

## 18-pin *Enhanced* FLASH/EEPROM 8-Bit Microcontroller

### High Performance RISC CPU Features:

- Only 35 single word instructions to learn
- All instructions single-cycle except for program branches which are two-cycle
- Operating speed: DC - 20 MHz clock input  
DC - 200 ns instruction cycle
- 1024 words of program memory
- 68 bytes of Data RAM
- 64 bytes of Data EEPROM
- 14-bit wide instruction words
- 8-bit wide data bytes
- 15 Special Function Hardware registers
- Eight-level deep hardware stack
- Direct, indirect and relative addressing modes
- Four interrupt sources:
  - External RB0/INT pin
  - TMR0 timer overflow
  - PORTB<7:4> interrupt-on-change
  - Data EEPROM write complete

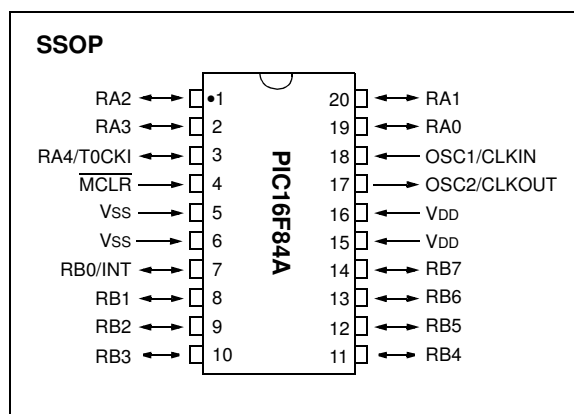
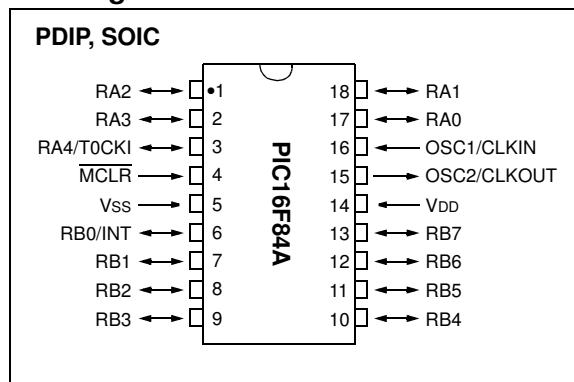
### Peripheral Features:

- 13 I/O pins with individual direction control
- High current sink/source for direct LED drive
  - 25 mA sink max. per pin
  - 25 mA source max. per pin
- TMR0: 8-bit timer/counter with 8-bit programmable prescaler

### Special Microcontroller Features:

- 10,000 erase/write cycles *Enhanced* FLASH Program memory typical
- 10,000,000 typical erase/write cycles EEPROM Data memory typical
- EEPROM Data Retention > 40 years
- In-Circuit Serial Programming™ (ICSP™) - via two pins
- Power-on Reset (POR), Power-up Timer (PWRT), Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own On-Chip RC Oscillator for reliable operation
- Code protection
- Power saving SLEEP mode
- Selectable oscillator options

### Pin Diagrams



### CMOS Enhanced FLASH/EEPROM Technology:

- Low power, high speed technology
- Fully static design
- Wide operating voltage range:
  - Commercial: 2.0V to 5.5V
  - Industrial: 2.0V to 5.5V
- Low power consumption:
  - < 2 mA typical @ 5V, 4 MHz
  - 15 µA typical @ 2V, 32 kHz
  - < 0.5 µA typical standby current @ 2V

## 1.0 DEVICE OVERVIEW

This document contains device specific information for the operation of the PIC16F84A device. Additional information may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023), which may be downloaded from the Microchip website. The Reference Manual should be considered a complementary document to this data sheet, and is highly recommended reading for a better understanding of the device architecture and operation of the peripheral modules.

The PIC16F84A belongs to the mid-range family of the PICmicro® microcontroller devices. A block diagram of the device is shown in Figure 1-1.

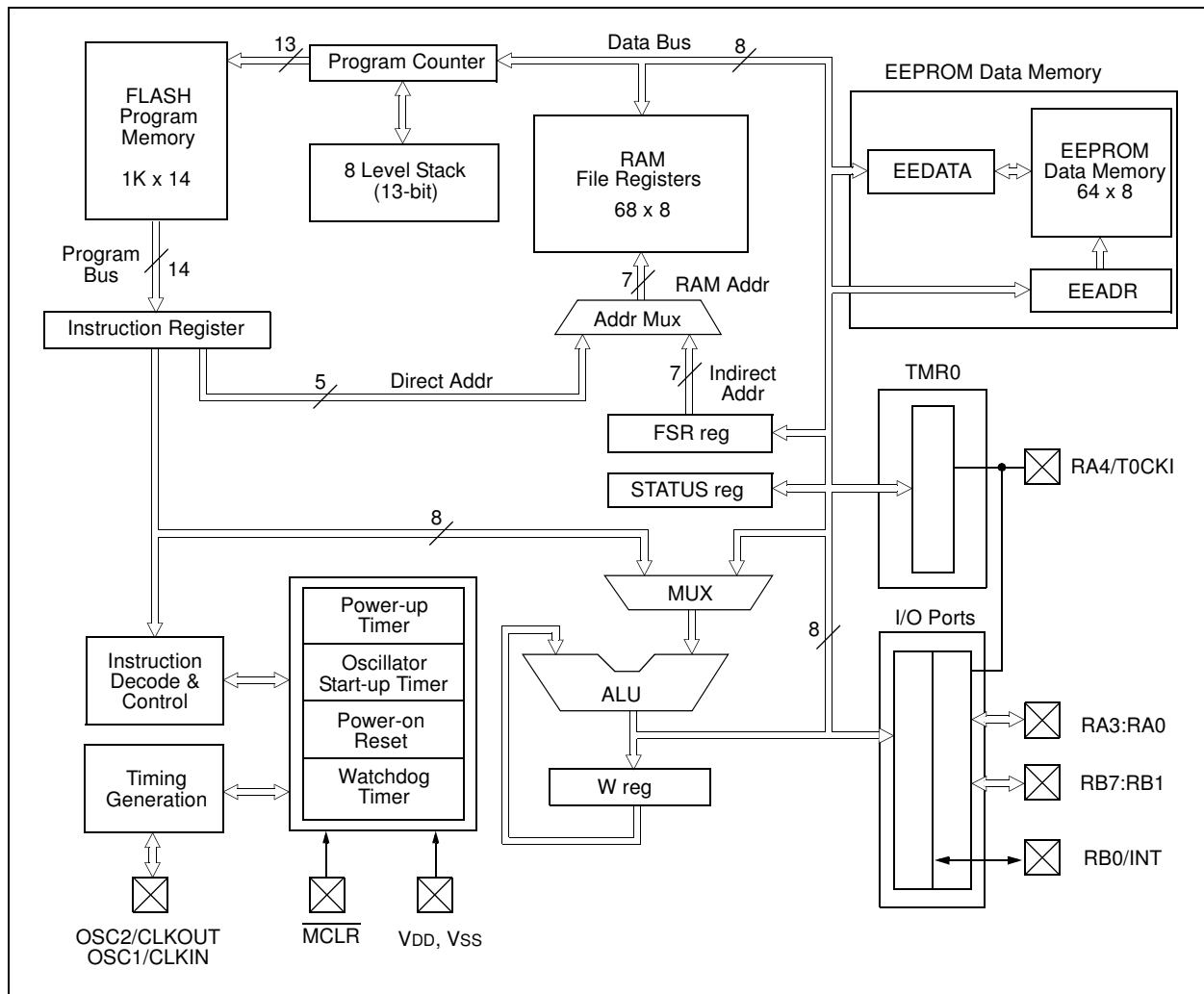
The program memory contains 1K words, which translates to 1024 instructions, since each 14-bit program memory word is the same width as each device instruction. The data memory (RAM) contains 68 bytes. Data EEPROM is 64 bytes.

There are also 13 I/O pins that are user-configured on a pin-to-pin basis. Some pins are multiplexed with other device functions. These functions include:

- External interrupt
- Change on PORTB interrupt
- Timer0 clock input

Table 1-1 details the pinout of the device with descriptions and details for each pin.

**FIGURE 1-1: PIC16F84A BLOCK DIAGRAM**



# PIC16F84A

**TABLE 1-1: PIC16F84A PINOUT DESCRIPTION**

Pin Name	PDIP No.	SOIC No.	SSOP No.	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	16	16	18	I	ST/CMOS <sup>(3)</sup>	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	15	15	19	O	—	Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. In RC mode, OSC2 pin outputs CLKOUT, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
MCLR	4	4	4	I/P	ST	Master Clear (Reset) input/programming voltage input. This pin is an active low RESET to the device.
RA0	17	17	19	I/O	TTL	<p>PORTA is a bi-directional I/O port.</p> <p>Can also be selected to be the clock input to the TMR0 timer/counter. Output is open drain type.</p>
RA1	18	18	20	I/O	TTL	
RA2	1	1	1	I/O	TTL	
RA3	2	2	2	I/O	TTL	
RA4/T0CKI	3	3	3	I/O	ST	
RB0/INT	6	6	7	I/O	TTL/ST <sup>(1)</sup>	<p>PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.</p> <p>RB0/INT can also be selected as an external interrupt pin.</p> <p>Interrupt-on-change pin.</p> <p>Interrupt-on-change pin.</p> <p>Interrupt-on-change pin.</p> <p>Serial programming clock.</p> <p>Interrupt-on-change pin.</p> <p>Serial programming data.</p>
RB1	7	7	8	I/O	TTL	
RB2	8	8	9	I/O	TTL	
RB3	9	9	10	I/O	TTL	
RB4	10	10	11	I/O	TTL	
RB5	11	11	12	I/O	TTL	
RB6	12	12	13	I/O	TTL/ST <sup>(2)</sup>	
RB7	13	13	14	I/O	TTL/ST <sup>(2)</sup>	
VSS	5	5	5,6	P	—	Ground reference for logic and I/O pins.
VDD	14	14	15,16	P	—	Positive supply for logic and I/O pins.

Legend: I = input    O = Output    I/O = Input/Output    P = Power  
 — = Not used    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.  
**3:** This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

## 2.0 MEMORY ORGANIZATION

There are two memory blocks in the PIC16F84A. These are the program memory and the data memory. Each block has its own bus, so that access to each block can occur during the same oscillator cycle.

The data memory can further be broken down into the general purpose RAM and the Special Function Registers (SFRs). The operation of the SFRs that control the “core” are described here. The SFRs used to control the peripheral modules are described in the section discussing each individual peripheral module.

The data memory area also contains the data EEPROM memory. This memory is not directly mapped into the data memory, but is indirectly mapped. That is, an indirect address pointer specifies the address of the data EEPROM memory to read/write. The 64 bytes of data EEPROM memory have the address range 0h-3Fh. More details on the EEPROM memory can be found in Section 3.0.

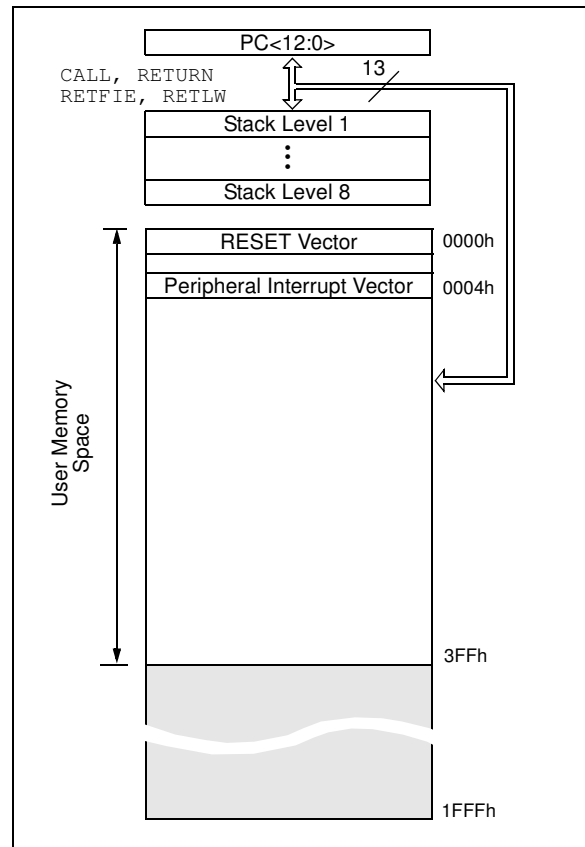
Additional information on device memory may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023).

### 2.1 Program Memory Organization

The PIC16FXX has a 13-bit program counter capable of addressing an 8K x 14 program memory space. For the PIC16F84A, the first 1K x 14 (0000h-03FFh) are physically implemented (Figure 2-1). Accessing a location above the physically implemented address will cause a wraparound. For example, for locations 20h, 420h, 820h, C20h, 1020h, 1420h, 1820h, and 1C20h, the instruction will be the same.

The RESET vector is at 0000h and the interrupt vector is at 0004h.

**FIGURE 2-1: PROGRAM MEMORY MAP AND STACK - PIC16F84A**



# PIC16F84A

## 2.2 Data Memory Organization

The data memory is partitioned into two areas. The first is the Special Function Registers (SFR) area, while the second is the General Purpose Registers (GPR) area. The SFRs control the operation of the device.

Portions of data memory are banked. This is for both the SFR area and the GPR area. The GPR area is banked to allow greater than 116 bytes of general purpose RAM. The banked areas of the SFR are for the registers that control the peripheral functions. Banking requires the use of control bits for bank selection. These control bits are located in the STATUS Register. Figure 2-2 shows the data memory map organization.

Instructions `MOVWF` and `MOVF` can move values from the W register to any location in the register file ("F"), and vice-versa.

The entire data memory can be accessed either directly using the absolute address of each register file or indirectly through the File Select Register (FSR) (Section 2.5). Indirect addressing uses the present value of the RP0 bit for access into the banked areas of data memory.

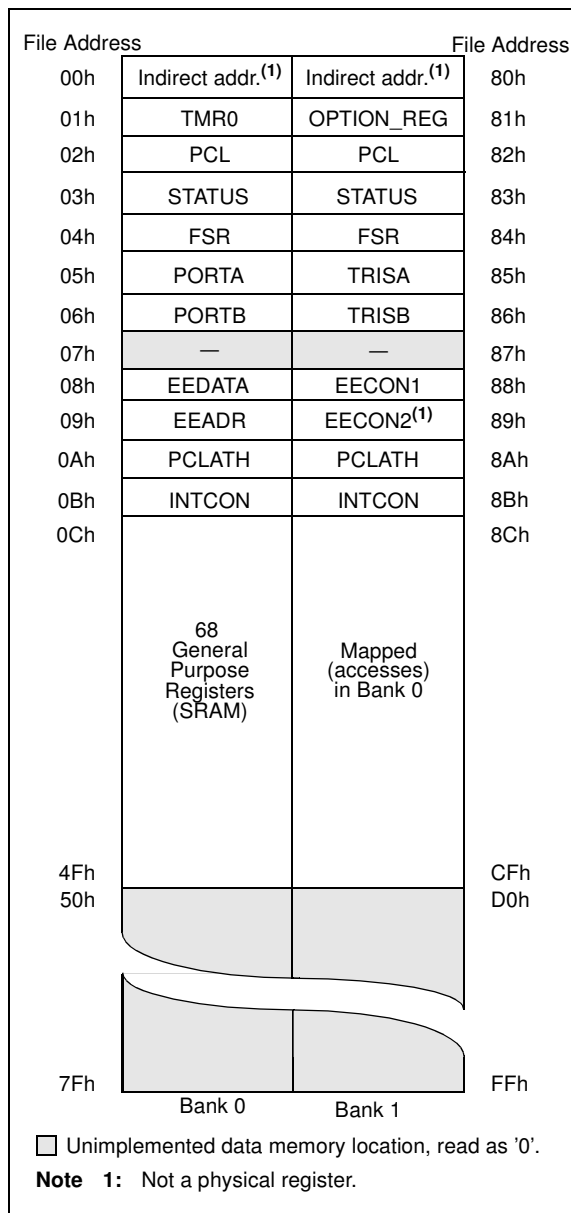
Data memory is partitioned into two banks which contain the general purpose registers and the special function registers. Bank 0 is selected by clearing the RP0 bit (`STATUS<5>`). Setting the RP0 bit selects Bank 1. Each Bank extends up to 7Fh (128 bytes). The first twelve locations of each Bank are reserved for the Special Function Registers. The remainder are General Purpose Registers, implemented as static RAM.

### 2.2.1 GENERAL PURPOSE REGISTER FILE

Each General Purpose Register (GPR) is 8-bits wide and is accessed either directly or indirectly through the FSR (Section 2.5).

The GPR addresses in Bank 1 are mapped to addresses in Bank 0. As an example, addressing location 0Ch or 8Ch will access the same GPR.

**FIGURE 2-2: REGISTER FILE MAP - PIC16F84A**



## 2.3 Special Function Registers

The Special Function Registers (Figure 2-2 and Table 2-1) are used by the CPU and Peripheral functions to control the device operation. These registers are static RAM.

The special function registers can be classified into two sets, core and peripheral. Those associated with the core functions are described in this section. Those related to the operation of the peripheral features are described in the section for that specific feature.

**TABLE 2-1: SPECIAL FUNCTION REGISTER FILE SUMMARY**

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on RESET	Details on page
Bank 0											
00h	INDF	Uses contents of FSR to address Data Memory (not a physical register)								---- --	11
01h	TMR0	8-bit Real-Time Clock/Counter								xxxx xxxx	20
02h	PCL	Low Order 8 bits of the Program Counter (PC)								0000 0000	11
03h	STATUS <sup>(2)</sup>	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	8
04h	FSR	Indirect Data Memory Address Pointer 0								xxxx xxxx	11
05h	PORTA <sup>(4)</sup>	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	16
06h	PORTB <sup>(5)</sup>	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	18
07h	—	Unimplemented location, read as '0'								—	—
08h	EEDATA	EEPROM Data Register								xxxx xxxx	13,14
09h	EEADR	EEPROM Address Register								xxxx xxxx	13,14
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of the PC <sup>(1)</sup>				---0 0000	11	
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10
Bank 1											
80h	INDF	Uses Contents of FSR to address Data Memory (not a physical register)								---- --	11
81h	OPTION_REG	RBPV	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	9
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	11
83h	STATUS <sup>(2)</sup>	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	8
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	11
85h	TRISA	—	—	—	PORTA Data Direction Register				---1 1111	16	
86h	TRISB	PORTB Data Direction Register								1111 1111	18
87h	—	Unimplemented location, read as '0'								—	—
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	13
89h	EECON2	EEPROM Control Register 2 (not a physical register)								---- --	14
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC <sup>(1)</sup>				---0 0000	11	
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10

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

**Note 1:** The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> are never transferred to PCLATH.

**2:** The  $\overline{TO}$  and  $\overline{PD}$  status bits in the STATUS register are not affected by a MCLR Reset.

**3:** Other (non power-up) RESETS include: external RESET through MCLR and the Watchdog Timer Reset.

**4:** On any device RESET, these pins are configured as inputs.

**5:** This is the value that will be in the port output latch.

# PIC16F84A

## 2.3.1 STATUS REGISTER

The STATUS register contains the arithmetic status of the ALU, the RESET status and the bank select bit for data memory.

As with any register, the STATUS register can be the destination for any instruction. If the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to device logic. Furthermore, the  $\overline{TO}$  and PD bits are not writable. 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 u1uu (where u = unchanged).

Only the `BCF`, `BSF`, `SWAPF` and `MOVWF` instructions should be used to alter the STATUS register (Table 7-2), because these instructions do not affect any status bit.

**Note 1:** The IRP and RP1 bits (STATUS<7:6>) are not used by the PIC16F84A and should be programmed as cleared. Use of these bits as general purpose R/W bits is NOT recommended, since this may affect upward compatibility with future products.

**2:** The C and DC bits operate as a borrow and digit borrow out bit, respectively, in subtraction. See the `SUBLW` and `SUBWF` instructions for examples.

**3:** When the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. The specified bit(s) will be updated according to device logic

### REGISTER 2-1: STATUS REGISTER (ADDRESS 03h, 83h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	PD	Z	DC	C
bit 7							bit 0

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

bit 5 **RP0:** Register Bank Select bits (used for direct addressing)

01 = Bank 1 (80h - FFh)

00 = Bank 0 (00h - 7Fh)

bit 4  **$\overline{TO}$ :** Time-out bit

1 = After power-up, `CLRWDT` instruction, or `SLEEP` instruction

0 = A WDT time-out occurred

bit 3 **PD:** Power-down bit

1 = After power-up or by the `CLRWDT` instruction

0 = By execution of the `SLEEP` instruction

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/borrow bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions) (for borrow, the polarity is reversed)

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

bit 0 **C:** Carry/borrow bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions) (for borrow, the polarity is reversed)

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:** 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

## 2.3.2 OPTION REGISTER

The OPTION register is a readable and writable register which contains various control bits to configure the TMR0/WDT prescaler, the external INT interrupt, TMR0, and the weak pull-ups on PORTB.

**Note:** When the prescaler is assigned to the WDT (PSA = '1'), TMR0 has a 1:1 prescaler assignment.

### REGISTER 2-2: OPTION REGISTER (ADDRESS 81h)

	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
	RBP	INTEDG	T0CS	T0SE	PSA	PS2	PS0
bit 7							bit 0

bit 7      **RBP**: PORTB Pull-up Enable bit  
1 = PORTB pull-ups are disabled  
0 = PORTB pull-ups are enabled by individual port latch values

bit 6      **INTEDG**: Interrupt Edge Select bit  
1 = Interrupt on rising edge of RB0/INT pin  
0 = Interrupt on falling edge of RB0/INT pin

bit 5      **T0CS**: TMR0 Clock Source Select bit  
1 = Transition on RA4/T0CKI pin  
0 = Internal instruction cycle clock (CLKOUT)

bit 4      **T0SE**: TMR0 Source Edge Select bit  
1 = Increment on high-to-low transition on RA4/T0CKI pin  
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3      **PSA**: Prescaler Assignment bit  
1 = Prescaler is assigned to the WDT  
0 = Prescaler is assigned to the Timer0 module

bit 2-0      **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

#### 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



# PIC16F84A

## 2.3.3 INTCON REGISTER

The INTCON register is a readable and writable register that contains the various enable bits for all interrupt sources.

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

### REGISTER 2-3: INTCON REGISTER (ADDRESS 0Bh, 8Bh)

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	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
bit 7							bit 0

- bit 7 **GIE:** Global Interrupt Enable bit  
1 = Enables all unmasked interrupts  
0 = Disables all interrupts
- bit 6 **EEIE:** EE Write Complete Interrupt Enable bit  
1 = Enables the EE Write Complete interrupts  
0 = Disables the EE Write Complete interrupt
- bit 5 **TOIE:** TMR0 Overflow Interrupt Enable bit  
1 = Enables the TMR0 interrupt  
0 = Disables the TMR0 interrupt
- bit 4 **INTE:** RB0/INT External Interrupt Enable bit  
1 = Enables the RB0/INT external interrupt  
0 = Disables the RB0/INT 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 **TOIF:** TMR0 Overflow Interrupt Flag bit  
1 = TMR0 register has overflowed (must be cleared in software)  
0 = TMR0 register did not overflow
- bit 1 **INTF:** RB0/INT External Interrupt Flag bit  
1 = The RB0/INT external interrupt occurred (must be cleared in software)  
0 = The RB0/INT external interrupt did not occur
- bit 0 **RBIF:** RB Port Change Interrupt Flag bit  
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)  
0 = None of the RB7:RB4 pins have changed state

#### 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

## 2.4 PCL and PCLATH

The program counter (PC) specifies the address of the instruction to fetch for execution. The PC is 13 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<12:8> bits and is not directly readable or writable. If the program counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP. All updates to the PCH register go through the PCLATH register.

### 2.4.1 STACK

The stack allows a combination of up to 8 program calls and interrupts to occur. The stack contains the return address from this branch in program execution.

Mid-range devices have an 8 level deep x 13-bit wide hardware stack. The stack space is not part of either program or data space and the stack pointer is not readable or writable. The PC is PUSHed onto the stack when a CALL instruction is executed or an interrupt causes a branch. The stack is POPed in the event of a RETURN, RETLW or a RETFIE instruction execution. PCLATH is not modified when the stack is PUSHed or POPed.

After the stack has been PUSHed eight times, the ninth push overwrites the value that was stored from the first push. The tenth push overwrites the second push (and so on).

## 2.5 Indirect Addressing; INDF and FSR Registers

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

### EXAMPLE 2-1: INDIRECT ADDRESSING

- Register file 05 contains the value 10h
- Register file 06 contains the value 0Ah
- Load the value 05 into the FSR register
- A read of the INDF register will return the value of 10h
- Increment the value of the FSR register by one (FSR = 06)
- A read of the INDF register now will return the value of 0Ah.

Reading INDF itself indirectly (FSR = 0) will produce 00h. Writing to the INDF register indirectly results in a no-operation (although STATUS bits may be affected).

A simple program to clear RAM locations 20h-2Fh using indirect addressing is shown in Example 2-2.

### EXAMPLE 2-2: HOW TO CLEAR RAM USING INDIRECT ADDRESSING

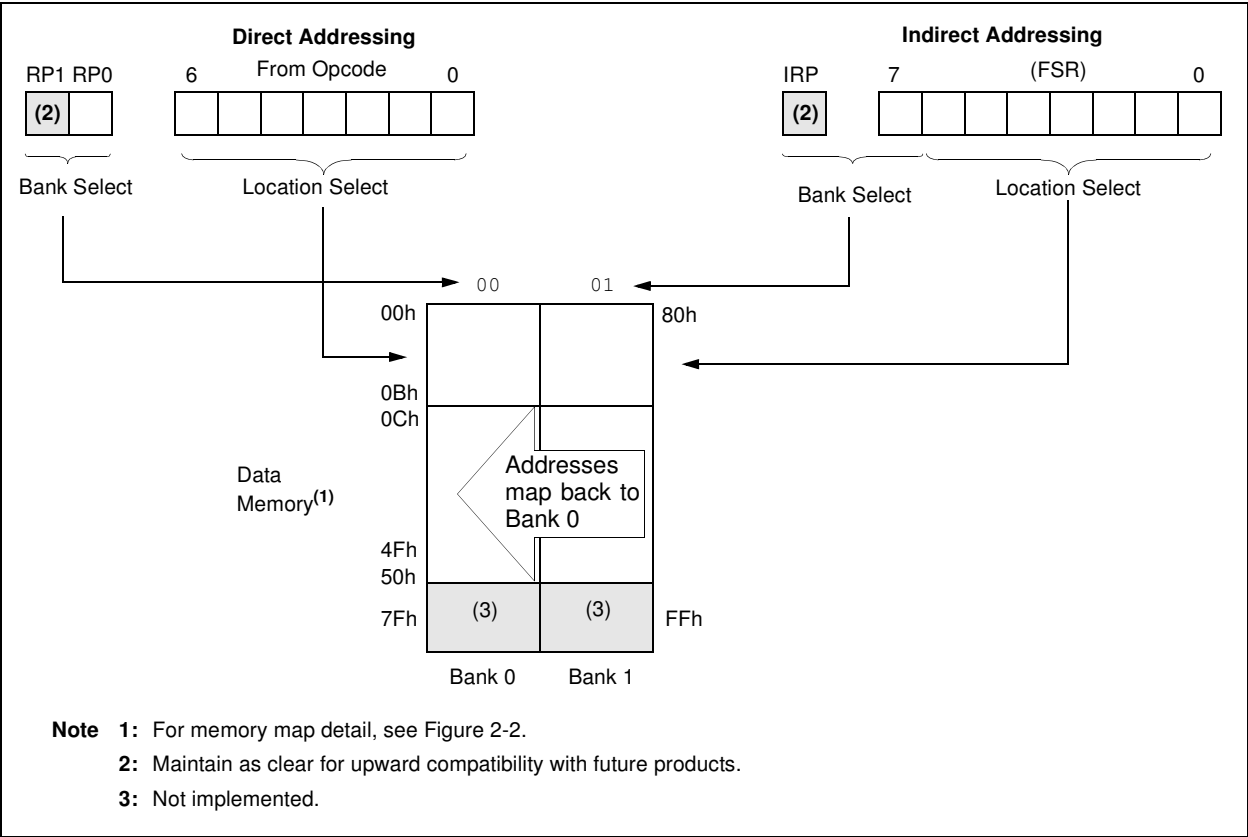
```

        movlw 0x20    ;initialize pointer
        movwf FSR     ;to RAM
NEXT    clrf  INDF     ;clear INDF register
        incf  FSR      ;inc pointer
        btfss FSR,4    ;all done?
        goto  NEXT     ;NO, clear next
CONTINUE
        :              ;YES, continue
    
```

An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (STATUS<7>), as shown in Figure 2-3. However, IRP is not used in the PIC16F84A.

# PIC16F84A

FIGURE 2-3: DIRECT/INDIRECT ADDRESSING



## 3.0 DATA EEPROM MEMORY

The EEPROM data memory 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
- EECON2 (not a physically implemented register)
- EEDATA
- EEADR

EEDATA holds the 8-bit data for read/write, and EEADR holds the address of the EEPROM location being accessed. PIC16F84A devices have 64 bytes of data EEPROM with an address range from 0h to 3Fh.

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 AC specifications for exact limits.

When the device is code protected, the CPU may continue to read and write the data EEPROM memory. The device programmer can no longer access this memory.

Additional information on the Data EEPROM is available in the PICmicro™ Mid-Range Reference Manual (DS33023).

### REGISTER 3-1: EECON1 REGISTER (ADDRESS 88h)

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

- bit 7-5 **Unimplemented:** Read as '0'
- bit 4 **EEIF:** EEPROM Write Operation Interrupt Flag bit  
 1 = The write operation completed (must be cleared in software)  
 0 = The write operation is not complete or has not been started
- bit 3 **WRERR:** EEPROM Error Flag bit  
 1 = A write operation is prematurely terminated (any 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 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

# PIC16F84A

## 3.1 Reading the EEPROM Data Memory

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

### EXAMPLE 3-1: DATA EEPROM READ

```
BCF    STATUS, RP0    ; Bank 0
MOVLW  CONFIG_ADDR    ;
MOVWF  EEADR           ; Address to read
BSF    STATUS, RP0    ; Bank 1
BSF    EECON1, RD      ; EE Read
BCF    STATUS, RP0    ; Bank 0
MOVF   EEDATA, W       ; W = EEDATA
```

## 3.2 Writing to the EEPROM Data Memory

To write an EEPROM data location, the user must first write the address to the EEADR register and the data to the EEDATA register. Then the user must follow a specific sequence to initiate the write for each byte.

### EXAMPLE 3-2: DATA EEPROM WRITE

```
BSF    STATUS, RP0    ; Bank 1
BCF    INTCON, GIE     ; Disable INTs.
BSF    EECON1, WREN    ; Enable Write
MOVLW  55h             ;
;
; Required Sequence
MOVWF  EECON2           ; Write 55h
MOVLW  AAh             ;
MOVWF  EECON2           ; Write AAh
BSF    EECON1, WR       ; Set WR bit
; begin write
BSF    INTCON, GIE     ; Enable INTs.
```

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

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

After a write sequence has been initiated, clearing the WREN bit will not affect this write cycle. The WR bit will be inhibited from being set unless the WREN bit is set.

At the completion of the write cycle, the WR bit is cleared in hardware and the EE Write Complete Interrupt Flag bit (EEIF) is set. The user can either enable this interrupt or poll this bit. EEIF must be cleared by software.

## 3.3 Write Verify

Depending on the application, good programming practice may dictate that the value written to the Data EEPROM should be verified (Example 3-3) to the desired value to be written. This should be used in applications where an EEPROM bit will be stressed near the specification limit.

Generally, the EEPROM write failure will be a bit which was written as a '0', but reads back as a '1' (due to leakage off the bit).

### EXAMPLE 3-3: WRITE VERIFY

```
BCF    STATUS, RP0    ; Bank 0
:      ; Any code
:      ; can go here
MOVF   EEDATA, W       ; Must be in Bank 0
BSF    STATUS, RP0    ; Bank 1
READ
BSF    EECON1, RD      ; YES, Read the
; value written
BCF    STATUS, RP0    ; Bank 0
;
; Is the value written
; (in W reg) and
; read (in EEDATA)
; the same?
;
SUBWF  EEDATA, W       ;
BTFS   STATUS, Z       ; Is difference 0?
GOTO   WRITE_ERR      ; NO, Write error
```

TABLE 3-1: REGISTERS/BITS ASSOCIATED WITH DATA EEPROM

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other RESETS
08h	EEDATA	EEPROM Data Register								xxxx xxxx	uuuu uuuu
09h	EEADR	EEPROM Address Register								xxxx xxxx	uuuu uuuu
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2	EEPROM Control Register 2								---- ----	---- ----

Legend: x = unknown, u = unchanged, — = unimplemented, read as '0', q = value depends upon condition.

Shaded cells are not used by data EEPROM.

## 4.0 I/O PORTS

Some pins for these I/O ports are multiplexed with an alternate function for the peripheral features on the device. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.

Additional information on I/O ports may be found in the PICmicro™ Mid-Range Reference Manual (DS33023).

### 4.1 PORTA and TRISA Registers

PORTA is a 5-bit wide, bi-directional port. The corresponding data direction register is TRISA. Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input (i.e., put the corresponding output driver in a Hi-Impedance mode). Clearing a TRISA bit (= 0) will make the corresponding PORTA pin an output (i.e., put the contents of the output latch on the selected pin).

**Note:** On a Power-on Reset, these pins are configured as inputs and read as '0'.

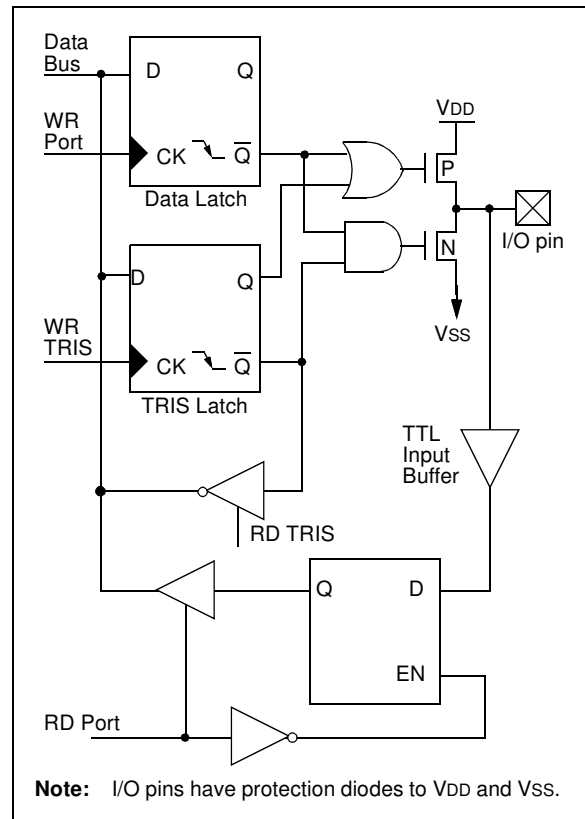
Reading the PORTA register reads the status of the pins, whereas writing to it will write to the port latch. All write operations are read-modify-write operations. Therefore, a write to a port implies that the port pins are read. This value is modified and then written to the port data latch.

Pin RA4 is multiplexed with the Timer0 module clock input to become the RA4/T0CKI pin. The RA4/T0CKI pin is a Schmitt Trigger input and an open drain output. All other RA port pins have TTL input levels and full CMOS output drivers.

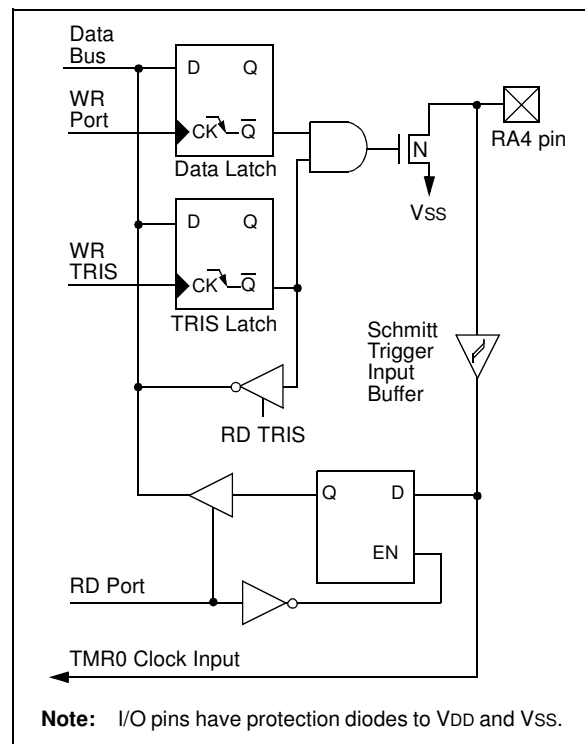
#### EXAMPLE 4-1: INITIALIZING PORTA

```
BCF    STATUS, RP0 ;
CLRF   PORTA       ; Initialize PORTA by
                   ; clearing output
                   ; data latches
BSF    STATUS, RP0 ; Select Bank 1
MOVLW  0x0F        ; Value used to
                   ; initialize data
                   ; direction
MOVWF  TRISA        ; Set RA<3:0> as inputs
                   ; RA4 as output
                   ; TRISA<7:5> are always
                   ; read as '0'.
```

**FIGURE 4-1: BLOCK DIAGRAM OF PINS RA3:RA0**



**FIGURE 4-2: BLOCK DIAGRAM OF PIN RA4**



# PIC16F84A

**TABLE 4-1: PORTA FUNCTIONS**

Name	Bit0	Buffer Type	Function
RA0	bit0	TTL	Input/output
RA1	bit1	TTL	Input/output
RA2	bit2	TTL	Input/output
RA3	bit3	TTL	Input/output
RA4/T0CKI	bit4	ST	Input/output or external clock input for TMR0. Output is open drain type.

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

**TABLE 4-2: SUMMARY OF REGISTERS ASSOCIATED WITH PORTA**

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other RESETS
05h	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
85h	TRISA	—	—	—	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	---1 1111	---1 1111

Legend: x = unknown, u = unchanged, — = unimplemented, read as '0'. Shaded cells are unimplemented, read as '0'.

## 4.2 PORTB and TRISB Registers

PORTB is an 8-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).

### EXAMPLE 4-2: INITIALIZING PORTB

```
BCF    STATUS, RP0 ;
CLRF   PORTB       ; Initialize PORTB by
                   ; clearing output
                   ; data latches
BSF    STATUS, RP0 ; Select Bank 1
MOVLW  0xCF        ; Value used to
                   ; initialize data
                   ; direction
MOVWF  TRISB        ; Set RB<3:0> as inputs
                   ; RB<5:4> as outputs
                   ; RB<7:6> as inputs
```

Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups. This is performed by clearing bit  $\overline{\text{RBP}}\text{U}$  (OPTION<7>). The 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.

Four of PORTB's pins, RB7:RB4, have an interrupt-on-change feature. Only pins configured as inputs can cause this interrupt to occur (i.e., any RB7:RB4 pin configured as an output is excluded from the interrupt-on-change comparison). The input pins (of RB7:RB4) are compared with the old value latched on the last read of PORTB. The "mismatch" outputs of RB7:RB4 are OR'ed together to generate the RB Port Change Interrupt with flag bit RBIF (INTCON<0>).

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

- Any read or write of PORTB. This will end the mismatch condition.
- Clear flag bit RBIF.

A mismatch condition will continue to set flag bit RBIF. Reading PORTB will end the mismatch condition and allow flag bit RBIF to be cleared.

The interrupt-on-change feature is recommended for wake-up on key depression operation and operations where PORTB is only used for the interrupt-on-change feature. Polling of PORTB is not recommended while using the interrupt-on-change feature.

FIGURE 4-3: BLOCK DIAGRAM OF PINS RB7:RB4

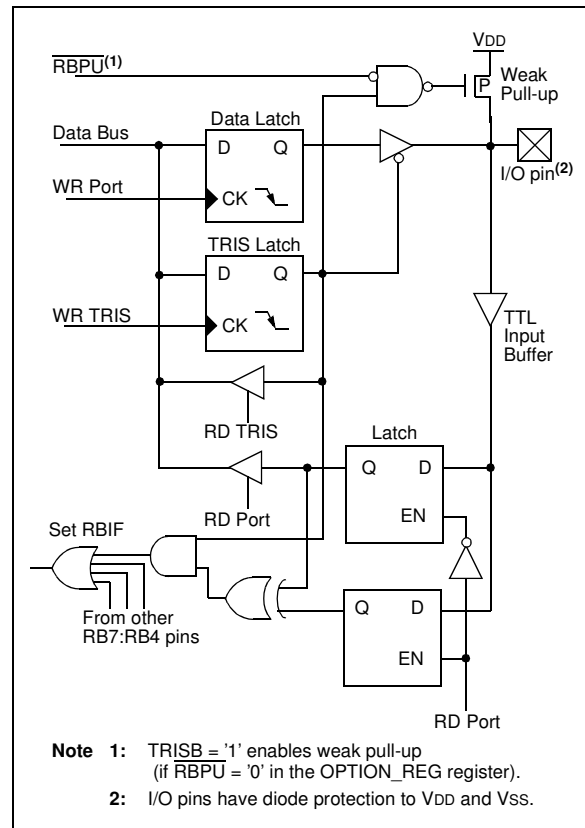
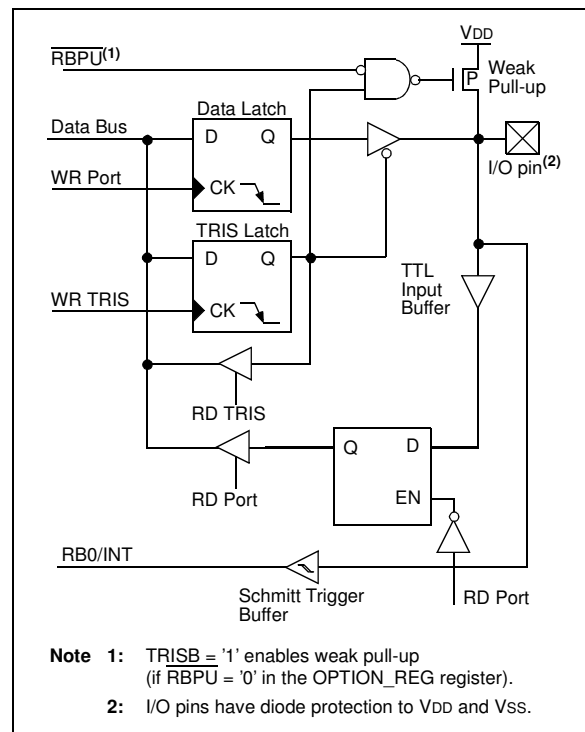


FIGURE 4-4: BLOCK DIAGRAM OF PINS RB3:RB0





# PIC16F84A

**TABLE 4-3: PORTB FUNCTIONS**

Name	Bit	Buffer Type	I/O Consistency Function
RB0/INT	bit0	TTL/ST <sup>(1)</sup>	Input/output pin or external interrupt input. Internal software programmable weak pull-up.
RB1	bit1	TTL	Input/output pin. Internal software programmable weak pull-up.
RB2	bit2	TTL	Input/output pin. Internal software programmable weak pull-up.
RB3	bit3	TTL	Input/output pin. Internal software programmable weak pull-up.
RB4	bit4	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB5	bit5	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB6	bit6	TTL/ST <sup>(2)</sup>	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up. Serial programming clock.
RB7	bit7	TTL/ST <sup>(2)</sup>	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up. Serial programming data.

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

**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 4-4: SUMMARY OF REGISTERS ASSOCIATED WITH PORTB**

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other RESETS
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	1111 1111
81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
0Bh,8Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBF	0000 000x	0000 000u

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

## 5.0 TIMER0 MODULE

The Timer0 module timer/counter has the following features:

- 8-bit timer/counter
- Readable and writable
- Internal or external clock select
- Edge select for external clock
- 8-bit software programmable prescaler
- Interrupt-on-overflow from FFh to 00h

Figure 5-1 is a simplified block diagram of the Timer0 module.

Additional information on timer modules is available in the PICmicro™ Mid-Range Reference Manual (DS33023).

### 5.1 Timer0 Operation

Timer0 can operate as a timer or as a counter.

Timer mode is selected by clearing bit T0CS (OPTION\_REG<5>). 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 bit T0CS (OPTION\_REG<5>). In Counter mode, Timer0 will increment, either on every rising or falling edge of pin RA4/T0CKI. The incrementing edge is determined by the Timer0 Source Edge Select bit, T0SE (OPTION\_REG<4>). Clearing bit T0SE 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.

Additional information on external clock requirements is available in the PICmicro™ Mid-Range Reference Manual, (DS33023).

### 5.2 Prescaler

An 8-bit counter is available as a prescaler for the Timer0 module, or as a postscaler for the Watchdog Timer, respectively (Figure 5-2). For simplicity, this counter is being referred to as “prescaler” throughout this data sheet. Note that there is only one prescaler available which is mutually exclusively shared between the Timer0 module and the Watchdog Timer. Thus, a prescaler assignment for the Timer0 module means that there is no prescaler for the Watchdog Timer, and vice-versa.

The prescaler is not readable or writable.

The PSA and PS2:PS0 bits (OPTION\_REG<3:0>) determine the prescaler assignment and prescale ratio.

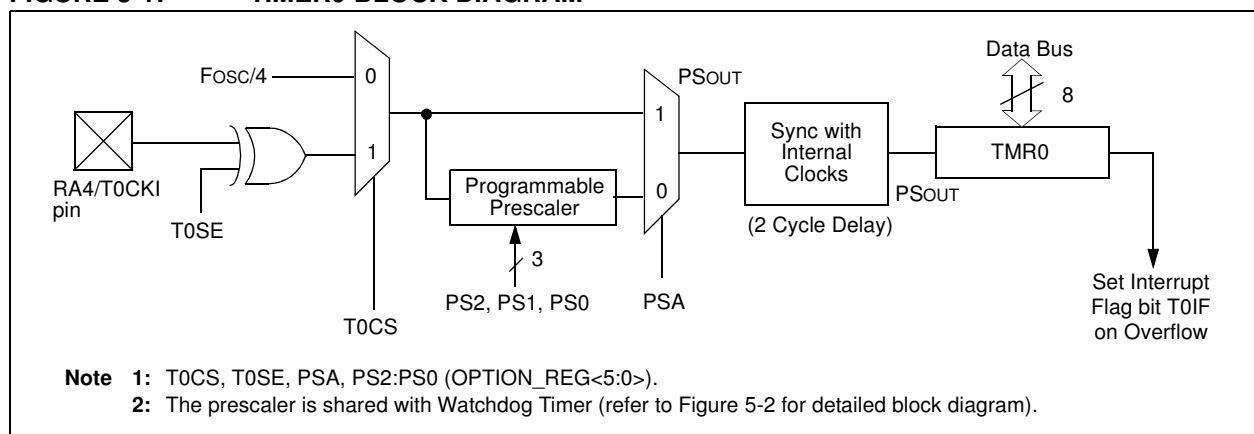
Clearing bit PSA will assign the prescaler to the Timer0 module. When the prescaler is assigned to the Timer0 module, prescale values of 1:2, 1:4, ..., 1:256 are selectable.

Setting bit PSA will assign the prescaler to the Watchdog Timer (WDT). When the prescaler is assigned to the WDT, prescale values of 1:1, 1:2, ..., 1:128 are selectable.

When assigned to the Timer0 module, all instructions writing to the TMR0 register (e.g., CLRF 1, MOVWF 1, BSF 1, etc.) will clear the prescaler. When assigned to WDT, a CLRWD instruction will clear the prescaler along with the WDT.

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

**FIGURE 5-1: TIMER0 BLOCK DIAGRAM**



# PIC16F84A

## 5.2.1 SWITCHING PRESCALER ASSIGNMENT

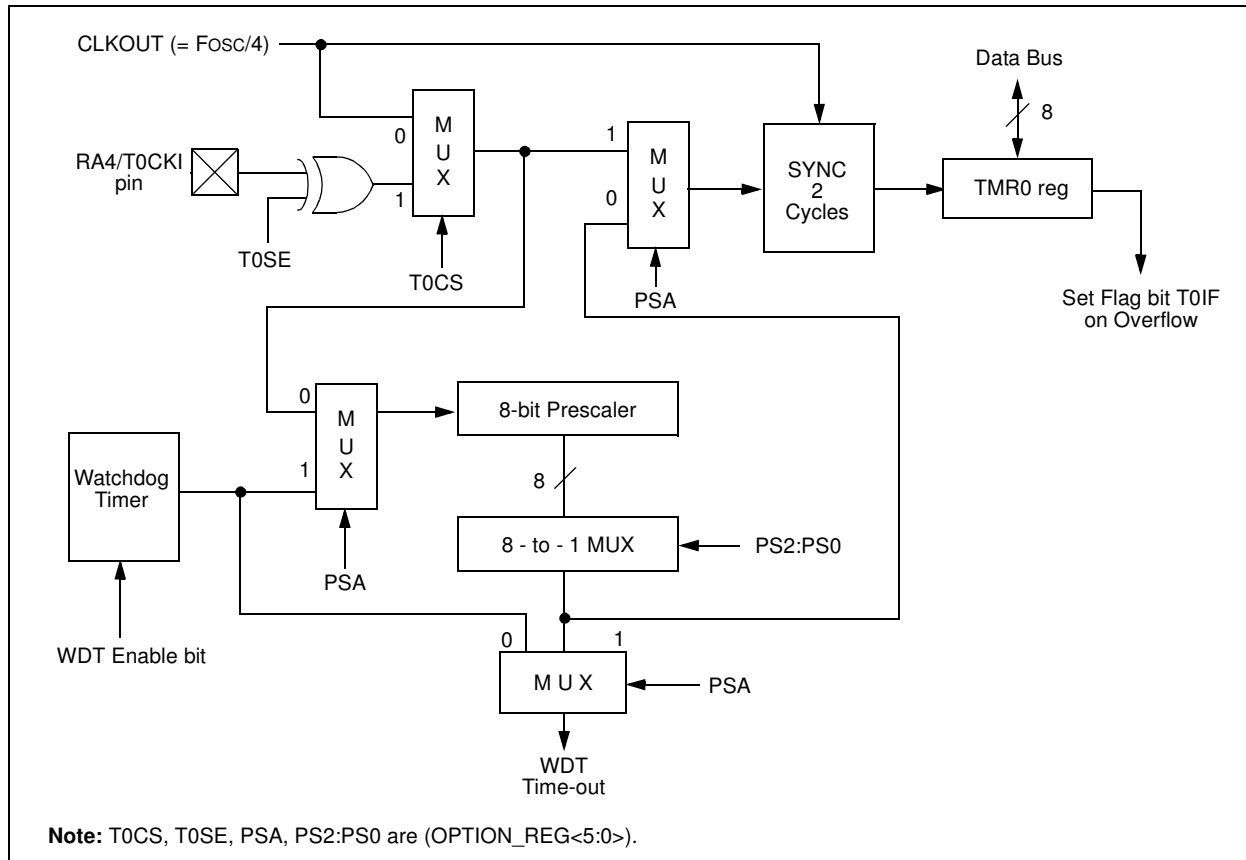
The prescaler assignment is fully under software control (i.e., it can be changed “on the fly” during program execution).

**Note:** To avoid an unintended device RESET, a specific instruction sequence (shown in the PICmicro™ Mid-Range Reference Manual, DS33023) must be executed when changing the prescaler assignment from Timer0 to the WDT. This sequence must be followed even if the WDT is disabled.

## 5.3 Timer0 Interrupt

The TMR0 interrupt is generated when the TMR0 register overflows from FFh to 00h. This overflow sets bit T0IF (INTCON<2>). The interrupt can be masked by clearing bit T0IE (INTCON<5>). Bit T0IF 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.

**FIGURE 5-2: BLOCK DIAGRAM OF THE TIMER0/WDT PRESCALER**



**TABLE 5-1: REGISTERS ASSOCIATED WITH TIMER0**

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
01h	TMR0	Timer0 Module Register								xxxx xxxx	uuuu uuuu
0Bh,8Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
85h	TRISA	—	—	—	PORTA Data Direction Register					---1 1111	---1 1111

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

## 6.8 Interrupts

The PIC16F84A has 4 sources of interrupt:

- External interrupt RB0/INT pin
- TMR0 overflow interrupt
- PORTB change interrupts (pins RB7:RB4)
- Data EEPROM write complete interrupt

The interrupt control register (INTCON) records individual interrupt requests in flag bits. It also contains the individual and global interrupt enable bits.

The global interrupt enable bit, GIE (INTCON<7>), enables (if set) all unmasked interrupts or disables (if cleared) all interrupts. Individual interrupts can be disabled through their corresponding enable bits in INTCON register. Bit GIE is cleared on RESET.

The “return from interrupt” instruction, `RETFIE`, exits interrupt routine as well as sets the GIE bit, which re-enables interrupts.

The RB0/INT pin interrupt, the RB port change interrupt and the TMR0 overflow interrupt flags are contained in the INTCON register.

When an interrupt is responded to, the GIE bit is cleared to disable any further interrupt, the return address is pushed onto the stack and the PC is loaded with 0004h. For external interrupt events, such as the RB0/INT pin or PORTB change interrupt, the interrupt latency will be three to four instruction cycles. The exact latency depends when the interrupt event occurs. The latency is the same for both one and two cycle instructions. Once in the Interrupt Service Routine, the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bit(s) must be cleared in software before re-enabling interrupts to avoid infinite interrupt requests.

**Note:** Individual interrupt flag bits are set regardless of the status of their corresponding mask bit or the GIE bit.

### 6.8.1 INT INTERRUPT

External interrupt on RB0/INT pin is edge triggered: either rising if INTEDG bit (OPTION\_REG<6>) is set, or falling if INTEDG bit is clear. When a valid edge appears on the RB0/INT pin, the INTF bit (INTCON<1>) is set. This interrupt can be disabled by clearing control bit INTE (INTCON<4>). Flag bit INTF must be cleared in software via the Interrupt Service Routine before re-enabling this interrupt. The INT interrupt can wake the processor from SLEEP (Section 6.11) only if the INTE bit was set prior to going into SLEEP. The status of the GIE bit decides whether the processor branches to the interrupt vector following wake-up.

### 6.8.2 TMR0 INTERRUPT

An overflow (FFh → 00h) in TMR0 will set flag bit T0IF (INTCON<2>). The interrupt can be enabled/disabled by setting/clearing enable bit T0IE (INTCON<5>) (Section 5.0).

### 6.8.3 PORTB INTERRUPT

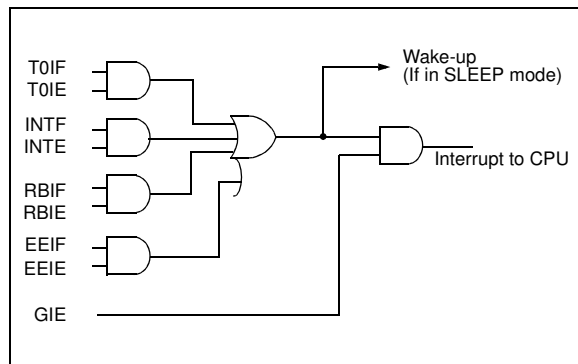
An input change on PORTB<7:4> sets flag bit RBIF (INTCON<0>). The interrupt can be enabled/disabled by setting/clearing enable bit RBIE (INTCON<3>) (Section 4.2).

**Note:** For a change on the I/O pin to be recognized, the pulse width must be at least  $T_{CY}$  wide.

### 6.8.4 DATA EEPROM INTERRUPT

At the completion of a data EEPROM write cycle, flag bit EEIF (EECON1<4>) will be set. The interrupt can be enabled/disabled by setting/clearing enable bit EEIE (INTCON<6>) (Section 3.0).

**FIGURE 6-10: INTERRUPT LOGIC**



# PIC16F84A

## 6.9 Context Saving During Interrupts

During an interrupt, only the return PC value is saved on the stack. Typically, users wish to save key register values during an interrupt (e.g., W register and STATUS register). This is implemented in software.

The code in Example 6-1 stores and restores the STATUS and W register's values. The user defined registers, W\_TEMP and STATUS\_TEMP are the temporary storage locations for the W and STATUS registers values.

Example 6-1 does the following:

- a) Stores the W register.
- b) Stores the STATUS register in STATUS\_TEMP.
- c) Executes the Interrupt Service Routine code.
- d) Restores the STATUS (and bank select bit) register.
- e) Restores the W register.

### EXAMPLE 6-1: SAVING STATUS AND W REGISTERS IN RAM

```
PUSH    MOVWF    W_TEMP      ; Copy W to TEMP register,
        SWAPF    STATUS,     W      ; Swap status to be saved into W
        MOVWF    STATUS_TEMP    ; Save status to STATUS_TEMP register
ISR      :
        :
        :      ; Interrupt Service Routine
        :      ; should configure Bank as required
        :
POP      SWAPF    STATUS_TEMP,W    ; Swap nibbles in STATUS_TEMP register
        :      ; and place result into W
        MOVWF    STATUS          ; Move W into STATUS register
        :      ; (sets bank to original state)
        SWAPF    W_TEMP,      F      ; Swap nibbles in W_TEMP and place result in W_TEMP
        SWAPF    W_TEMP,      W      ; Swap nibbles in W_TEMP and place result into W
```

## 6.10 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 RC oscillator of the OSC1/CLKIN pin. That means that the WDT will run even if the clock on the OSC1/CLKIN and OSC2/CLKOUT 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. If the device is in SLEEP mode, a WDT wake-up causes the device to wake-up and continue with normal operation. The WDT can be permanently disabled by programming configuration bit WDTE as a '0' (Section 6.1).

### 6.10.1 WDT PERIOD

The WDT has a nominal time-out period of 18 ms, (with no prescaler). The time-out periods vary with temperature, VDD and process variations from part to part (see DC specs). If longer time-out periods are desired, a prescaler with a division ratio of up to 1:128 can be assigned to the WDT under software control by writing to the OPTION\_REG register. Thus, time-out periods up to 2.3 seconds can be realized.

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.

The  $\overline{TO}$  bit in the STATUS register will be cleared upon a WDT time-out.

## 7.0 INSTRUCTION SET SUMMARY

Each PIC16CXX instruction is a 14-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 PIC16CXX instruction set summary in Table 7-2 lists **byte-oriented**, **bit-oriented**, and **literal and control** operations. Table 7-1 shows the opcode field descriptions.

For **byte-oriented** instructions, '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 W register. If 'd' is one, the result is placed in the file register specified in the instruction.

For **bit-oriented** instructions, 'b' represents a bit field designator which selects the number of the bit affected by the operation, while 'f' represents the address of the file in which the bit is located.

For **literal and control** operations, 'k' represents an eight or eleven bit constant or literal value.

**TABLE 7-1: OPCODE FIELD DESCRIPTIONS**

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 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.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1
PC	Program Counter
TO	Time-out bit
PD	Power-down bit

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

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal and control** operations

All instructions are executed within one single instruction cycle, unless a conditional test is true or the program counter is changed as a result of an instruction. In this case, the execution takes two instruction cycles with the second cycle executed as a NOP. One instruction cycle consists of four oscillator periods. Thus, for an oscillator frequency of 4 MHz, the normal instruction execution time is 1  $\mu$ s. If a conditional test is true or the program counter is changed as a result of an instruction, the instruction execution time is 2  $\mu$ s.

Table 7-2 lists the instructions recognized by the MPASM™ Assembler.

Figure 7-1 shows the general formats that the instructions can have.

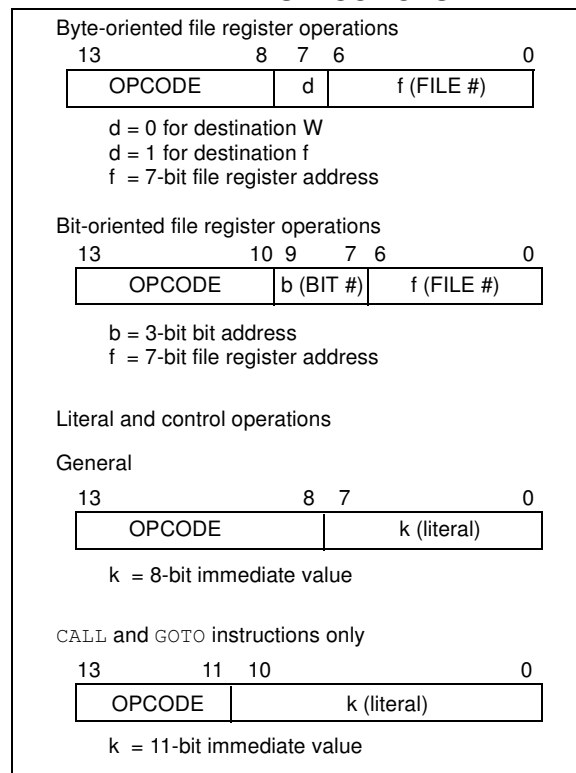
**Note:** To maintain upward compatibility with future PIC16CXX products, do not use the **OPTION** and **TRIS** instructions.

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

0xhh

where h signifies a hexadecimal digit.

**FIGURE 7-1: GENERAL FORMAT FOR INSTRUCTIONS**



A description of each instruction is available in the PICmicro™ Mid-Range Reference Manual (DS33023).

# PIC16F84A

**TABLE 7-2: PIC16CXXX INSTRUCTION SET**

Mnemonic, Operands		Description	Cycles	14-Bit Opcode				Status Affected	Notes
				MSb		LSb			
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECf	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1 (2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1 (2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDt	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}, \overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}, \overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

- Note 1:** When an I/O register is modified as a function of itself ( e.g., `MOVF PORTB, 1`), 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 to the Timer0 Module.
- 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.

**Note:** Additional information on the mid-range instruction set is available in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023).

## 7.1 Instruction Descriptions

### **ADDLW**      **Add Literal and W**

Syntax:      `[label] ADDLW k`  
 Operands:     $0 \leq k \leq 255$   
 Operation:     $(W) + k \rightarrow (W)$   
 Status Affected: C, DC, Z  
 Description:    The contents of the W register are added to the eight-bit literal 'k' and the result is placed in the W register.

### **BCF**      **Bit Clear f**

Syntax:      `[label] BCF f,b`  
 Operands:     $0 \leq f \leq 127$   
                   $0 \leq b \leq 7$   
 Operation:     $0 \rightarrow (f<b>)$   
 Status Affected: None  
 Description:    Bit 'b' in register 'f' is cleared.

### **ADDWF**      **Add W and f**

Syntax:      `[label] ADDWF f,d`  
 Operands:     $0 \leq f \leq 127$   
                   $d \in [0,1]$   
 Operation:     $(W) + (f) \rightarrow (\text{destination})$   
 Status Affected: C, DC, Z  
 Description:    Add the contents of the W register with register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.

### **BSF**      **Bit Set f**

Syntax:      `[label] BSF f,b`  
 Operands:     $0 \leq f \leq 127$   
                   $0 \leq b \leq 7$   
 Operation:     $1 \rightarrow (f<b>)$   
 Status Affected: None  
 Description:    Bit 'b' in register 'f' is set.

### **ANDLW**      **AND Literal with W**

Syntax:      `[label] ANDLW k`  
 Operands:     $0 \leq k \leq 255$   
 Operation:     $(W) .\text{AND.} (k) \rightarrow (W)$   
 Status Affected: Z  
 Description:    The contents of W register are AND'ed with the eight-bit literal 'k'. The result is placed in the W register.

### **BTFSS**      **Bit Test f, Skip if Set**

Syntax:      `[label] BTFSS f,b`  
 Operands:     $0 \leq f \leq 127$   
                   $0 \leq b < 7$   
 Operation:    skip if  $(f<b>) = 1$   
 Status Affected: None  
 Description:    If bit 'b' in register 'f' is '0', the next instruction is executed.  
                  If bit 'b' is '1', then the next instruction is discarded and a NOP is executed instead, making this a 2TCY instruction.

### **ANDWF**      **AND W with f**

Syntax:      `[label] ANDWF f,d`  
 Operands:     $0 \leq f \leq 127$   
                   $d \in [0,1]$   
 Operation:     $(W) .\text{AND.} (f) \rightarrow (\text{destination})$   
 Status Affected: Z  
 Description:    AND the W register with register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.



# PIC16F84A

---

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

---

Syntax:      *[label]* BTFSC *f*,*b*

Operands:       $0 \leq f \leq 127$   
                     $0 \leq b \leq 7$

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

Status Affected:      None

Description:      If bit 'b' in register 'f' is '1', the next instruction is executed.  
                    If bit 'b' in register 'f' is '0', the next instruction is discarded, and a NOP is executed instead, making this a 2Tcy instruction.

---

## **CLRWDT**      **Clear Watchdog Timer**

---

Syntax:      *[label]* CLRWDT

Operands:      None

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

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

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

---

## **CALL**      **Call Subroutine**

---

Syntax:      *[label]* CALL *k*

Operands:       $0 \leq k \leq 2047$

Operation:      (PC)+1 → TOS,  
                    *k* → PC<10:0>,  
                    (PCLATH<4:3>) → PC<12:11>

Status Affected:      None

Description:      Call Subroutine. First, return address (PC+1) is pushed onto the stack. The eleven-bit immediate address is loaded into PC bits <10:0>. The upper bits of the PC are loaded from PCLATH. CALL is a two-cycle instruction.

---

## **COMF**      **Complement f**

---

Syntax:      *[label]* COMF *f*,*d*

Operands:       $0 \leq f \leq 127$   
                    *d* ∈ [0,1]

Operation:      ( $\bar{f}$ ) → (destination)

Status Affected:      Z

Description:      The contents of register 'f' are complemented. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f'.

---

## **CLRF**      **Clear f**

---

Syntax:      *[label]* CLRF *f*

Operands:       $0 \leq f \leq 127$

Operation:      00h → (*f*)  
                    1 → Z

Status Affected:      Z

Description:      The contents of register 'f' are cleared and the Z bit is set.

---

## **DECF**      **Decrement f**

---

Syntax:      *[label]* DECF *f*,*d*

Operands:       $0 \leq f \leq 127$   
                    *d* ∈ [0,1]

Operation:      (*f*) - 1 → (destination)

Status Affected:      Z

Description:      Decrement register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.

---

## **CLRW**      **Clear W**

---

Syntax:      *[label]* CLRW

Operands:      None

Operation:      00h → (W)  
                    1 → Z

Status Affected:      Z

Description:      W register is cleared. Zero bit (Z) is set.

## DECFSZ      Decrement f, Skip if 0

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

**Operands:**       $0 \leq f \leq 127$   
 $d \in [0,1]$

**Operation:**       $(f) - 1 \rightarrow (\text{destination});$   
skip if result = 0

**Status Affected:**      None

**Description:**      The contents of register 'f' are decremented. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'.  
If the result is 1, the next instruction is executed. If the result is 0, then a NOP is executed instead, making it a 2Tcy instruction.

## INCFSZ      Increment f, Skip if 0

**Syntax:**      [ *label* ] INCFSZ f,d

**Operands:**       $0 \leq f \leq 127$   
 $d \in [0,1]$

**Operation:**       $(f) + 1 \rightarrow (\text{destination}),$   
skip if result = 0

**Status Affected:**      None

**Description:**      The contents of register 'f' are incremented. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'.  
If the result is 1, the next instruction is executed. If the result is 0, a NOP is executed instead, making it a 2Tcy instruction.

## GOTO      Unconditional Branch

**Syntax:**      [ *label* ] GOTO k

**Operands:**       $0 \leq k \leq 2047$

**Operation:**       $k \rightarrow PC<10:0>$   
 $PCLATH<4:3> \rightarrow PC<12:11>$

**Status Affected:**      None

**Description:**      GOTO is an unconditional branch. The eleven-bit immediate value is loaded into PC bits <10:0>. The upper bits of PC are loaded from PCLATH<4:3>. GOTO is a two-cycle instruction.

## IORLW      Inclusive OR Literal with W

**Syntax:**      [ *label* ] IORLW k

**Operands:**       $0 \leq k \leq 255$

**Operation:**       $(W) .OR. k \rightarrow (W)$

**Status Affected:**      Z

**Description:**      The contents of the W register are OR'ed with the eight-bit literal 'k'. The result is placed in the W register.

## INCF      Increment f

**Syntax:**      [ *label* ] INCF f,d

**Operands:**       $0 \leq f \leq 127$   
 $d \in [0,1]$

**Operation:**       $(f) + 1 \rightarrow (\text{destination})$

**Status Affected:**      Z

**Description:**      The contents of register 'f' are incremented. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'.

## IORWF      Inclusive OR W with f

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

**Operands:**       $0 \leq f \leq 127$   
 $d \in [0,1]$

**Operation:**       $(W) .OR. (f) \rightarrow (\text{destination})$

**Status Affected:**      Z

**Description:**      Inclusive OR the W register with register 'f'. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'.

# PIC16F84A

---

## **MOVF**                      **Move f**

---

Syntax:            [ *label* ]   MOVF   f,d

Operands:         $0 \leq f \leq 127$   
                     $d \in [0,1]$

Operation:        (f)  $\rightarrow$  (destination)

Status Affected:   Z

Description:      The contents of register f are moved to a destination dependant upon the status of d. If d = 0, destination is W register. If d = 1, the destination is file register f itself. d = 1 is useful to test a file register, since status flag Z is affected.

## **RETFIE**                    **Return from Interrupt**

---

Syntax:            [ *label* ]   RETFIE

Operands:        None

Operation:        TOS  $\rightarrow$  PC,  
                    1  $\rightarrow$  GIE

Status Affected:   None

## **MOVLW**                    **Move Literal to W**

---

Syntax:            [ *label* ]   MOVLW   k

Operands:         $0 \leq k \leq 255$

Operation:        k  $\rightarrow$  (W)

Status Affected:   None

Description:      The eight-bit literal 'k' is loaded into W register. The don't cares will assemble as 0's.

## **RETLW**                    **Return with Literal in W**

---

Syntax:            [ *label* ]   RETLW   k

Operands:         $0 \leq k \leq 255$

Operation:        k  $\rightarrow$  (W);  
                    TOS  $\rightarrow$  PC

Status Affected:   None

Description:      The W register is loaded with the eight-bit literal 'k'. The program counter is loaded from the top of the stack (the return address). This is a two-cycle instruction.

## **MOVWF**                    **Move W to f**

---

Syntax:            [ *label* ]   MOVWF   f

Operands:         $0 \leq f \leq 127$

Operation:        (W)  $\rightarrow$  (f)

Status Affected:   None

Description:      Move data from W register to register 'f'.

## **RETURN**                    **Return from Subroutine**

---

Syntax:            [ *label* ]   RETURN

Operands:        None

Operation:        TOS  $\rightarrow$  PC

Status Affected:   None

Description:      Return from subroutine. The stack is POPed and the top of the stack (TOS) is loaded into the program counter. This is a two-cycle instruction.

## **NOP**                        **No Operation**

---

Syntax:            [ *label* ]   NOP

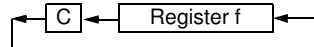
Operands:        None

Operation:        No operation

Status Affected:   None

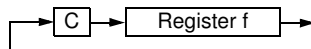
Description:      No operation.

RLF	Rotate Left f through Carry
Syntax:	[ <i>label</i> ] RLF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	See description below
Status Affected:	C
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 the W register. If 'd' is 1, the result is stored back in register 'f'.



SUBLW	Subtract W from Literal
Syntax:	[ <i>label</i> ] SUBLW k
Operands:	$0 \leq k \leq 255$
Operation:	$k - (W) \rightarrow (W)$
Status Affected:	C, DC, Z
Description:	The W register is subtracted (2's complement method) from the eight-bit literal 'k'. The result is placed in the W register.

RRF	Rotate Right f through Carry
Syntax:	[ <i>label</i> ] RRF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	See description below
Status Affected:	C
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 the W register. If 'd' is 1, the result is placed back in register 'f'.



SUBWF	Subtract W from f
Syntax:	[ <i>label</i> ] SUBWF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	$(f) - (W) \rightarrow (\text{destination})$
Status Affected:	C, DC, Z
Description:	Subtract (2's complement method) W register from register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.

SLEEP	
Syntax:	[ <i>label</i> ] SLEEP
Operands:	None
Operation:	00h $\rightarrow$ WDT, 0 $\rightarrow$ WDT prescaler, 1 $\rightarrow$ $\overline{\text{TO}}$ , 0 $\rightarrow$ $\overline{\text{PD}}$
Status Affected:	$\overline{\text{TO}}$ , $\overline{\text{PD}}$
Description:	The power-down status bit, $\overline{\text{PD}}$ is cleared. Time-out status bit, $\overline{\text{TO}}$ is set. Watchdog Timer and its prescaler are cleared. The processor is put into SLEEP mode with the oscillator stopped.

SWAPF	Swap Nibbles in f
Syntax:	[ <i>label</i> ] SWAPF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	$(f<3:0>) \rightarrow (\text{destination}<7:4>),$ $(f<7:4>) \rightarrow (\text{destination}<3:0>)$
Status Affected:	None
Description:	The upper and lower nibbles of register 'f' are exchanged. If 'd' is 0, the result is placed in W register. If 'd' is 1, the result is placed in register 'f'.

# PIC16F84A

---

<b>XORLW</b>	<b>Exclusive OR Literal with W</b>
Syntax:	<i>[label]</i> XORLW k
Operands:	$0 \leq k \leq 255$
Operation:	(W) .XOR. k $\rightarrow$ (W)
Status Affected:	Z
Description:	The contents of the W register are XOR'ed with the eight-bit literal 'k'. The result is placed in the W register.

<b>XORWF</b>	<b>Exclusive OR W with f</b>
Syntax:	<i>[label]</i> XORWF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	(W) .XOR. (f) $\rightarrow$ (destination)
Status Affected:	Z
Description:	Exclusive OR the contents of the W register with register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.