7>

Status Affected: C,N,Z

Encoding:

0011	00da	ffff	ffff

Description: The contents of register 'f' are rotated one bit to the right through the Carry Flag. 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' is 1, the Bank will be selected as per the BSR value.



Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: RRCF REG, W

Before Instruction
REG = 1110 0110
C = 0
N = ?
Z = ?

After Instruction
REG = 1110 0110
WREG = 0111 0011
C = 0
N = 0
Z = 0

RRNCF **Rotate Right f (no carry)**

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

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

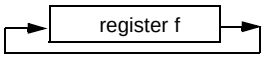
Operation: (f<n>) → dest<n-1>,
(f<0>) → dest<7>

Status Affected: N,Z

Encoding:

0100	00da	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' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.



Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1:
```
RRNCF   REG
```
Before Instruction
```
REG  =  1101 0111
N    =  ?
Z    =  ?
```
After Instruction
```
REG  =  1110 1011
N    =  1
Z    =  0
```

Example 2:
```
RRNCF   REG, 0, 0
```
Before Instruction
```
WREG =  ?
REG  =  1101 0111
N    =  ?
Z    =  ?
```
After Instruction
```
WREG =  1110 1011
REG  =  1101 0111
N    =  1
Z    =  0
```

SETF **Set f**

Syntax: [*label*] SETF f [,a]

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

Operation: FFh → f

Status Affected: None

Encoding:

0110	100a	ffff	ffff

Description: The contents of the specified register are set to FFh. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example:
```
SETF    REG
```
Before Instruction
```
REG  =  0x5A
```
After Instruction
```
REG  =  0xFF
```

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, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

SUBFWB (Cont.)

Example 1: `SUBFWB REG`

Before Instruction
REG = 3
WREG = 2
C = 1

After Instruction
REG = 0xFF
WREG = 2
C = 0
Z = 0
N = 1 ; result is negative

Example 2: `SUBFWB REG`

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`

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: $k - (WREG) \rightarrow WREG$

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 = 0xFF ; (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, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

SUBWF **Subtract WREG from f (cont'd)**

Example 1:
```
SUBWF   REG
```
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, W
```
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
```
Before Instruction
```
REG   =  1
WREG  =  2
C     =  ?
```
After Instruction
```
REG   =  0xFF  ;(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, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

SUBWFB **Subtract WREG from f with Borrow (cont'd)**

Example 1: `SUBWFB REG`

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, W`

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`

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 nibbles in 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, the Bank will be selected as per the BSR value.
Words:	1
Cycles:	1

Q Cycle Activity:

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

Example: `SWAPF REG`

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

TBLWT **Table Write**

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

Operands: None

Operation: if TBLWT*,
(TABLAT) → Prog Mem (TBLPTR) or Holding Register;
TBLPTR - No Change;
if TBLWT*+,
(TABLAT) → Prog Mem (TBLPTR) or Holding Register;
(TBLPTR) +1 → TBLPTR;
if TBLWT*-,
(TABLAT) → Prog Mem (TBLPTR) or Holding Register;
(TBLPTR) -1 → TBLPTR;
if TBLWT+*,
(TBLPTR) +1 → TBLPTR;
(TABLAT) → Prog Mem (TBLPTR) or Holding Register;

Status Affected: None

Encoding:

0000	0000	0000	11nn nn=0 * =1 *+ =2 *- =3 +*

Description: This instruction is used to program the contents of Program Memory (P.M.).

The TBLPTR (a 21-bit pointer) points to each byte in the program memory. TBLPTR has a 2 MBtye address range. The LSb of the TBLPTR selects which byte of the program memory location to access.

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

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

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

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

Words: 1

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

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation
No operation	No operation (Read TABLAT)	No operation	No operation (Write to Holding Register or Memory)

TBLWT **Table Write (Continued)**

Example 1: TBLWT *+;

Before Instruction
TABLAT = 0x55
TBLPTR = 0x00A356
MEMORY(0x00A356) = 0xFF

After Instructions (table write completion)
TABLAT = 0x55
TBLPTR = 0x00A357
MEMORY(0x00A356) = 0x55

Example 2: TBLWT +*;

Before Instruction
TABLAT = 0x34
TBLPTR = 0x01389A
MEMORY(0x01389A) = 0xFF
MEMORY(0x01389B) = 0xFF

After Instruction (table write completion)
TABLAT = 0x34
TBLPTR = 0x01389B
MEMORY(0x01389A) = 0xFF
MEMORY(0x01389B) = 0x34

TSTFSZ **Test f, skip if 0**

Syntax: [*label*] TSTFSZ f [,a]

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

Operation: skip if f = 0

Status Affected: None

Encoding:

0110	011a	ffff	ffff

Description: If 'f' = 0, the next instruction, fetched during the current instruction execution, is discarded and a NOP is executed making this a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

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

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:
```
HERE    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: [*label*] XORLW k

Operands: $0 \le k \le 255$

Operation: (WREG) .XOR. k → WREG

Status Affected: N,Z

Encoding:

0000	1010	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'	Process Data	Write to WREG

Example: XORLW 0xAF

Before Instruction
WREG = 0xB5
N = ?
Z = ?

After Instruction
WREG = 0x1A
N = 0
Z = 0

XORWF **Exclusive OR WREG with f**

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

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

Operation: (WREG) .XOR. (f) → dest

Status Affected: N,Z

Encoding:

0001	10da	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' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: XORWF REG

Before Instruction
REG = 0xAF
WREG = 0xB5
N = ?
Z = ?

After Instruction
REG = 0x1A
WREG = 0xB5
N = 0
Z = 0

14.0 DEVELOPMENT SUPPORT

The PICmicro® microcontrollers are supported with a full range of hardware and software development tools:

- Integrated Development Environment
  - MPLAB® IDE Software
- Assemblers/Compilers/Linkers
  - MPASM™ Assembler
  - MPLAB C17 and MPLAB C18 C Compilers
  - MPLINK™ Object Linker/ MPLIB™ Object Librarian
- Simulators
  - MPLAB SIM Software Simulator
- Emulators
  - MPLAB ICE 2000 In-Circuit Emulator
  - ICEPIC™ In-Circuit Emulator
- In-Circuit Debugger
  - MPLAB ICD
- Device Programmers
  - PRO MATE® II Universal Device Programmer
  - PICSTART® Plus Entry-Level Development Programmer
- Low Cost Demonstration Boards
  - PICDEM™ 1 Demonstration Board
  - PICDEM 2 Demonstration Board
  - PICDEM 3 Demonstration Board
  - PICDEM 17 Demonstration Board
  - KEELOQ® Demonstration Board

14.1 MPLAB Integrated Development Environment Software

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

- An interface to debugging tools
  - simulator
  - programmer (sold separately)
  - emulator (sold separately)
  - in-circuit debugger (sold separately)
- A full-featured editor
- A project manager
- Customizable toolbar and key mapping
- A status bar
- On-line help

The MPLAB IDE allows you to:

- Edit your source files (either assembly or ‘C’)
- One touch assemble (or compile) and download to PICmicro emulator and simulator tools (automatically updates all project information)
- Debug using:
  - source files
  - absolute listing file
  - machine code

The ability to use MPLAB IDE with multiple debugging tools allows users to easily switch from the cost-effective simulator to a full-featured emulator with minimal retraining.

14.2 MPASM Assembler

The MPASM assembler is a full-featured universal macro assembler for all PICmicro MCU’s.

The MPASM assembler has a command line interface and a Windows shell. It can be used as a stand-alone application on a Windows 3.x or greater system, or it can be used through MPLAB IDE. The MPASM assembler generates relocatable object files for the MPLINK object linker, Intel® standard HEX files, MAP files to detail memory usage and symbol reference, an absolute LST file that contains source lines and generated machine code, and a COD file for debugging.

The MPASM assembler features include:

- Integration into MPLAB IDE projects.
- User-defined macros to streamline assembly code.
- Conditional assembly for multi-purpose source files.
- Directives that allow complete control over the assembly process.

14.3 MPLAB C17 and MPLAB C18 C Compilers

The MPLAB C17 and MPLAB C18 Code Development Systems are complete ANSI ‘C’ compilers for Microchip’s PIC17CXXX and PIC18CXXX family of microcontrollers, respectively. These compilers provide powerful integration capabilities and ease of use not found with other compilers.

For easier source level debugging, the compilers provide symbol information that is compatible with the MPLAB IDE memory display.

14.4 MPLINK Object Linker/ MPLIB Object Librarian

The MPLINK object linker combines relocatable objects created by the MPASM assembler and the MPLAB C17 and MPLAB C18 C compilers. It can also link relocatable objects from pre-compiled libraries, using directives from a linker script.

The MPLIB object librarian is a librarian for pre-compiled code to be used with the MPLINK object linker. When a routine from a library is called from another source file, only the modules that contain that routine will be linked in with the application. This allows large libraries to be used efficiently in many different applications. The MPLIB object librarian manages the creation and modification of library files.

The MPLINK object linker features include:

- Integration with MPASM assembler and MPLAB C17 and MPLAB C18 C compilers.
- Allows all memory areas to be defined as sections to provide link-time flexibility.

The MPLIB object librarian features include:

- Easier linking because single libraries can be included instead of many smaller files.
- Helps keep code maintainable by grouping related modules together.
- Allows libraries to be created and modules to be added, listed, replaced, deleted or extracted.

14.5 MPLAB SIM Software Simulator

The MPLAB SIM software simulator allows code development in a PC-hosted environment by simulating the PICmicro series microcontrollers on an instruction level. On any given instruction, the data areas can be examined or modified and stimuli can be applied from a file, or user-defined key press, to any of the pins. The execution can be performed in single step, execute until break, or trace mode.

The MPLAB SIM simulator fully supports symbolic debugging using the MPLAB C17 and the MPLAB C18 C compilers and the MPASM assembler. The software simulator offers the flexibility to develop and debug code outside of the laboratory environment, making it an excellent multi-project software development tool.

14.6 MPLAB ICE High Performance Universal In-Circuit Emulator with MPLAB IDE

The MPLAB ICE universal in-circuit emulator is intended to provide the product development engineer with a complete microcontroller design tool set for PICmicro microcontrollers (MCUs). Software control of the MPLAB ICE in-circuit emulator is provided by the MPLAB Integrated Development Environment (IDE), which allows editing, building, downloading and source debugging from a single environment.

The MPLAB ICE 2000 is a full-featured emulator system with enhanced trace, trigger and data monitoring features. Interchangeable processor modules allow the system to be easily reconfigured for emulation of different processors. The universal architecture of the MPLAB ICE in-circuit emulator allows expansion to support new PICmicro microcontrollers.

The MPLAB ICE in-circuit emulator system has been designed as a real-time emulation system, with advanced features that are generally found on more expensive development tools. The PC platform and Microsoft® Windows environment were chosen to best make these features available to you, the end user.

14.7 ICEPIC In-Circuit Emulator

The ICEPIC low cost, in-circuit emulator is a solution for the Microchip Technology PIC16C5X, PIC16C6X, PIC16C7X and PIC16CXXX families of 8-bit One-Time-Programmable (OTP) microcontrollers. The modular system can support different subsets of PIC16C5X or PIC16CXXX products through the use of interchangeable personality modules, or daughter boards. The emulator is capable of emulating without target application circuitry being present.

14.8 MPLAB ICD In-Circuit Debugger

Microchip's In-Circuit Debugger, MPLAB ICD, is a powerful, low cost, run-time development tool. This tool is based on the FLASH PICmicro MCUs and can be used to develop for this and other PICmicro microcontrollers. The MPLAB ICD utilizes the in-circuit debugging capability built into the FLASH devices. This feature, along with Microchip's In-Circuit Serial Programming™ protocol, offers cost-effective in-circuit FLASH debugging from the graphical user interface of the MPLAB Integrated Development Environment. This enables a designer to develop and debug source code by watching variables, single-stepping and setting break points. Running at full speed enables testing hardware in real-time.

14.9 PRO MATE II Universal Device Programmer

The PRO MATE II universal device programmer is a full-featured programmer, capable of operating in stand-alone mode, as well as PC-hosted mode. The PRO MATE II device programmer is CE compliant.

The PRO MATE II device programmer has programmable V_{DD} and V_{PP} supplies, which allow it to verify programmed memory at V_{DD} min and V_{DD} max for maximum reliability. It has an LCD display for instructions and error messages, keys to enter commands and a modular detachable socket assembly to support various package types. In stand-alone mode, the PRO MATE II device programmer can read, verify, or program PICmicro devices. It can also set code protection in this mode.

14.10 PICSTART Plus Entry Level Development Programmer

The PICSTART Plus development programmer is an easy-to-use, low cost, prototype programmer. It connects to the PC via a COM (RS-232) port. MPLAB Integrated Development Environment software makes using the programmer simple and efficient.

The PICSTART Plus development programmer supports all PICmicro devices with up to 40 pins. Larger pin count devices, such as the PIC16C92X and PIC17C76X, may be supported with an adapter socket. The PICSTART Plus development programmer is CE compliant.

14.11 PICDEM 1 Low Cost PICmicro Demonstration Board

The PICDEM 1 demonstration board 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 user can program the sample microcontrollers provided with the PICDEM 1 demonstration board on a PRO MATE II device programmer, or a PICSTART Plus development programmer, and easily test firmware. The user can also connect the PICDEM 1 demonstration board to the MPLAB ICE in-circuit emulator and download the firmware to the emulator for testing. A 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.

14.12 PICDEM 2 Low Cost PIC16CXX Demonstration Board

The PICDEM 2 demonstration board 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 demonstration board on a PRO MATE II device programmer, or a PICSTART Plus development programmer, and easily test firmware. The MPLAB ICE in-circuit emulator may also be used with the PICDEM 2 demonstration board to test firmware. A 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 I^2C™ bus and separate headers for connection to an LCD module and a keypad.

14.13 PICDEM 3 Low Cost PIC16CXXX Demonstration Board

The PICDEM 3 demonstration board 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 an 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 demonstration board on a PRO MATE II device programmer, or a PICSTART Plus development programmer with an adapter socket, and easily test firmware. The MPLAB ICE in-circuit emulator may also be used with the PICDEM 3 demonstration board to test firmware. A prototype area has been provided to the user for adding 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 thermistor and separate headers for connection to an external LCD module and a keypad. Also provided on the PICDEM 3 demonstration board is a LCD panel, with 4 commons and 12 segments, that is capable of displaying time, temperature and day of the week. The PICDEM 3 demonstration board provides an additional RS-232 interface and Windows 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.

14.14 PICDEM 17 Demonstration Board

The PICDEM 17 demonstration board is an evaluation board that demonstrates the capabilities of several Microchip microcontrollers, including PIC17C752, PIC17C756A, PIC17C762 and PIC17C766. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5-inch disk. A programmed sample is included and the user may erase it and program it with the other sample programs using the PRO MATE II device programmer, or the PICSTART Plus development programmer, and easily debug and test the sample code. In addition, the PICDEM 17 demonstration board supports downloading of programs to and executing out of external FLASH memory on board. The PICDEM 17 demonstration board is also usable with the MPLAB ICE in-circuit emulator, or the PICMASTER emulator and all of the sample programs can be run and modified using either emulator. Additionally, a generous prototype area is available for user hardware.

14.15 KEELOQ Evaluation and Programming Tools

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

TABLE 14-1: DEVELOPMENT TOOLS FROM MICROCHIP

		PIC12CXXX	PIC14000	PIC16C5X	PIC16C6X	PIC16CXXX	PIC16F62X	PIC16C7X	PIC16C7XX	PIC16C8X	PIC16F8XX	PIC16C9XX	PIC17C4X	PIC17C7XX	PIC18CXX2	PIC18FXXX	24CXX/ 25CXX/ 93CXX	HCSXXX	MCRFXXX	MCP2510
Software Tools	MPLAB® Integrated Development Environment	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓				
	MPLAB® C17 C Compiler												✓	✓						
	MPLAB® C18 C Compiler														✓	✓				
	MPASM™ Assembler/ MPLINK™ Object Linker	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
Emulators	MPLAB® ICE In-Circuit Emulator	✓		✓	✓	✓	✓**		✓	✓	✓	✓	✓	✓	✓	✓				
	ICEPIC™ In-Circuit Emulator	✓		✓	✓	✓			✓	✓		✓								
Debugger	MPLAB® ICD In-Circuit Debugger				✓*			✓*			✓					✓				
Programmers	PICSTART® Plus Entry Level Development Programmer	✓	✓	✓	✓	✓	✓**	✓	✓	✓	✓	✓	✓	✓	✓	✓				
	PRO MATE® II Universal Device Programmer	✓	✓	✓	✓	✓	✓**	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
Demo Boards and Eval Kits	PICDEM™ 1 Demonstration Board			✓		✓		✓†		✓			✓							
	PICDEM™ 2 Demonstration Board				✓†			✓†							✓	✓				
	PICDEM™ 3 Demonstration Board											✓								
	PICDEM™ 14A Demonstration Board		✓																	
	PICDEM™ 17 Demonstration Board													✓						
	KEELOQ® Evaluation Kit																	✓		
	KEELOQ® Transponder Kit																	✓		
	microID™ Programmer's Kit																		✓	
	125 kHz microID™ Developer's Kit																		✓	
	125 kHz Anticollision microID™ Developer's Kit																		✓	
	13.56 MHz Anticollision microID™ Developer's Kit																		✓	
	MCP2510 CAN Developer's Kit																			✓

* Contact the Microchip Technology Inc. web site at www.microchip.com for information on how to use the MPLAB® ICD In-Circuit Debugger (DV164001) with PIC16C62, 63, 64, 65, 72, 73, 74, 76, 77.

** Contact Microchip Technology Inc. for availability date.

† Development tool is available on select devices.

NOTES:

15.0 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings (†)

Ambient temperature under bias ..... -55°C to +125°C
Storage temperature ..... -65°C to +150°C
Voltage on any pin with respect to V_{SS} (except V_{DD} and $\overline{MCLR}$) ..... -0.3V to (V_{DD} + 0.3V)
Voltage on V_{DD} with respect to V_{SS} ..... -0.3V to +7.5V
Voltage on $\overline{MCLR}$ with respect to V_{SS} **(Note 2)** ..... 0V to +13.25V
Total power dissipation **(Note 1)** 1.0W
Maximum current out of V_{SS} pin ..... 300 mA
Maximum current into V_{DD} pin ..... 250 mA
Input clamp current, I_{IK} (V_I < 0 or V_I > V_{DD}) ..... ±20 mA
Output clamp current, I_{OK} (V_O < 0 or V_O > V_{DD}) ..... ±20 mA
Maximum output current sunk by any I/O pin ..... 25 mA
Maximum output current sourced by any I/O pin ..... 25 mA
Maximum current sunk by PORTB ..... 150 mA
Maximum current sourced by PORTB ..... 150 mA

Note 1: Power dissipation is calculated as follows:

$$Pdis = V_{DD} \times \{I_{DD} - \sum I_{OH}\} + \sum \{(V_{DD}-V_{OH}) \times I_{OH}\} + \sum(Vol \times I_{OL})$$

† NOTICE: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

FIGURE 15-1: PIC18F010/020 VOLTAGE-FREQUENCY GRAPH

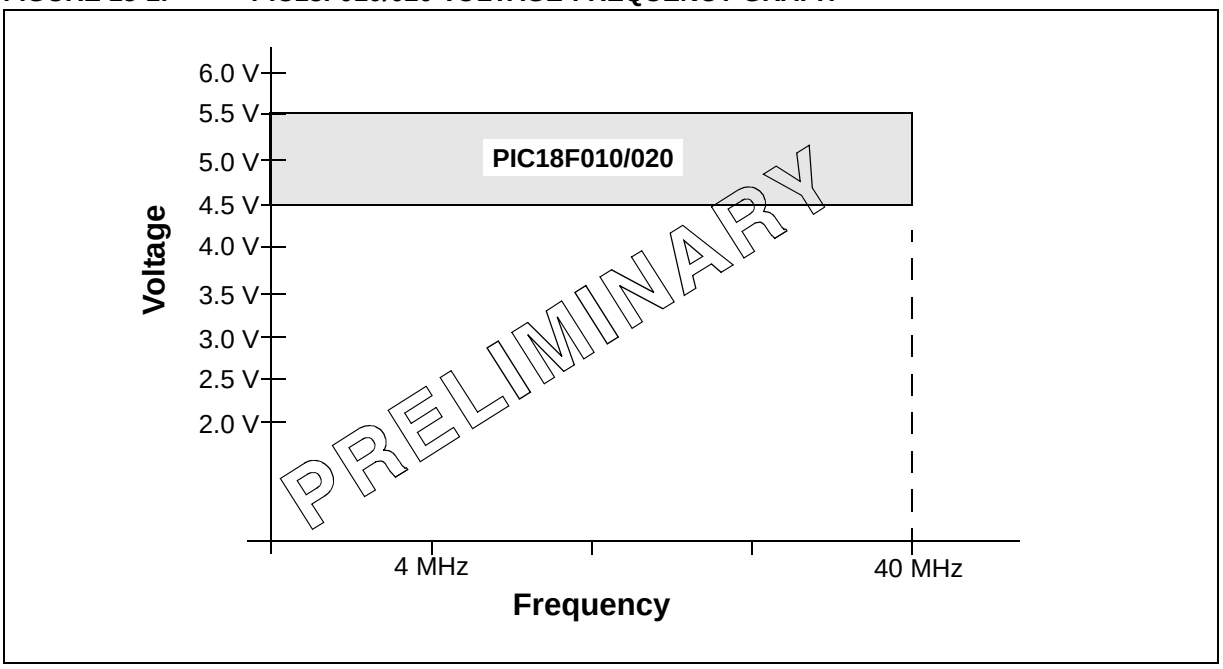
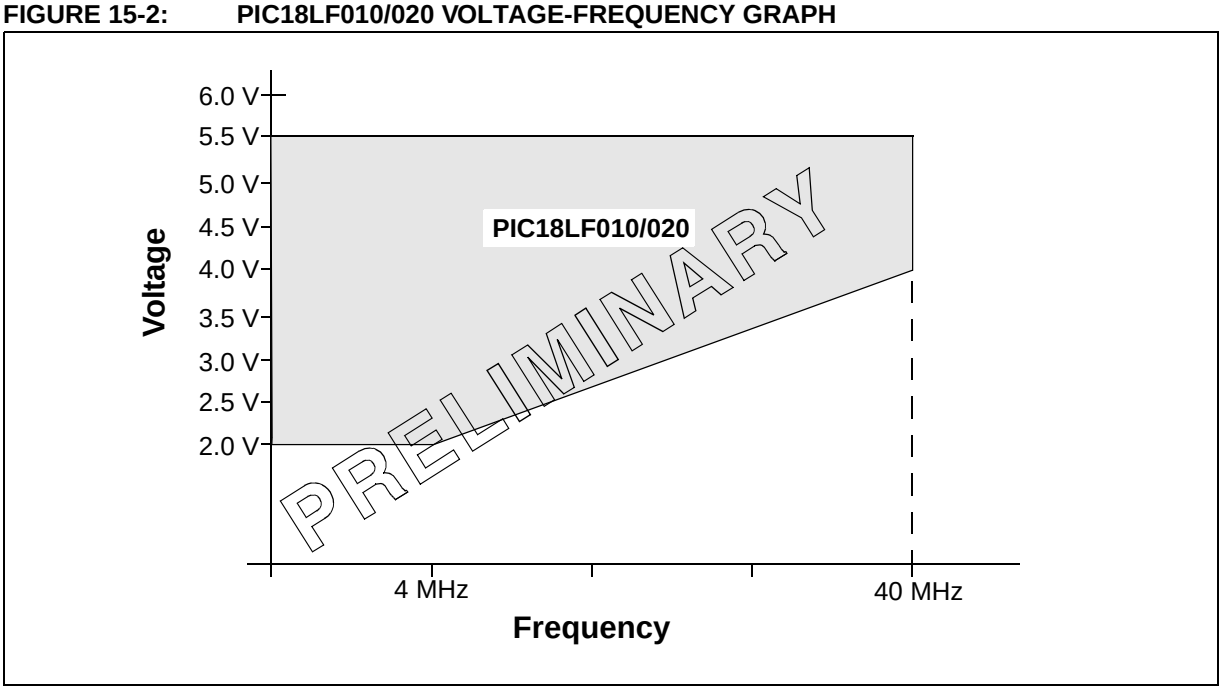


FIGURE 15-2: PIC18LF010/020 VOLTAGE-FREQUENCY GRAPH



15.1 DC Characteristics

PIC18F010/020 (Industrial unless otherwise stated)			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial				
Param No.	**Symbol**	**Characteristic**	**Min**	**Typ†**	**Max**	**Units**	**Conditions**
D001	VDD	**Supply Voltage**	2.0	—	5.5	V	XT, LP, RC, EC and Internal osc mode
D001A			4.5	—	5.5	V	HS osc mode
D002*	VDR	**RAM Data Retention Voltage(1)**	1.5	—	—	V	
D003	VPOR	**VDD Start Voltage** to ensure internal Power-on Reset signal	—	VSS	—	V	
D004*	SVDD	**VDD Rise Rate** to ensure internal Power-on Reset signal	0.05	—	—	V/ms	
D010	IDD	**Supply Current(2)**					XT, RC, Internal osc modes
			—	TBD	4	mA	FOSC = 4 MHz, VDD = 3.0V
							HS osc mode
			—	TBD	50	mA	FOSC = 25 MHz, VDD = 5.5V
							EC osc mode
			—	TBD	45	mA	FOSC = 40 MHz, VDD = 5.5V
							LP osc mode
			—	TBD	48	µA	FOSC = 32 kHz, VDD = 3.0V
D020	IPD	**Power-down Current(3)**	—	<1	—	µA	VDD = 3.0V, -40°C to +85°C
		Module Differential Current(5)					
D021	ΔIWDT	**Watchdog Timer**	—	6.5	12	µA	VDD = 3.0V
D423	ΔILVD	**Low Voltage Detect**	—	30	50	µA	Brown-out disabled
D022A	ΔIBOR	**Brown-out Reset**	—	30	50	µA	Low Voltage Detect disabled

* These parameters are characterized, but not tested.

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

Note 1: This is the limit to which VDD can be lowered without losing RAM data.

2: The supply current is mainly a function of the operating voltage and frequency. Other factors, such as I/O pin loading and switching rate, oscillator type, internal code execution pattern and temperature, also have an impact on the current consumption.

The test conditions for all IDD measurements in active operation mode are:
OSC1 = external square wave, from rail to rail; all I/O pins tri-stated, pulled to VDD
$\overline{MCLR}$ = VDD; WDT enabled/disabled as specified.

3: The power-down current in SLEEP mode does not depend on the oscillator type. Power-down current is measured with the part in SLEEP mode, with all I/O pins in hi-impedance state and tied to VDD or VSS.

4: For RC osc mode, current through REXT is not included. The current through the resistor can be estimated by the formula Ir = VDD/2REXT (mA) with REXT in kOhm.

5: The Δ current is the additional current consumed when the peripheral is enabled. This current should be added to the base current.

15.2 DC Characteristics: PIC18F010/020 (Industrial unless otherwise stated)

DC CHARACTERISTICS			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial				
Param No.	**Symbol**	**Characteristic**	**Min**	**Typ†**	**Max**	**Units**	**Conditions**
	VIL	**Input Low Voltage**					
		I/O ports:					
D030		with TTL buffer	VSS	—	0.15VDD	V	4.5V ≤ VDD ≤ 5.5V
D030A			VSS	—	0.8V	V	4.5V ≤ VDD ≤ 5.5V
D031A		All others (Schmitt Trigger)	VSS	—	0.2VDD	V	For entire VDD range
D032		$\overline{MCLR}$	VSS	—	0.2VDD	V	
D032A		OSC1 (XT, HS, LP modes)	VSS	—	0.2VDD	V	
D033		OSC1 (RC mode)	VSS	—	0.3VDD	V	**(Note 1)**
	VIH	**Input High Voltage**					
		I/O ports:					
D040		with TTL buffer	2.0	—	VDD	V	4.5V ≤ VDD ≤ 5.5V
D040A			0.25VDD + 0.8V	—	VDD	V	For entire VDD range
D041A		All others (Schmitt Trigger)	0.8VDD	—	VDD	V	For entire VDD range
D042		$\overline{MCLR}$	0.8VDD	—	VDD	V	
D042A		OSC1 (XT, HS and LP modes)	0.7VDD	—	VDD	V	**(Note 1)**
D043		OSC1 (RC mode)	0.9VDD	—	VDD	V	
D070	IPURB	**PORTB Weak Pull-up Current**	50	250	400	µA	VDD = 5V, VPIN = VSS
	IIL	**Input Leakage Current (Notes 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, XT, HS, LP and EC osc mode
	VOL	**Output Low Voltage**					
D080		I/O ports	—	—	0.6	V	IOL = 8.5mA, VDD = 4.5V, -40°C to +85°C
D083		OSC2/CLKOUT (RC or EC osc mode)	—	—	0.6	V	IOL = 1.6mA, VDD = 4.5V, -40°C to +85°C
	VOH	**Output High Voltage**					
D090		I/O ports **(Note 3)**	VDD - 0.7	—	—	V	IOH = -3.0mA, VDD = 4.5V, -40°C to +85°C
D092		OSC2/CLKOUT (RC or EC osc mode)	VDD - 0.7	—	—	V	IOH = -1.3mA, VDD = 4.5V, -40°C to +85°C
		Capacitive Loading Specs on Output Pins					
D100*	COSC2	OSC2 pin	—	—	15	pF	In XT, HS and LP modes when external clock is used to drive OSC1.
D101*	CIO	All I/O pins and OSC2 (Internal or EC osc mode)	—	—	50	pF	

* These parameters are characterized, but not tested.

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

Note 1: In Internal Oscillator mode, the OSC1/CLKIN pin is a Schmitt Trigger input. It is not recommended that the PICmicro MCU be driven with an external clock in Internal Oscillator 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.

15.3 DC Characteristics: LVD-BOR

FIGURE 15-3: LOW VOLTAGE DETECT CHARACTERISTICS

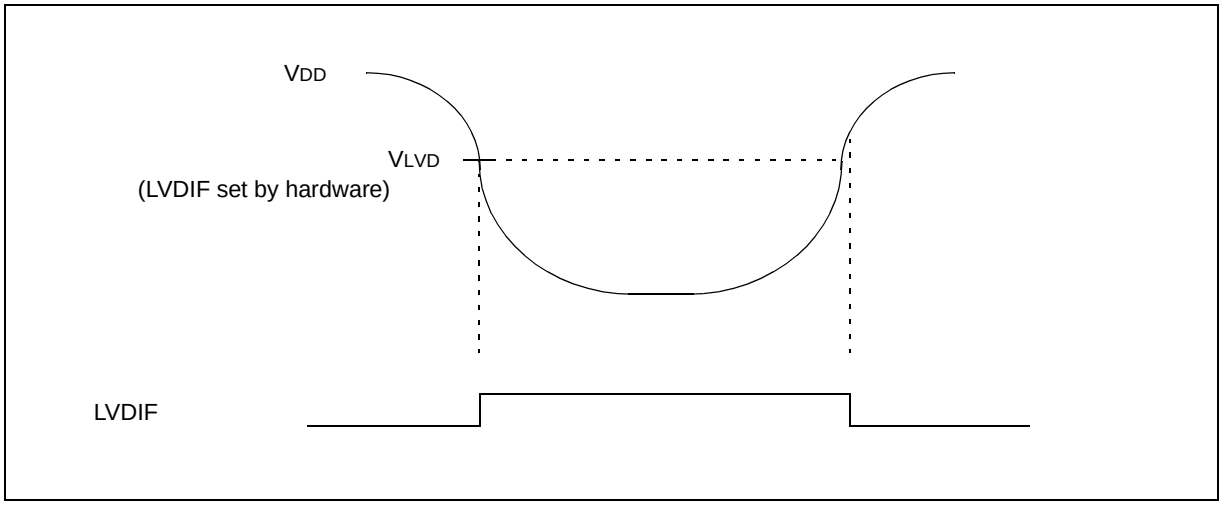


TABLE 15-1: ELECTRICAL CHARACTERISTICS: LVD

			VCC = 2.5V to 5.5V Industrial (I): TAMB = -40°C to +85°C					
Param No.	**Characteristic**		**Symbol**	**Min**	**Typ†**	**Max**	**Units**	**Conditions**
D420	LVD Voltage on VDD transition high to low	LVV = 0000	VPLVD	—	1.9	—	V	VPLVD = 2.0V selected
		LVV = 0001		—	2.0	—	V	VPLVD = 2.1V selected
		LVV = 0010		—	2.1	—	V	VPLVD = 2.2V selected
		LVV = 0011		—	2.2	—	V	VPLVD = 2.3V selected
		LVV = 0100		—	2.3	—	V	VPLVD = 2.4V selected
		LVV = 0101		—	2.4	—	V	VPLVD = 2.5V selected
		LVV = 0110		—	2.5	—	V	VPLVD = 2.6V selected
		LVV = 0111		—	2.6	—	V	VPLVD = 2.7V selected
		LVV = 1000		—	2.7	—	V	VPLVD = 2.8V selected
		LVV = 1001		—	2.8	—	V	VPLVD = 2.9V selected
		LVV = 1010		—	2.9	—	V	VPLVD = 3.0V selected
		LVV = 1011		—	3.0	—	V	VPLVD = 3.1V selected
		LVV = 1100		—	3.5	—	V	VPLVD = 3.2V selected
		LVV = 1101		—	4.0	—	V	VPLVD = 4.4V selected
		LVV = 1110		—	—	—	V	VPLVD = 4.7V selected
D421	LVD Voltage Drift Temperature coefficient		TCVOUT	—	15	50	ppm/°C	
D422	LVD Voltage Drift with respect to VDD Regulation		ΔVLVD/ ΔVDD	—	—	50	µV/V	

Note 1: Production tested at TAMB = 25°C. Specifications over temp limits are insured by characterization.

FIGURE 15-4: BROWN-OUT RESET CHARACTERISTICS

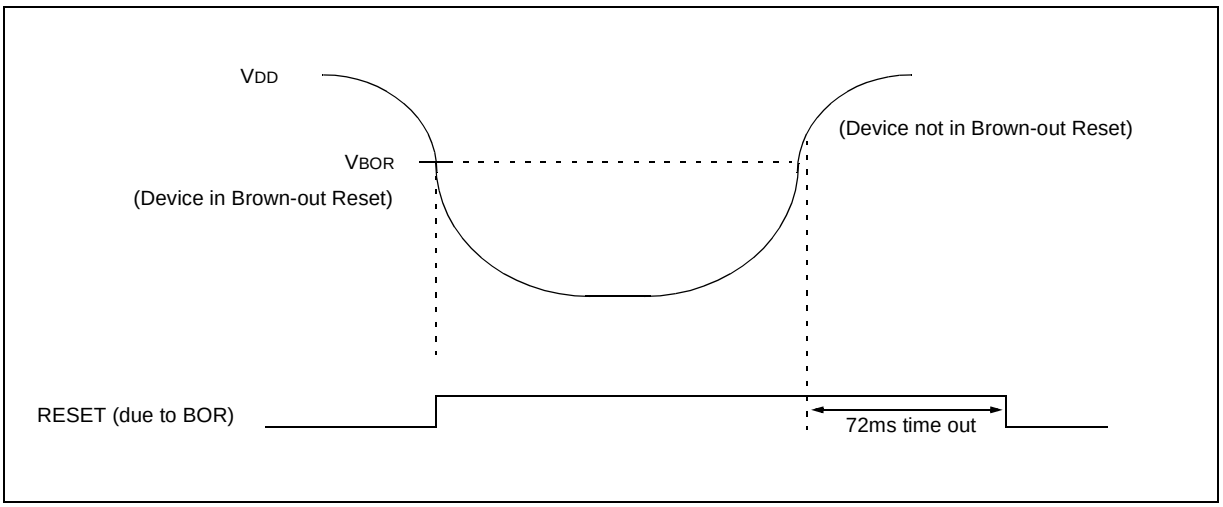


TABLE 15-2: ELECTRICAL CHARACTERISTICS: BOR

		VCC = 2.5V to 5.5V Industrial (I): TAMB = -40°C to +85°C					
Param No.	**Characteristic**	**Symbol**	**Min**	**Typ**	**Max**	**Units**	**Conditions**
D005	BOR Voltage on VDD transition high to low	VBOR	2.0	—	2.15	V	
D006	BOR Voltage Drift Temperature coefficient	TCVOUT	—	15	50	ppm/°C	
D006A	BOR Voltage Drift with respect to VDD Regulation	ΔVBOR/ ΔVDD	—	—	50	μV/V	

Note 1: Production tested at TAMB = 25°C. Specifications over temp limits are insured by characterization.

15.4 AC Characteristics: (Commercial, Industrial)

15.4.1 TIMING PARAMETER SYMBOLOGY

The timing parameter symbols have been created following one of the following formats:

1. TppS2ppS		2. TppS	
T			
F	Frequency	T	Time

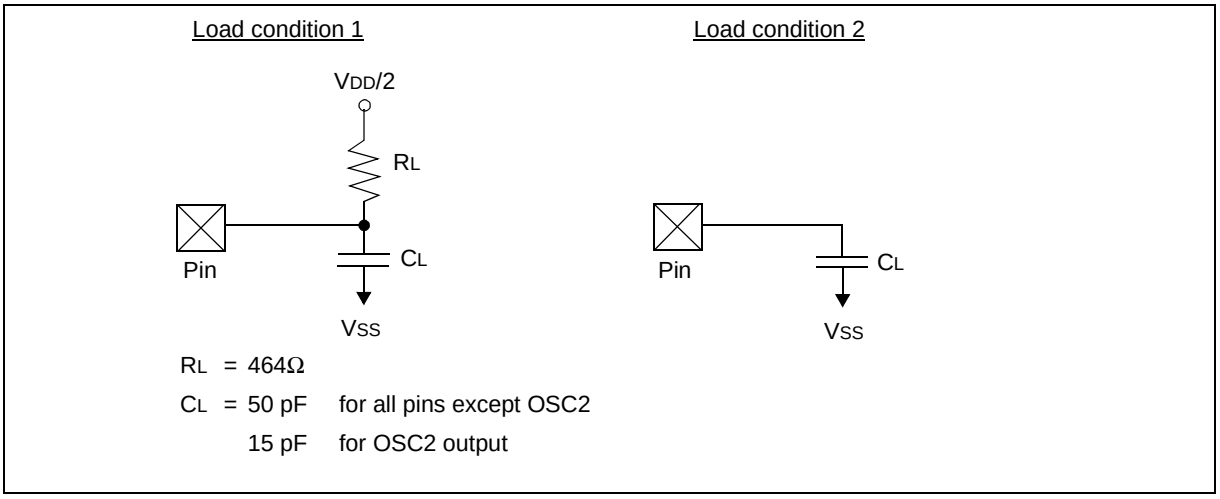
Lowercase letters (pp) and their meanings:

pp			
cc	CCP1	osc	OSC1
ck	CLKOUT	rd	$\overline{RD}$
cs	$\overline{CS}$	rw	$\overline{RD}$ or $\overline{WR}$
di	SDI	sc	SCK
do	SDO	ss	$\overline{SS}$
dt	Data in	t0	T0CKI
io	I/O port	t1	T1CKI
mc	$\overline{MCLR}$	wr	$\overline{WR}$

Uppercase letters and their meanings:

S			
F	Fall	R	Rise
H	High	V	Valid
I	Invalid (Hi-impedance)	Z	Hi-impedance
L	Low		
P	Period	High	High
		Low	Low

FIGURE 15-5: LOAD CONDITIONS



15.4.2 TIMING DIAGRAMS AND SPECIFICATIONS

FIGURE 15-6: EXTERNAL CLOCK TIMING

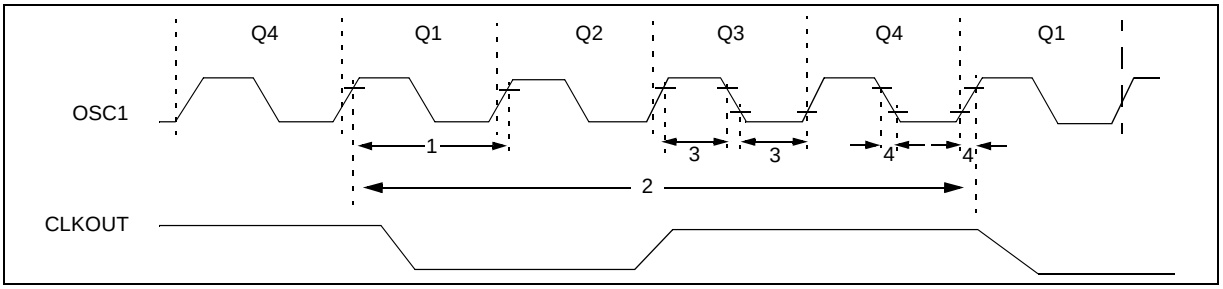


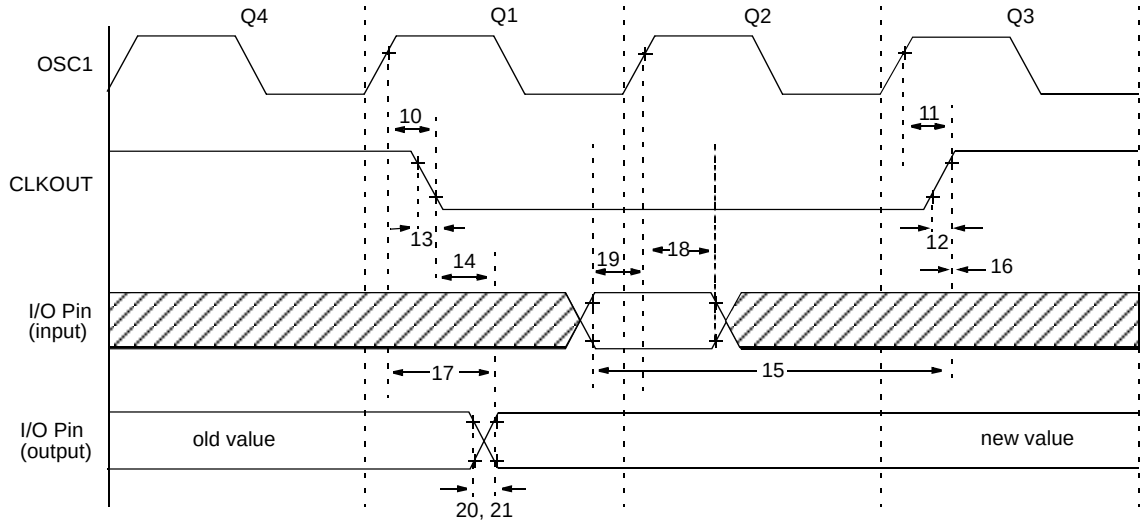
TABLE 15-3: EXTERNAL CLOCK TIMING REQUIREMENTS

Param. No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions
	FOSC	**External CLKIN Frequency (Note 1)**	DC	—	4	MHz	RC osc mode
			DC	—	4	MHz	XT osc mode
			DC	—	25	MHz	HS osc mode
			DC	—	40	MHz	EC osc mode
			DC	—	200	kHz	LP osc mode
		Oscillator Frequency (Note 1)	DC	—	4	MHz	RC osc mode
			0.1	—	4	MHz	XT osc mode
			4	—	25	MHz	HS osc mode
			4	—	8.25	MHz	HS osc mode
			5	—	200	kHz	LP osc mode
1	TOSC	**External CLKIN Period (Note 1)**	250	—	—	ns	RC osc mode
			100	—	—	ns	XT osc mode
			40	—	—	ns	HS osc mode
			120	—	—	ns	HS osc mode
			30	—	—	ns	EC osc mode
			5	—	—	µs	LP osc mode
		Oscillator Period (Note 1)	250	—	—	ns	RC osc mode
			0.1	—	10	µs	XT osc mode
			40	—	100	ns	HS osc mode
			120	—	100	ns	HS osc mode
			5	—	—	µs	LP osc mode
2	TCY	**Instruction Cycle Time (Note 1)**	100	Tcy	DC	ns	TCY = 4/System Clock, 40 MHz max
3	TosL, TosH	**External Clock in (OSC1) High or Low Time**	30	—	—	ns	XT oscillator
			2.5	—	—	µs	LP oscillator
			10	—	—	ns	HS oscillator
4	TosR, TosF	**External Clock in (OSC1) Rise or Fall Time**	—	—	20	ns	XT oscillator
			—	—	50	ns	LP oscillator
			—	—	7.5	ns	HS oscillator

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

Note 1: Instruction cycle period (TCY) equals four times the input oscillator time-base period. All specified values are based on characterization data for that particular oscillator type under standard operating conditions with the device executing code. Exceeding these specified limits may result in an unstable oscillator operation and/or higher than expected current consumption. All devices are tested to operate at "min." values with an external clock applied to the OSC1/CLKIN pin. When an external clock input is used, the "max." cycle time limit is "DC" (no clock) for all devices.

FIGURE 15-7: CLKOUT AND I/O TIMING



Note: Refer to Figure 15-5 for load conditions.

TABLE 15-4: CLKOUT AND I/O TIMING REQUIREMENTS

Parameter No.	Symbol	Characteristic	Min	Typ†	Max	Units	Conditions
10*	TosH2ckL	OSC1↑ to CLKOUT↓	—	75	200	ns	**(Note 1)**
11*	TosH2ckH	OSC1↑ to CLKOUT↑	—	75	200	ns	**(Note 1)**
12*	TckR	CLKOUT rise time	—	35	100	ns	**(Note 1)**
13*	TckF	CLKOUT fall time	—	35	100	ns	**(Note 1)**
14*	TckL2ioV	CLKOUT ↓ to Port out valid	—	—	0.5TCY + 10	ns	**(Note 1)**
15*	TioV2ckH	Port in valid before CLKOUT ↑	0.25TCY + 25	—	—	ns	**(Note 1)**
16*	TckH2ioI	Port in hold after CLKOUT ↑	0	—	—	ns	**(Note 1)**
17*	TosH2ioV	OSC1↑ (Q1 cycle) to Port out valid	—	50	150	ns	
18*	TosH2ioI	OSC1↑ (Q2 cycle) to Port input invalid (I/O in hold time)	100	—	—	ns	
19*	TioV2osH	Port input valid to OSC1↑ (I/O in setup time)	0	—	—	ns	
20*	TioR	Port output rise time	—	10	25	ns	
21*	TioF	Port output fall time	—	10	25	ns	
23††*	Trbp	RB5:RB0 change INT high or low time	TCY	—	—	ns	

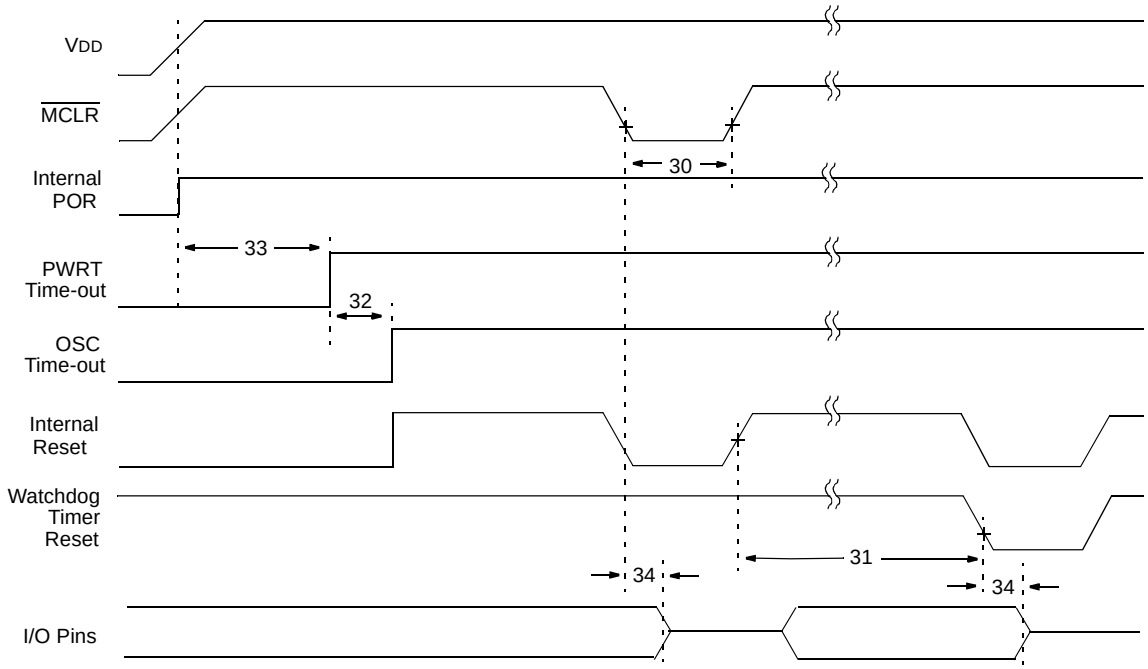
* These parameters are characterized but not tested.

† 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 asynchronous events not related to any internal clock edges.

Note 1: Measurements are taken in Internal Oscillator mode where CLKOUT output is 4 x TOSC.

FIGURE 15-8: RESET, WATCHDOG TIMER, OSCILLATOR START-UP TIMER AND POWER-UP TIMER TIMING



Note: Refer to Figure 15-5 for load conditions.

FIGURE 15-9: BROWN-OUT RESET TIMING

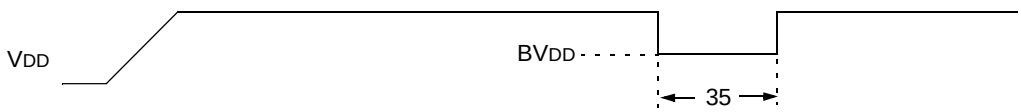


TABLE 15-5: RESET, WATCHDOG TIMER, OSCILLATOR START-UP TIMER, POWER-UP TIMER AND BROWN-OUT RESET REQUIREMENTS

Parameter No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions
30	TmcL	$\overline{\text{MCLR}}$ Pulse Width (low)	100	—	—	ns	VDD = 5V, -40°C to +85°C
31*	TWDT	Watchdog Timer Time-out Period (No Prescaler)	7	18	33	ms	VDD = 5V, -40°C to +85°C
32	TOST	Oscillation Start-up Timer Period	—	1024TOSC	—	—	TOSC = OSC1 period
33*	TPWRT	Power up Timer Period	28	72	132	ms	VDD = 5V, -40°C to +85°C
34	TIOZ	I/O Hi-Impedance from $\overline{\text{MCLR}}$ Low or Watchdog Timer Reset	—	—	100	ns	
35	TBOR	Brown-out Reset pulse width	100	—	—	µs	VDD ≤ BVDD (D005)

* 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.

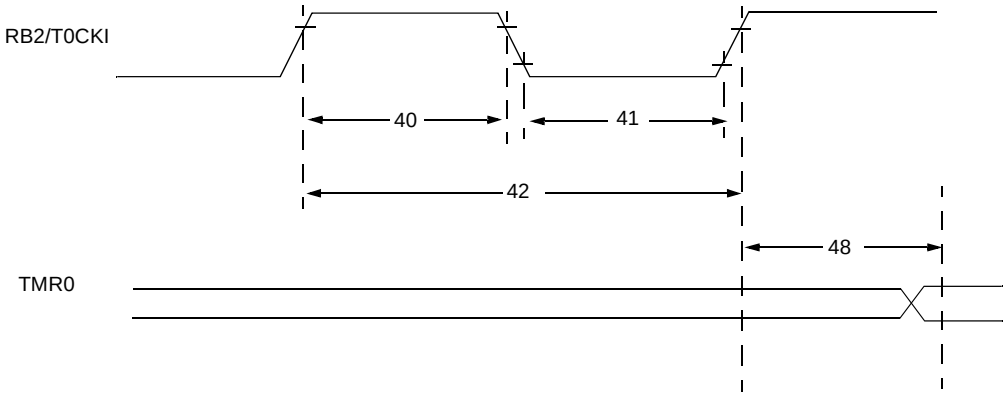
TABLE 15-6: BANDGAP START-UP TIME

Parameter No.	Symbol	Characteristic	Min	Typ†	Max	Units	Conditions
36*	TIVR	Internal Voltage Reference start-up time	—	20	50	µs	Defined as the time between the instant that the Internal Voltage Reference is enabled and the moment that the Internal Voltage Reference is stable.

* 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.

FIGURE 15-10: TIMER0 EXTERNAL CLOCK TIMINGS



Note: Refer to Figure 15-5 for load conditions.

TABLE 15-7: TIMER0 EXTERNAL CLOCK REQUIREMENTS

Param No.	Sym	Characteristic		Min	Typ†	Max	Units	Conditions
40*	Tt0H	T0CKI High Pulse Width	No Prescaler	0.5TCY + 5	—	—	ns	Must also meet parameter 42
			With Prescaler	10	—	—	ns	
41*	Tt0L	T0CKI Low Pulse Width	No Prescaler	0.5TCY + 5	—	—	ns	Must also meet parameter 42
			With Prescaler	10	—	—	ns	
42*	Tt0P	T0CKI Period	No Prescaler	TCY + 10	—	—	ns	
			With Prescaler	Greater of: 20 or $\frac{T_{CY} + 20}{N}$	—	—	ns	N = prescale value (2, 4, ..., 256)
48	TCKEZtmr1	Delay from external clock edge to timer increment		2Tosc	—	7Tosc	—	

* 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.

16.0 DC AND AC CHARACTERISTICS GRAPHS AND TABLES

Graphs and Tables not available at this time.

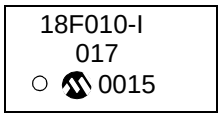
NOTES:

17.0 PACKAGING INFORMATION

17.1 Package Marking Information

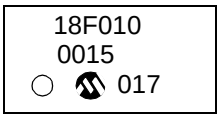
8-Lead PDIP (Skinny DIP)

Example



8-Lead SOIC

8-Lead SOIC

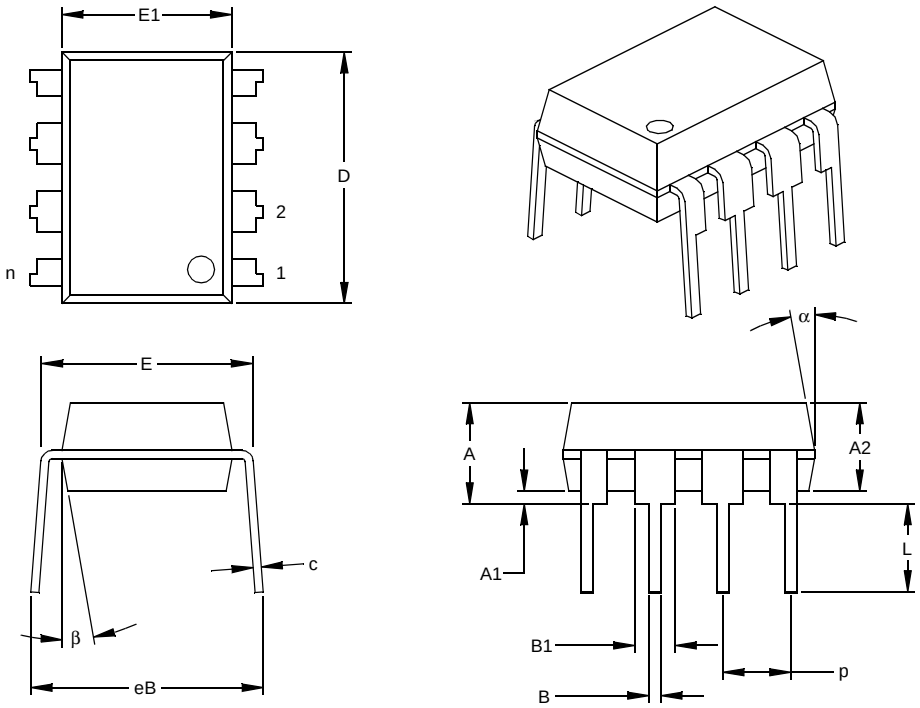


Legend:	XX...X	Customer specific information*
	YY	Year code (last 2 digits of calendar year)
	WW	Week code (week of January 1 is week ‘01’)
	NNN	Alphanumeric traceability code

Note: In the event the full Microchip part number cannot be marked on one line, it will be carried over to the next line thus limiting the number of available characters for customer specific information.

***** Standard PICmicro device marking consists of Microchip part number, year code, week code, and traceability code. For PICmicro device marking beyond this, certain price adders apply. Please check with your Microchip Sales Office. For QTP devices, any special marking adders are included in QTP price.

8-Lead Plastic Dual In-line (P) – 300 mil (PDIP)

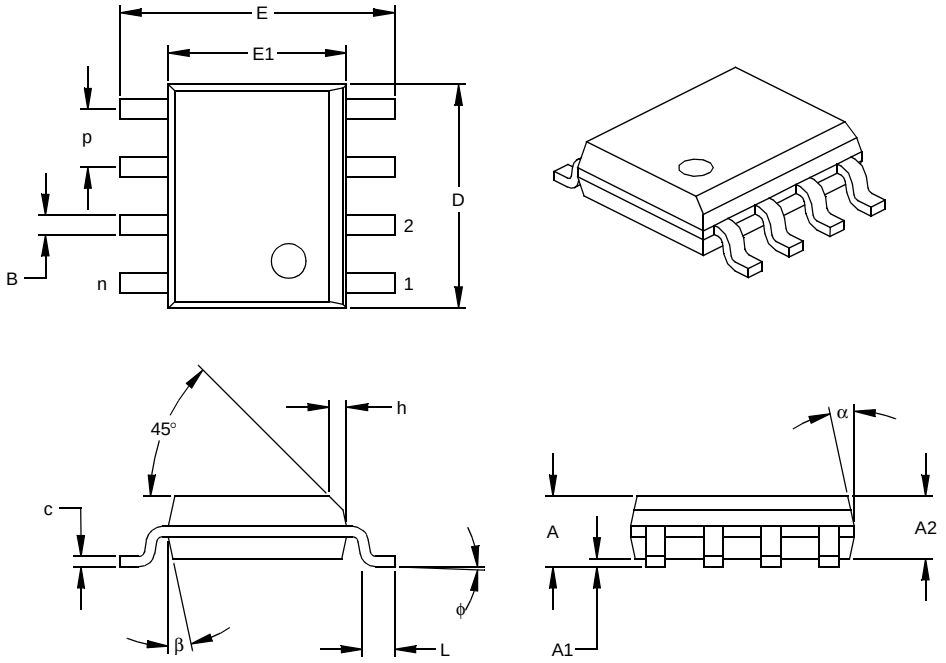


Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		8			8	
Pitch	p		.100			2.54	
Top to Seating Plane	A	.140	.155	.170	3.56	3.94	4.32
Molded Package Thickness	A2	.115	.130	.145	2.92	3.30	3.68
Base to Seating Plane	A1	.015			0.38		
Shoulder to Shoulder Width	E	.300	.313	.325	7.62	7.94	8.26
Molded Package Width	E1	.240	.250	.260	6.10	6.35	6.60
Overall Length	D	.360	.373	.385	9.14	9.46	9.78
Tip to Seating Plane	L	.125	.130	.135	3.18	3.30	3.43
Lead Thickness	c	.008	.012	.015	0.20	0.29	0.38
Upper Lead Width	B1	.045	.058	.070	1.14	1.46	1.78
Lower Lead Width	B	.014	.018	.022	0.36	0.46	0.56
Overall Row Spacing §	eB	.310	.370	.430	7.87	9.40	10.92
Mold Draft Angle Top	α	5	10	15	5	10	15
Mold Draft Angle Bottom	β	5	10	15	5	10	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-001
Drawing No. C04-018

8-Lead Plastic Small Outline (SN) – Narrow, 150 mil (SOIC)



Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		8			8	
Pitch	p		.050			1.27	
Overall Height	A	.053	.061	.069	1.35	1.55	1.75
Molded Package Thickness	A2	.052	.056	.061	1.32	1.42	1.55
Standoff §	A1	.004	.007	.010	0.10	0.18	0.25
Overall Width	E	.228	.237	.244	5.79	6.02	6.20
Molded Package Width	E1	.146	.154	.157	3.71	3.91	3.99
Overall Length	D	.189	.193	.197	4.80	4.90	5.00
Chamfer Distance	h	.010	.015	.020	0.25	0.38	0.51
Foot Length	L	.019	.025	.030	0.48	0.62	0.76
Foot Angle	φ	0	4	8	0	4	8
Lead Thickness	c	.008	.009	.010	0.20	0.23	0.25
Lead Width	B	.013	.017	.020	0.33	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-012
Drawing No. C04-057

NOTES:

APPENDIX A: CONVERSION CONSIDERATIONS

This appendix discusses the considerations for converting from previous version 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 B: 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

APPENDIX C: MIGRATION FROM MID-RANGE TO ENHANCED DEVICES

This section discusses how to migrate from a Mid-Range device (i.e., PIC16CXXX) to an Enhanced device (i.e., PIC18CXXX).

The following are the list of modifications over the PIC16CXXX microcontroller family:

Not Currently Available

APPENDIX D: MIGRATION FROM HIGH-END TO ENHANCED DEVICES

This section discusses how to migrate from a High-End device (i.e., PIC17CXXX) to an Enhanced MCU device (i.e., PIC18CXXX).

The following are the list of modifications over the PIC17CXXX microcontroller family:

Not Currently Available

INDEX

I

K

L

M

ON-LINE SUPPORT

Microchip provides on-line support on the Microchip World Wide Web (WWW) site.

The web site is used by Microchip as a means to make files and information easily available to customers. To view the site, the user must have access to the Internet and a web browser, such as Netscape or Microsoft Explorer. Files are also available for FTP download from our FTP site.

Connecting to the Microchip Internet Web Site

The Microchip web site is available by using your favorite Internet browser to attach to:

www.microchip.com

The file transfer site is available by using an FTP service to connect to:

ftp://ftp.microchip.com

The web site and file transfer site provide a variety of services. Users may download files for the latest Development Tools, Data Sheets, Application Notes, User's Guides, Articles and Sample Programs. A variety of Microchip specific business information is also available, including listings of Microchip sales offices, distributors and factory representatives. Other data available for consideration is:

- Latest Microchip Press Releases
- Technical Support Section with Frequently Asked Questions
- Design Tips
- Device Errata
- Job Postings
- Microchip Consultant Program Member Listing
- Links to other useful web sites related to Microchip Products
- Conferences for products, Development Systems, technical information and more
- Listing of seminars and events

Systems Information and Upgrade Hot Line

The Systems Information and Upgrade Line provides system users a listing of the latest versions of all of Microchip's development systems software products. Plus, this line provides information on how customers can receive any currently available upgrade kits.The Hot Line Numbers are:

1-800-755-2345 for U.S. and most of Canada, and

1-480-792-7302 for the rest of the world.

001024

READER RESPONSE

It is our intention to provide you with the best documentation possible to ensure successful use of your Microchip product. If you wish to provide your comments on organization, clarity, subject matter, and ways in which our documentation can better serve you, please FAX your comments to the Technical Publications Manager at (480) 792-4150.

Please list the following information, and use this outline to provide us with your comments about this Data Sheet.

To: Technical Publications Manager Total Pages Sent

RE: Reader Response

From: Name _____

Company _____

Address _____

City / State / ZIP / Country _____

Telephone: (_____) _____ - _____ FAX: (_____) _____ - _____

Application (optional):

Would you like a reply? ___Y ___N

Device: **PIC18F010/020** Literature Number: **DS41142A**

Questions:

1. What are the best features of this document?

2. How does this document meet your hardware and software development needs?

3. Do you find the organization of this data sheet easy to follow? If not, why?

4. What additions to the data sheet do you think would enhance the structure and subject?

5. What deletions from the data sheet could be made without affecting the overall usefulness?

6. Is there any incorrect or misleading information (what and where)?

7. How would you improve this document?

8. How would you improve our software, systems, and silicon products?

PIC18F010/020 PRODUCT IDENTIFICATION SYSTEM

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

PART NO.	–	X	/XX	XXX
Device		Temperature Range	Package	Pattern

Device	PIC18F0X0[1], PIC18F0X0T[2]; V_{DD} range 4.5V to 5.5V PIC18LF0X0[1], PIC18LF0X0T[2]; V_{DD} range 2.0V to 5.5V
Temperature Range	I = -40°C to +85°C (Industrial)
Package	SO = SOIC P = PDIP
Pattern	QTP, SQTP, Code or Special Requirements (blank otherwise)

Examples:

a) PIC18LF010 - I/P 301 = Industrial temp., PDIP package, 40 MHz, Extended V_{DD} limits, QTP pattern #301.

b) PIC18LF020 - I/SO = Industrial temp., SOIC package, Extended V_{DD} limits.

c) PIC18F020 - I/P = Industrial temp., PDIP package, 40MHz, normal V_{DD} limits.

Note 1: F = Standard Voltage range
LF = Wide Voltage Range
2: T = in tape and reel - SOIC

Sales and Support

Data Sheets
Products supported by a preliminary Data Sheet may have an errata sheet describing minor operational differences and recommended workarounds. To determine if an errata sheet exists for a particular device, please contact one of the following:

1. Your local Microchip sales office
2. The Microchip Corporate Literature Center U.S. FAX: (480) 792-7277
3. The Microchip Worldwide Site (www.microchip.com)

Please specify which device, revision of silicon and Data Sheet (include Literature #) you are using.

New Customer Notification System
Register on our web site (www.microchip.com/cn) to receive the most current information on our products.

NOTES:

NOTES:

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-7456

Atlanta
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Austin
Analog Product Sales
8303 MoPac Expressway North
Suite A-201
Austin, TX 78759
Tel: 512-345-2030 Fax: 512-345-6085

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Boston
Analog Product Sales
Unit A-8-1 Millbrook Tarry Condominium
97 Lowell Road
Concord, MA 01742
Tel: 978-371-6400 Fax: 978-371-0050

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Dayton
Two Prestige Place, Suite 130
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

Mountain View
Analog Product Sales
1300 Terra Bella Avenue
Mountain View, CA 94043-1836
Tel: 650-968-9241 Fax: 650-967-1590

New York
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing
Microchip Technology Beijing Office
Unit 915
New China Hong Kong Manhattan Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Shanghai
Microchip Technology Shanghai Office
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

Hong Kong
Microchip Asia Pacific
RM 2101, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

ASIA/PACIFIC (continued)

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan
Microchip Technology Taiwan
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark
Microchip Technology Denmark ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Arizona Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann Ring 125
D-81739 Munich, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Germany
Analog Product Sales
Lochhamer Strasse 13
D-82152 Martinsried, Germany
Tel: 49-89-895650-0 Fax: 49-89-895650-22

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

01/30/01