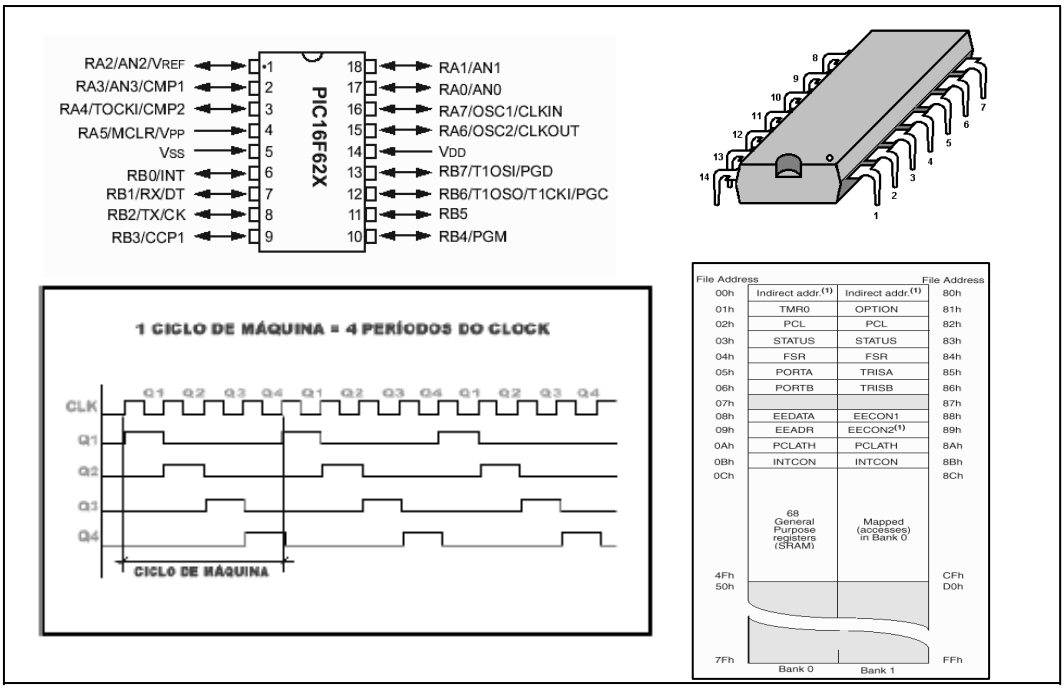
# Microcontroladores PIC




Prof. Valdir Dugo Zaragoza

28/07/2015

## MICROCONTROLADORES PIC

Os microcontroladores PIC são componentes integrados programáveis fabricados pela empresa americana Microchip. Possuem grande vocação para automação de equipamentos, sendo que sua utilização começou a popularizar-se no Brasil no final da década de 80. Até esse momento as alternativas existentes eram os microprocessadores para automação ou os Controladores Lógicos Programáveis (PLC ou CLP) que eram mais utilizados em aplicações industriais.

**É um pequeno componente eletrônico, utilizado geralmente para controle de processos lógicos em funções específicas. Possui instruções mais simples que um microprocessador, o que o torna muito mais fácil de ser programado.**

Os microcontroladores possuem as seguintes características:

- Baixo custo em relação aos microprocessadores ou CLP's
- Facilidade de aplicação e grande versatilidade.
- Possui internamente todos os módulos necessários para processamento de dados, como memória de dados (RAM), memória de programa (PROM ou FLASH), contadores, Interfaces de I/O, etc.
- Não necessita de componentes periféricos para a maioria das aplicações.
- Trabalha com frequências de clock de até 20 MHz, sendo que alguns modelos possuem oscilador interno de 4 MHz, dispensando gerador externo (cristal ou RC).
- Alguns modelos possuem internamente periféricos dedicados como conversores A/D, comunicação serial I2C, SPI, USART, etc.
- Os modelos com memória FLASH podem ser regravados milhares de vezes, facilitando o desenvolvimento de protótipos.
- O desenvolvimento de programas para microcontroladores requer profissionais qualificados, pois na maioria dos casos são escritos em Assembler ou linguagem C. Existem compiladores para outras linguagens como o Basic, mas são utilizados em menor escala.

Existem várias famílias de circuitos integrados que executam uma função lógica pré-determinada, na qual podemos destacar os PAL (Programable Array Logic) e os PLD (Programable Logic Device). Estes dispositivos tiveram uma larga aplicação na indústria eletrônica, reduzindo drasticamente as placas de circuito impresso, custo de fabricação e manutenção.

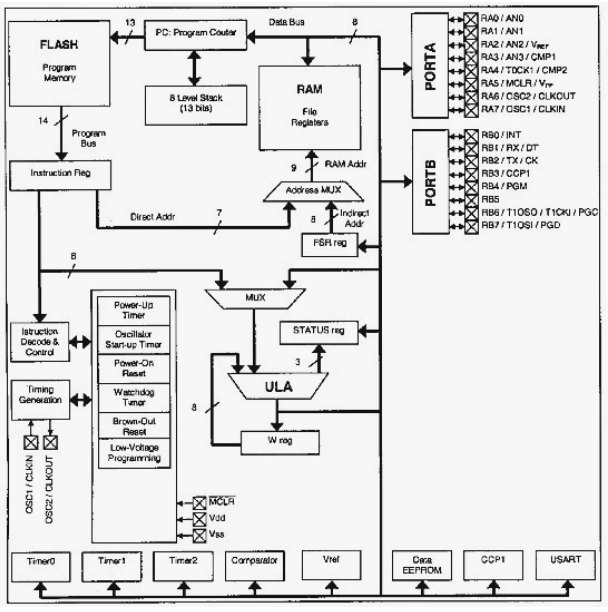
Seu ponto fraco é que uma vez produzido o chip, não há como alterar suas funções, sendo necessário desenvolver um novo modelo.

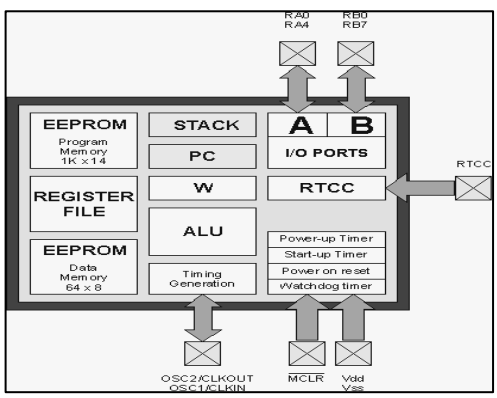
Com a introdução dos microcontroladores, os circuitos integrados programáveis foram sendo substituídos gradativamente, em face da sua facilidade de programação e alteração de suas funções, sem ter que substituir o chip, sendo possível mudar as funções de um projeto simplesmente alterando seu programa.

## Microcontrolador PIC16F84A

O microcontrolador 16F84A da Microchip, possui 18 pinos, 13 portas configuráveis como entrada ou saída, 4 interrupções, memória de programação EEPROM FLASH, via de programação com 14 bits e 35 instruções.

A estrutura interna do PIC 16F628A é mostrada na figura abaixo :



As instruções de 14 bits são divididas da seguinte forma :

Instrução - ( x x x x x x x x x x x x x x) 14 bits
| 6 bits | 8 bits |

Da forma como é utilizada a instrução, os primeiros 8 bits podem trazer uma informação de endereço (como no caso das instruções MOVF, ADDWF, SUBWF, etc.) ou um dado literal de 8 bits (como no caso das instruções MOVLW, ADDLW, SUBLW, etc.).

Note que nas instruções que utilizam endereço existe a letra F (referente ao endereço do registro F na memória RAM) e nas instruções que utilizam dado literal existe a letra W (referente ao acumulador interno W). O dado literal de 8 bits é gravado na própria instrução conforme veremos mais adiante.

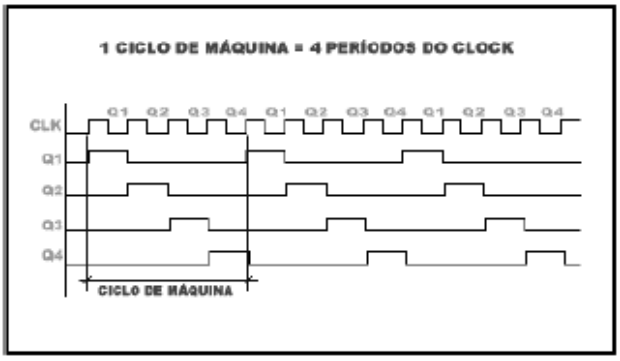
Os últimos 6 bits são usados para codificar a instrução a ser executada. Observe que com 6 bits conseguimos $2^6$ = 64 combinações diferentes, ou seja, com essa estrutura seria possível utilizar um total de 64 instruções diferentes.

## Ciclo de Máquina

Um microcontrolador pode ser entendido como sendo uma máquina que executa operações em ciclos.

Todos os sinais necessários para a busca ou execução de uma determinada instrução devem ser gerados dentro de um período de tempo denominado Ciclo de Máquina. Nos PIC´s com memória de programa de 12 e 14 bits um Ciclo de Máquina corresponde a quatro períodos de clock (1:4) denominados Q1,Q2,Q3 e Q4, conforme pode ser verificado na figura abaixo.

Ciclo de Máquina

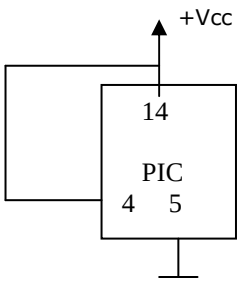



## Características elétricas

- Tensão de trabalho.....2,0V a 6,0V
- Tensão máxima no pino Vdd (em relação ao Vss ).....-0,3V a 7,5V
- Tensão máxima no pino /MCLR ( em relação ao Vss ).....-0,3V a 14V
- Tensão máxima nos demais pinos ( em relação ao Vss )...-0,6V a Vdd+0,6V
- Dissipação máxima de energia.....800 mW
- Temperatura de trabalho.....-55 ºC a +125 ºC
- Temperatura de armazenamento..... -65 ºC a +150 ºC
- Corrente máxima de saída no pino Vss.....150 mA
- Corrente máxima de entrada no pino Vdd .....100 mA
- Corrente máxima de entrada de um pino .....25 mA
- Corrente máxima de saída de um pino .....20 mA
- Corrente máxima de entrada no PORTA.....80 mA
- Corrente máxima de saída do PORTA.....50 mA
- Corrente máxima de entrada no PORTB.....150 mA
- Corrente máxima de saída do PORTB.....100 mA
- Consumo típico em 5 Volts / 4 MHz .....2 mA
- Consumo em standby ( sleep )..... < 1 µA

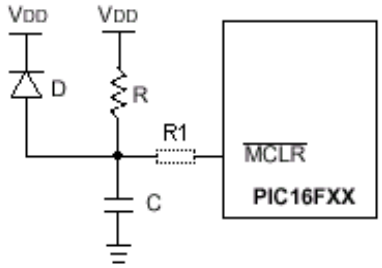
Circuito de Reset

O PIC permite que tenhamos um reset automático sem componentes externos ( POR – Power On Reset )




Obs: Isto só pode ser feito se a tensão da fonte não subir lentamente.

Uma outra opção é utilizar um circuito RC




Valores praticados:
D – 1N4007
R < 40KΩ
R1 – 100Ω a 1kΩ
C – 0,1 uF

O resistor R1 é utilizado para limitar corrente.

*Tipos de memória de programa*

Basicamente temos 5 tipos de memória, na qual iremos armazenar os dados e os programas a serem executados:

**ROM** máscara – Os dados são gravados no momento de sua fabricação, e não podem ser excluídos

**OTP** ( One Time Program ) – Com auxilio de um gravador e um PC, esta memória pode ser gravada uma única vez, e os dados armazenados não podem ser excluídos.

**EPROM** – Com auxilio de um gravador e um PC, os dados podem ser gravados e excluídos várias vezes , submetendo o chip por alguns minutos a raios ultravioleta através de uma janela de cristal na face superior do chip.

**EEPROM** ou **$E_2$PROM** – Este tipo de memória os dados podem ser gravados e excluídos eletricamente com um gravador, sem o auxilio dos raios ultravioleta.

**FLASH** – Esta memória é similar à EEPROM, com auxilio de um programador conectado ao PC, o usuário pode gravar e excluir os dados por várias vezes.

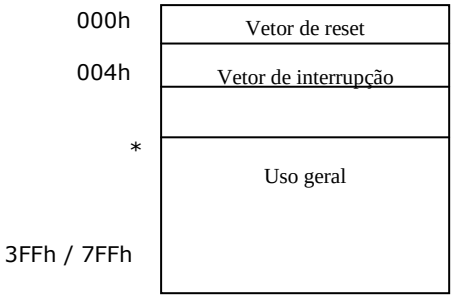
Organização da Memória nos microcontroladores PIC16F84A e PIC16F628A

A memória dos microcontroladores PIC16F84A e PIC16F628A está organizada em memória de dados e memória de programa.

O PIC16F628A possui três tipos de memória disponíveis :

- 2048 endereços de memória de programa (FLASH) de 14 bits (1024 no PIC16F84A). Pode ser regravada milhares de vezes.
- 128 endereços de memória EEPROM ou $E_2$PROM (64 endereços no PIC16F84A). Por ser uma memória não volátil a $E_2$PROM é utilizada para armazenar dados que devam permanecer gravados mesmo após o micro ser desligado.
- Memória de dados (Ram estática volátil) distribuída em Registradores de Funções Especiais (SFR) mais 224 endereços de memória de dados de uso geral.

## Memória de programa




Tamanho: 1K x 14
Vetor de reset : : 000h
Vetor de interrupção : 004h

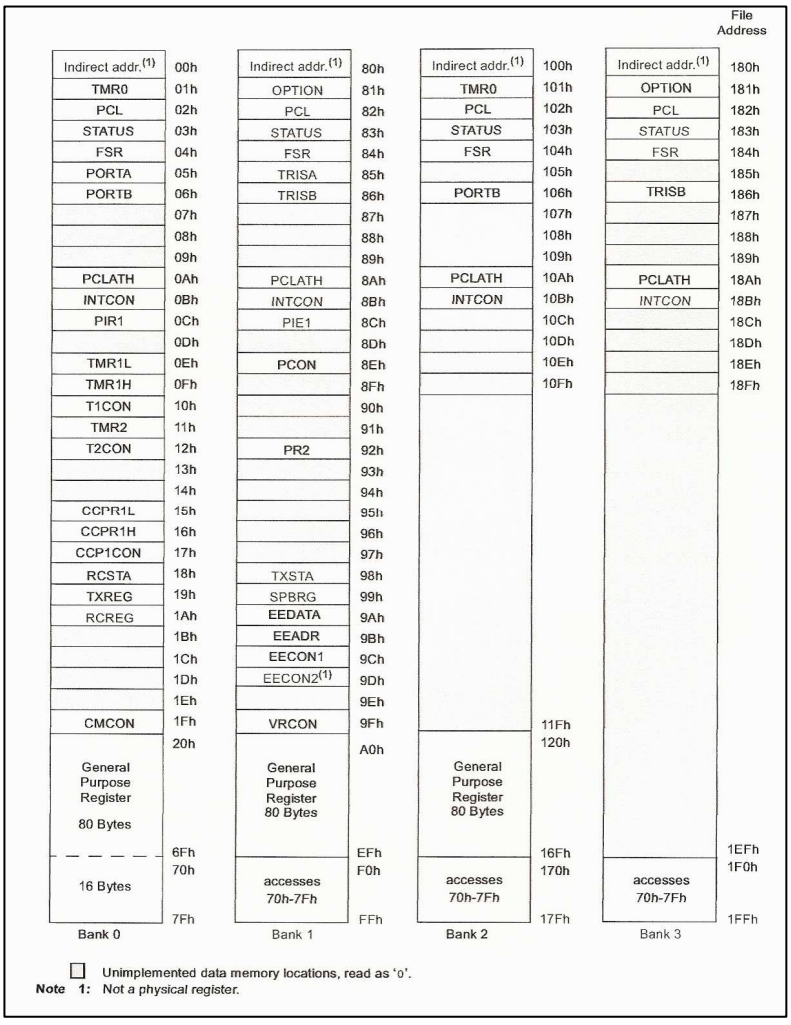
* - 0Ch no PIC16F84A e 20h no PIC16F627/8A

O microcontrolador PIC16F84A possui uma memória de programa do tipo Flash, com capacidade de 1024 palavras de 14 bits. A memória do tipo Flash permite milhares de ciclos de gravação e exclusão. Seu contador de programa possui 13 linhas de endereçamento, sendo possível de endereçar até 8K x 14 bits.

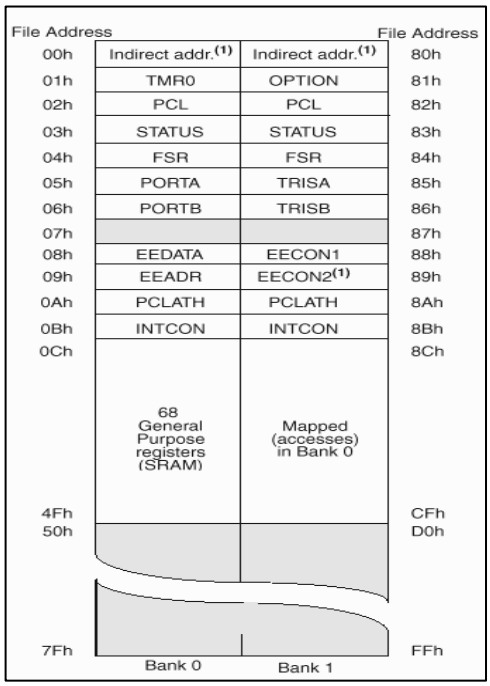
Memória de dados (Ram estática volátil)

A memória de dados é distribuída em 4 bancos no PIC16F627/8A e 2 bancos no PIC16F84A conforme os quadros mostrados abaixo.

### Mapa de memória do PIC16F628A



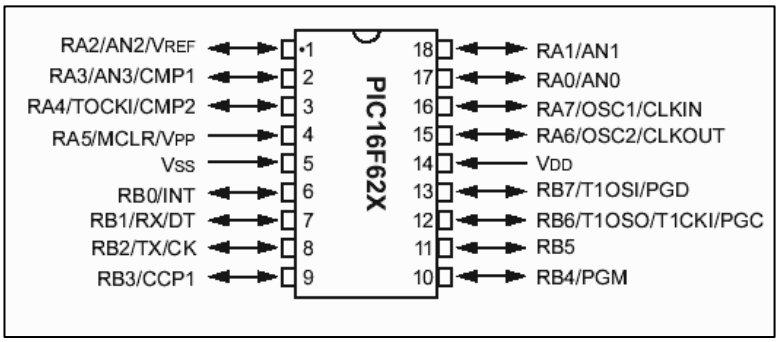

### Mapa de memória do PIC16F84A



Encapsulamento

A figura abaixo mostra o encapsulamento DIP 18 do PIC16F627, PIC16F628A e do PIC16F84A. Estão disponíveis de 13 a 16 terminais de I/O divididos em duas portas (Porta A e Porta B). Os terminais RA0 a RA 7 formam a chamada PORTA, enquanto os terminais RB0 a RB7 formam PORTB.

Observe que muitos terminais possuem mais do que uma função definida, como por exemplo RA7, que pode ser utilizado como o bit mais significativo da porta A, ou então terminal de conexão com um oscilador externo (cristal ou RC) ou ainda entrada de sinal de um gerador de clock externo. Essas funções são configuradas durante a gravação do programa na memória Flash. A alimentação é feita através dos terminais VDD (pino 14 + ) e VSS (pino 5 - ). A alimentação do PIC segue o padrão TTL, ou seja, 5 V.




## Pinagem do PIC16F84A

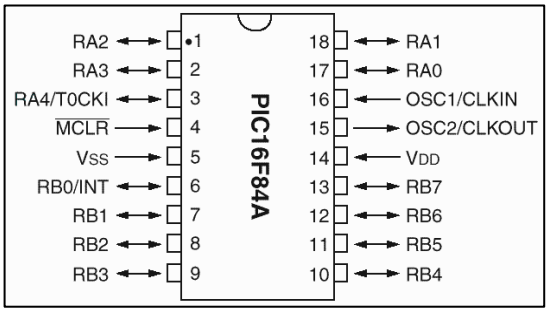
O PIC 16F84A possui 18 terminais.

2 termin. alimentação – pino 5 - GND; pino 14 - Vcc

2 termin. oscilador – pino 15; pino 16

1 termin. de Reset – pino 4

13 termin. de I /O – Port A – pinos 1, 2, 3, 17 e 18; Port B – pinos 6, 7, 8, 9, 10, 11, 12 e 13




## Descrição dos terminais

Alimentação

O PIC pode ser alimentado com tensão continua de 2 a 6 Volts

Circuito oscilador

O PIC possui um hardware incorporado que permite adaptar-se por diferentes tipos de circuitos osciladores ( clock ).

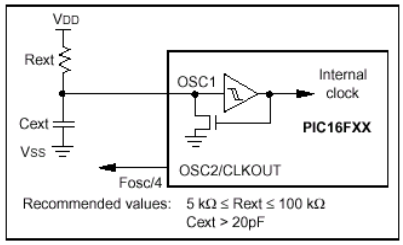
LP – Low Power Crystal – 32KHz a 200KHz
XT – Crystal – 2MHz a 4MHz
HS – High Speed Crystal – 4MHz a 10MHz
RC – Resistor / Capacitor

Circuito RC

Este tipo é o mais simples e o mais barato,

deixando a desejar quanto sua precisão,

devido à tolerância dos componentes.



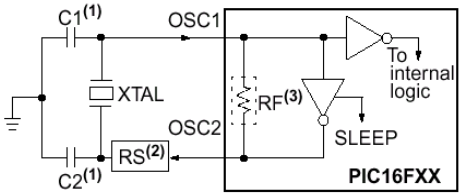
Circuito com cristal ( XT, HS, LP )

Valores fornecidos pela Microchip:

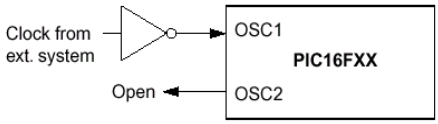
XTAL : até 10MHz

C1 e C2 : entre 15pF a 33pF

RS : Resistor série ~ 100Ω




Com entrada de clock externo ( XT, HS ou LP )




*Terminais de I / O*

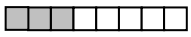
O PIC16F84A possui 13 terminais de I/O, divididos em PORTA e PORTB.

PORTA possui 5 terminais e o PORTB possui 8 terminais, e pode-se definir qualquer terminal como entrada ou saída. Para se fazer tal programação, deve-se utilizar o registrador **TRISA e TRISB**.

Cada bit destes registradores corresponde ao respectivo terminal de I/O, quando o bit 0 for igual a 0 (zero), o terminal passa a ser saída, e quando o bit 0 for igual a 1 o terminal passa a ser entrada.

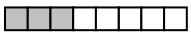
***PORT A***

7 6 5 4 3 2 1 0



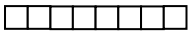
***TRISA***

7 6 5 4 3 2 1 0



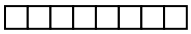
***PORT B***

7 6 5 4 3 2 1 0



***TRISB***

7 6 5 4 3 2 1 0



Exemplo:

Para que se tenha os quatro bits mais significativos como entrada e os outros quatro como saída do PORTB

***PORT B***

| E | E | E | E | S | S | S | S |
|---|---|---|---|---|---|---|---|

Deve-se programar o registrador TRISB com:

***TRISB***

| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

F 0

Portanto o registrador TRISB deve ser carregado com F0h

Descrição dos terminais do PIC16F84A

| | |
|---|---|
| **1** | **RA2**<br>É uma linha de I/O programável em entrada ou saída da unidade.<br>Corresponde ao **BIT 2** da **PORTA A**. |
| **2** | **RA3**<br>É uma linha de I/O programável em entrada ou saída da unidade.<br>Corresponde ao **BIT 3** da **PORTA A**. |
| **3** | **RA4 / RTCC**<br>É um pino multi função que pode ser programado como uma linha normal de I/O ou como linha de clock para entrada em sentido ao contador RTCC.<br>Se programada como linha de I/O corresponde ao **BIT 4** da **PORTA A** ao contrario de outra linha de I/O, Quando esta linha funciona como saída, trabalha em coletor aberto. |
| **4** | **MCLR / VPP**<br>Em condição normal de funcionamento desenvolve a função de **M**aster **CL**ea**R** ou seja Reset estará ativo a nível 0. Pode ser conectado a um circuito de reset externo ou simplesmente conectando-o ao positivo da alimentação.<br>Quando o PIC vier posto em Program Mode será utilizado como entrada para a tensão de programação Vpp. |
| **5** | **VSS**<br>É o pino que vai conectado ao **negativo** da tensão de alimentação. |
| **6** | **RB0**<br>É uma linha de I/O programável em entrada ou em saída.<br>Corresponde ao **BIT 0** da **PORTA B** e pode ser programada para gerar interrupção. |
| **7** | **RB1**<br>É uma linha de I/O programável em entrada ou em saída.<br>Corresponde ao **BIT 1** da **PORTA B**. |
| **8** | **RB2**<br>É uma linha de I/O programável em entrada ou em saída.<br>Corresponde ao **BIT 2** da **PORTA B**. |
| **9** | **RB3**<br>É uma linha de I/O programável em entrada ou em saída.<br>Corresponde ao **BIT 3** da **PORTA B**. |
| **10** | **RB4**<br>É uma linha de I/O programável em entrada ou em saída.<br>Corresponde ao **BIT 4** da **PORTA B**. |
| **11** | **RB5**<br>É uma linha de I/O programável em entrada ou em saída.<br>Corresponde ao **BIT 5** da **PORTA B**. |
| **12** | **RB6**<br>É uma linha de I/O programável em entrada ou saída.<br>Corresponde ao **BIT 6** da **PORTA B**. |
| **13** | **RB7**<br>É uma linha de I/O programável em entrada ou saída.<br>Corresponde ao **BIT 7** da **PORTA B**. |
| **14** | **VDD**<br>É o terminal positivo de alimentação do PIC.<br>em todas as três versões disponíveis do **PIC16F84A** (comercial, industrial e automotiva) a tensão pode assumir um valor que vai de um mínimo de **2.0 volts** a um máximo de **6.0 volts**. |
| **15** | **OSC2 / CLKOUT**<br>É um pino de conexão no caso de se utilizar um cristal de quartzo para gerar o clock. E como saída de clock caso for aplicado um oscilador RC externo. |
| **16** | **OSC1 / CLKIN**<br>É um pino de conexão para o caso de se utilizar um cristal de quartzo ou um circuito RC para gerar o clock. E também como entrada caso utilizemos oscilador externo. |
| **17** | **RA0**<br>É uma linha de I/O programável em entrada ou saída.<br>Corresponde ao **BIT 0** da **PORTA A**. |
| **18** | **RA1**<br>É uma linha de I/O programável em entrada ou saída.<br>Corresponde ao **BIT 1** da **PORTA A**. |

## Registradores de Funções Especiais - SFR

### Registrador STATUS

| Bit 7 | | | | | | | Bit 0 |
|---|---|---|---|---|---|---|---|
| **IRP** | **RP1** | **RP0** | **TO** | **PD** | **Z** | **DC** | **C** |

| | |
|---|---|
| Bit 7 – **IRP**<br><br>R/W | Seleciona Bancos (Endereçamento Indireto)<br><br>0 = Bancos 0 e 1 (00h – FFh)<br>1 = Bancos 2 e 3 (100h – 1FFh) (Manter em 0 para o PIC16F84A) |
| Bit 6 – **RP1** | (Manter em 0 para o PIC16F84A) |
| Bit 5 – **RP0**<br><br>R/W | Seleciona Bancos (Endereçamento Direto)<br><br>00 = Banco 0 – Banco 0 (00h – 7Fh)<br>01 = Banco 1 – Banco 1 (80h – FFh)<br>10 = Banco 1 – Banco 2 (100h – 17Fh) (Não implementados no PIC16F84A)<br>11 = Banco 1 – Banco 3 (180h – 1FFh) |
| Bit 4 – **TO**<br><br>R | Bit sinalizador de Time-out<br><br>1 = Após o power-up, pela instrução CLRWDT ou SLEEP<br>0 = Ocorreu Time-out do Watch Dog |
| Bit 3 – **PD**<br><br>R | Bit sinalizador de Power-down<br><br>1 = Após o power-up ou pela instrução CLRWDT<br>0 = Pela execução da instrução SLEEP |
| Bit 2 – **Z**<br><br>R/W | Bit sinalizador de Zero<br><br>1 = O registro está com o valor 00<br>0 = O registro não está com o valor 00 |
| Bit 1 – **DC**<br><br>R/W | Bit sinalizador de Digit Carry/Borrow<br><br>1 = Ocorreu Carry-out do 3º para o 4º Bit<br>0 = Não ocorreu Carry-out |
| Bit 0 – **C**<br><br>R/W | Bit sinalizador de Carry/Borrow<br><br>1 = Ocorreu Carry-out no 7º Bit<br>0 = Não ocorreu Carry-out |

### Registrador OPTION

| Bit 7 | | | | | | | Bit 0 |
|---|---|---|---|---|---|---|---|
| **RBPU** | **INTEDG** | **TOCS** | **TOSE** | **PSA** | **PS2** | **PS1** | **PS0** |

| | |
|---|---|
| Bit 7 – **RBPU** | Habilita Pull-up da PORTB<br><br>1 = Pull-up da PORTB Desabilitados<br>0 = Pull-up da PORTB Habilitados |
| Bit 6 – **INTEDG**<br><br>R/W | Define o modo de aceitação da interrupção INT<br><br>1 = Na **SUBIDA** do sinal no pino RB0/INT<br>0 = Na **DESCIDA** do sinal no pino RB0/INT |
| Bit 5 – **TOCS** | Define a fonte de Clock do Timer 0<br><br>1 = Na transição do pino RB4/TOCK1<br>0 = Clock interno (CLKOUT = f/4) |
| Bit 4 – **TOSE** | Define o modo como o clock externo incrementará o Timer 0<br><br>1 = Na **DESCIDA** do sinal no pino RB4/TOCK1<br>0 = Na **SUBIDA** do sinal no pino RB4/TOCK1 |
| Bit 3 – **PSA** | Atribuição do **Prescaler**<br><br>1 = Prescaler atribuído ao Watch Dog<br>0 = Prescaler atribuído ao Timer 0 |

| | |
|---|---|
| Bit's 2, 1 e 0<br>**PS2, PS1 e PS0**<br><br>R/W | Seleção da taxa do **Prescaler** |

| **PS2** | **PS1** | **PS0** | **Divisão – Timer 0** | **Divisão –** Watch Dog |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 : 2 | 1 : 1 |
| 0 | 0 | 1 | 1 : 4 | 1 : 2 |
| 0 | 1 | 0 | 1 : 8 | 1 : 4 |
| 0 | 1 | 1 | 1 : 16 | 1 : 8 |
| 1 | 0 | 0 | 1 : 32 | 1 : 16 |
| 1 | 0 | 1 | 1 : 64 | 1 : 32 |
| 1 | 1 | 0 | 1 : 128 | 1 : 64 |
| 1 | 1 | 1 | 1 : 256 | 1 : 128 |

Registrador INTCON

| Bit 7 | | | | | | | Bit 0 |
|---|---|---|---|---|---|---|---|
| **GIE** | **EEIE** | **TOIE** | **INTE** | **RBIE** | **TOIF** | **INTF** | **RBIF** |

| | |
|---|---|
| Bit 7 – **GIE**<br><br>R/W | Habilitação Global das Interrupções (Global Interrupt Enable)<br><br>1 = Habilita todas as Interrupções selecionadas individualmente<br>0 = Desabilita **TODAS** as Interrupções |

| | |
|---|---|
| Bit 6 – **EEIE**<br><br>R/W | Interrupção de fim de escrita na EEPROM<br><br>1 = Habilita<br>0 = Desabilita |

| | |
|---|---|
| Bit 5 – **TOIE**<br><br>R/W | Interrupção de Overflow no Timer 0<br><br>1 = Habilita<br>0 = Desabilita |

| | |
|---|---|
| Bit 4 – **INTE**<br><br>R/W | Interrupção externa RB0/INT<br><br>1 = Habilita<br>0 = Desabilita |

| | |
|---|---|
| Bit 3 – **RBIE**<br><br>R/W | Interrupção por mudanças na PORTB<br><br>1 = Habilita<br>0 = Desabilita |

| | |
|---|---|
| Bit 2 – **TOIF**<br><br>R/W | Sinaliza Interrupção de Overflow no Timer 0 (*)<br><br>1 = Ocorreu Overflow no Timer 0<br>0 = Não ocorreu Overflow no Timer 0 |

| | |
|---|---|
| Bit 1 – **INTF**<br><br>R/W | Sinaliza Interrupção externa no pino RB0/INT (*)<br><br>1 = Ocorreu pedido de Interrupção no pino RB0/INT<br>0 = Não ocorreu pedido de Interrupção no pino RB0/INT |

| | |
|---|---|
| Bit 0 – **RBIF**<br><br>R/W | Sinaliza Interrupção de mudanças no PORTB (*)<br><br>1 = Um ou mais Bit's RB4 – RB7 mudou de estado<br>0 = Nenhum dos Bit's RB4 – RB7 sofreu alteração |

(*) Devem ser zerados pelo software

## Interrupções

O microcontrolador PIC possui 4 interrupções básicas :

- Interrupção direta no pino RB0/INT
- Interrupção por mudança de nível lógico no PORTB
- Interrupção por término da escrita na E2PROM
- Interrupção por estouro na contagem (Overflow) do TIMER0.

Qualquer que seja a interrupção acionada, o microcontrolador interrompe a execução do programa principal e desvia para o vetor de interrupção (endereço 004h da memória de programa). O endereço de retorno fica armazenado em um dos 8 níveis da pilha (Stack Pointer) , sendo que o retorno é acionado pelo comando RETFIE.

Registrador EECON1

| Bit 7 | | | | | | | Bit 0 |
|---|---|---|---|---|---|---|---|
| **- 0 -** | **- 0 -** | **- 0 -** | **EEIF** | **WRERR** | **WREN** | **WR** | **RD** |

| | |
|---|---|
| Bit's 7, 6 e 5 | Não Implementados |

| | |
|---|---|
| Bit 4 – **EEIF**<br><br>R/W | Sinaliza Interrupção de fim de escrita<br><br>1 = Processo de escrita na EEPROM finalizado (zerar por software)<br>0 = Processo de escrita na EEPROM ainda não finalizado |

| | |
|---|---|
| Bit 3 – **WRERR**<br><br>R/W | Flag de erro na escrita da EEPROM<br><br>1 = Parado por Reset ou Watch Dog<br>0 = Completado com sucesso |

| | |
|---|---|
| Bit 2 – **WREN**<br><br>R/W | Habilitação de escrita na EEPROM<br><br>1 = Habilitada<br>0 = Desabilitada |

| | |
|---|---|
| Bit 1 – **WR**<br><br>R/W | Bit de início de escrita (pode ser lido, mas apenas ser setados pelo software)<br><br>1 = Iniciado processo de escrita na EEPROM (zerado pelo hardware no final)<br>0 = Escrita finalizada |

| | |
|---|---|
| Bit 0 – **RD**<br><br>R/W | Bit de início de leitura (pode ser lido, mas apenas ser setados pelo software)<br><br>1 = Iniciado processo de leitura na EEPROM (zerado pelo hardware no final)<br>0 = Leitura finalizada |

**Obs:** O processo de escrita leva em média 10 ms, enquanto que a leitura é realizada em apenas um ciclo de máquina.

Set de instruções do PIC16F84A

| Mnemônico | | Descrição | Ciclos | Código de 14 bits MSb | | | | Flags afetados |
|---|---|---|---|---|---|---|---|---|
| **Operações com registrador orientadas a byte** | | | | | | | | |
| ADDWF | f,d | Adiciona o valor de W ao valor de f (d=f+W) | 1 | 00 | 0111 | dfff | ffff | C,DC,Z |
| ANDWF | f,d | AND lógico de W com f (d=f and W) | 1 | 00 | 0101 | dfff | ffff | Z |
| CLRF | f | Limpa conteúdo de f (zera bits) (f=0) | 1 | 00 | 0001 | lfff | ffff | Z |
| CLRW | | Limpa conteúdo de W (zera bits) (W=0) | 1 | 00 | 0001 | 0xxx | xxxx | Z |
| COMF | f,d | Complementa f (inverte bits) (d= $\bar{f}$ ) | 1 | 00 | 1001 | dfff | ffff | Z |
| DECF | f,d | Decrementa f (d=f-1) | 1 | 00 | 0011 | dfff | ffff | Z |
| DECFSZ | f,d | Decrementa f (d=f-1) e pula se igual a zero | 1 (2) | 00 | 1011 | dfff | ffff | Nenhum |
| INCF | f,d | Incrementa f (d=f+1) | 1 | 00 | 1010 | dfff | ffff | Z |
| INCFSZ | f,d | Incrementa f (d=f+1) e pula se igual a zero | 1 (2) | 00 | 1111 | dfff | ffff | Nenhum |
| IORWF | f,d | OR lógico entre W e f (d=f or W) | 1 | 00 | 0100 | dfff | ffff | Z |
| MOVF | f,d | Copia o valor de f para o destino d (d=f) | 1 | 00 | 1000 | dfff | ffff | Z |
| MOVWF | f | Copia o valor de W para f (f=W) | 1 | 00 | 0000 | lfff | ffff | Nenhum |
| NOP | | Nenhuma operação | 1 | 00 | 0000 | 0xx0 | 0000 | Nenhum |
| RLF | f,d | Desloca f uma posição à esquerda | 1 | 00 | 1101 | dfff | ffff | C |
| RRF | f,d | Desloca f uma posição à direita | 1 | 00 | 1100 | dfff | ffff | C |
| SUBWF | f,d | Subtrai W de F (d=f-W) | 1 | 00 | 0010 | dfff | ffff | C,DC,Z |
| SWAPF | f,d | Troca Nibbles de f (xxxxyyyy -> yyyyxxxx) | 1 | 00 | 1110 | dfff | ffff | Nenhum |
| XORWF | f,d | XOR lógico entre W e f (d=f xor W) | 1 | 00 | 0110 | dfff | ffff | Z |
| BTFSC | f,d | Testa bit b do registrador f, pula se igual a zero | 1 (2) | 01 | 10bb | bfff | ffff | Nenhum |
| BTFSS | f,d | Testa bit b do registrador f, pula se igual a um | 1 (2) | 01 | 11bb | bfff | ffff | Nenhum |
| **Operações de controle e com constantes** | | | | | | | | |
| ADDLW | k | Soma a constante k ao registrador W (W=W+k) | 1 | 11 | 111x | kkkk | kkkk | C,DC,Z |
| ANDLW | k | AND lógico entre k e W (W=W and k) | 1 | 11 | 1001 | kkkk | kkkk | Z |
| CALL | k | Chamada da sub-rotina especificada por k | 2 | 10 | 0kkk | kkkk | kkkk | Nenhum |
| CLRWDT | | Limpa a contagem do watchdog | 1 | 00 | 000 | 0110 | 0100 | $\overline{TO}$ , $\overline{PD}$ |
| GOTO | k | Desvia o programa para o endereço k | 2 | 10 | 1kkk | kkkk | kkkk | Nenhum |
| IORLW | k | OR lógico entre k e W (W=W or k) | 1 | 11 | 1000 | kkkk | kkkk | Z |
| MOVLW | k | Copia a constante k para o registrador W (W=k) | 1 | 11 | 00xx | kkkk | kkkk | Nenhum |
| RETFIE | | Retorno da interrupção (seta GIE para 1) | 2 | 00 | 0000 | 0000 | 1001 | Nenhum |
| RETLW | k | Retorna da sub-rotina, copia k para registrador W | 2 | 11 | 01xx | kkkk | kkkk | Nenhum |
| RETURN | | Retorno de sub-rotina | 2 | 00 | 0000 | 0000 | 1000 | Nenhum |
| SLEEP | | Ativa modo de baixa potência | 1 | 00 | 0000 | 0110 | 0011 | $\overline{TO}$ , $\overline{PD}$ |
| SUBLW | k | Subtrai W de k (W=k-W) | 1 | 11 | 110x | kkkk | kkkk | C,DC,Z |
| XORLW | k | XOR lógico entre k e W (W=W xor k) | 1 | 11 | 1010 | kkkk | kkkk | Z |

## Programação - Fluxogramas

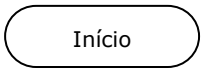
Fluxograma é uma representação de um processo, através de um diagrama, com símbolos previamente definidos. No nosso caso, utilizaremos o fluxograma para representar o algoritmo do programa, de forma que seja facilitada a observação, a lógica e também uma futura alteração do programa efetuado.

Após a conclusão do fluxograma, fica mais fácil a transcrição para a linguagem de programação desejada. Utilizaremos a linguagem C.

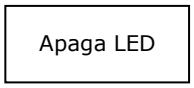
Uma vez transcrito o código, será utilizado um compilador C para converter o código fonte do programa em linguagem de máquina, que será então gravada na memória do microcontrolador PIC.

Existem diversos compiladores C para PIC no mercado, como o MikroC, CC5X, CCS, etc. Os comandos utilizados nesses compiladores são iguais, sendo padrão para a linguagem C, porém, cada um possui diretivas de compilação diferentes, que são comandos utilizados nas configurações iniciais, configuração de "Fuses" internos, declaração de variáveis, etc., devendo o programador utiliza-las de forma adequada.

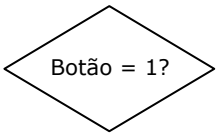
Simbologia utilizada nos fluxogramas:




- Início e final de programa.

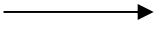



- Execução de comando ou ação.




- Tomada de decisão.


- Desvio do fluxo.



- Indicação de sentido do fluxo.

## Principais comandos utilizados:

**void** – É utilizado para declarar rotinas principais ou sub-rotinas que não retornam resultado.
Ex: - void inicio()

**main** – Corpo do programa principal.
Ex: - void main()

**if** – Comando condicional (se). Executa o comando imediatamente abaixo ou o conjunto de comandos entre chaves ({ }) caso a condição declarada seja verdadeira.

```
Ex: - if (botão==1)
        led =1;
```

**else** – Comando condicional (caso contrário). É utilizado junto com o comando "if" mostrado acima. Executa o comando imediatamente abaixo ou o conjunto de comandos entre chaves ({ }) caso a condição declarada seja falsa.

```
  Ex: - if (botão==1)
          led =1;
      else
          led =0;
```

**while** – Laço de programa. É utilizado para executar um comando ou um conjunto de comandos entre chaves ({ }) enquanto a condição declarada for verdadeira.
Ex: - while(true) if (botão==1)

**for** – É geralmente utilizado como laço fechado em contadores para contagem de tempo ou "delays". Observe no exemplo abaixo. Para a variável "i", enquanto "i" for menor ou igual a zero, incremente "i".
Ex.: for (i=1;i<=10;++i)

**//** - Faz com que o compilador ignore, na linha de comando, tudo que estiver escrito após as duas barras. É utilizado para comentários no programa. Adicionar comentários no programa é muito importante para um futuro entendimento da lógica de programação, caso seja necessária uma alteração ou manutenção do programa.

**/*** - Inicia uma sequência de múltiplas linhas de comentários. É encerrado com o comando " ***/** "

**Obs.** – É muito importante não se esquecer de colocar o sinal ";" no final da linha onde for utilizado comando incondicional, e também nas declarações de variáveis, pois, caso contrário, gerará um erro de compilação.
Ex: - led =1;
int contador;
int contador;

## Tipos de variáveis em linguagem C

| | | Faixa | | |
|---|---|---|---|---|
| **Tipo** | **Tamanho** | **Unsigned** | **Signed** | **Dígitos** |
| **int1** | 1 bit | 0 a 1 | N/A | 1/2 |
| **int8** | 8 bits | 0 a 255 | -128 a 127 | 2-3 |
| **int16** | 16 bits | 0 a 65535 | -32768 a 32767 | 4-5 |
| **int32** | 32 bits | 0 a 4294967295 | -2147483648 a 2147483647 | 9-10 |
| **float32** | 32 bits float | $-1.5 \times 10^{45}$ a $3.4 \times 10^{38}$ | | 7-8 |

| C Standard | Default |
|---|---|
| short | int1 |
| char | unsigned int8 |
| int | int8 |
| long | int16 |
| long long | int32 |
| float | float32 |

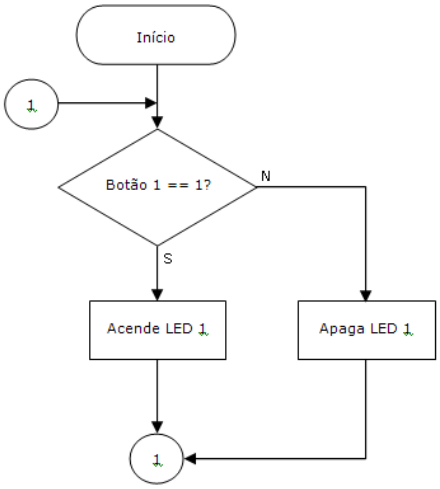
## Operadores da linguagem C

| | |
|---|---|
| + | Operador de Adição |
| += | Operador de atribuição de adição, x+=y, é o mesmo que x=x+y |
| &= | Operador de atribuição AND bit a bit, x&=y, é o mesmo que x=x&y |
| & | Operador de Endereço do operando |
| & | Operador AND bit a bit |
| ^= | Operador de atribuição EXOR bit a bit, x^=y, é o mesmo que x=x^y |
| ^ | Operador EXOR bit a bit |
| l= | Operador de atribuição OR bit a bit,, xl=y, é o mesmo que x=xly |
| l | Operador OR bit a bit |
| ?: | Operador de expressão condicional |
| - - | Decremento |
| /= | Operador de atribuição de divisão, x/=y, é o mesmo que x=x/y |
| / | Operador de divisão |
| == | Igualdade |
| > | Operador "maior que" |
| >= | Operador "maior ou igual" |
| ++ | Incremento |
| * | Operador de conteúdo do endereço do operando |
| != | Inigualdade (diferente de) |
| <<= | Oper. de atribuição de deslocam. p/ esquerda , x<<=y, é o mesmo que x=x<<y |
| < | Operador "menor que" |
| << | Operador de deslocamento p/ esquerda |
| <= | Operador "menor ou igual" |
| && | Operador lógico AND |
| ! | Operador lógico NOT |
| ll | Operador lógico OR |
| %= | Operador de atribuição de Módulos x%=y, é o mesmo que x=x%y |
| % | Operador de Módulos |
| *= | Operador de atribuição de multiplicação, x*=y, é o mesmo que x=x*y |
| * | Operador de multiplicação |
| ~ | Operador de "complemento de um" |
| >>= | Oper. de atribuição de deslocam. p/ direita, x>>=y, é o mesmo que x=x>>y |
| >> | Operador de deslocamento p/ direita |
| -> | Ponteiro de elemento de estrutura |
| -= | Operador de atribuição de subtração |
| - | Operador de subtração |
| sizeof | Determina tamanho em bytes de operandos |

## Programas de exemplo para compilador CCS

### Exemplo 1 – Programa para acender um Led quando um botão for pressionado :

Fluxograma:




Observe no fluxograma ao lado o teste do bit botão 1. Caso o teste seja verdadeiro (true), o LED 1 será acionado, caso contrário, o LED 1 será desligado. A condição contrária do teste do bit ou variável sugere o uso da estrutura "IF – ELSE" conforme mostrado abaixo :

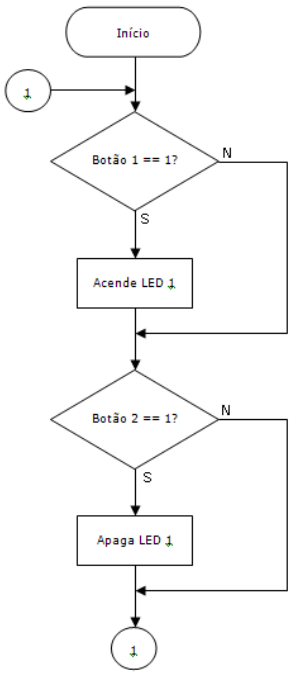
Programa em linguagem C :

```
void main(){                    // Programa principal

set_tris_a(0b11111111);         // Definição das entradas e saídas
set_tris_b(0b00000000);         // (PORT_A = entrada e PORT_B = saída)

PORT_A = 0;                     // Inicialização das variáveis
PORT_B = 0;

//------------------------------------------- Laço de repetição ---------------------------------  ----------------------

while(true){                    // Laço fechado para execução contínua do programa

if (botao1){                    // Se botao1 for pressionado (nível lógico 1) – Acende Led
led1=1;
}

else{                           // Caso contrário (botao1 solto - nível lógico 0) – Apaga Led
led1=0;
}

}
}
```

---

### Exemplo 2 – Programa para acender um Led quando um botão for pressionado e apagar em outro botão:

Fluxograma:




Observe no fluxograma o teste do bit botão 1. Caso o teste seja verdadeiro (true), o LED 1 será acionado. É feito então o teste do bit botão 2. Caso esse seja verdadeiro (true), então o LED 1 será apagado. Nesse caso, poderemos, simplesmente, utilizar a estrutura "IF", pois somente é executado o comando em caso positivo,conforme mostrado abaixo :

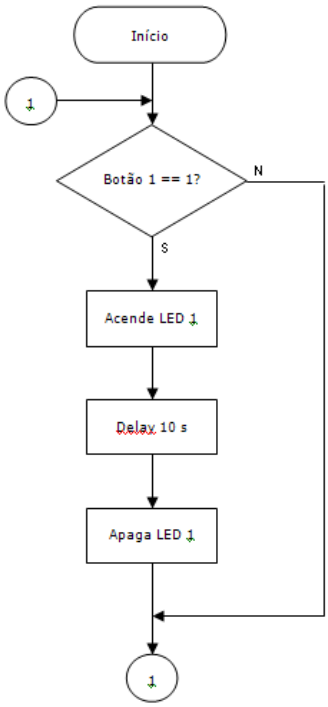
Programa em linguagem C :

```
void main(){                        // Programa principal

set_tris_a(0b11111111);             // Definição das entradas e saídas (PORT_A = entrada e PORT_B = saída)
set_tris_b(0b00000000);

PORT_A = 0;                         // Inicialização das variáveis
PORT_B = 0;

//---------------------------------------------- Laço de repetição  ---------------------------------  ----------------------

while(true){                        // Laço fechado para execução contínua do programa

if (botao1){                        // Se botao1 for pressionado (nível lógico 1) – Acende Led

led1=1;

}

if (botao2){                        // Se botao2 for pressionado (nível lógico 1) – Apaga Led

led1=0;

}

}
}
```

---

Exemplo 3 – Programa para acender um Led por 10 segundos quando um botão for pressionado :

Fluxograma:



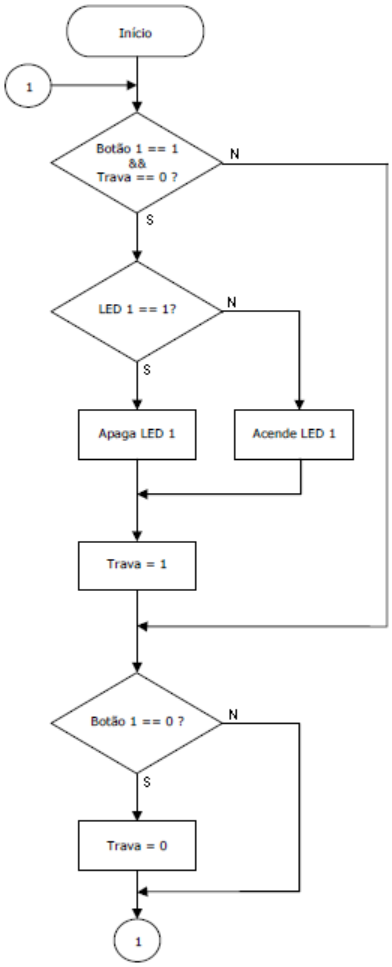

Observe no fluxograma o teste do bit botão 1. Caso o teste seja verdadeiro (true), o LED 1 será acionado, permanecendo aceso por 10 segundos e depois será apagado. Nesse caso, novamente, como visto anteriormente, podemos simplesmente utilizar a estrutura "IF",pois somente é executado o comando caso o teste seja positivo, conforme mostrado abaixo :

Programa em linguagem C :

```
void main(){                        // Programa principal

set_tris_a(0b11111111);             //  Definição das entradas e saídas
set_tris_b(0b00000000);            // (PORT_A = entrada e PORT_B = saída)

PORT_A = 0;                         // Inicialização das variáveis
PORT_B = 0;

//---------------------------------------------- Laço de repetição  ----------------------

while(true){                        // Laço fechado para execução contínua do programa

if (botao1){                        // Se botao1 for pressionado (nível lógico 1) – Acende Led por 10s

led1=1;
delay_ms(10000);
led1=0;
}

}
}
```

Exemplo 4 – Programa para acender e apagar um Led pressionando-se o mesmo botão :

Fluxograma:



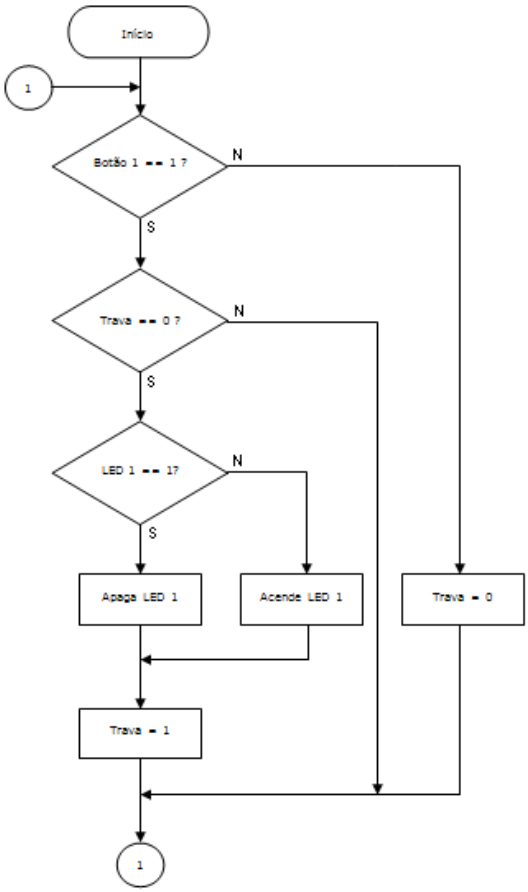

Nesse caso, torna-se necessário criar uma trava lógica, que permite executar apenas um comando para cada vez que o botão for pressionado. Sem a utilização dessa trava, o programa acenderá e apagará o Led continuamente, sendo impossível definir o estado desejado. Será utilizado o bit "trava", o qual habilitará a execução de apenas um comando (acender ou apagar) quando estiver em nível lógico 0 (zero).

As declarações das variáveis devem ser feitas previamente, ao início do programa principal:

Programa em linguagem C :

```
--------------------------------------------------------------------------------
int flag_1=0;              // Declaração das variáveis utilizadas
#bit trava = flag_1.0      // "trava" é o bit número 0 (zero) da variável
                           //"flag_1"
--------------------------------------------------------------------------------

void main(){               // Programa principal

set_tris_a(0b11111111);    // Definição das entradas e saídas (PORT_A = entrada e PORT_B = saída)
set_tris_b(0b00000000);

PORT_A = 0;                // Inicialização das variáveis
PORT_B = 0;

//-------------------------------------------- Laço de repetição  ----------------------------------

while(true){               // Laço fechado para execução contínua do programa

if ((botao1)&&(!trava)){   // Se botão estiver pressionado (nível 1) E trava = 0

  if (led1){               // Se led estiver aceso, então apaga led
  led1=0;
  }
  else{                    // Caso contrário, led apagado, então acende led
  led1=1;
  }

trava=1;                   // Marca trava = 1 para desabilitar a execução de mais comandos
}

if (!botao1) {             // Se botão estiver solto (nível 0), então libera trava (faz com que trava=0)
trava=0;                   // liberando novas execuções.
}

}
}
```

Uma outra solução é mostrada na figura ao lado.




Fluxograma:

Nesse caso, é feito primeiramente o teste do bit "botão", sendo que a variável "trava" é levada a "um" quando o estado do LED for alterado. Isso faz com que o estado da variável de saída LED seja alterado uma única vez quando o botão estiver pressionado. Observe que a lógica do programa é a mesma, apenas alterando a ordem de teste dos bits.

As declarações das variáveis também devem ser feitas previamente, ao início do programa principal:

Programa em linguagem C :

```
-------------------------------------------------------------------
int flag_1=0;              // Declaração das variáveis utilizadas
#bit trava = flag_1.0      // "trava" é o bit número 0 (zero) da
                           //variável "flag_1"
-------------------------------------------------------------------

void main(){               // Programa principal

set_tris_a(0b11111111);    // Definição das entradas e saídas (PORT_A = entrada e PORT_B = saída)
set_tris_b(0b00000000);

PORT_A = 0;                // Inicialização das variáveis
PORT_B = 0;

//-------------------------------------------- Laço de repetição  ----------------------------------

while(true){               // Laço fechado para execução contínua do programa

if (botao1){               // Se botão estiver pressionado (nível 1)

  if(!trava){              // Se trava = 0  (nível 0, ou seja, execução liberada)

    if (led1){             // Se led estiver aceso, então apaga led
    led1=0;
    }
    else{                  // Caso contrário, led apagado, então acende led
    led1=1;
    }

  trava=1;                 // Marca trava = 1 para desabilitar a execução de mais comandos
  }
}

else {                     // Se botão estiver solto (nível 0), então libera trava (faz com que trava=0)
trava=0;                   // liberando novas execuções.
}


}
}
```

```
/*
-----   Cabeçalho para programa em linguagem C - Compilador CCS - Rev.2   ----

Microcontroladores PIC - PIC16F628A  -- Interrupção Timer 0 -- Tratamento Manual

------   Prof. Valdir Dugo Zaragoza      ------                28/07/2015
*/
//                        Inicialização
//-------------------------------------------------------------------------
#include <16F628A.h>

//Osc. cristal XT 4 Mhz, Watch Dog Timer Off, Power Up Timer On e Code Protect Off.

//#FUSES XT,NOWDT,PUT,NOPROTECT   // Usar para PIC16F84A
#FUSES XT,NOWDT,MCLR,PUT,NOPROTECT,NOCPD,NOBROWNOUT,NOLVP // Usar para PIC16F628A

#use delay(clock=4M)

//                      Definições de SFR's
//-------------------------------------------------------------------------
//Bytes:
//*
#byte TIMER_0 = 0x01
#byte STATUS = 0x03
#byte FSR = 0x04
#byte PORT_A = 0x05
#byte PORT_B = 0x06
#byte PCLATH = 0x0A
#byte INTCON = 0x0B

//Bit's:
#bit T0IF = INTCON.2
//*/
//                      Definições de Flag's
//-------------------------------------------------------------------------
//Bytes:

//int flag_1=0;

//Bit's:
//#bit flag_250u = flag_1.0


//                      Declarações de entradas
//-------------------------------------------------------------------------
#bit botao1 = PORT_A.0
//#bit botao2 = PORT_A.1
//#bit botao3 = PORT_A.2
//#bit botao4 = PORT_A.3
//#bit botao5 = PORT_A.4

//                      Declarações de saídas
//-------------------------------------------------------------------------
#bit Led1 = PORT_B.0
//#bit Led2 = PORT_B.1
//#bit Led3 = PORT_B.2
//#bit Led4 = PORT_B.3
//#bit Led5 = PORT_B.4
//#bit Led6 = PORT_B.5
//#bit Led7 = PORT_B.6
//#bit Led8 = PORT_B.7

//                      Declarações de Variáveis
//-------------------------------------------------------------------------

//utilizadas p/ salvar contexto:
//static int W_TEMP, STATUS_TEMP, FSR_TEMP, PCLATH_TEMP;

// Uso no programa:

//int x;
//                      Inicialização das Portas
//-------------------------------------------------------------------------
#use fast_io(a)
#use fast_io(b)
```

```
//                              Sub-rotinas
//-------------------------------------------------------------------------
//        ----------------   Interrupção do Timer_0   -------------------

//#int_timer0          // Usada no tratamento automático da interrupção por Timer 0
/*#inline
void int_timer_0(){
T0IF = 0;
TIMER_0=TIMER_0+6;           //GERA OVERFLOW DO TIMER_0 A CADA 250uS

//                              Adicionar o tratamento da int timer0 abaixo:
//flag_250u=1;

}
*/
//        --------------------   Salva Contexto   -------------------------
//void salva_contexto() {
/*#asm
     MOVWF W_TEMP
     SWAPF STATUS,W
     MOVWF STATUS_TEMP
     MOVF  FSR,W
     MOVWF FSR_TEMP
     MOVF  PCLATH,W
     MOVWF PCLATH_TEMP
     CLRF  PCLATH
     CLRF  STATUS
#endasm
}
*/
//        --------------------   Restaura Contexto   -------------------------
/*void restaura_contexto() {
#asm
     MOVF  PCLATH_TEMP,W
     MOVWF PCLATH
     MOVF  FSR_TEMP,W
     MOVWF FSR
     SWAPF STATUS_TEMP,W
     MOVWF STATUS
     SWAPF W_TEMP,F
     SWAPF W_TEMP,W
#endasm
}
*/
//-----------------------------  Programa Principal    --------------------------

void main(){        // Programa principal

set_tris_a(0b11111111);      // Definição das entradas e saídas (PORT_A = entrada e PORT_B = saída)
set_tris_b(0b00000000);

//setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);//setup_wdt(WDT_18MS);  // Overflow 256us e WDT não utilizado

PORT_A = 0;         // Inicialização das variáveis
PORT_B = 0;

//enable_interrupts(GLOBAL|INT_RTCC);   // Habilita interrupções – Global (tratamento manual) e Timer 0

//-----------------------------  Laço de repetição   ----------------------------

while(true){
}
}
//        ---------   Tratamento Global das Interrupções   --------------
/*#int_global         // Interrupção Global (tratamento manual)
void trata_int(void){

     salva_contexto();
     if (T0IF) {
     int_timer_0();
     }
     restaura_contexto();
}
*/
```

```
/*
-------    Cabeçalho para programa em linguagem C - Compilador XC8   ------
Microcontroladores PIC - PIC16F628A --
------ Prof. Valdir Dugo Zaragoza ------ 26/07/2015
*/
// Inicialização
//-----------------------------------------------------------------------------
#include <PIC16F628A.H> //Arquivo de cabeçalho do PIC16F628A.
#include <xc.h>         //Arquivo de cabeçalho - Diretivas XC8

// FUSES
//Osc. cristal XT 4 Mhz, Watch Dog Timer Off, Power Up Timer On e Code Protect Off.
 #pragma config FOSC = XT     //Definido Oscilador a cristal <= 4MHz.
 #pragma config MCLRE = ON    //Master Clear ON.
 #pragma config CP = OFF      //Proteção contra leitura da Memória de Porgrama - OFF.
 #pragma config CPD = OFF     //Proteção contra leitura da Memória de Dados - OFF.
 #pragma config WDTE = OFF    //Desabilita o Watchdog Timer (WDT).
 #pragma config PWRTE = ON    //Habilita o Power-up Timer (PWRT).
 #pragma config BOREN = OFF   //Brown-out Reset (BOR) habilitado somente no hardware.
 #pragma config LVP = OFF     //Desabilita o Low Voltage Program.

//#define _XTAL_FREQ 4000000    //Define frequência do Oscilador p/ uso do Delay

// Declarações de Variáveis
//-----------------------------------------------------------------------------

// Uso no programa:

//unsigned char contador=0;

// Definições de Flag's
//-----------------------------------------------------------------------------
//Bit's:

//short flag250u = 0;  //Bit de flag

// Declarações de entradas
//-----------------------------------------------------------------------------
//#define  botao1 PORTAbits.RA0    //
//#define  botao2 PORTAbits.RA1    //
//#define  botao3 PORTAbits.RA2    //
//#define  botao4 PORTAbits.RA3    //
//#define  botao5 PORTAbits.RA4    //

// Declarações de saídas
//-----------------------------------------------------------------------------
//#define  Led1 PORTBbits.RB0      //
//#define  Led2 PORTBbits.RB1      //
//#define  Led3 PORTBbits.RB2      //
//#define  Led4 PORTBbits.RB3      //
//#define  Led5 PORTBbits.RB4      //
//#define  Led6 PORTBbits.RB5      //
//#define  Led7 PORTBbits.RB6      //
//#define  Led8 PORTBbits.RB7      //

// --------- Tratamento Global das Interrupções --------------
/*
void interrupt trata_int(void){  //  Subrotina de tratamento de Interrupções

   INTCONbits.GIE=0;

   if (INTCONbits.T0IF) {  //Interrupção Overflow do TIMER0

      // 4/4MHz = 1us (ciclo de máquina)
      // 256 - 6 = 250 (contagem)
      // Prescaler = 1:1
      // t = ciclo de maq * Prescaler * contagem
      // t = 1us * 1 * 250 = 250 us
   TMR0 = TMR0+6;
   INTCONbits.T0IF=0;
   flag250u = 1;
}
   INTCONbits.GIE=1;
}
*/
```

```
//----------------------------------- Programa Principal ----------------------

void main(void){ // Programa principal
TRISA = (0b11111111); // Definição das entradas e saídas (PORT_A = entrada e PORT_B = saída)
TRISB = (0b00000000);

PORTA = 0; // Inicialização das variáveis
PORTB = 0;

//------------------------------------ Laço de repetição -------------------------

while(1){

//--------------   Seu programa entra aqui!  ----------------------------

//__delay_ms(1000);             //aguarda 1 seg


  }

}
```

Cabeçalho para programação em linguagem C – Compilador CC5X

```
// Definição do Microcontrolador
//-------------------------------------------------------------------------------------------------
//#pragma chip PIC16F84a   // Para ser usado com compilador associado ao MPLab
//#include <16f84a.h>       // Para ser usado com compilador – modo texto
//-------------------------------------------------------------------------------------------------
//
// Definição das sub-rotinas
//-------------------------------------------------------------------------------------------------
//#include <time.c>          // Sub-rotina time.c
//
//-------------------------------------------------------------------------------------------------
// Definição dos terminais – Podemos atribuir um nome a um dado terminal, como botao, sensor, motor, etc.
//-------------------------------------------------------------------------------------------------
//#pragma bit botao1 @ PORTA.0
//#pragma bit sensor1 @ PORTA.1
//#pragma bit led1 @ PORTB.0
//#pragma bit motor1 @ PORTB.1
//
// Configuração dos Fuses internos (PWRTE, LVP, BODEN, WDT, Oscilador, etc.)
//-------------------------------------------------------------------------------------------------
//#define CP_off |= 0x3F21    // Utilizado com PIC16F628a
//#define CP_off |= 0x3FF1     // Utilizado com PIC16F84a
//
#pragma  config WDTE=off,FOSC=XT,PWRTE=on,CP_off
//
// Início do programa principal  -  Cuidado para não esquecer o  " ; " no final de cada comando
//-------------------------------------------------------------------------------------------------
void main()
 {
//
// Declaração de variáveis (int, char, uns, etc.)
//-------------------------------------------------------------------------------------------------
 // int x;
//  int c;
//
// Configuração dos Registradores internos SFR's
//-------------------------------------------------------------------------------------------------
//  TRISA =0b11111111;   //BINARIO
//  TRISB =0x00;      //HEXADECIMAL
//  RP0=0;             //Formato DECIMAL  -  DEFINE O BANCO 0
//  CMCON=0x07;    //Deve obrigatoriamente ser usado com o PIC16F628a
//  GIE=0;             //DESABILITA TODAS INTERRUPÇÔES
//
// Inicialização das variáveis
//-------------------------------------------------------------------------------------------------
//  PORTB=0;
//x=0;
//c=0;
//
// Laço principal – executado infinitamente
//-------------------------------------------------------------------------------------------------
  while(1)
  {

//------------------------------ Corpo do programa ----------------------------------------------




//
//------------------------------ Final do programa principal ------------------------------------

   }
 }
```

```
;*****************************************************************************
;                                CABEÇALHO P/ PROGRAMA EM ASSEMBLER COMPILADOR MPASM (MPLAB)
;
;Prof. Valdir D. Zaragoza                                        27/06/2013
;*****************************************************************************
;                                    ARQUIVOS DE DEFINICOES
;*****************************************************************************
#INCLUDE <P16F84A.INC>      ;ARQUIVO PADRAO MICROCHIP PARA O PIC16F84A
;#INCLUDE <P16F628A.INC>     ;ARQUIVO PADRAO MICROCHIP PARA O PIC16F628A
;*****************************************************************************
;                                    BITS DE CONFIGURACAO
;*****************************************************************************
__CONFIG _CP_OFF&_PWRTE_ON&_WDT_OFF&_XT_OSC                                    ;  PIC16F84A
;__CONFIG _BOREN_OFF&_CP_OFF&_PWRTE_ON&_WDT_OFF&_LVP_OFF&DATA_CP_OFF&_MCLRE_ON&_XT_OSC;PIC16F628A
;*****************************************************************************
;                                    PAGINACAO DA MEMORIA
;*****************************************************************************
;COMANDOS PARA ALTERACAO DE PAGINA DE MEMORIA
#DEFINE     BANK0   BCF     STATUS,RP0      ;SETA BANCO 0 DE MEMORIA
#DEFINE     BANK1   BSF     STATUS,RP0      ;SETA BANCO 1 DE MEMORIA
;*****************************************************************************
;                                    VARIAVEIS
;*****************************************************************************
;ENDERECOS DAS VARIAVEIS UTILIZADAS PELO SISTEMA

            CBLOCK  0x0C                    ;ENDERECO INICIAL DA MEMORIA DO USUARIO p/ PIC16F84A
;           CBLOCK  0x20                    ;ENDERECO INICIAL DA MEMORIA DO USUARIO p/ PIC16F628A

            ENDC                                                ;FIM DO BLOCO DE MEMORIA
;*****************************************************************************
;                                    CONSTANTES
;*****************************************************************************
;CONSTANTES UTILIZADAS PELO SISTEMA
;VAR!                       EQU     .200            ;INICIALIZACAO
;*****************************************************************************
;                                    ENTRADAS
;*****************************************************************************
;PINOS QUE SERAO UTILIZADOS COMO ENTRADA
#DEFINE  BOTAO   PORTA,1                    ;1 --> PRESSIONADO
                                            ;0 --> LIBERADO
;*****************************************************************************
;                                    SAIDAS
;*****************************************************************************
;                   PINOS QUE SERAO UTILIZADOS COMO SAIDA
#DEFINE    LED      PORTB,0                 ;0 --> APAGADO
                                            1 --> ACESO
;*****************************************************************************
;                                    VETOR DE RESET
;*****************************************************************************
            ORG     0x00                    ;ENDERECO INICIAL DE PROCESSAMENTO
            GOTO    INICIO
;*****************************************************************************
;                                    INTERRUPCAO
;*****************************************************************************
;AS INTERRUPCOES
            ORG     0x04                                        ;ENDERECO INICAL DA INTERRUPCAO
            RETFIE                                              ;RETORNA DA INTERRUPCAO
;*****************************************************************************
;                                    INICIO
;*****************************************************************************
INICIO:
            BANK1                                               ;ALTERA PARA BANCO1
            MOVLW   B'11111111'
            MOVWF   TRISA               ;DEFINE RA1 COMO ENTRADA e DEMAIS
                                                                ;COMO SAIDA
            MOVLW   B'00000000'
            MOVWF   TRISB                           ;DEFINE O PORTB COMO SAIDA

            MOVLW   B'10000000'
            MOVWF   OPTION_REG                      ;PULL_UPS DESABILIADOS <7>
                                                    ;DEMAIS  BITS IRRELEVANTES
            MOVLW   B'00000000'
            MOVWF   INTCON                          ;CHAVE GERAL DE INTERRUPCAO DESATIVADA
                                                    ;DEMAIS  BITS IRRELEVANTES
            BANK0                                   ;ATIVA BANCO 0
;           MOVLW   B'00000111'                     ; somente p/ PIC16F628A - DESTIVA COMPARADOR INTERNO
;           MOVWF   CMCON                           ;somente p/ PIC16F628A
;
;*****************************************************************************
;                                    INICIALIZACAO DAS VARIAVEIS
;*****************************************************************************
CLRF        PORTA                           ;LIMPA PORTA
            CLRF    PORTB                   ;LIMPA PORTB
;*****************************************************************************
MAIN:

            GOTO    MAIN                    ;DESVIA
;*****************************************************************************
END                                                             ;FIM DO PROGRAMA
;*****************************************************************************
```

**SPECIAL REGISTERS SUMMARY BANK0**

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on POR Reset(1) | Details on Page |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bank 0** | | | | | | | | | | | |
| 00h | INDF | Addressing this location uses contents of FSR to address data memory (not a physical register) | | | | | | | | xxxx xxxx | 28 |
| 01h | TMR0 | Timer0 Module's Register | | | | | | | | xxxx xxxx | 45 |
| 02h | PCL | Program Counter's (PC) Least Significant Byte | | | | | | | | 0000 0000 | 28 |
| 03h | STATUS | IRP | RP1 | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C | 0001 1xxx | 22 |
| 04h | FSR | Indirect Data Memory Address Pointer | | | | | | | | xxxx xxxx | 28 |
| 05h | PORTA | RA7 | RA6 | RA5 | RA4 | RA3 | RA2 | RA1 | RA0 | xxxx 0000 | 31 |
| 06h | PORTB | RB7 | RB6 | RB5 | RB4 | RB3 | RB2 | RB1 | RB0 | xxxx xxxx | 36 |
| 07h | — | Unimplemented | | | | | | | | — | — |
| 08h | — | Unimplemented | | | | | | | | — | — |
| 09h | — | Unimplemented | | | | | | | | — | — |
| 0Ah | PCLATH | — | — | — | Write Buffer for upper 5 bits of Program Counter | | | | | ---0 0000 | 28 |
| 0Bh | INTCON | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF | 0000 000x | 24 |
| 0Ch | PIR1 | EEIF | CMIF | RCIF | TXIF | — | CCP1IF | TMR2IF | TMR1IF | 0000 -000 | 26 |
| 0Dh | — | Unimplemented | | | | | | | | — | — |
| 0Eh | TMR1L | Holding Register for the Least Significant Byte of the 16-bit TMR1 Register | | | | | | | | xxxx xxxx | 48 |
| 0Fh | TMR1H | Holding Register for the Most Significant Byte of the 16-bit TMR1 Register | | | | | | | | xxxx xxxx | 48 |
| 10h | T1CON | — | — | T1CKPS1 | T1CKPS0 | T1OSCEN | $\overline{T1SYNC}$ | TMR1CS | TMR1ON | --00 0000 | 48 |
| 11h | TMR2 | TMR2 Module's Register | | | | | | | | 0000 0000 | 52 |
| 12h | T2CON | — | TOUTPS3 | TOUTPS2 | TOUTPS1 | TOUTPS0 | TMR2ON | T2CKPS1 | T2CKPS0 | -000 0000 | 52 |
| 13h | — | Unimplemented | | | | | | | | — | — |
| 14h | — | Unimplemented | | | | | | | | — | — |
| 15h | CCPR1L | Capture/Compare/PWM Register (LSB) | | | | | | | | xxxx xxxx | 55 |
| 16h | CCPR1H | Capture/Compare/PWM Register (MSB) | | | | | | | | xxxx xxxx | 55 |
| 17h | CCP1CON | — | — | CCP1X | CCP1Y | CCP1M3 | CCP1M2 | CCP1M1 | CCP1M0 | --00 0000 | 55 |
| 18h | RCSTA | SPEN | RX9 | SREN | CREN | ADEN | FERR | OERR | RX9D | 0000 000x | 72 |
| 19h | TXREG | USART Transmit Data Register | | | | | | | | 0000 0000 | 77 |
| 1Ah | RCREG | USART Receive Data Register | | | | | | | | 0000 0000 | 80 |
| 1Bh | — | Unimplemented | | | | | | | | — | — |
| 1Ch | — | Unimplemented | | | | | | | | — | — |
| 1Dh | — | Unimplemented | | | | | | | | — | — |
| 1Eh | — | Unimplemented | | | | | | | | — | — |
| 1Fh | CMCON | C2OUT | C1OUT | C2INV | C1INV | CIS | CM2 | CM1 | CM0 | 0000 0000 | 61 |

**SPECIAL FUNCTION REGISTERS SUMMARY BANK1**

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on POR Reset(1) | Details on Page |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bank 1** | | | | | | | | | | | |
| 80h | INDF | Addressing this location uses contents of FSR to address data memory (not a physical register) | | | | | | | | xxxx xxxx | 28 |
| 81h | OPTION | $\overline{RBPU}$ | INTEDG | T0CS | T0SE | PSA | PS2 | PS1 | PS0 | 1111 1111 | 23 |
| 82h | PCL | Program Counter's (PC) Least Significant Byte | | | | | | | | 0000 0000 | 28 |
| 83h | STATUS | IRP | RP1 | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C | 0001 1xxx | 22 |
| 84h | FSR | Indirect Data Memory Address Pointer | | | | | | | | xxxx xxxx | 28 |
| 85h | TRISA | TRISA7 | TRISA6 | TRISA5 | TRISA4 | TRISA3 | TRISA2 | TRISA1 | TRISA0 | 1111 1111 | 31 |
| 86h | TRISB | TRISB7 | TRISB6 | TRISB5 | TRISB4 | TRISB3 | TRISB2 | TRISB1 | TRISB0 | 1111 1111 | 36 |
| 87h | — | Unimplemented | | | | | | | | — | — |
| 88h | — | Unimplemented | | | | | | | | — | — |
| 89h | — | Unimplemented | | | | | | | | — | — |
| 8Ah | PCLATH | — | — | — | Write Buffer for upper 5 bits of Program Counter | | | | | ---0 0000 | 28 |
| 8Bh | INTCON | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF | 0000 000x | 24 |
| 8Ch | PIE1 | EEIE | CMIE | RCIE | TXIE | — | CCP1IE | TMR2IE | TMR1IE | 0000 -000 | 25 |
| 8Dh | — | Unimplemented | | | | | | | | — | — |
| 8Eh | PCON | — | — | — | — | OSCF | — | $\overline{POR}$ | $\overline{BOR}$ | ---- 1-0x | 27 |
| 8Fh | — | Unimplemented | | | | | | | | — | — |
| 90h | — | Unimplemented | | | | | | | | — | — |
| 91h | — | Unimplemented | | | | | | | | — | — |
| 92h | PR2 | Timer2 Period Register | | | | | | | | 1111 1111 | 52 |
| 93h | — | Unimplemented | | | | | | | | — | — |
| 94h | — | Unimplemented | | | | | | | | — | — |
| 95h | — | Unimplemented | | | | | | | | — | — |
| 96h | — | Unimplemented | | | | | | | | — | — |
| 97h | — | Unimplemented | | | | | | | | — | — |
| 98h | TXSTA | CSRC | TX9 | TXEN | SYNC | — | BRGH | TRMT | TX9D | 0000 -010 | 71 |
| 99h | SPBRG | Baud Rate Generator Register | | | | | | | | 0000 0000 | 73 |
| 9Ah | EEDATA | EEPROM Data Register | | | | | | | | xxxx xxxx | 89 |
| 9Bh | EEADR | EEPROM Address Register | | | | | | | | xxxx xxxx | 90 |
| 9Ch | EECON1 | — | — | — | — | WRERR | WREN | WR | RD | ---- x000 | 90 |
| 9Dh | EECON2 | EEPROM Control Register 2 (not a physical register) | | | | | | | | ---- ---- | 90 |
| 9Eh | — | Unimplemented | | | | | | | | — | — |
| 9Fh | VRCON | VREN | VROE | VRR | — | VR3 | VR2 | VR1 | VR0 | 000- 0000 | 67 |

## SPECIAL FUNCTION REGISTERS SUMMARY BANK2

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on POR Reset(1) | Details on Page |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bank 2** | | | | | | | | | | | |
| 100h | INDF | Addressing this location uses contents of FSR to address data memory (not a physical register) | | | | | | | | xxxx xxxx | 28 |
| 101h | TMR0 | Timer0 Module's Register | | | | | | | | xxxx xxxx | 45 |
| 102h | PCL | Program Counter's (PC) Least Significant Byte | | | | | | | | 0000 0000 | 28 |
| 103h | STATUS | IRP | RP1 | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C | 0001 1xxx | 22 |
| 104h | FSR | Indirect Data Memory Address Pointer | | | | | | | | xxxx xxxx | 28 |
| 105h | — | Unimplemented | | | | | | | | — | — |
| 106h | PORTB | RB7 | RB6 | RB5 | RB4 | RB3 | RB2 | RB1 | RB0 | xxxx xxxx | 36 |
| 107h | — | Unimplemented | | | | | | | | — | — |
| 108h | — | Unimplemented | | | | | | | | — | — |
| 109h | — | Unimplemented | | | | | | | | — | — |
| 10Ah | PCLATH | — | — | — | Write Buffer for upper 5 bits of Program Counter | | | | | ---0 0000 | 28 |
| 10Bh | INTCON | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF | 0000 000x | 24 |
| 10Ch | — | Unimplemented | | | | | | | | — | — |
| 10Dh | — | Unimplemented | | | | | | | | — | — |
| 10Eh | — | Unimplemented | | | | | | | | — | — |
| 10Fh | — | Unimplemented | | | | | | | | — | — |
| 110h | — | Unimplemented | | | | | | | | — | — |
| 111h | — | Unimplemented | | | | | | | | — | — |
| 112h | — | Unimplemented | | | | | | | | — | — |
| 113h | — | Unimplemented | | | | | | | | — | — |
| 114h | — | Unimplemented | | | | | | | | — | — |
| 115h | — | Unimplemented | | | | | | | | — | — |
| 116h | — | Unimplemented | | | | | | | | — | — |
| 117h | — | Unimplemented | | | | | | | | — | — |
| 118h | — | Unimplemented | | | | | | | | — | — |
| 119h | — | Unimplemented | | | | | | | | — | — |
| 11Ah | — | Unimplemented | | | | | | | | — | — |
| 11Bh | — | Unimplemented | | | | | | | | — | — |
| 11Ch | — | Unimplemented | | | | | | | | — | — |
| 11Dh | — | Unimplemented | | | | | | | | — | — |
| 11Eh | — | Unimplemented | | | | | | | | — | — |
| 11Fh | — | Unimplemented | | | | | | | | — | — |

## SPECIAL FUNCTION REGISTERS SUMMARY BANK3

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on POR Reset(1) | Details on Page |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bank 3** | | | | | | | | | | | |
| 180h | INDF | Addressing this location uses contents of FSR to address data memory (not a physical register) | | | | | | | | xxxx xxxx | 28 |
| 181h | OPTION | $\overline{RBPU}$ | INTEDG | T0CS | T0SE | PSA | PS2 | PS1 | PS0 | 1111 1111 | 23 |
| 182h | PCL | Program Counter's (PC) Least Significant Byte | | | | | | | | 0000 0000 | 28 |
| 183h | STATUS | IRP | RP1 | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C | 0001 1xxx | 22 |
| 184h | FSR | Indirect Data Memory Address Pointer | | | | | | | | xxxx xxxx | 28 |
| 185h | — | Unimplemented | | | | | | | | — | — |
| 186h | TRISB | TRISB7 | TRISB6 | TRISB5 | TRISB4 | TRISB3 | TRISB2 | TRISB1 | TRISB0 | 1111 1111 | 36 |
| 187h | — | Unimplemented | | | | | | | | — | — |
| 188h | — | Unimplemented | | | | | | | | — | — |
| 189h | — | Unimplemented | | | | | | | | — | — |
| 18Ah | PCLATH | — | — | — | Write Buffer for upper 5 bits of Program Counter | | | | | ---0 0000 | 28 |
| 18Bh | INTCON | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF | 0000 000x | 24 |
| 18Ch | — | Unimplemented | | | | | | | | — | — |
| 18Dh | — | Unimplemented | | | | | | | | — | — |
| 18Eh | — | Unimplemented | | | | | | | | — | — |
| 18Fh | — | Unimplemented | | | | | | | | — | — |
| 190h | — | Unimplemented | | | | | | | | — | — |
| 191h | — | Unimplemented | | | | | | | | — | — |
| 192h | — | Unimplemented | | | | | | | | — | — |
| 193h | — | Unimplemented | | | | | | | | — | — |
| 194h | — | Unimplemented | | | | | | | | — | — |
| 195h | — | Unimplemented | | | | | | | | — | — |
| 196h | — | Unimplemented | | | | | | | | — | — |
| 197h | — | Unimplemented | | | | | | | | — | — |
| 198h | — | Unimplemented | | | | | | | | — | — |
| 199h | — | Unimplemented | | | | | | | | — | — |
| 19Ah | — | Unimplemented | | | | | | | | — | — |
| 19Bh | — | Unimplemented | | | | | | | | — | — |
| 19Ch | — | Unimplemented | | | | | | | | — | — |
| 19Dh | — | Unimplemented | | | | | | | | — | — |
| 19Eh | — | Unimplemented | | | | | | | | — | — |
| 19Fh | — | Unimplemented | | | | | | | | — | — |

Anotações.

/////// Fuses para PIC16F628A no compilador CCS
/////// Fuses: NOWDT,WDT,PUT,NOPUT,LP,XT,HS,EC_IO,INTRC_IO,INTRC,RC_IO,RC
/////// Fuses: NOMCLR,MCLR,NOBROWNOUT,BROWNOUT,NOLVP,LVP,CPD,NOCPD,PROTECT
/////// Fuses: NOPROTECT

/////// Fuses para PIC16F84A no compilador CCS
/////// Fuses: LP,XT,HS,RC,NOWDT,WDT,PUT,NOPUT,PROTECT,NOPROTECT