UFRGS / EE / DELET

Semana Acadêmica 2004

CURSO PIC

Compilado por
Prof. A. Junqueira
Outubro/2004

ÍNDICE

PIC

Peripheral Interface Controller

Microcontroladores

- Low-End
- MidRange
- High-End

Low-End

- Conjunto reduzido de instruções
- Sem tratamento de interrupções
- Menor disponibilidade de memória RAM
- Sem I/O avançado
- Ex: PIC 12C5xx
- Não deve ser utilizado para novas aplicações ou para aprendizado

MidRange

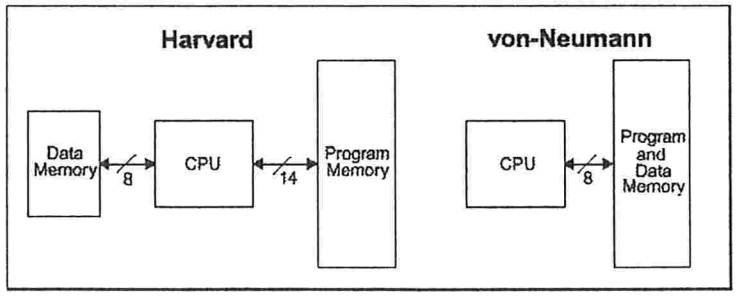
- Baseado na organização dos Low-End
- Tratamento do interrupções
- Timers
- Quantidade razoável de memória RAM
- Tipos avançados de I/O
- Ex: PIC 16C7xx
- Caracterizado como sendo de uso geral

High-End

- Tudo dos anteriores e mais...
- Acesso a todos os registradores diretamente
- Múltiplos vetores de interrupção
- Ex: PIC 17Cxx

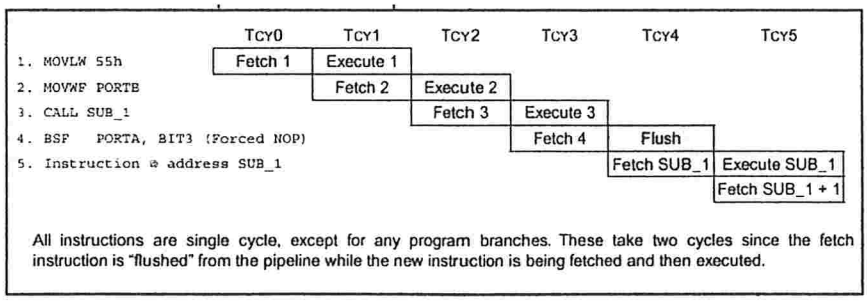
Arquitetura:

- Arquitetura Harvard:



- Instruções longas:
  - Utilização mais eficiente da memória de programa.
- Instruções de apenas uma palavra:
  - Cada instrução é recuperada da memória em apenas um ciclo.

- Pipeline de instruções:
  - Pipeline de 2 estágios.
  - Um ciclo para fetch e outro para execução.
  - Uma instrução é executada por ciclo.
  - Intruções de "Branch" gastam 2 ciclos.



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.

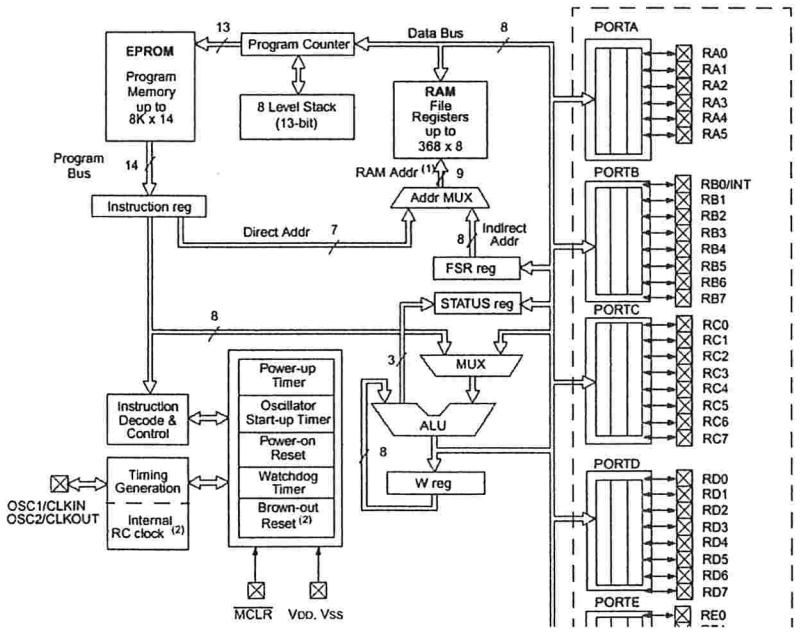
- Conjunto reduzido de Instruções:
  - Facilidade de aprendizado.
  - Menor tempo de programação.
  - 35 instruções.
- Registradores são mapeados em memória:
  - Todos os registradores de funções especiais, incluindo o program counter, podem ser direta ou indiretamente acessados.

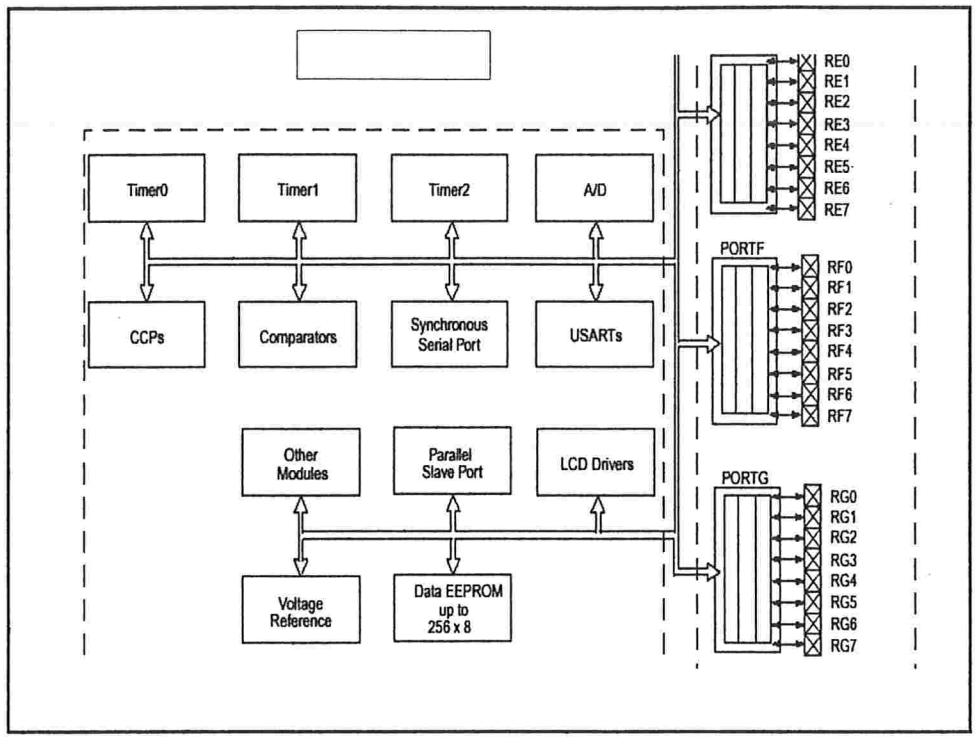
- Interrupções:
  - Controle de até 12 origens de interrupção independentes.
- Timers:
  - 3 timers podem ser programados para controlar entradas, saídas ou eventos internos ao PIC.

- Power-on reset e brown-out protection:
  - Garantem que o PIC só irá operar quando os valores de voltagem estiverem dentro das especificações.
- Watchdog timer:
  - Reseta o PIC se ele apresentar qualquer sinal de mal-funcionamento.

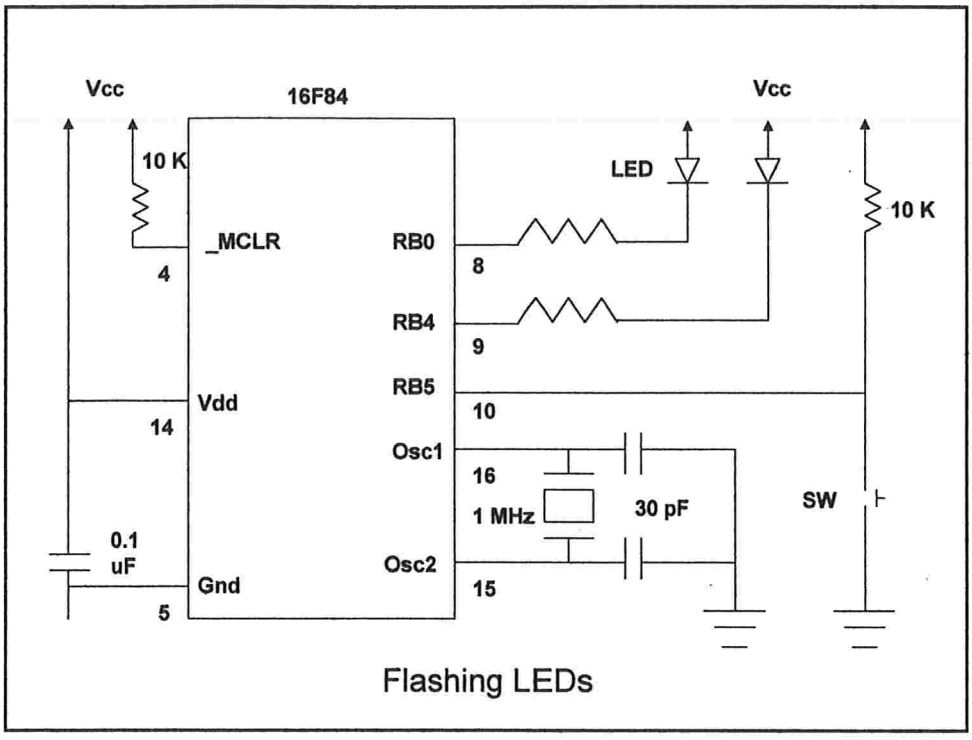
- Power-up Timer
  - Permite que a corrente de entrada alcance um patamar aceitável de operação.
  - Atraso de 72 ms.
- Oscillator Start-up Timer
  - Garante que o oscilador foi iniciado e está estável.
  - Gera um atraso correspondente a 1024 ciclos do oscilador.

- Sleep Mode:
  - Quando o comando "sleep" é executado, o oscilador que gera o clock para o circuito é desligado.
  - É o estado que consome menos energia.
  - O usuário pode "acordar" o PIC através de um reset externo, do reset do Watchdog timer ou através de uma interrupção.

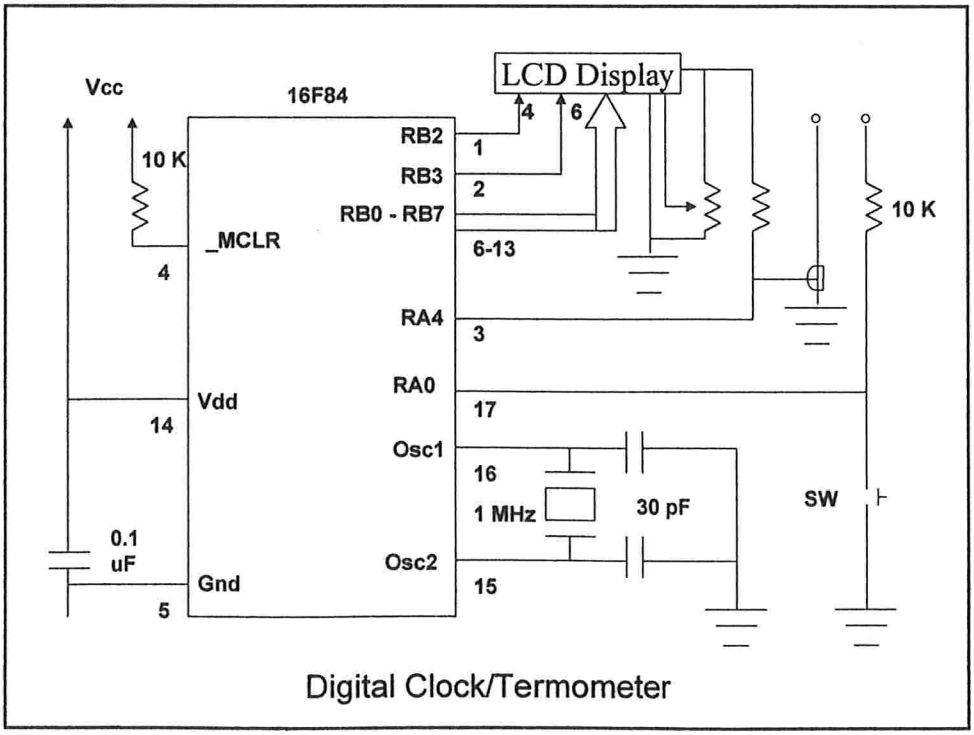




- Classificação quanto a memória:
  - PIC 16 Cxxx - Memória EPROM
  - PIC 16 CRxxx - Memória ROM
  - PIC 16 Fxxx - Memória FLASH



Flashing LEDs



Digital Clock/Termometer

- *MPASM*
  - Assembler capaz de gerar código para todos os dispositivos da linha PICMicro;
  - Disponível em linha de comando "DOS" ou interface Windows;
- *MPLAB*
  - desenvolver e editar aplicações;
  - simular a aplicação com estímulos externos e modificar registros internos;
  - programar um chip;
  - prover uma interface de emulador de hardware similar ao simulador;

- KeeLog - algoritmo e fonte PICMicro para permitir controle externo de dispositivos;
- Fuzzy Tech - sistema de desenvolvimento para simplificar o esforço em desenvolver aplicações com lógica fuzzy para PICMicro

Microchip

- PIC16F84A Data Sheet
- PIC16F84A Errata Sheet (2)
- PIC16F8X EEPROM Memory Programming Specification
- SQTP^{SM} Specification for PIC16/17

PIC16F84A
Data Sheet

18-pin Enhanced FLASH/EEPROM
8-bit Microcontroller

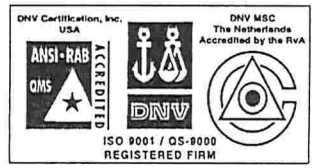
Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable".
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

Trademarks

Printed on recycled paper.



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.

MICROCHIP

PIC16F84A

Enhanced

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:

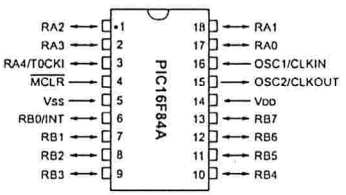
- 13 I/O pins with individual direction control
- High current sink/source for direct LED drive
  - 25 mA sink max. per pin
  - 25 mA source max. per pin
- TMR0: 8-bit timer/counter with 8-bit programmable prescaler

Special Microcontroller Features:

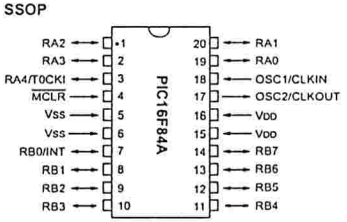
- 10,000 erase/write cycles *Enhanced* FLASH Program memory typical
- 10,000,000 typical erase/write cycles EEPROM Data memory typical
- EEPROM Data Retention > 40 years
- In-Circuit Serial Programming™ (ICSP™) - via two pins
- Power-on Reset (POR), Power-up Timer (PWRT), Oscillator Start-up Timer (OST)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

Pin Diagrams

PDIP, SOIC



SSOP



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

PIC16F84A

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.

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'.

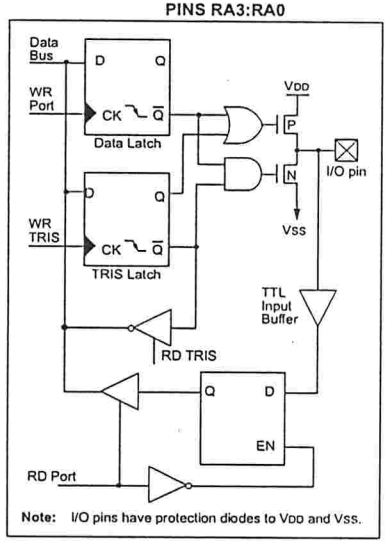
Reading the PORTA register reads the status of the pins, whereas writing to it will write to the port latch. All write operations are read-modify-write operations. Therefore, a write to a port implies that the port pins are read. This value is modified and then written to the port data latch.

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

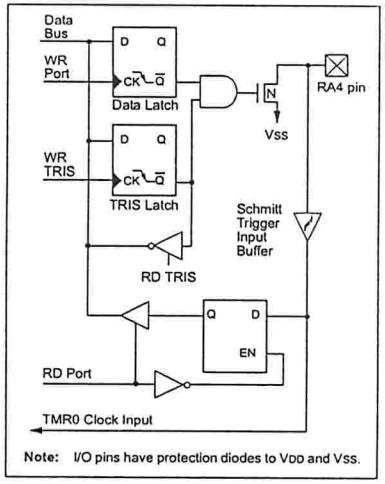
```
BCF     STATUS, RP0 ;
CLRF    PORTA       ; Initialize PORTA by
                    ; clearing output
                    ; data latches
BSF     STATUS, RP0 ; Select Bank 1
MOVLW   0x0F        ; Value used to
                    ; initialize data
                    ; direction
MOVWF   TRISA       ; 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



Note: I/O pins have protection diodes to VDD and VSS.

FIGURE 4-2: BLOCK DIAGRAM OF PIN RA4



Note: I/O pins have protection diodes to VDD and VSS.

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 $\overline{\text{RBPU}}$ (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:

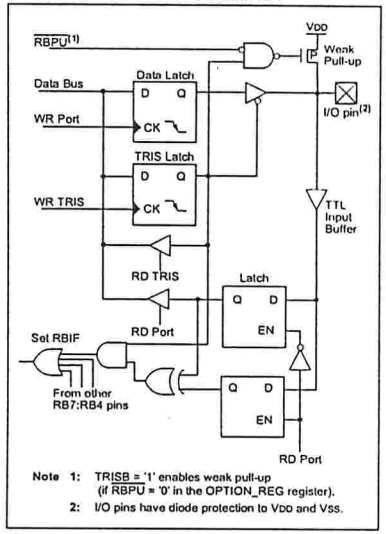
a) Any read or write of PORTB. This will end the mismatch condition.

b) 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.

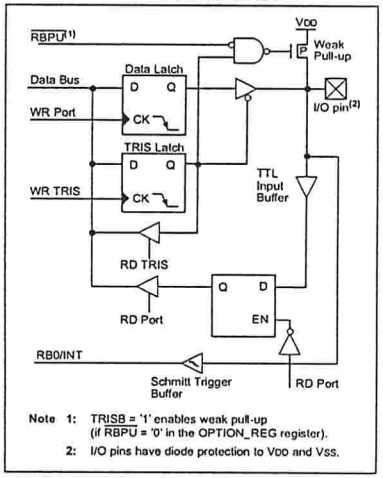
The interrupt-on-change feature is recommended for wake-up on key depression operation and operations where PORTB is only used for the interrupt-on-change feature. Polling of PORTB is not recommended while using the interrupt-on-change feature.

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



Note 1: TRISB = '1' enables weak pull-up (if $\overline{\text{RBPU}}$ = '0' in the OPTION_REG register).
2: I/O pins have diode protection to VDD and VSS.

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



Note 1: TRISB = '1' enables weak pull-up (if $\overline{\text{RBPU}}$ = '0' in the OPTION_REG register).
2: I/O pins have diode protection to VDD and VSS.

TABLE 4-3: PORTB FUNCTIONS

Name	Bit	Buffer Type	I/O Consistency Function
RB0/INT	bit0	TTL/ST(1)	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(2)	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up. Serial programming clock.
RB7	bit7	TTL/ST(2)	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	RBIF	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.

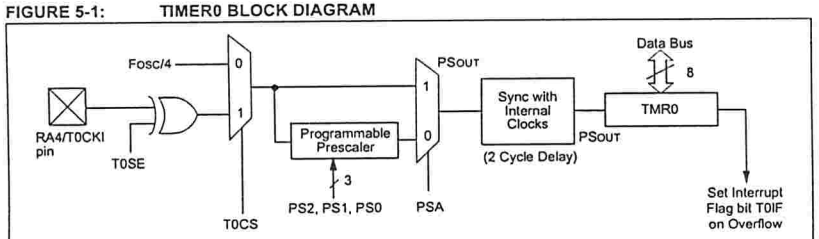
Clearing bit PSA will assign the prescaler to the Timer0 module. When the prescaler is assigned to the Timer0 module, prescale values of 1:2, 1:4, ..., 1:256 are selectable.

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 `CLRWDT` 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



Note 1: T0CS, T0SE, PSA, PS2:PS0 (OPTION_REG<5:0>).
2: The prescaler is shared with Watchdog Timer (refer to Figure 5-2 for detailed block diagram).

ERRATA

Module: Timer0

The TMR0 register may increment when the WDT postscaler is switched to the Timer0 prescaler. If TMR0 = FFh, this will cause TMR0 to overflow (setting T0IF).

Work Around

Follow the following sequence:

1) Read the 8-bit TMR0 register into the W register
2) Clear the TMR0 register
3) Assign WDT postscaler to Timer0
4) Write W register to TMR0

5.2.1 SWITCHING PRESCALER ASSIGNMENT

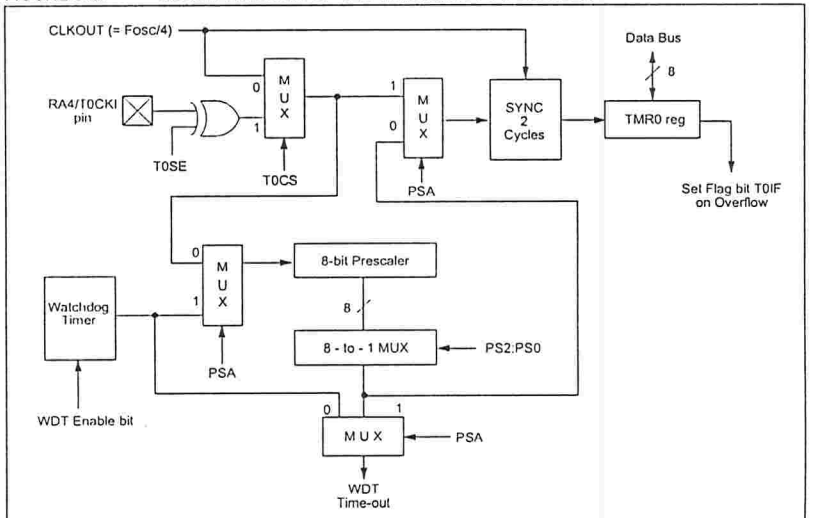
The prescaler assignment is fully under software control (i.e., it can be changed "on the fly" during program execution).

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



Note: T0CS, T0SE, PSA, PS2:PS0 are (OPTION_REG<5:0>).

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.

6.0 SPECIAL FEATURES OF THE CPU

What sets a microcontroller apart from other processors are special circuits to deal with the needs of real time applications. The PIC16F84A has a host of such features intended to maximize system reliability, minimize cost through elimination of external components, provide power saving operating modes and offer code protection. These features are:

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

The PIC16F84A has a Watchdog Timer which can be shut-off only through configuration bits. It runs off its own RC 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 of 72 ms (nominal) on power-up only. This design keeps the device in RESET while the power supply stabilizes. With these two timers on-chip, most applications need no external RESET circuitry.

SLEEP mode offers a very low current power-down mode. The user can wake-up from SLEEP through external RESET, Watchdog Timer Time-out or through an interrupt. Several oscillator options are provided to allow the part to fit the application. The RC oscillator option saves system cost while the LP crystal option saves power. A set of configuration bits are used to select the various options.

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

6.1 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 in program memory location 2007h.

Address 2007h is beyond the user program memory space and it belongs to the special test/configuration memory space (2000h - 3FFFh). This space can only be accessed during programming.

REGISTER 6-1: PIC16F84A CONFIGURATION WORD

R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u
CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	$\overline{\text{PWRTE}}$	WDTE	F0SC1	F0SC0
bit13													bit0

bit 13-4 **CP**: Code Protection bit
1 = Code protection disabled
0 = All program memory is code protected

bit 3 $\overline{\text{PWRTE}}$: Power-up Timer Enable bit
1 = Power-up Timer is disabled
0 = Power-up Timer is enabled

bit 2 **WDTE**: Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled

bit 1-0 **FOSC1:FOSC0**: Oscillator Selection bits
11 = RC oscillator
10 = HS oscillator
01 = XT oscillator
00 = LP oscillator

6.2 Oscillator Configurations

6.2.1 OSCILLATOR TYPES

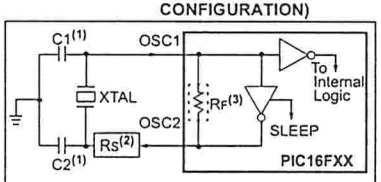
The PIC16F84A can be operated in four different oscillator modes. The user can program two configuration bits (FOSC1 and FOSC0) to select one of these four modes:

- LP Low Power Crystal
- XT Crystal/Resonator
- HS High Speed Crystal/Resonator
- RC Resistor/Capacitor

6.2.2 CRYSTAL OSCILLATOR/CERAMIC RESONATORS

In XT, LP, or HS modes, a crystal or ceramic resonator is connected to the OSC1/CLKIN and OSC2/CLKOUT pins to establish oscillation (Figure 6-1).

FIGURE 6-1: CRYSTAL/CERAMIC RESONATOR OPERATION (HS, XT OR LP OSC CONFIGURATION)



Note 1: See Table 6-1 for recommended values of C1 and C2.

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

The PIC16F84A oscillator design requires the use of a parallel cut crystal. Use of a series cut crystal may give a frequency out of the crystal manufacturers specifications. When in XT, LP, or HS modes, the device can have an external clock source to drive the OSC1/CLKIN pin (Figure 6-2).

FIGURE 6-2: EXTERNAL CLOCK INPUT OPERATION (HS, XT OR LP OSC CONFIGURATION)

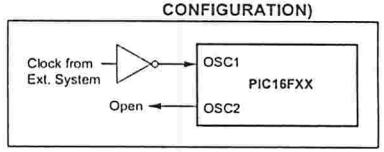


TABLE 6-1: CAPACITOR SELECTION FOR CERAMIC RESONATORS

Ranges Tested:			
Mode	**Freq**	**OSC1/C1**	**OSC2/C2**
XT	455 kHz	47 - 100 pF	47 - 100 pF
	2.0 MHz	15 - 33 pF	15 - 33 pF
	4.0 MHz	15 - 33 pF	15 - 33 pF
HS	8.0 MHz	15 - 33 pF	15 - 33 pF
	10.0 MHz	15 - 33 pF	15 - 33 pF

Note: Recommended values of C1 and C2 are identical to the ranges tested in this table. Higher capacitance increases the stability of the oscillator, but also increases the start-up time. These values are for design guidance only. Since each resonator has its own characteristics, the user should consult the resonator manufacturer for the appropriate values of external components.

Note: When using resonators with frequencies above 3.5 MHz, the use of HS mode rather than XT mode, is recommended. HS mode may be used at any VDD for which the controller is rated.

TABLE 6-2: CAPACITOR SELECTION FOR CRYSTAL OSCILLATOR

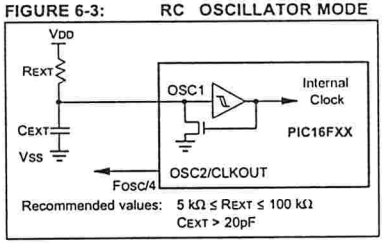
Mode	Freq	OSC1/C1	OSC2/C2
LP	32 kHz	68 - 100 pF	68 - 100 pF
	200 kHz	15 - 33 pF	15 - 33 pF
XT	100 kHz	100 - 150 pF	100 - 150 pF
	2 MHz	15 - 33 pF	15 - 33 pF
	4 MHz	15 - 33 pF	15 - 33 pF
HS	4 MHz	15 - 33 pF	15 - 33 pF
	20 MHz	15 - 33 pF	15 - 33 pF

Note: Higher capacitance increases the stability of the oscillator, but also increases the start-up time. These values are for design guidance only. Rs may be required in HS mode, as well as XT mode, to avoid overdriving crystals with low drive level specification. Since each crystal has its own characteristics, the user should consult the crystal manufacturer for appropriate values of external components.
For VDD > 4.5V, C1 = C2 ≈ 30 pF is recommended.

6.2.3 RC OSCILLATOR

For timing insensitive applications, the RC device option offers additional cost savings. The RC oscillator frequency is a function of the supply voltage, the resistor (REXT) values, capacitor (CEXT) values, and the operating temperature. In addition to this, the oscillator frequency will vary from unit to unit due to normal process parameter variation. Furthermore, the difference in lead frame capacitance between package types also affects the oscillation frequency, especially for low CEXT values. The user needs to take into account variation, due to tolerance of the external R and C components. Figure 6-3 shows how an R/C combination is connected to the PIC16F84A.

FIGURE 6-3: RC OSCILLATOR MODE



6.3 RESET

The PIC16F84A differentiates between various kinds of RESET:

- Power-on Reset (POR)
- $\overline{MCLR}$ during normal operation
- $\overline{MCLR}$ during SLEEP
- WDT Reset (during normal operation)
- WDT Wake-up (during SLEEP)

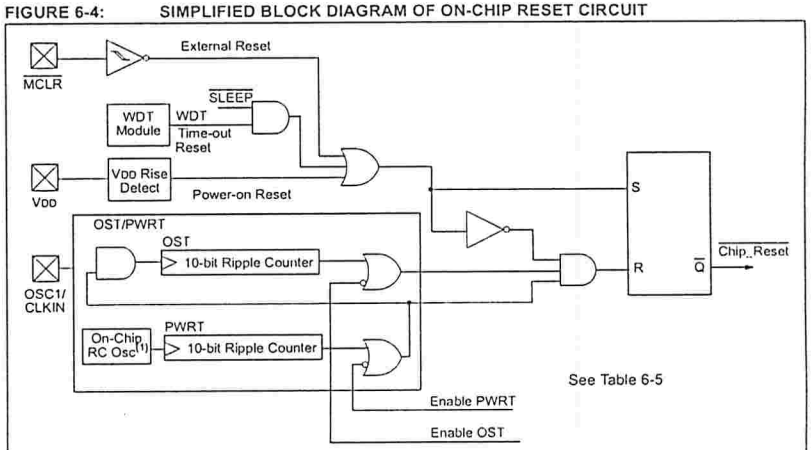
Figure 6-4 shows a simplified block diagram of the On-Chip RESET Circuit. The $\overline{MCLR}$ Reset path has a noise filter to ignore small pulses. The electrical specifications state the pulse width requirements for the $\overline{MCLR}$ pin.

Some registers are not affected in any RESET condition; their status is unknown on a POR and unchanged in any other RESET. Most other registers are reset to a "RESET state" on POR, $\overline{MCLR}$ or WDT Reset during normal operation and on $\overline{MCLR}$ during SLEEP. They are not affected by a WDT Reset during SLEEP, since this RESET is viewed as the resumption of normal operation.

Table 6-3 gives a description of RESET conditions for the program counter (PC) and the STATUS register. Table 6-4 gives a full description of RESET states for all registers.

The $\overline{TO}$ and $\overline{PD}$ bits are set or cleared differently in different RESET situations (Section 6.7). These bits are used in software to determine the nature of the RESET.

FIGURE 6-4: SIMPLIFIED BLOCK DIAGRAM OF ON-CHIP RESET CIRCUIT



Note 1: This is a separate oscillator from the RC oscillator of the CLKIN pin.
2: See Table 6-5.

TABLE 6-3: RESET CONDITION FOR PROGRAM COUNTER AND THE STATUS REGISTER

Condition	Program Counter	STATUS Register
Power-on Reset	000h	0001 1xxx
$\overline{MCLR}$ during normal operation	000h	000u uuuu
$\overline{MCLR}$ during SLEEP	000h	0001 0uuu
WDT Reset (during normal operation)	000h	0000 1uuu
WDT Wake-up	PC + 1	uuu0 0uuu
Interrupt wake-up from SLEEP	PC + 1(1)	uuu1 0uuu

Legend: u = unchanged, x = unknown
Note 1: When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h).

TABLE 6-4: RESET CONDITIONS FOR ALL REGISTERS

Register	Address	Power-on Reset	MCLR during: – normal operation – SLEEP WDT Reset during normal operation	Wake-up from SLEEP: – through interrupt – through WDT Time-out
W	—	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF	00h	---- ----	---- ----	---- ----
TMR0	01h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCL	02h	0000 0000	0000 0000	PC + 1(2)
STATUS	03h	0001 1xxx	000q quuu(3)	uuuq quuu(3)
FSR	04h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTA(4)	05h	---x xxxx	---u uuuu	---u uuuu
PORTB(5)	06h	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEDATA	08h	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEADR	09h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCLATH	0Ah	---0 0000	---0 0000	---u uuuu
INTCON	0Bh	0000 000x	0000 000u	uuuu uuuu(1)
INDF	80h	---- ----	---- ----	---- ----
OPTION_REG	81h	1111 1111	1111 1111	uuuu uuuu
PCL	82h	0000 0000	0000 0000	PC + 1(2)
STATUS	83h	0001 1xxx	000q quuu(3)	uuuq quuu(3)
FSR	84h	xxxx xxxx	uuuu uuuu	uuuu uuuu
TRISA	85h	---1 1111	---1 1111	---u uuuu
TRISB	86h	1111 1111	1111 1111	uuuu uuuu
EECON1	88h	---0 x000	---0 q000	---0 uuuu
EECON2	89h	---- ----	---- ----	---- ----
PCLATH	8Ah	---0 0000	---0 0000	---u uuuu
INTCON	8Bh	0000 000x	0000 000u	uuuu uuuu(1)

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

Note 1: One or more bits in INTCON will be affected (to cause wake-up).

2: When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h).

3: Table 6-3 lists the RESET value for each specific condition.

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

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

6.4 Power-on Reset (POR)

A Power-on Reset pulse is generated on-chip when VDD rise is detected (in the range of 1.2V - 1.7V). To take advantage of the POR, just tie the MCLR pin directly (or through a resistor) to VDD. This will eliminate external RC components usually needed to create Power-on Reset. A minimum rise time for VDD must be met for this to operate properly. See Electrical Specifications for details.

When the device starts normal operation (exits the RESET condition), device operating parameters (voltage, frequency, temperature, etc.) 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.

For additional information, refer to Application Note AN607, "*Power-up Trouble Shooting.*"

The POR circuit does not produce an internal RESET when VDD declines.

6.5 Power-up Timer (PWRT)

The Power-up Timer (PWRT) provides a fixed 72 ms nominal time-out (TPWRT) from POR (Figures 6-6 through 6-9). The Power-up Timer operates on an internal RC oscillator. The chip is kept in RESET as long as the PWRT is active. The PWRT delay allows the VDD to rise to an acceptable level (possible exception shown in Figure 6-9).

A configuration bit, PWRTE, can enable/disable the PWRT. See Register 6-1 for the operation of the PWRTE bit for a particular device.

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

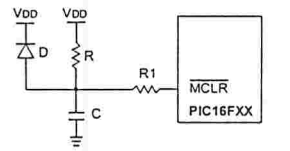
6.6 Oscillator Start-up Timer (OST)

The Oscillator Start-up Timer (OST) provides a 1024 oscillator cycle delay (from OSC1 input) after the PWRT delay ends (Figure 6-6, Figure 6-7, Figure 6-8 and Figure 6-9). This ensures the crystal oscillator or resonator has started and stabilized.

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

When VDD rises very slowly, it is possible that the TPWRT time-out and TOST time-out will expire before VDD has reached its final value. In this case (Figure 6-9), an external Power-on Reset circuit may be necessary (Figure 6-5).

FIGURE 6-5: EXTERNAL POWER-ON RESET CIRCUIT (FOR SLOW VDD POWER-UP)



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

2: R < 40 kΩ is recommended to make sure that voltage drop across R does not exceed 0.2V (max leakage current spec on MCLR pin is 5 μA). A larger voltage drop will degrade VIH level on the MCLR pin.

3: R1 = 100Ω to 1 kΩ will limit any current flowing into MCLR from external capacitor C, in the event of a MCLR pin breakdown due to ESD or EOS.

FIGURE 6-6: TIME-OUT SEQUENCE ON POWER-UP (MCLR NOT TIED TO VDD): CASE 1

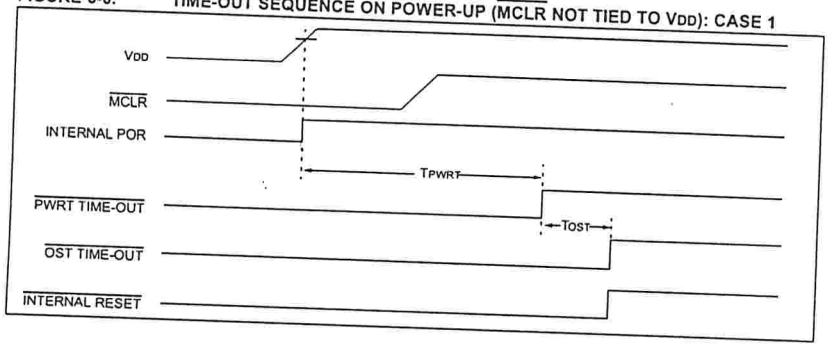


FIGURE 6-7: TIME-OUT SEQUENCE ON POWER-UP (MCLR NOT TIED TO VDD): CASE 2

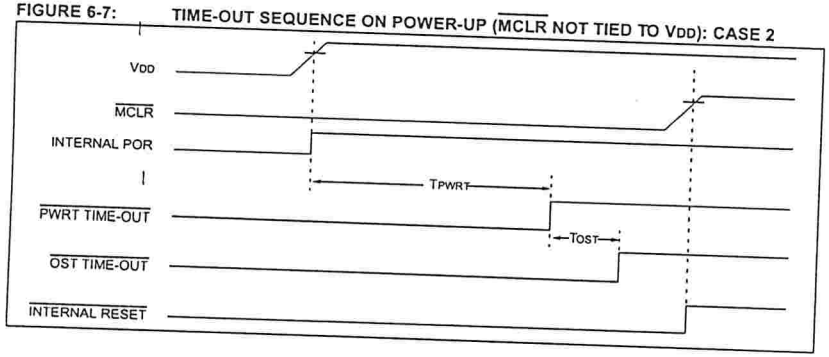


FIGURE 6-8: TIME-OUT SEQUENCE ON POWER-UP (MCLR TIED TO VDD): FAST VDD RISE TIME

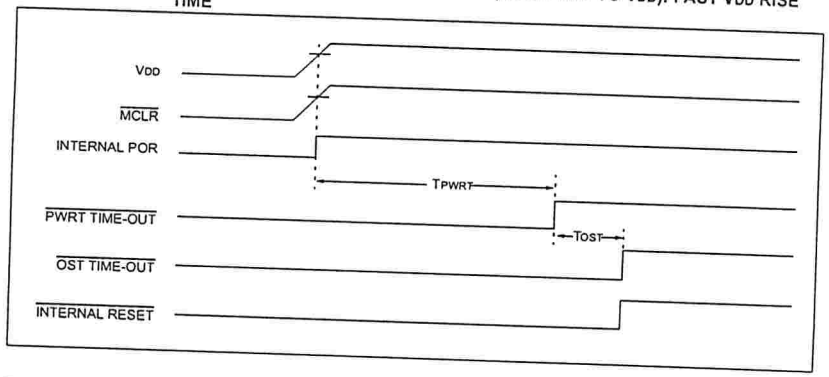
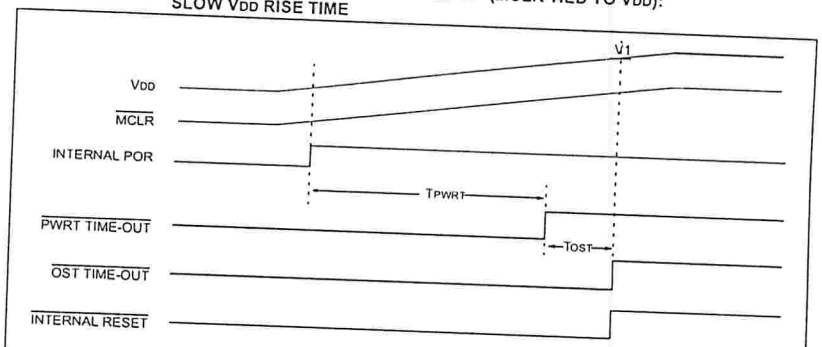


FIGURE 6-9: TIME-OUT SEQUENCE ON POWER-UP (MCLR TIED TO VDD): SLOW VDD RISE TIME



When VDD rises very slowly, it is possible that the TPWRT time-out and TOST time-out will expire before VDD has reached its final value. In this example, the chip will reset properly if, and only if, V1 ≥ VDD min.

6.7 Time-out Sequence and Power-down Status Bits (TO/PD)

On power-up (Figures 6-6 through 6-9), the time-out sequence is as follows:

1. PWRT time-out is invoked after a POR has expired.
2. Then, the OST is activated.

The total time-out will vary based on oscillator configuration and PWRTE configuration bit status. For example, in RC mode with the PWRT disabled, there will be no time-out at all.

TABLE 6-5: TIME-OUT IN VARIOUS SITUATIONS

Oscillator Configuration	Power-up		Wake-up from SLEEP
	PWRT Enabled	PWRT Disabled	
XT, HS, LP	72 ms + 1024Tosc	1024Tosc	1024Tosc
RC	72 ms	—	—

Since the time-outs occur from the POR pulse, if MCLR is kept low long enough, the time-outs will expire. Then bringing MCLR high, execution will begin immediately (Figure 6-6). This is useful for testing purposes or to synchronize more than one PIC16F84A device when operating in parallel.

Table 6-6 shows the significance of the TO and PD bits. Table 6-3 lists the RESET conditions for some special registers, while Table 6-4 lists the RESET conditions for all the registers.

TABLE 6-6: STATUS BITS AND THEIR SIGNIFICANCE

TO	PD	Condition
1	1	Power-on Reset
0	x	Illegal, TO is set on POR
x	0	Illegal, PD is set on POR
0	1	WDT Reset (during normal operation)
0	0	WDT Wake-up
1	1	MCLR during normal operation
1	0	MCLR during SLEEP or interrupt wake-up from SLEEP

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.

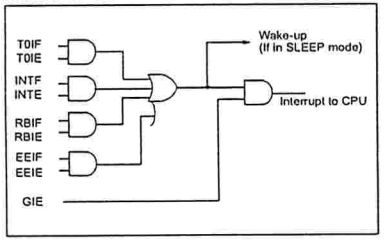
The "return from interrupt" instruction, RETFIE, exits interrupt routine as well as sets the GIE bit, which re-enables interrupts.

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.

FIGURE 6-10: INTERRUPT LOGIC



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 TCY 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).

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.

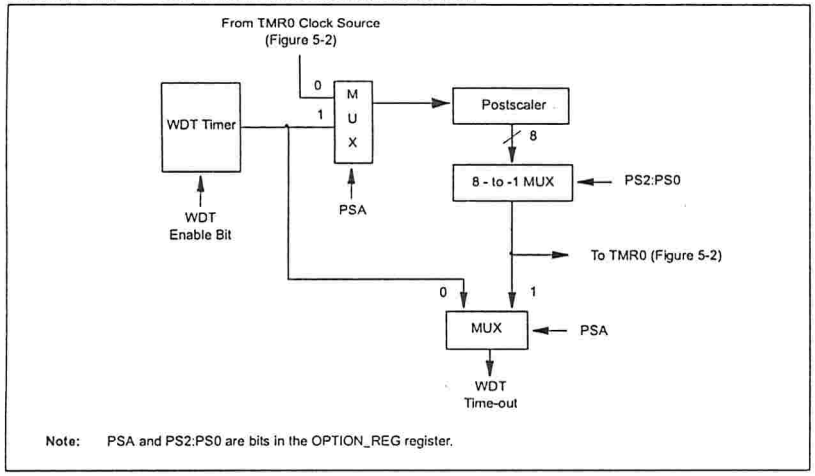
The CLRWDT and SLEEP instructions clear the WDT and the postscaler (if assigned to the WDT) and prevent it from timing out and generating a device RESET condition.

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

6.10.2 WDT PROGRAMMING CONSIDERATIONS

It should also be taken into account that under worst case conditions (V_{DD} = Min., Temperature = Max., Max. WDT Prescaler), it may take several seconds before a WDT time-out occurs.

FIGURE 6-11: WATCHDOG TIMER BLOCK DIAGRAM



Note: PSA and PS2:PS0 are bits in the OPTION_REG register.

TABLE 6-7: SUMMARY OF REGISTERS ASSOCIATED WITH THE WATCHDOG TIMER

Addr	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
2007h	Config. bits	(2)	(2)	(2)	(2)	$\overline{PWRTE}$(1)	WDTE	FOSC1	FOSC0	(2)	
81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown. Shaded cells are not used by the WDT.

Note 1: See Register 6-1 for operation of the $\overline{PWRTE}$ bit.

2: See Register 6-1 and Section 6.12 for operation of the code and data protection bits.

6.11 Power-down Mode (SLEEP)

A device may be powered down (SLEEP) and later powered up (wake-up from SLEEP).

6.11.1 SLEEP

The Power-down mode is entered by executing the SLEEP instruction.

If enabled, the Watchdog Timer is cleared (but keeps running), the $\overline{PD}$ bit (STATUS<3>) is cleared, the $\overline{TO}$ bit (STATUS<4>) 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 the lowest current consumption in SLEEP mode, place all I/O pins at either V_{DD} or V_{SS}, with no external circuitry drawing current from the I/O pins, and disable external clocks. I/O pins that are hi-impedance inputs should be pulled high or low externally to avoid switching currents caused by floating inputs. The T0CKI input should also be at V_{DD} or V_{SS}. The contribution from on-chip pull-ups on PORTB should be considered.

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

It should be noted that a $\overline{RESET}$ generated by a WDT time-out does not drive the $\overline{MCLR}$ pin low.

6.11.2 WAKE-UP FROM SLEEP

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

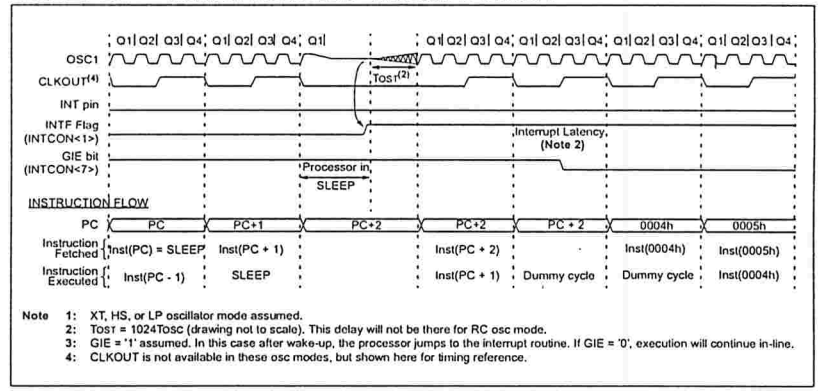
1. External RESET input on $\overline{MCLR}$ pin.
2. WDT wake-up (if WDT was enabled).
3. Interrupt from RB0/INT pin, RB port change, or data EEPROM write complete.

Peripherals cannot generate interrupts during SLEEP, since no on-chip Q clocks are present.

The first event ($\overline{MCLR}$ Reset) will cause a device RESET. The two latter events are considered a continuation of program execution. The $\overline{TO}$ and $\overline{PD}$ bits can be used to determine the cause of a 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).

While the SLEEP instruction is being executed, the next instruction (PC + 1) is pre-fetched. For the device to wake-up through an interrupt event, the corresponding interrupt enable bit must be set (enabled). Wake-up occurs 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 (0004h). In cases where the execution of the instruction following SLEEP is not desirable, the user should have a NOP after the SLEEP instruction.

FIGURE 6-12: WAKE-UP FROM SLEEP THROUGH INTERRUPT



Note 1: XT, HS, or LP oscillator mode assumed.

2: TOST = 1024TOSC (drawing not to scale). This delay will not be there for RC osc mode.

3: GIE = '1' assumed. In this case after wake-up, the processor jumps to the interrupt routine. If GIE = '0', execution will continue in-line.

4: CLKOUT is not available in these osc modes, but shown here for timing reference.

6.11.3 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 the interrupt 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 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.

6.12 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.

6.13 ID Locations

Four memory locations (2000h - 2004h) are designated as ID locations to store checksum or other code identification numbers. These locations are not accessible during normal execution but are readable and writable only during program/verify. Only the four Least Significant bits of ID location are usable.

6.14 In-Circuit Serial Programming

PIC16F84A microcontrollers can be serially programmed while in the end application circuit. This is simply done with two lines for clock and data, and three other lines for power, ground, and the programming voltage. Customers can manufacture boards with unprogrammed devices, and then program the microcontroller just before shipping the product, allowing the most recent firmware or custom firmware to be programmed.

For complete details of Serial Programming, please refer to the In-Circuit Serial Programming™ (ICSP™) Guide, (DS30277).

NOTES:

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 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.

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

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

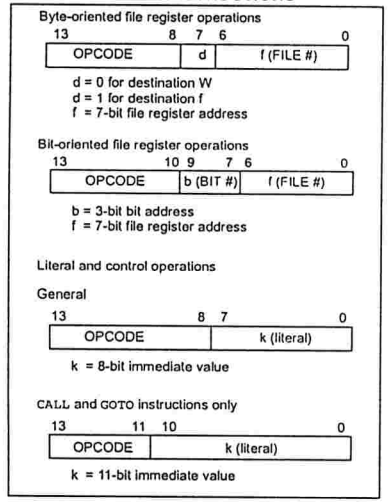
Note: To maintain upward compatibility with future PIC16CXX products, do not use the OPTION and TRIS instructions.

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).

TABLE 7-2: PIC16CXXX INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb	LSb	Status Affected	Notes
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 ≤ k ≤ 255
Operation:	(W) + k → (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.

ADDWF	Add W and f
Syntax:	[*label*] ADDWF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(W) + (f) → (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'.

ANDLW	AND Literal with W
Syntax:	[*label*] ANDLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) .AND. (k) → (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.

ANDWF	AND W with f
Syntax:	[*label*] ANDWF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(W) .AND. (f) → (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'.

BCF	Bit Clear f
Syntax:	[*label*] BCF f,b
Operands:	0 ≤ f ≤ 127 0 ≤ b ≤ 7
Operation:	0 → (f<b>)
Status Affected:	None
Description:	Bit 'b' in register 'f' is cleared.

BSF	Bit Set f
Syntax:	[*label*] BSF f,b
Operands:	0 ≤ f ≤ 127 0 ≤ b ≤ 7
Operation:	1 → (f<b>)
Status Affected:	None
Description:	Bit 'b' in register 'f' is set.

BTFSS	Bit Test f, Skip if Set
Syntax:	[*label*] BTFSS f,b
Operands:	0 ≤ f ≤ 127 0 ≤ 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.

BTFSC	Bit Test, Skip if Clear
Syntax:	[*label*] BTFSC f,b
Operands:	0 ≤ f ≤ 127 0 ≤ b ≤ 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.

CALL	Call Subroutine
Syntax:	[*label*] CALL k
Operands:	0 ≤ k ≤ 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.

CLRF	Clear f
Syntax:	[*label*] CLRF f
Operands:	0 ≤ f ≤ 127
Operation:	00h → (f) 1 → Z
Status Affected:	Z
Description:	The contents of register 'f' are cleared and the Z bit is set.

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.

CLRWDT	Clear Watchdog Timer
Syntax:	[*label*] CLRWDT
Operands:	None
Operation:	00h → WDT 0 → $\overline{\text{WDT}}$ prescaler, 1 → $\overline{\text{TO}}$ 1 → $\overline{\text{PD}}$
Status Affected:	$\overline{\text{TO}}$, $\overline{\text{PD}}$
Description:	CLRWDT instruction resets the Watchdog Timer. It also resets the prescaler of the WDT. Status bits $\overline{\text{TO}}$ and $\overline{\text{PD}}$ are set.

COMF	Complement f
Syntax:	[*label*] COMF f,d
Operands:	0 ≤ f ≤ 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'.

DECF	Decrement f
Syntax:	[*label*] DECF f,d
Operands:	0 ≤ f ≤ 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'.

DECFSZ	Decrement f, Skip If 0
Syntax:	[*label*] DECFSZ f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) - 1 → (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.

GOTO	Unconditional Branch
Syntax:	[*label*] GOTO k
Operands:	0 ≤ k ≤ 2047
Operation:	k → PC<10:0> PCLATH<4:3> → 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.

INCF	Increment f
Syntax:	[*label*] INCF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) + 1 → (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'.

INCFSZ	Increment f, Skip If 0
Syntax:	[*label*] INCFSZ f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) + 1 → (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.

IORLW	Inclusive OR Literal with W
Syntax:	[*label*] IORLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) .OR. k → (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.

IORWF	Inclusive OR W with f
Syntax:	[*label*] IORWF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(W) .OR. (f) → (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'.

MOVF	Move f
Syntax:	[*label*] MOVF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) → (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.

MOVLW	Move Literal to W
Syntax:	[*label*] MOVLW k
Operands:	0 ≤ k ≤ 255
Operation:	k → (W)
Status Affected:	None
Description:	The eight-bit literal 'k' is loaded into W register. The don't cares will assemble as 0's.

MOVWF	Move W to f
Syntax:	[*label*] MOVWF f
Operands:	0 ≤ f ≤ 127
Operation:	(W) → (f)
Status Affected:	None
Description:	Move data from W register to register 'f'.

NOP	No Operation
Syntax:	[*label*] NOP
Operands:	None
Operation:	No operation
Status Affected:	None
Description:	No operation.

RETFIE	Return from Interrupt
Syntax:	[*label*] RETFIE
Operands:	None
Operation:	TOS → PC, 1 → GIE
Status Affected:	None

RETLW	Return with Literal in W
Syntax:	[*label*] RETLW k
Operands:	0 ≤ k ≤ 255
Operation:	k → (W); TOS → 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.

RETURN	Return from Subroutine
Syntax:	[*label*] RETURN
Operands:	None
Operation:	TOS → 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.

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

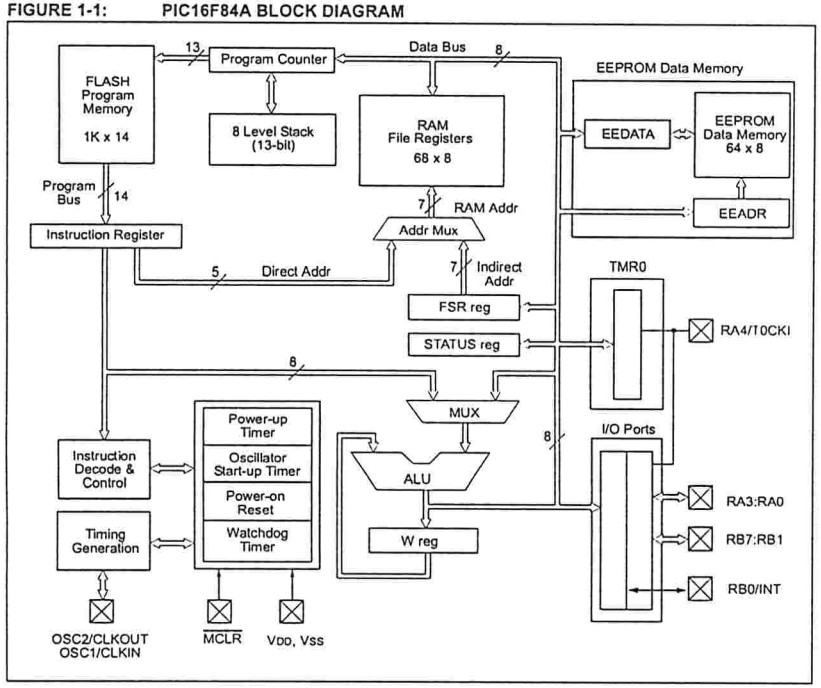


TABLE 1-1: PIC16F84A PINOUT DESCRIPTION

Pin Name	PDIP No.	SOIC No.	SSOP No.	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	16	16	18	I	ST/CMOS(3)	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.
$\overline{MCLR}$	4	4	4	I/P	ST	Master Clear (Reset) input/programming voltage input. This pin is an active low RESET to the device.
						PORTA is a bi-directional I/O port.
RA0	17	17	19	I/O	TTL	
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	Can also be selected to be the clock input to the TMR0 timer/counter. Output is open drain type.
						PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	6	6	7	I/O	TTL/ST(1)	RB0/INT can also be selected as an external interrupt pin.
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	Interrupt-on-change pin.
RB5	11	11	12	I/O	TTL	Interrupt-on-change pin.
RB6	12	12	13	I/O	TTL/ST(2)	Interrupt-on-change pin. Serial programming clock.
RB7	13	13	14	I/O	TTL/ST(2)	Interrupt-on-change pin. Serial programming data.
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.

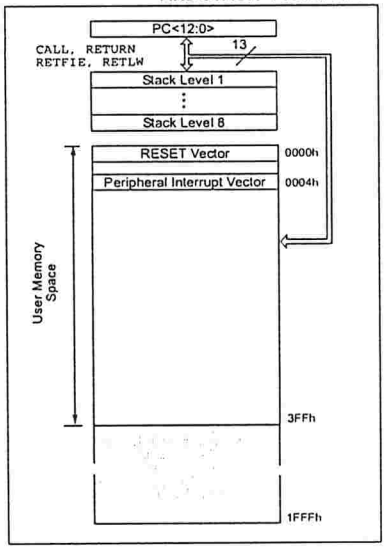
Additional information on device memory may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023).

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



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

File Address	Bank 0	Bank 1	File Address
00h	Indirect addr.(1)	Indirect addr.(1)	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	—	—	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2(1)	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	68 General Purpose Registers (SRAM)	Mapped (accesses) in Bank 0	8Ch
4Fh			CFh
50h			D0h
7Fh			FFh

☐ Unimplemented data memory location, read as '0'.

Note 1: Not a physical register.

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(2)	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(4)	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	16
06h	PORTB(5)	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(1)					---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	RBPU	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 (2)	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(1)					---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 $\overline{MCLR}$ Reset.

3: Other (non power-up) RESETS include: external RESET through $\overline{MCLR}$ 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.

ERRATA

1. **Module: CPU (STATUS bit)**
   The operation of the power-down ($\overline{PD}$) bit in the STATUS register may not function correctly for temperatures below - 20 °C.
   Work Around
   None

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 $\overline{PD}$ 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 $\overline{borrow}$ and digit $\overline{borrow}$ 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	$\overline{TO}$	$\overline{PD}$	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 **$\overline{PD}$:** 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/$\overline{borrow}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions) (for $\overline{borrow}$, 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/$\overline{borrow}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions) (for $\overline{borrow}$, 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	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$:** 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

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	T0IE	INTE	RBIE	T0IF	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 **T0IE:** 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 **T0IF:** 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).

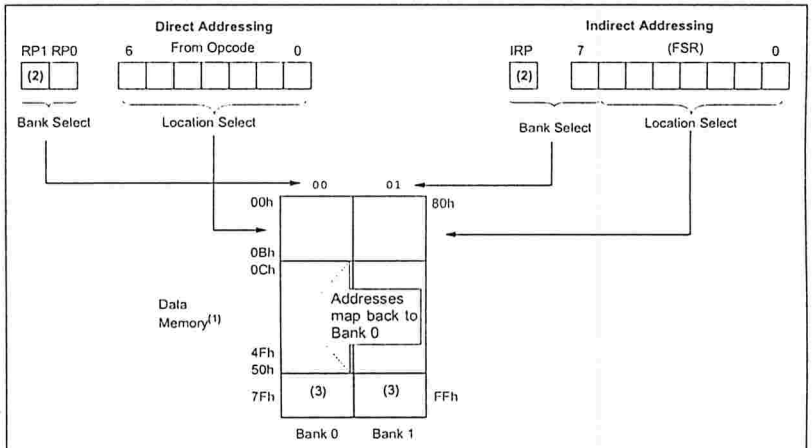
A simple program to clear RAM locations 20h-2Fh using indirect addressing is shown in Example 2-2.

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.

FIGURE 2-3: DIRECT/INDIRECT ADDRESSING



Note 1: For memory map detail, see Figure 2-2.
2: Maintain as clear for upward compatibility with future products.
3: Not implemented.

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

ERRATA

Module: Data EEPROM

Do not perform a modify (set or clear a bit) of the EECON1 register one instruction cycle after an EEPROM read. This will corrupt the EEDATA register.

Example:

```
BSF   EECON1, RD
BCF   EECON1, WREN
```

Work Around

Use either of the following two code segment in place of the above example.

```
BSF   EECON1, RD
NOP
BCF   EECON1, WREN
```

or

```
BCF   EECON1, WREN
BSF   EECON1, RD
```


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

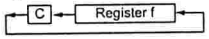
```
        BCF   STATUS,RP0 ; Bank 0
        :                ; Any code
        :                ; can go here
        MOVF  EEDATA,W   ; Must be in Bank 0
        BSF   STATUS,RP0 ; Bank 1
READ
        BSF   EECON1, RD ; YES, Read the
                         ; value written
        BCF   STATUS,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

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

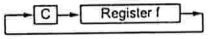
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other RESETS
08h	EEDATA	EEPROM Data Register								xxxx xxxx	uuuu uuuu
09h	EEADR	EEPROM Address Register								xxxx xxxx	uuuu uuuu
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2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

RLF	Rotate Left f through Carry
Syntax:	[*label*] RLF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [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'.



RRF	Rotate Right f through Carry
Syntax:	[*label*] RRF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [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'.



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

SUBLW	Subtract W from Literal
Syntax:	[*label*] SUBLW k
Operands:	0 ≤ k ≤ 255
Operation:	k - (W) → (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.

SUBWF	Subtract W from f
Syntax:	[*label*] SUBWF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) - (W) → (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'.

SWAPF	Swap Nibbles in f
Syntax:	[*label*] SWAPF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f<3:0>) → (destination<7:4>), (f<7:4>) → (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'.

XORLW	Exclusive OR Literal with W
Syntax:	[*label*] XORLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) .XOR. k → (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.

XORWF	Exclusive OR W with f
Syntax:	[*label*] XORWF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(W) .XOR. (f) → (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'.

8.0 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

8.1 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.

8.2 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.

8.3 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.

8.4 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.

8.5 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.

8.6 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.

8.7 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.

8.8 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.

8.9 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 VDD and VPP supplies, which allow it to verify programmed memory at VDD min and VDD 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.

8.10 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.

8.11 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.

8.12 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 I2CTM bus and separate headers for connection to an LCD module and a keypad.

8.13 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.

8.14 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.

8.15 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 8-1: 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:

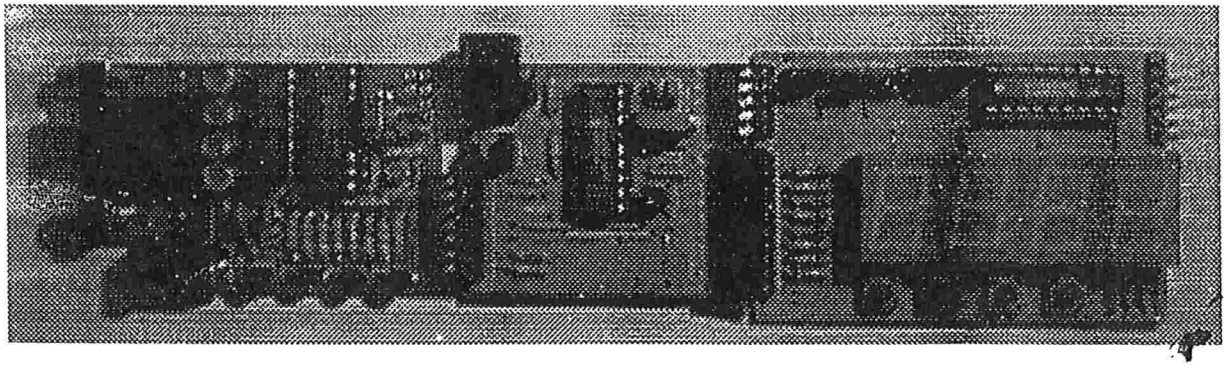
INDEX

NOTES:

FLYPIC!
Gravador e Sistema de desenvolvimento em microcontroladores PIC

Manual de Utilização - Kit Didático de Ensino

Programa de Ensino à Distância



Prof. Elmo Dutra da Silveira Filho, Msc
Centro Tecnológico de Mecatrônica - SENAI - RS - LMM - DEMEC - UFRGS

Miguel dos Santos
Técnico em Informática Industrial

FLYPIC – KIT DIDÁTICO DE ENSINO E SISTEMA DE DESENVOLVIMENTO EM PICs
PROF ELMO DUTRA – CENTRO TECNOLÓGICO DE MECATRÔNICA – SENAI – RS

O FLYPIC foi desenvolvido com base em 16 anos de experiência na área de microcontroladores – ensino e projetos. A linha de chips da Microchip tem sido largamente utilizada pela sua versatilidade – você encontra em versões de 6 a 33 I/Os, com diversos periféricos, tudo em um mesmo chip.

Este kit didático atualmente é composto de 7 módulos:

Gravador de PICs: Possui conector para ligação ao PC para transferência de arquivos **.HEX.** Permite a gravação de PICs da linha básica e média de protocolo ICSP (In Circuit Serial Programing) – atualmente 75 tipos de PICs – conforme lista do fabricante.

Placa soquete 18 pinos: esta placa permite o trabalho com o chip mais utilizado do momento: o **16F84A** (flash, 13 I/Os). Tem alimentação e clock independente, e possui saída lateral para as 13 I/Os. Para a gravação de outros PICs basta trocar o soquete.

Placa de aplicações I: para estudo e exercícios básicos em PICs, possui 8 leds e teclado 3X4 telefônico (Portb), 2 teclas, conector para display LCD e buzzer.

Placa de aplicações II: também desenvolvida para aplicações e estudos, possui 4 teclas e 4 displays de 7 segmentos ligados multiplexados – o objetivo é implementar contadores e um relógio digital ou timer programável.

Soquetes: 8, 28 e 40 pinos: estas plaquetas apenas servem de suporte na gravação de outros tamanhos de PICs que o fabricante dispõe – veja no site do fabricante: www.microchip.com.

Os esquemáticos constam neste manual. Foram incluídas dezenas de rotinas de exemplos que podem ser modificadas e implementadas no kit com as devidas modificações de hardware e sotware. O pacote inclui:

Disco1: disquete que contém versão do compilador MPASM e simulador MPSIM do fabricante Microchip, além de software de transferência executável FLYPIC (porta COM-2) e FLYCOM1 (porta COM-1). Estes softwares rodam no modo DOS. Também foram incluídas diversas aplicações e exemplos para estudo e modificação.

Disco2: novo disco com curso em HTML de PICs, nova versão do MPASM e dezenas de novas aplicações e exemplos para seu kit didático de ensino.

Para descompactar utilize o software Pkunzip.exe (incluso) ou ainda melhor o Winzip (o curso em HTML – excelente – deve ser descompactado com winzip – com os subdiretórios restaurados.

PROCEDIMENTO DE INSTALAÇÃO E TESTES INICIAIS DO FLYPIC:

Ligue o cabo de comunicação ao Porto Serial do PC (DB-25). Se seu computador tiver saída do tipo DB-9 adquira uma adaptador DB-9 / DB-25. Certifique-se que a COM do seu micro está habilitada e sem conflitos. Para utilizar a COM-2 use o software executável FLYPIC. EXE de transferência do arquivo .HEX do compilador. Se utilizar a COM-1 use o software de transferência FLYCOM1.EXE.

Ligue o cabo de comunicação ao gravador, o gravador à placa soquete 18 pinos com o PIC 16F84ª . A alimentação pode ser tanto pela placa gravadora quanto pela placa soquete. Utilize uma fonte de alimentação com tensão de **12 a 15 Volts – conector positivo do meio. Não alimentar com tensão menor – não grava!**

Ligue a placa soquete a placa de 4 displays 7 segmentos (aplicações II). Como a placa soquete veio já gravada com um software demonstração, irá aparecer nos displays um contador.

Você pode ligar a placa soquete a placa de aplicações I (leds) e testar com o seguinte exemplo (estando já no diretório do disquete 1):

flypic seqxt.hex

O software para DOS aparece. Você pode então apagar o PIC gravado instantaneamente (comando E – erase), e gravar por cima (comando P – programação). Irá aparecer nos leds um seqüencial.

Outro exemplo:

Digite S (sair) e no prompt digite:

flypic ledxt.hex

Você entra novamente no menu do software e regrava por cima um novo programa: led pisca pisca. O apagamento é instantâneo e a gravação leva alguns segundos (10 ms por cada byte gravado, mais o check). Programas muito longos podem demorar minutos!

Seu kit foi amplamente testado antes de ser enviado. Oferecemos garantia de funcionamento, se seu micro estiver corretamente configurado e com a COM funcionando. A garantia cobre somente uso **NORMAL** do kit, não cobre sobrecargas ou uso indevido fora dos parâmetros do fabricante (consulte datasheet).

A garantia não cobre sobrecargas de corrente das portas de I/Os ou uso indevido. Alguns periféricos "carregam" a porta na gravação (exemplo: display LCD – não deve ser gravado na placa –pode sobrecarregar os portos do PIC. Você grava sem o display e após a gravação coloca o display).

A garantia também não cobre problemas como transientes (excesso de tensão, picos, raios, espúrios). Todo o microcontrolador é sensível a ruído elétrico.

A garantia cobre defeitos normais do equipamento e o reparo é feito mediante o envio do kit para conserto. Os cursos de transporte correm por parte do usuário.

Não esquecer que cada tipo de PIC requer configuração própria!!!

LEIA O MANUAL ATENTAMENTE !!!!

Estude as rotinas e exemplos, faça as devidas adaptações que forem necessárias para o devido funcionamento no kit. Não se esqueça: o fabricante possui dezenas de chips de diversas capacidades de memória e periféricos embutidos.

NÃO SE ESQUEÇA DE COLOCAR A DIRETIVA CONFIG NO SEU PROGRAMA .ASM – O FLYPIC NÃO FUNCIONA SEM O CONFIG PRÓPRIO – VEJA NO MANUAL – O ESQUECIMENTO DO CONFIG PODE DANIFICAR A GRAVAÇÃO DO PIC!!!!!!!!!!!!!!!

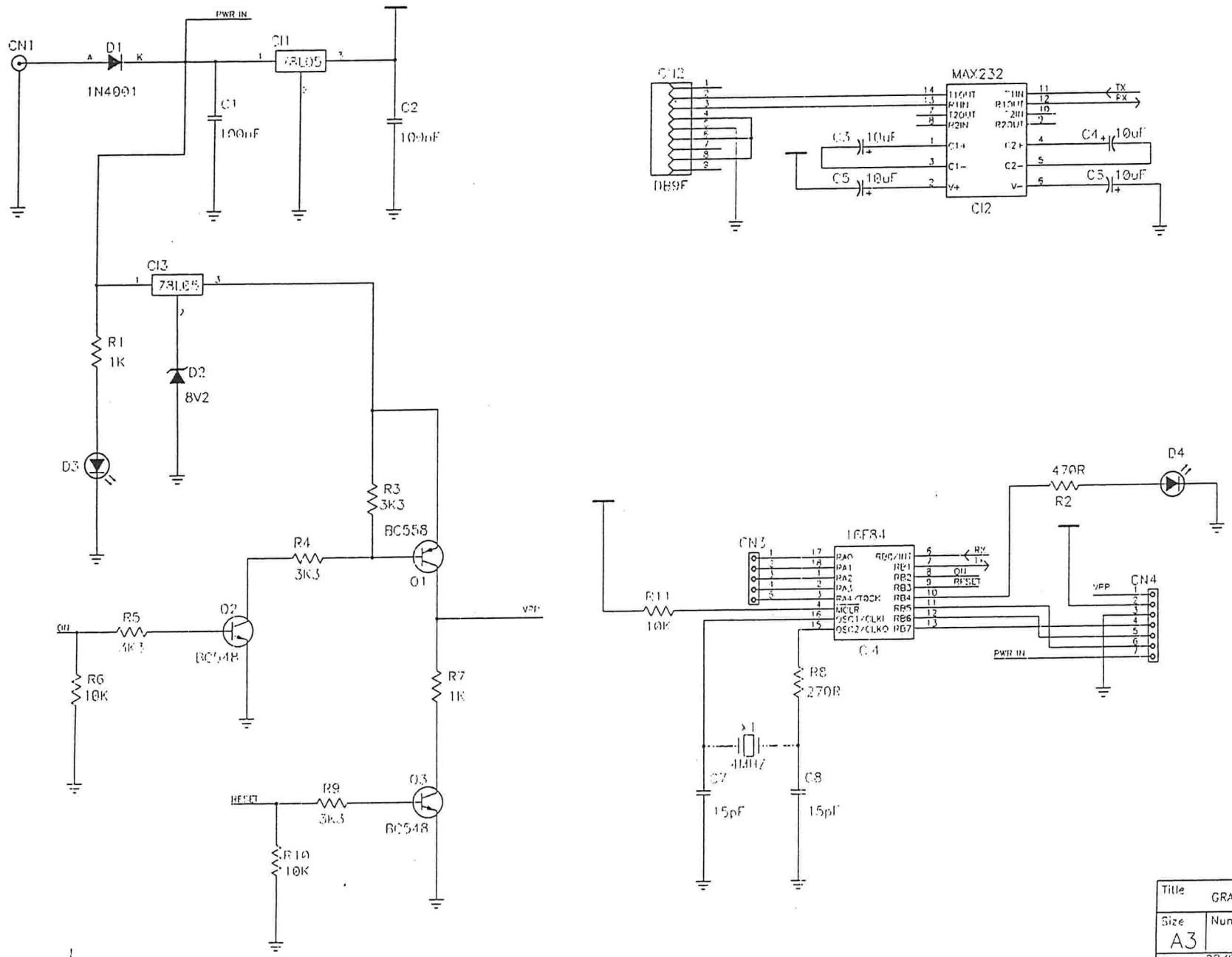
PARA DÚVIDAS E ESCLARECIMENTOS – CONSULTE POR E-MAIL:

Elmo@malbanet.com.br **(tudo minúsculo)**

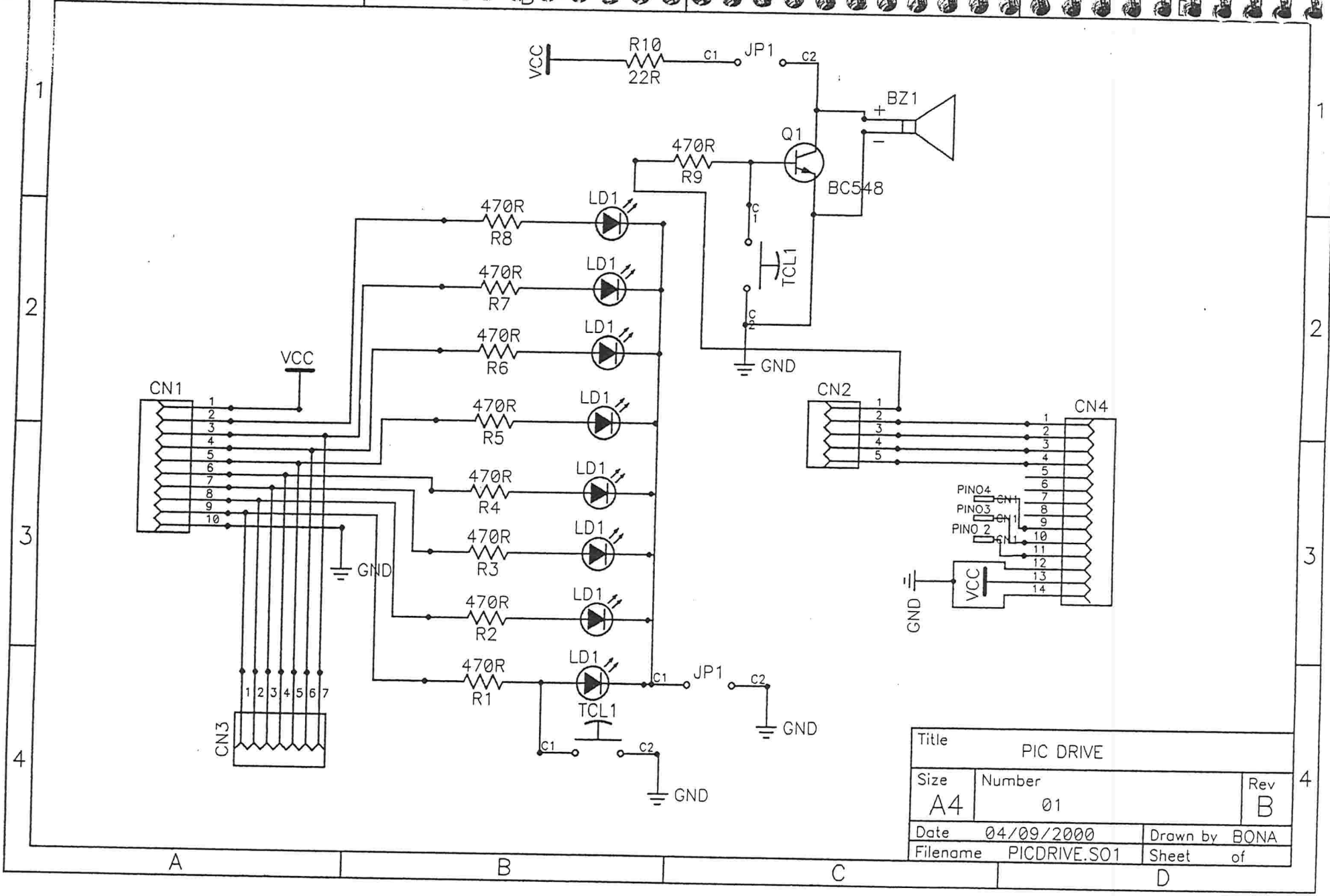
Site: www.malbanet.com.br/professorelmo **(breve novo site – portal em Automação!!!!)**

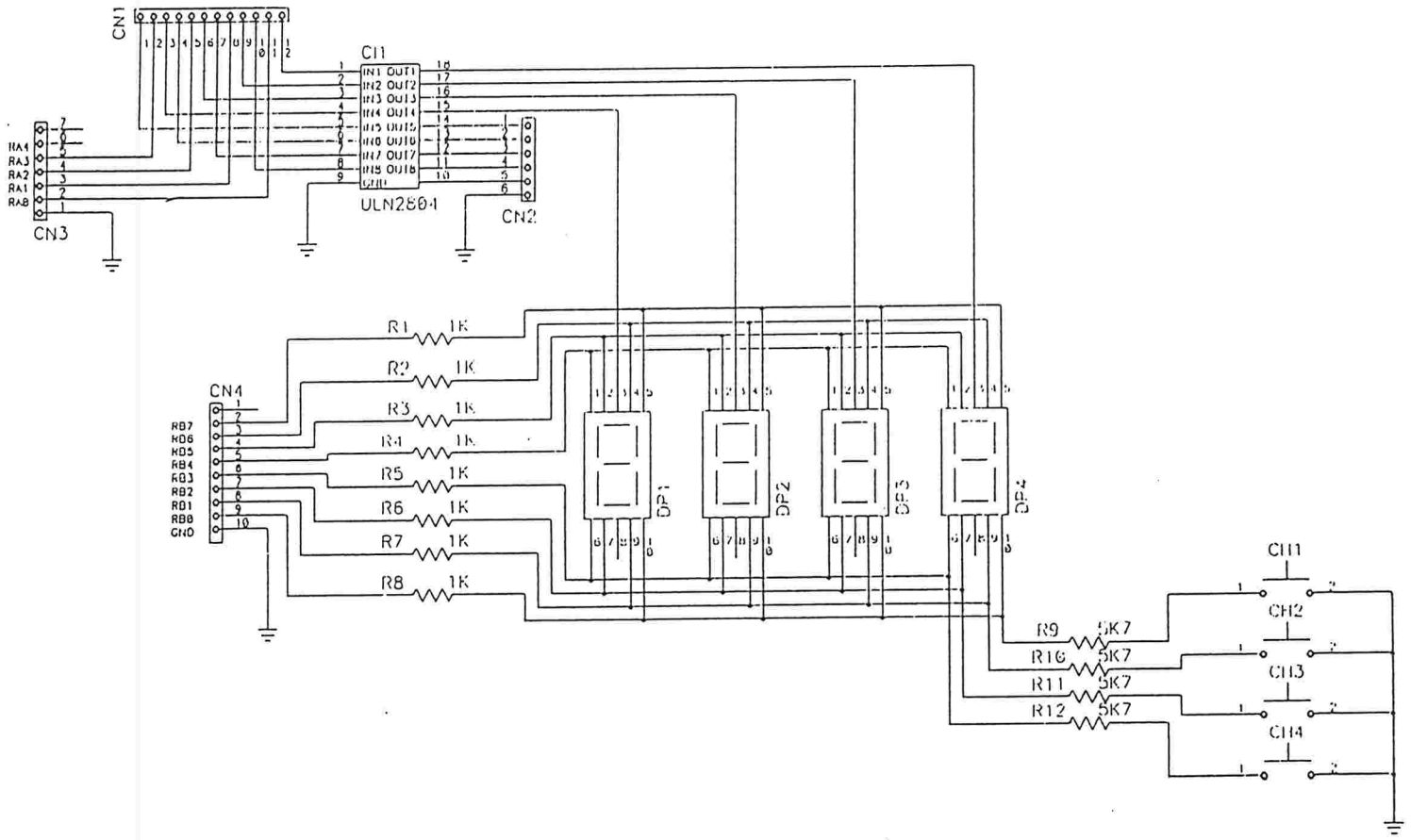
Faça um bom uso de seu kit didático – INVISTA SEU TEMPO EM CRESCIMENTO E ATUALIZAÇÃO PROFISSIONAL! Fico à disposição – Um grande abraço

PROF ELMO DUTRA, Msc – Centro Tecnológico de Mecatrônica – SENAI - RS



Title	GRAVADOR DE PIC
Size	A3
Number	
Rev	01
Date	28/03/99
Drawn by	MIGUEL
Filename	PICG02.S01
Sheet	1 of 1


VCC
R10
22R
C1
JP1
C2
BZ1
+
-
Q1
BC548
470R
R9
470R
R8
470R
R7
470R
R6
470R
R5
470R
R4
470R
R3
470R
R2
470R
R1
LD1
C1
TCL1
C2
GND
CN1
1
2
3
4
5
6
7
8
9
10
CN2
CN4
PIN04
PIN03
PIN02
CN1
11
12
13
14
CN3
1 2 3 4 5 6 7
Title
PIC DRIVE
Size
A4
Number
01
Rev
B
Date
04/09/2000
Drawn by
BONA
Filename
PICDRIVE.SO1
Sheet
of
A
B
C
D



Title		
Size A3	Number	Rev
Date		Drawn by
Filename		Sheet of

```
;*******************************************************************************
; Programa: sequencial de led no Portb
; arquivo: seqxt.asm
;*******************************************************************************
            list p=16f84
;
            include "P16F84.INC"
;
            __config _CP_OFF & _PWRTE_ON & _WDT_OFF & _XT_OSC
;*******************************************************************************
;           area de declaracao de equ
;*******************************************************************************
led         equ     0               ;determina saida para o led
;*******************************************************************************
;           vetor do reset
;*******************************************************************************
            org 0h                  ;endereco do vetor do reset
            goto start              ;vai para o inicio do programa
;*******************************************************************************
;           Area de subrotinas
;*******************************************************************************
;           delay (utiliza TMR0 sem habilitar interrupcao)
;*******************************************************************************
delay:      clrf TMR0               ;zera TMR0
            movlw 0ffh              ;valor maximo do TMR0
            xorwf TMR0,W            ;compara W com TMR0
            btfss STATUS,Z          ;se igual finaliza funcao
            goto $-3                ;senao repete o ciclo
            return
;*******************************************************************************
;           segmento de inicializacao
;*******************************************************************************
start:      bsf STATUS,RP0          ;ativa Bank1
            movlw 0xc7              ;configura o registro option habilitando
            movwf OPTION_REG        ;timer0 com prescaler 1:256
            clrf TRISB              ;configura PORTB como saida (todos os bits)
            bcf STATUS,RP0          ;ativa Bank0
;*******************************************************************************
;           segmento principal
;*******************************************************************************
            movlw 0x01
            movwf PORTB
            bcf STATUS,C            ;zera carry
loop:       call delay
            call delay
            rlf PORTB,f
            goto loop               ;repete o ciclo
;*******************************************************************************
            end
;*******************************************************************************
```

```
;*******************************************************************************
; Programa: pisca led no Portb
; arquivo: ledxt.asm
;*******************************************************************************
            list p=16f84
;
            include "P16F84.INC"
;
            __config _CP_OFF & _PWRTE_ON & _WDT_OFF & _XT_OSC
;*******************************************************************************
;           area de declaracao de equ
;*******************************************************************************
led1        equ     0           ;determina saida para o led
led2        equ     7
;*******************************************************************************
;           vetor do reset
;*******************************************************************************
            org 0h              ;endereco do vetor do reset
            goto start          ;vai para o inicio do programa
;*******************************************************************************
;           Area de subrotinas
;*******************************************************************************
;           delay (utiliza TMR0 sem habilitar interrupcao)
;*******************************************************************************
delay:      clrf TMR0           ;zera TMR0
            movlw 0ffh          ;valor maximo do TMR0
            xorwf TMR0,W        ;compara W com TMR0
            btfss STATUS,Z      ;se igual finaliza funcao
            goto $-3            ;senao repete o ciclo
            return
;*******************************************************************************
;           segmento de inicializacao
;*******************************************************************************
start:      bsf STATUS,RP0      ;ativa Bank1
            movlw 0xc7          ;configura o registro option habilitando
            movwf OPTION_REG    ;timer0 com prescaler 1:256
            clrf TRISB          ;configura PORTB como saida (todos os bits)
            bcf STATUS,RP0      ;ativa Bank0
;*******************************************************************************
;           segmento principal
;*******************************************************************************
            movlw 0             ;inicializa PORTB
            movwf PORTB
loop:       bsf PORTB,led1
            bcf PORTB,led2
            call delay
            call delay
            call delay
            bcf PORTB,led1
            bsf PORTB,led2
            call delay
            call delay
            btfss PORTA,0
            goto loop           ;repete o ciclo
            btfsc PORTA,0
            goto $ - 1
            goto loop
;*******************************************************************************
            end
;*******************************************************************************
```

ANEXOS

- "basic code template for assembly code generation on the PICmicro PIC16F84A"

- "header file defines configurations, registers, and other useful bits of information for the PIC16F84"

- Primeiro lab.: programa elementar de exemplo (prog0.asm)

- Segundo lab.: Cronômetro de 1 seg na PORTB do FLYPIC! utilizando timer e interrupção (prog1.asm)

- Exemplo de complexidade alta de uso do PIC:
  "FILE : LCD_DIGI.ASM
  CONTENTS : Simple low-cost 7-digit digital Scale using a PIC16F84"

- Links para site do curso e afins

```
;*****************************************************************
;   This file is a basic code template for assembly code generation   *
;   on the PICmicro PIC16F84A. This file contains the basic code      *
;   building blocks to build upon.                                    *
;                                                                     *
;   If interrupts are not used all code presented between the ORG     *
;   0x004 directive and the label main can be removed. In addition    *
;   the variable assignments for 'w_temp' and 'status_temp' can       *
;   be removed.                                                       *
;                                                                     *
;   Refer to the MPASM User's Guide for additional information on     *
;   features of the assembler (Document DS33014).                     *
;                                                                     *
;   Refer to the respective PICmicro data sheet for additional        *
;   information on the instruction set.                               *
;                                                                     *
;   Template file assembled with MPLAB V4.00.00 and MPASM V2.20.12.   *
;                                                                     *
;*****************************************************************
;                                                                     *
;    Filename:        xxx.asm                                         *
;    Date:                                                           *
;    File Version:                                                   *
;                                                                     *
;    Author:                                                         *
;    Company:                                                        *
;                                                                     *
;                                                                     *
;*****************************************************************
;                                                                     *
;    Files required:                                                 *
;                                                                     *
;                                                                     *
;                                                                     *
;*****************************************************************
;                                                                     *
;    Notes:                                                          *
;                                                                     *
;                                                                     *
;                                                                     *
;                                                                     *
;*****************************************************************


      list      p=16F84A            ; list directive to define processor
      #include <p16F84A.inc>        ; processor specific variable definitions

      __CONFIG    _CP_OFF & _WDT_ON & _PWRTE_ON & _RC_OSC

; '__CONFIG' directive is used to embed configuration data within .asm file.
; The lables following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.



;***** VARIABLE DEFINITIONS
w_temp        EQU     0x0C        ; variable used for context saving
status_temp   EQU     0x0D        ; variable used for context saving
```

```
;*****************************************************************
            ORG     0x000             ; processor reset vector
            goto    main              ; go to beginning of program


            ORG     0x004             ; interrupt vector location
            movwf   w_temp            ; save off current W register contents
            movf  STATUS,w            ; move status register into W register
            movwf status_temp         ; save off contents of STATUS register

; isr code can go here or be located as a call subroutine elsewhere


            movf    status_temp,w     ; retrieve copy of STATUS register
            movwf STATUS              ; restore pre-isr STATUS register contents
            swapf   w_temp,f
            swapf   w_temp,w          ; restore pre-isr W register contents
            retfie                    ; return from interrupt


main

; remaining code goes here




            END                       ; directive 'end of program'
```

```
        LIST
; P16F84A.INC  Standard Header File, Version 2.00    Microchip Technology, Inc.
        NOLIST

; This header file defines configurations, registers, and other useful bits of
; information for the PIC16F84 microcontroller.  These names are taken to match
; the data sheets as closely as possible.

; Note that the processor must be selected before this file is
; included.  The processor may be selected the following ways:

;       1. Command line switch:
;               C:\ MPASM MYFILE.ASM /PIC16F84A
;       2. LIST directive in the source file
;               LIST   P=PIC16F84A
;       3. Processor Type entry in the MPASM full-screen interface

;==========================================================================
;
;       Revision History
;
;==========================================================================

;Rev:   Date:    Reason:

;1.00   2/15/99 Initial Release

;==========================================================================
;
;       Verify Processor
;
;==========================================================================

        IFNDEF __16F84A
           MESSG "Processor-header file mismatch.  Verify selected processor."
        ENDIF

;==========================================================================
;
;       Register Definitions
;
;==========================================================================

W                               EQU     H'0000'
F                               EQU     H'0001'

;----- Register Files------------------------------------------------------

INDF                            EQU     H'0000'
TMR0                            EQU     H'0001'
PCL                             EQU     H'0002'
STATUS                          EQU     H'0003'
FSR                             EQU     H'0004'
PORTA                           EQU     H'0005'
PORTB                           EQU     H'0006'
EEDATA                          EQU     H'0008'
EEADR                           EQU     H'0009'
PCLATH                          EQU     H'000A'
INTCON                          EQU     H'000B'

OPTION_REG                      EQU     H'0081'
TRISA                           EQU     H'0085'
TRISB                           EQU     H'0086'
EECON1                          EQU     H'0088'
EECON2                          EQU     H'0089'
```

```
;----- STATUS Bits --------------------------------------------------

IRP                             EQU     H'0007'
RP1                             EQU     H'0006'
RP0                             EQU     H'0005'
NOT_TO                          EQU     H'0004'
NOT_PD                          EQU     H'0003'
Z                               EQU     H'0002'
DC                              EQU     H'0001'
C                               EQU     H'0000'

;----- INTCON Bits --------------------------------------------------

GIE                             EQU     H'0007'
EEIE                            EQU     H'0006'
T0IE                            EQU     H'0005'
INTE                            EQU     H'0004'
RBIE                            EQU     H'0003'
T0IF                            EQU     H'0002'
INTF                            EQU     H'0001'
RBIF                            EQU     H'0000'

;----- OPTION_REG Bits ----------------------------------------------

NOT_RBPU                        EQU     H'0007'
INTEDG                          EQU     H'0006'
T0CS                            EQU     H'0005'
T0SE                            EQU     H'0004'
PSA                             EQU     H'0003'
PS2                             EQU     H'0002'
PS1                             EQU     H'0001'
PS0                             EQU     H'0000'

;----- EECON1 Bits --------------------------------------------------

EEIF                            EQU     H'0004'
WRERR                           EQU     H'0003'
WREN                            EQU     H'0002'
WR                              EQU     H'0001'
RD                              EQU     H'0000'

;=========================================================================
;
;       RAM Definition
;
;=========================================================================

        __MAXRAM H'CF'
        __BADRAM H'07', H'50'-H'7F', H'87'

;=========================================================================
;
;       Configuration Bits
;
;=========================================================================

_CP_ON                          EQU     H'000F'
_CP_OFF                         EQU     H'3FFF'
_PWRTE_ON                       EQU     H'3FF7'
_PWRTE_OFF                      EQU     H'3FFF'
_WDT_ON                         EQU     H'3FFF'
_WDT_OFF                        EQU     H'3FFB'
_LP_OSC                         EQU     H'3FFC'
_XT_OSC                         EQU     H'3FFD'
_HS_OSC                         EQU     H'3FFE'
_RC_OSC                         EQU     H'3FFF'

        LIST
```

```
;*****************************************************************
;   This file is a basic code template for assembly code generation   *
;   on the PICmicro PIC16F84A. This file contains the basic code      *
;   building blocks to build upon.                                    *
;                                                                     *
;   If interrupts are not used all code presented between the ORG     *
;   0x004 directive and the label main can be removed. In addition    *
;   the variable assignments for 'w_temp' and 'status_temp' can       *
;   be removed.                                                       *
;                                                                     *
;   Refer to the MPASM User's Guide for additional information on     *
;   features of the assembler (Document DS33014).                     *
;                                                                     *
;   Refer to the respective PICmicro data sheet for additional        *
;   information on the instruction set.                               *
;                                                                     *
;   Template file assembled with MPLAB V4.00.00 and MPASM V2.20.12.   *
;                                                                     *
;*****************************************************************
                                                                      *
    Filename: PROG0.ASM                                               *
    Date: OUT/2004                                                    *
    File Version: 1                                                   *
                                                                      *
    Author: AAJ                                                       *
    Company: UFRGS/EE/DELET                                           *
                                                                      *
                                                                      *
;*****************************************************************
                                                                      *
    Files required:                                                   *
                                                                      *
                                                                      *
                                                                      *
;*****************************************************************
                                                                      *
    Notes: PRIMEIRO EXEMPLO DE LAB                                    *
                                                                      *
                                                                      *
                                                                      *
                                                                      *
;*****************************************************************
```

```
;-----------------------------------------------------------------
;          *** TEMPLATE FLYPIC ***
;          list p=16f84
;          include "P16F84.INC"
           __config _CP_OFF & _PWRTE_ON & _WDT_OFF & _XT_OSC
;-----------------------------------------------------------------


     list      p=16F84A              ; list directive to define processor
     #include <p16F84A.inc>          ; processor specific variable definitions

;    __CONFIG    _CP_OFF & _WDT_ON & _PWRTE_ON & _RC_OSC

; '__CONFIG' directive is used to embed configuration data within .asm file.
; The lables following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.


;***** VARIABLE DEFINITIONS
w_temp        EQU     0x0C        ; variable used for context saving
status_temp   EQU     0x0D        ; variable used for context saving


;**********************************************************************
          ORG     0x000             ; processor reset vector
          goto    main              ; go to beginning of program

;**********************************************************************
          ORG     0x004             ; interrupt vector location
;         movwf   w_temp            ; save off current W register contents
;         movf    STATUS,w          ; move status register into W register
          movwf   status_temp       ; save off contents of STATUS register
;    isr code can go here or be located as a call subroutine elsewhere

;         movf    status_temp,w     ; retrieve copy of STATUS register
          movwf   STATUS            ; restore pre-isr STATUS register contents
          swapf   w_temp,f
;         swapf   w_temp,w          ; restore pre-isr W register contents
          retfie                    ; return from interrupt

;**********************************************************************
main
; remaining code goes here


num       EQU       0x0D        ; variável a decrementar de 10 até 0

          movlw d'10'
          movwf num

de_novo
          decfsz num, f
          goto de_novo

fim
          goto fim


          END                       ; directive 'end of program'
;**********************************************************************
```

```
;*****************************************************************
;                                                                *
;    Filename: PROG1.ASM                                         *
;    Date: OUT/2004                                              *
;    File Version: 1.1                                           *
;                                                                *
;    Author: AAJ                                                 *
;    Company: UFRGS/EE/DELET                                     *
;                                                                *
;*****************************************************************
;                                                                *
;    Files required:                                             *
;                                                                *
;*****************************************************************
;                                                                *
;    Notes: PROGRAMA OTIMIZADO POR:                              *
;                                                                *
;                    Ricardo Kusiak                              *
;                    José Mariano Arigony                        *
;                    Jonathan Henrique Oliveira                  *
;                                                                *
;*****************************************************************

;-----------------------------------------------------------------
;       *** TEMPLATE FLYPIC ***
;       list p=16f84
;       include "P16F84.INC"
        __config _CP_OFF & _PWRTE_ON & _WDT_OFF & _XT_OSC
;-----------------------------------------------------------------
        list      p=16F84A            ; list directive to define processor
        #include <p16F84A.inc>        ; processor specific variable definitions
;       __CONFIG   _CP_OFF & _WDT_ON & _PWRTE_ON & _RC_OSC

; '__CONFIG' directive is used to embed configuration data within .asm file.
; The lables following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.

;       errorlevel -302

;***** VARIABLE DEFINITIONS

cont1   EQU     0x11
cont2   EQU     0x12
cont3   EQU     0x13

valor1  EQU     D'012'
valor2  EQU     D'254'
valor3  EQU     D'255'
seg     EQU     0x16

;*****************************************************************
                ORG     0x000             ; processor reset vector
                goto    main              ; go to beginning of program

;*****************************************************************
                ORG     0x004             ; interrupt vector location
                bcf     INTCON, T0IF ; para desligar o bit de overflow

        incfsz  cont1, f
                goto    fimint
                movlw   valor1
                movwf   cont1

                incfsz  cont2, f
                goto    fimint
                movlw   valor2
                movwf   cont2

                incfsz  cont3, f
                goto    fimint
                movlw   valor3
                movwf   cont3

                incf    seg, f
```

```
                movf    seg, w
                sublw   D'60'
                btfss   STATUS, Z
                goto    fimint
                movlw   d'0'
                movwf   seg

fimint
                retfie                  ; return from interrupt

;*****************************************************************
main

; remaining code goes here
                bsf             STATUS, RP0
                clrf    TRISB
; configuração do timer
                bcf     OPTION_REG, T0CS        ; escolhe função Timer, não Contador
                bcf     OPTION_REG, PSA                 ; habilita saída do prescaler

                bcf     OPTION_REG, PS2                 ; PS2, PS1, PS0: coloca 1:8 no prescaler
                bsf     OPTION_REG, PS1                 ;
                bcf     OPTION_REG, PS0                 ;

; configuração das interrupções
                bsf             INTCON, GIE                     ; habilita todas int
                bsf             INTCON, T0IE            ; habilita int pelo timer

; principal
                movlw   valor1
                movwf   cont1

                movlw   valor2
                movwf   cont2

                movlw   valor3
                movwf   cont3

                bcf             STATUS, RP0
                clrw
                movwf   TMR0

espera
                movf    seg, w
                movwf   PORTB
                goto    espera

fim
                goto    fim

                END                             ; directive 'end of program'
;*****************************************************************
```

```
; ------------------------------------------------------------------------
; FILE     : LCD_DIGI.ASM                                                 *
; CONTENTS : Simple low-cost 7-digit digital scale using a PIC16F84       *
; COPYRIGHT: Peter Halicky  OM3CPH                                        *
; AUTHOR   : Peter Halicky  OM3CPH & Peter Halicky Jr., OM2APH            *
; PCB      : Tibor Madarasz OM2ATM                                        *
;-------------------------------------------------------------------------
; E-Mail: halicky@cepoe.minv.sk or om3cph@oe3xbs.aut.eu
;         peto-h@writeme.com  or om2aph@om0pbm.svk.eu
;
; Bratislava, Slovakia, December 1998
;
;------------------------------------------------------------------------
; This is 7-digit digital scale counting up to 35 MHz. The decimal point
; is after MHz digit.
;
; It adds or substracts RF according signal level at RA2:
;              +5V - adds RF
;     disconnected - substracts RF
;
; Hardware is very simple:
;
; It contains     : PIC 16F84
;                   1 NPN low power HF Si transistor,
;                   16 character (2x8) in 1 Line LCD display,
;                   Xtal 1..10 MHz,
;                   some resistors, capacitors and 2 Si switching diodes...
;                   (see schematic in lcd_digi.pcx)
; Note:
; LCD display is 16 character in 1 line LCD display PVC160101PTN which
; seems to be compatible with TWO LINES HITACHI LCD display, exept
; that it has only 8 characters in 1 line.
;
; The counter uses internal prescaler of PIC as low byte of counter,
; TMR0 as middle byte and some register as high byte of counter.
;
; Some ideas were taken from "Simple low-cost digital frequency meter
;                             using a PIC 16C54" (frqmeter.asm)
;                             written by James Hutchby, MadLab Ltd. 1996
; LCD interfacing was taken from AN587 and mainly from LCD.ASM written
; by Peter Ouwehand.
; ------------------------------------------------------------------------
;
; This software is free for private usage. It was created for HAM radio
; community members. Commercial exploatation is allowed only with permission
; of authors.
;
; ------------------------------------------------------------------------
;
```

```
; The measuring period is 100 000 us.
; Procesor cycle is T = 4/fx [us,MHz], fx is Xtal frequency
;
; Number of procesor cycles per measuring period:
;
;       N = 100 000/T procesor cycles
;       N = fx x 100 000/4 = 25 000 x fx
;
;       The main steps of measuring period:
;
;       1. decode 3-byte value into 7 decimal numbers,
;       2. decode decimal value of digit to chars,
;       3. set decimal point if needed,
;       4. output to PORTB (LCD),
;       5. start measurement,
;       6. test TMR0 overflow bite, if YES increase TimerH,
;       7. goto 5 until measuring period is done,
;       8. stop measurement,
;       9. shift out precounter content,
;      10. Add/substract RF according signal from optocoupler,
;      11. goto 1
;
; ------------------------------------------------------------------------
;
; Total timing formula:
;
; N = 25 000 x fosc[MHz]
;
; Example:  fosc = 4 MHz
;
; N = 25 000 x 4 = 100 000
; N = 25 000 x fx = ((9*T1+4)*T2+4)*T3+1+(9+X)*T4+6
;
; ------------------------------------------------------------------------

    include <p16c84.inc>

; ------------------------------------------------------------------------

Index       equ         0Ch         ; dummy register
Count       equ         0Dh         ; inkremental register
Help        equ         0Eh         ; dummy register

LED0        equ         0Fh
LED1        equ        010h
LED2        equ        011h
LED3        equ        012h
LED4        equ        013h
LED5        equ        014h
LED6        equ        015h

LCD_TEMP    equ        016h         ; LCD subroutines internal use

TimerH      equ        017h         ; the highest byte of SW counter

LowB        equ        018h         ; low byte of resulted frequency
MidB        equ        019h         ; middle byte of resulted frequency
HigB        equ        01Ah         ; high byte of resulted frequency

TEMP        equ        01Bh         ; temporary register
HIndex      equ        01Ch         ; index register
```

```
LEDIndex    equ       01Dh          ; LED pointer

R1          equ       01Eh          ; Timing counters
R2          equ       01Fh
R3          equ       020h

; -----------------------------------------------------------------------
; LCD variables
; -----------------------------------------------------------------------

;FREQ       EQU      .4196          ; Xtal frequency in kHz
DELAY15     EQU      .19            ; (HIGH((4000/FREQ)*15000/770))+1

LINE0    equ       0
LINE1    equ       040h
                                    ; PORTA bits
E        equ         2              ; LCD Enable control line
R_W      equ         1              ; LCD Read/Write control line
RS       equ         0              ; LCD Register-Select control line

; -----------------------------------------------------------------------
                                    ; timing loop values
                                    ; must be from 1 to 255!!!
T1          equ       .255          ; first timing loop
T2          equ       .9            ; second timing loop
T3          equ       .5            ; third timing loop
T4          equ       .149          ; last timing loop
                                    ; values for 4 194 kHz Xtal
; -----------------------------------------------------------------------
            org        0

Start
            clrf       STATUS          ; Do initialization, Select bank 0
         clrf       INTCON           ; Clear int-flags, Disable interrupts
         clrf       PCLATH           ; Keep in lower 2KByte
         clrf       PORTA            ; ALL PORT output should output Low.
         clrf       PORTB

            clrf       Index
            clrf       LEDIndex

            clrf       LED0
            clrf       LED1
            clrf       LED2
            clrf       LED3
            clrf       LED4
            clrf       LED5
            clrf       LED6

            clrf       LowB
            clrf       MidB
            clrf       HigB

            bsf        STATUS,RP0

            movlw      b'00010000' ; RA0..RA3 outputs
            movwf      TRISA       ; RA4 input

            movlw      b'00000000' ; RB0..RB7 outputs
            movwf      TRISB
         bsf           OPTION_REG,NOT_RBPU
```

```
            clrwdt                  ; Disable PORTB pull-ups
            movlw       b'10100111' ; Prescaler -> TMR0,
            movwf       OPTION_REG  ; 1:256, rising edge
            bcf         STATUS,RP0  ;
                                    ; Initilize LC-Display Module
                        ; Busy-flag is not yet valid
          clrf        PORTA       ; ALL PORT output should output Low.

            movlw       DELAY15     ; Wait for 15ms for LCD to get powered up
            movwf       R1
            clrf        R2
LCycle
            decfsz      R2,F
            goto        LCycle      ; 3*256
            decfsz      R1,F        ; 3*256+1
            goto        LCycle      ;(3*256+2)*R1=770*R1 in procesor cycles

          movlw       B'00111000' ; 8-bit-interface, 2-lines
          call        PutCMD

          movlw       B'00001000' ; disp.off, curs.off, no-blink
          call        PutCMD

          movlw       1           ; LCD clear
          call        PutCMD

            movlw       B'00001100' ; disp.on, curs.off
          call        PutCMD

          movlw       B'00000110' ; auto-inc (shift-cursor)
          call        PutCMD

            goto        Go

;*****************************************************************************
; LCD Module Subroutines
;*****************************************************************************
; Busy: Returns when LCD busy-flag is inactive
;
; PORTA returns as RA0..RA2 output, RA3,RA4 input
; PORTB returns as full output
;=============================================================================
Busy
          bsf           STATUS,RP0  ; Select Register page 1

          movlw       0FFh      ; Set PORTB for input
          movwf       TRISB

            movlw       b'00011000' ; PORTA should be set RA0..RA2 output
            movwf       TRISA       ; RA3,RA4 input

          bcf         STATUS,RP0  ; Select Register page 0

          bcf           PORTA,RS    ; Set LCD for command mode
          bsf           PORTA,R_W   ; Setup to read busy flag
          bsf           PORTA,E     ; LCD E-line High

          movf        PORTB,W         ; Read busy flag + DDram address

          bcf           PORTA,E     ; LCD E-line Low
```

```
        andlw   080h        ; Check Busy flag, High = Busy
        btfss   STATUS,Z
        goto    Busy

          bcf       PORTA,R_W

        bsf         STATUS,RP0  ; Select Register page 1

        movlw   0
        movwf   TRISB     ; Set PORTB for output

        bcf         STATUS,RP0  ; Select Register page 0
        return

;*****************************************************************
; PUTCHAR Sends character to LCD, Required character must be in W
;*****************************************************************
PutCHAR
        movwf  LCD_TEMP   ; Character to be sent is from W saved
        call   Busy     ; Wait for LCD to be ready
                                ; Busy routine sets PORTA&PORTB adequately
        bcf             PORTA,R_W  ; Set LCD in read mode
        bsf             PORTA,RS   ; Set LCD in data mode
        bsf             PORTA,E    ; LCD E-line High
        movf   LCD_TEMP,W ; Restore character into W
        movwf  PORTB    ; Send data to LCD
        bcf             PORTA,E    ; LCD E-line Low
        return

;*****************************************************************
; PutCMD Sends command to LCD, Required command must be in W
;*****************************************************************
PutCMD
        movwf  LCD_TEMP   ; Command to be sent is from W saved

        call   Busy     ; Wait for LCD to be ready
                                ; Busy routine sets PORTA&PORTB adequately
        bcf             PORTA,R_W  ; Set LCD in read mode
        bcf             PORTA,RS   ; Set LCD in command mode
        bsf             PORTA,E    ; LCD E-line High
        movf   LCD_TEMP,W
        movwf  PORTB    ; Send data to LCD
        bcf             PORTA,E    ; LCD E-line Low
        return

;*****************************************************************
; End of LCD Module Subroutines
;*****************************************************************
; Numeric routines
;-----------------------------------------------------------------
; 3 byte substraction of the constant from the table which sets carry if
; result is negative
;-----------------------------------------------------------------

Subc24      clrf        TEMP        ; it will TEMPorary save C
            movf        Index,W     ; pointer to low byte of constant
            movwf       HIndex      ; W -> HIndex
            call        DecTable    ; W returned with low byte of constant
            bsf         STATUS,C    ; set C
            subwf       LowB,F      ; LowB - W -> LowB
                                    ; if underflow -> C=0
```

```
            btfsc       STATUS,C
            goto        Step1
            bsf         STATUS,C
            movlw       1
            subwf       MidB,F      ; decrement MidB
                                    ; if underflow -> C=0
            btfsc       STATUS,C
            goto        Step1

            bsf         STATUS,C
            movlw       1
            subwf       HigB,F      ; decrement HigB
            btfsc       STATUS,C    ; if underflow -> C=0
            goto        Step1
            bsf         TEMP,C      ; set C

Step1       decf        HIndex,F
            movf        HIndex,W    ; pointer to middle byte of const
            call        DecTable
            bsf         STATUS,C
            subwf       MidB,F      ; MidB - W -> MidB
            btfsc       STATUS,C    ; if underflow -> C=0
            goto        Step2
            bsf         STATUS,C
            movlw       1
            subwf       HigB,1      ; decrement HigB
            btfsc       STATUS,C    ; if underflow -> C=0
            goto        Step2
            bsf         TEMP,C      ; set C

Step2       decf        HIndex,F
            movf        HIndex,W    ; pointer to middle byte of constatnt
            call        DecTable
            bsf         STATUS,C
            subwf       HigB,F      ; HigB - W -> HigB
            btfsc       STATUS,C    ; if underflow -> C=0
            goto        ClearCF
            bsf         STATUS,C
            goto        SubEnd
ClearCF     rrf         TEMP,C      ; C -> STATUS
SubEnd      retlw       0

; ------------------------------------------------------------------------
; 3 byte addition of the constant from the table which sets carry if
; result overflows
; ------------------------------------------------------------------------

Addc24      clrf        TEMP        ; register for TEMPorary storage of C
            movf        Index,W     ; pointer to lower byte of const into W
            movwf       HIndex      ; save it into HIndex
            call        DecTable    ; W contains low byte of const
            bcf         STATUS,C    ; clear C
            addwf       LowB,1      ; W + LowB -> LowB
            btfss       STATUS,C    ; test overflow
            goto        Add2
            bcf         STATUS,C    ; clear C
            movlw       1
            addwf       MidB,F      ; increment MidB
            btfss       STATUS,C
            goto        Add2
            bcf         STATUS,C
```

```
            movlw       1
            addwf       HigB,F      ; increment HigB
            btfss       STATUS,C    ; test overflow
            goto        Add2
            bsf         TEMP,C      ; store C
Add2        decf        HIndex,F    ; pointer to middle byte into W
            movf        HIndex,W
            call        DecTable
            bcf         STATUS,C
            addwf       MidB,1      ; W + MidB -> MidB
            btfss       STATUS,C
            goto        Add3
            bcf         STATUS,C    ; clear C
            movlw       1
            addwf       HigB,1      ; increment HigB
            btfss       STATUS,C
            goto        Add3
            bsf         TEMP,C
Add3        decf        HIndex,F    ; pointer to higher byte into W
            movf        HIndex,W
            call        DecTable
            bsf         STATUS,C
            addwf       HigB,F      ; W + HigB -> HigB,
            btfss       STATUS,C
            goto        ClarCF
            bsf         STATUS,C
            goto        AddEnd
ClarCF      rrf         TEMP,C      ; C -> STATUS
AddEnd      retlw       0

;-------------------------------------------------------------------------
; Tables for 3 byte constants
;-------------------------------------------------------------------------
; Table of decades
;-------------------------------------------------------------------------

DecTable    addwf       PCL,F       ; W + PCL -> PCL
            retlw       0           ; 10
            retlw       0           ;
            retlw       0Ah         ;

            retlw       0           ; 100
            retlw       0           ;
            retlw       064h        ;

            retlw       0           ; 1 000
            retlw       03h         ;
            retlw       0E8h        ;

            retlw       0           ; 10 000
            retlw       027h        ;
            retlw       010h        ;

            retlw       01h         ; 100 000
            retlw       086h        ;
            retlw       0A0h        ;

            retlw       0Fh         ; 1 000 000
            retlw       042h        ;
            retlw       040h        ;
```

```
;-------------------------------------------------------------------------
; Table for RF shift - it is needed in the case of digiscale
; Example: 10.7 MHz is set as 1 070 000 = 10 53 B0 hex
; Note: direct definition using compiler directive is possible too...
;-------------------------------------------------------------------------

MFTable     addwf     PCL,F
            retlw     010h
            retlw     053h
            retlw     0B0h

;*************************************************************************
; Entry point for main cycle
;*************************************************************************

;-------------------------------------------------------------------------
; Routine for the conversion of 3 byte number into 7 decimal numbers
;-------------------------------------------------------------------------

Go
            movlw     6*3-1         ; pointer to dec. table
            movwf     Index         ; 6*3-1 -> Index

            movlw     9             ; maximum of substractions
            movwf     Count         ; 9 -> Count

            clrf      Help

            movlw     6
            movwf     LEDIndex

Divide      call      Subc24        ; substract untill result is negative,
            btfsc     STATUS,C      ; add last substracted number
            goto      Add24         ; next digit
            incf      Help,F
            decf      Count,F
            btfss     STATUS,Z
            goto      Divide
            movlw     3
            subwf     Index,F
            goto      Next

Add24       call      Addc24
            movlw     03h
            subwf     Index,F

Next        movlw     9
            movwf     Count
            movlw     LED1          ; LED1 -> W
            addwf     LEDIndex,W    ; LED1 + LEDIndex -> W
            movwf     TEMP
            decf      TEMP,F        ; LEDIndex+LED1-1 -> TEMP
            movf      TEMP,W

            movwf     FSR           ; W -> FSR
            movf      Help,W        ; Help -> W
            clrf      Help          ; save result at LEDx
            movwf     INDF          ; W -> LED(6..1)
            decf      LEDIndex,F

            movlw     1
```

```
            addwf       Index,W
            btfss       STATUS,Z
            goto        Divide

            movf        LowB,W
            movwf       LED0        ; the rest -> LED0

;-------------------------------------------------------------------------
; registers LED0..LED6 are filled with values - ready to be displayed
;-------------------------------------------------------------------------

            movlw       6
            movwf       LEDIndex

            movlw       LINE0
          iorlw       080h        ; Position cursor leftmost on first line
          call        PutCMD

LEDCycle
            movlw       LED0        ; LED0 -> W
            addwf       LEDIndex,W  ; LED1 + LEDIndex -> W

            movwf       FSR         ; W -> FSR
            movf        INDF,W      ; LED(0..6) -> W

            iorlw       030h
            call        PutCHAR     ; Display character

            movlw       5           ; test for decimal point
            bsf         STATUS,Z
            subwf       LEDIndex,W
            btfss       STATUS,Z
            goto        NoDot
            movlw       '.'         ; this can be ' ' or ',' ......
            call        PutCHAR     ; Display character
NoDot
            decfsz      LEDIndex,F
            goto        LEDCycle    ; continue with next number

            movlw       LED0        ; LED0 -> W
            addwf       LEDIndex,W  ; LED0 + LEDIndex -> W

            movwf       FSR         ; W -> FSR
            movf        INDF,W      ; [FSR] -> W

            iorlw       030h
            call        PutCHAR     ; Display character

            movlw       LINE1       ; continue at right half of display
          iorlw       080h        ; Function set
          call        PutCMD      ; Position cursor leftmost on first line

            movlw       ' '
            call        PutCHAR     ; Display character
            movlw       'M'
            call        PutCHAR     ; Display character
            movlw       'H'
            call        PutCHAR     ; Display character
            movlw       'z'
            call        PutCHAR     ; Display character
```

```
          movlw     LINE0
        iorlw     080h        ; Function set
        call      PutCMD

;-------------------------------------------------------------------------
; It is time to prepare new measuring cycle
;-------------------------------------------------------------------------

          clrf      TimerH
          clrf      TMR0
          nop                 ; it is SUGGESTED...
          nop

          clrf      LEDIndex

          movlw     T1          ; set initial counter values
          movwf     R1
          movlw     T2
          movwf     R2
          movlw     T3
          movwf     R3

          clrf      INTCON      ; global INT disable, TMR0 INT disable
                                ; clear TMR0 overflow bite

; ------------------------------------------------------------------------
; Start measurement:  RA3 + RA4 set input
; ------------------------------------------------------------------------

          movlw     b'00010000' ; all ports set L, RA4 set H
          movwf     PORTA

          bsf       STATUS,RP0
          movlw     b'00011111' ; RA0..RA4 input
          movwf     TRISA
          bcf       STATUS,RP0

; ------------------------------------------------------------------------
; It is opened now...
; ------------------------------------------------------------------------
Cycle
          btfss     INTCON,2 ; 1   Test for TMR0 overflow
          goto      Nothing  ; 3
          incf      TimerH,F ; 3
          bcf       INTCON,2 ; 4
          goto      Nxt      ; 6
Nothing
          nop                ; 4
          nop                ; 5
          nop                ; 6
Nxt
          decfsz    R1,F     ; 7
          goto      Cycle    ; 9
          movlw     T1       ; 9*T1

          movwf     R1       ; 9*T1+1
          decfsz    R2,F     ; 9*T1+2
          goto      Cycle    ; 9*T1+4
          movlw     T2       ;(9*T1+4)*T2

          movwf     R2       ;(9*T1+4)*T2+1
```

```
            decfsz      R3,F        ;(9*T1+4)*T2+2
            goto        Cycle       ;(9*T1+4)*T2+4
                                    ;((9*T1+4)*T2+4)*T3-1

; ------------------------------------------------------------------------
; Short fine tuning delay - enable it if needed.....
; ------------------------------------------------------------------------
;            movlw       .1
;            movwf       Help
;CCC
;            decfsz      Help,F
;            goto        CCC

            nop                     ;((9*T1+4)*T2+4)*T3

; ------------------------------------------------------------------------
; Final test for TMR0 overflow
; ------------------------------------------------------------------------

            movlw       T4          ;((9*T1+4)*T2+4)*T3
            movwf       Help        ;((9*T1+4)*T2+4)*T3+1
Cycle2
            btfss       INTCON,2    ; 1
            goto        Not2Do      ; 3
            incf        TimerH,F    ; 3
            bcf         INTCON,2    ; 4
            goto        Nx          ; 6
Not2Do
            nop                     ; 4
            nop                     ; 5
            nop                     ; 6
Nx
;           nop                     ; fine tuning nops
;           nop
;           nop

            decfsz      Help,F      ; 7+X
            goto        Cycle2      ; 9+X

; ------------------------------------------------------------------------
; Stop the measurement
; ------------------------------------------------------------------------

            clrw                    ; (9+X)*T4
            movwf       PORTB       ; 1
            movlw       b'00010000' ; 2   RA0..RA3 = 0
            movwf       PORTA       ; 3   W -> PORTA

            bsf         STATUS,RP0  ; 4
            movlw       b'00010111' ; 5   RA3 output
            movwf       TRISA       ; 6   RA0..RA2,RA4 input
            bcf         STATUS,RP0  ;

; ------------------------------------------------------------------------
; Analyse precounter and store counted value in registers
; ------------------------------------------------------------------------

            movf        TMR0,W
            movwf       MidB        ; TMR0 -> MidB

            movf        TimerH,W
```

```
            movwf       HigB        ; TimerH -> HigB

            clrf        TEMP
CountIt
            incf        TEMP,F
            bsf         PORTA,3     ; _| false impulz
            bcf         PORTA,3     ;    |_

            bcf         INTCON,2
            movf        TMR0,W      ; actual TMR0 -> W
            bcf         STATUS,Z
            subwf       MidB,W
            btfsc       STATUS,Z
            goto        CountIt
            incf        TEMP,F
            comf        TEMP,F
            incf        TEMP,F
            incf        TEMP,W

            movwf       LowB

; ---------------------------------------------------------------------
; Frequency shift according value on RB0 pin
; Both routines are simplified Subc24 and Addc24 routines
; ---------------------------------------------------------------------

            movlw       b'00010000'
            movwf       PORTA

            bsf         STATUS,RP0
            movlw       b'00000100' ; set RA2 as input
            movwf       TRISA
            bcf         STATUS,RP0

            btfss       PORTA,E
            goto        SubMF
            goto        MFAdd
SubMF
            clrf        TEMP
            movlw       2
            call        MFTable
            bsf         STATUS,C
            subwf       LowB,F

            btfsc       STATUS,C
            goto        S1

            bsf         STATUS,C
            movlw       1
            subwf       MidB,F

            btfsc       STATUS,C
            goto        S1

            bsf         STATUS,C
            movlw       1
            subwf       HigB,F

            btfsc       STATUS,C
            goto        S1
            bsf         TEMP,C
```

```
S1
            movlw       1
            call        MFTable
            bsf         STATUS,C
            subwf       MidB,1

            btfsc       STATUS,C
            goto        S2
            bsf         STATUS,C
            movlw       1
            subwf       HigB,F

            btfsc       STATUS,C
            goto        S2
            bsf         TEMP,C
S2
            clrw
            call        MFTable
            bsf         STATUS,C
            subwf       HigB,1

            btfsc       STATUS,C
            goto        CCT
            goto        Zero
CCT
            btfss       TEMP,C
            goto        ToMFEnd
Zero
            clrf        LowB
            clrf        MidB
            clrf        HigB
            goto        ToMFEnd
MFAdd
            clrf        TEMP
            movlw       2
            call        MFTable
            bcf         STATUS,C
            addwf       LowB,F
            btfss       STATUS,C
            goto        AddMF2
            bcf         STATUS,C
            movlw       1
            addwf       MidB,F
            btfss       STATUS,C
            goto        AddMF2
            bcf         STATUS,C
            movlw       1
            addwf       HigB,F
            btfss       STATUS,C
            goto        AddMF2
            bsf         TEMP,C
AddMF2
            movlw       1
            call        MFTable
            bcf         STATUS,C
            addwf       MidB,F
            btfss       STATUS,C
            goto        AddMF3
            bcf         STATUS,C
            movlw       1
            addwf       HigB,F
```

```
            btfss       STATUS,C
            goto        AddMF3
            bsf         TEMP,C
AddMF3
            clrw
            call        MFTable
            addwf       HigB,F
ToMFEnd
            goto        Go          ;  start new cycle - line 434

; -----------------------------------------------------------------------

            end
```

UFRGS / EE / DELET

Semana Acadêmica 2004

CURSO PIC

Compilado por
Prof. A. Junqueira
Outubro/2004

--

A maior parte do material PIC e deste CD está disponível em:

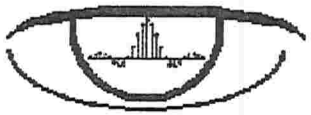
http://www.lapsi.eletro.ufrgs.br/Disciplinas/ENG-SemAcad-PIC/

Veja também material adicional em:

http://www.lapsi.eletro.ufrgs.br/Disciplinas/ENG04475/ (Microprocessadores)

http://www.lapsi.eletro.ufrgs.br/Disciplinas/ENG ELETRICA/ (diversos...)

--



LaPSI

Universidade Federal do Rio Grande do Sul
Escola de Engenharia
Departamento de Engenharia Elétrica

Laboratório de Processamento de Sinais e Imagens

Coordenação: Prof. Dr. Altamiro Amadeu Susin
susin@eletro.ufrgs.br

Av. Osvaldo Aranha, 103 - Sala 206-B
Porto Alegre/RS - BRASIL
CEP 90035-190
Fone +55 51 3316-3136
Fax +55 51 3316-3293

Contatos sobre esta página
schuka@eletro.ufrgs.br, negreiro@eletro.ufrgs.br

Última Atualização: Thu, 15 May 2003 22:16:14 GMT

LaPSI

Apresentação
Laboratório
Equipe
Publicações
Pesquisa
Disciplinas
Links
Download
Salão/Feira IC 2003

Acesso restrito
Software
Projetos
Intranet

English version

Index of /Disciplinas

Name	Last modified	Size	Description
Parent Directory	16-May-2003 12:03	-	
ELE00005/	13-Oct-2004 17:37	-	
ELE00006/	13-May-2004 14:58	-	
ELE00007/	25-Mar-2004 16:38	-	
ELE00009/	13-Nov-2003 15:51	-	
ENG-SemAcad-PIC/	26-Oct-2004 13:49	-	
ENG04009/	04-Aug-2004 11:17	-	
ENG04018/	09-Feb-2003 16:22	-	
ENG04427/	04-Aug-2004 11:17	-	
ENG04461/	20-Oct-2004 13:42	-	
ENG04475/	26-Mar-2004 19:33	-	
ENG04476/	03-Jul-2003 12:51	-	
ENG04477/	15-Mar-2004 15:56	-	
ENG ELETRICA/	26-Mar-2004 19:07	-	
PKUNZIP.EXE	01-Mar-1999 02:50	34k	
cmp117/	26-Oct-2004 20:31	-	
pkzip-msdos-all.exe	05-Nov-2003 19:44	184k	
winzip81.exe	13-Mar-2003 16:36	1.8M	
wrar320.exe	31-Jul-2003 16:45	965k	

Apache/1.3.26 Server at www.lapsi.eletro.ufrgs.br Port 80

LaPSI

Apresentação
Laboratório
Equipe
Publicações
Pesquisa
Disciplinas
Links
Download
Salão/Feira IC 2003

Acesso restrito
Software
Projetos
Intranet

English version

Index of /Disciplinas/ENG-SemAcad-PIC

Name	Last modified	Size	Description
Parent Directory	26-Oct-2004 13:45	-	
0 - PROGRAMADORES/	26-Oct-2004 13:46	-	
0 - PROGS AUXILIARES/	26-Oct-2004 13:47	-	
App Notes/	26-Oct-2004 13:47	-	
Curso SENAI/	26-Oct-2004 13:47	-	
Kit FlyPic/	26-Oct-2004 13:47	-	
Links/	26-Oct-2004 13:47	-	
PIC/	26-Oct-2004 13:48	-	
Site Mosaico/	26-Oct-2004 13:49	-	
Tutoriais PIC/	26-Oct-2004 13:49	-	

Apache/1.3.26 Server at www.lapsi.eletro.ufrgs.br Port 80

3