

**UNIVERSIDADE FEDERAL DA BAHIA**

**ESCOLA POLITÉCNICA**

**LUCAS MENEZES PEREIRA**

**MATHEUS OLIVER DE CARVALHO CERQUEIRA**

**RELATÓRIO DE SINTESE DE CIRCUITOS DE ESTAÇÃO DE MONITORAMENTO ATMOSFÉRICO USANDO MICROCONTROLADOR ARDUINO**

Salvador

2017

1. **APRESENTAÇÃO**

A linguagem *Assembly* é muito utilizada, tanto para *Software*, quanto para *Hardware*. É a responsável por interfacear os comandos expressos pelo programador para linguagem de máquina, e, consequentemente, configuração dos circuitos internos do dispositivo. Nesse contexto, cada fabricante utiliza essa linguagem em um dispositivo específico, para determinado fim, muitas vezes criando a própria arquitetura (conjunto de atributos que, organizados de tal forma possibilitam à sintaxe da linguagem interagir sobre os circuitos), como um protecionismo de propriedade industrial.

Dentre os dispositivos eletrônicos mais usados, estão os Microprocessadores e os Microcontroladores. Os primeiros contém apenas uma *CPU* (do inglês, Unidade Central de Processamento) interna, servindo como computadores de propósito geral. Enquanto isso, os últimos, além de CPU, contém Memórias, barramentos e periféricos (dispositivos de entrada/saída). Uma Unidade Central de Processamento, por sua vez, é composta de uma ALU (do inglês, Unidade Lógico-Aritmética), registradores, conexões e controlador de dados. Os microcontroladores, portanto, são mais utilizados em tarefas específicas, geralmente relacionadas ao gerenciamento de entrada e saída.

O microcontrolador *PIC* (do inglês, Controlador de Interface Programável), fabricado atualmente pela *Microchip Technology Inc.*, é um dos mais conhecidos, tendo diversas aplicações, com uma comunidade de usuários e desenvolvedores ativa. Junto com o 8051 e a Placa Arduino, é um dos Microcontroladores mais ensinados e porta de entrada ao desenvolvimento de projetos em dispositivos embarcados.

1. **OBJETIVOS**
   1. **Geral**

Apresentar e analisar o Microcontrolador PIC16 e suas possíveis aplicações em projetos.

* 1. **Específicos**
* Introduzir o PIC16;
* Detalhar a Arquitetura;
* Apresentar um projeto prático que utilize a plataforma.

1. **SOBRE O MICROCONTROLADOR**
   1. **Introdução**

Existem vários modelos do circuito integrado, categorizados em Famílias, que podem ser PIC10, PIC12, PIC16, PIC18 e muitos outros, com diferenças principalmente em relação à CPU (tamanho de palavra, ou conjunto de informações que podem ser resgatadas por ciclo de máquina, mais explicado posteriormente), memória e conjunto de mnemônicos (um mnemônico é uma instrução que, para a programação do CI, representa uma configuração de circuito interna, que realiza uma única operação). Todos são atendidos pela IDE em comum, MPLAB, onde os programas, no interesse do relatório, *Assembly*, são escritos, compilados, depurados, e gravados no chio através de um módulo conectado via comunicação USB. Para exemplificar, o modelo abordado será o PIC16F84A.

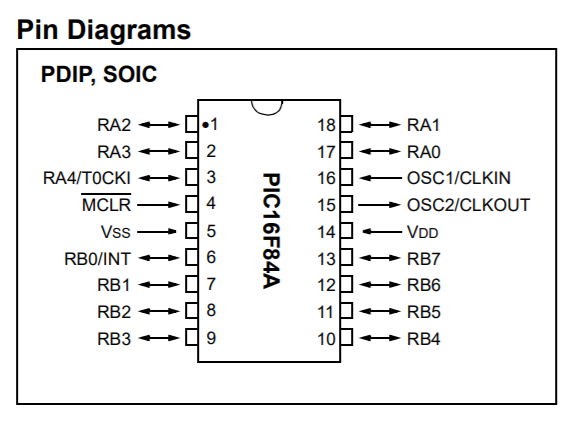


Figura 1 – Diagrama de Pinos do PIC16F84A.

Fonte: Microchip. PIC16F84A Data Sheet. 2001.

O modelo escolhido possui 18 pinos, sendo eles:

* 2 para frequência de operação (CLOCK), um de entrada (16) e um de saída (15);
* 2 de alimentação de circuito, sendo um de maior potencial (14) e um de menor potencial (5), associados a um positivo e um negativo de uma fonte ou bateria, por exemplo;
* 1 reset (4), que limpa o estado do programa e as entradas ao receber valor lógico “0” (tensão de 0V);
* 5 Pinos para o PORTA (lê-se PORT A), sendo pinos bidirecionais de Entrada/Saída (17,18,1,2,3);
* 8 Pinos para o PORTB, sendo pinos bidirecionais de Entrada/Saída (6,7,8,9,10,11,12,13), que também podem ser usados para programação serial ou para ativar interrupções.

Programas escritos para PIC são passados através de um gravador específico, que consiste em um circuito impresso, com uma entrada usb e um conector para pôr e remover o microcontrolador.

* 1. **Arquitetura**

O PIC utiliza arquitetura Harvard, caracterizada pelo acesso à memória de dados separado da memória de programa existindo um barramento para cada componente (dados, programa e entrada/saída), à unidade de controle. Como ele possui uma ULA, todos os comandos internos são baseados em operações matemáticas de soma, subtração, negação, multiplicação e outras.

As memórias de Programa e de Dados, são o que pode se chamar de memórias dedicadas. Abstendo um pouco sobre os tipos de memória (RAM – Memória de Acesso aleatório, ROM – Memória de Apenas Leitura, EEPROM – Memória Programável de Apenas Leitura Eletricamente apagável, e outras), ao se escrever um programa para PIC, ele identifica alguns registradores específicos para o programa e os acessa.

A memória é dividida em dois bancos de registradores. O de Registradores de Propósitos Gerais (GPR), manipulado pelo programador, e o dos Registradores de Funções Especiais, configurados na fabricação. O que determina a escolha de um registrador é o endereço representado na instrução de máquina. Assim, o TRISB (importante para definir se um pino é de entrada ou saída, por exemplo), encontra-se no endereço 86h (os endereços são dispostos de 00h, ou 00000000 até FFh, ou 11111111). Esse endereço é usado na memória de dados ao ser passado como operador de uma função.

Abaixo, alguns Registradores Importantes:

* STATUS: Registrador 8 bits, os quais podem ser citados:
  + Bit 7 - 6: livres;
  + Bit 5: Seletor de Banco (1 para banco 1, 80h-FFh; 0 para banco 0, 00h-7Fh);
  + Bit 2: Informa se o resultado de uma operação aritmética é zero (1 para sim, zero para não);
  + Bit 1 - 0: Carry de uma operação.
* PORTA: Armazena os bits dos pinos de Entrada/Saída A; definido pelo Bit 5 do STATUS (se 0, PORTA, se 1, TRISA). O mesmo ocorre com PORTB;
* TRISA: Define a direção dos Pinos mencionados em PORTA (se Entrada ou Saída);
* INTCON: Tanto para escrita ou leitura, responsável pelo tratamento de interrupções.

Ao executar um programa, cuja ordem aliás é dada de cima para baixo, começando no endereço 0x00h, frequentemente ocorrem interrupções, em sua grande maioria, definidas pelo usuário. São muito úteis para atrasar um programa (delay proposital, para sincronizar com outros dispositivos que sejam controlados pelo PIC, por exemplo), entrar em um loop (cuja mudança de estado seja assíncrona, como um contador), e muitos outros desvios.

A interrupção ocorre quando a chamada no programa é feita, de forma que o endereço da última instrução executada é armazenado, o ponteiro na pilha de execução (outra memória disponível no PIC) é movido para o endereço da instrução que inicia o laço (O endereço usado, por costume dos programadores e por orientação da fabricante, é 0x04h). Ao terminar a condição que mantém o programa no laço, o ponteiro aponta para a posição imediatamente posterior à armazenada antes da interrupção.

* 1. **Dilema CISC x RISC**

Sabendo que qualquer CPU tem um conjunto de instruções (no caso do PIC16 escolhido, será apresentado mais adiante), que é reconhecido e ao qual responde, sendo a base para todos os programas a serem gravados na memória dedicada. Assim, historicamente houve um debate sobre como se daria a evolução tecnológica nos microcontroladores e afins, com o objetivo de respostas cada vez mais rápidas: Aumentar o conjunto de instruções básicas, ou CISC – (do inglês, Computador de Conjunto de de Instruções Complexas), ou prezar por instruções mais simples em um conjunto menor, RISC (do inglês, Computador de Conjunto de Expressões Reduzidas).

Em uma visão superficial, o CISC parece melhor, pois torna a tarefa do programador mais fácil, considerando que as instruções já estão prontas. Entretanto, como, nessa camada de programação, cada instrução representa uma configuração de circuito interno, quanto maior a complexidade da instrução, maior o circuito necessário, e, portanto, maior a dificuldade de implementação. Assim, boa parte dos Microprocessadores, incluindo o PIC, favorece a construção RISC (embora atualmente, os CI’s apresentem uma mistura entre as duas opções).

* 1. **Instruções no PIC16**

Como mencionado, o PIC é um microcontrolador predominantemente RISC, então a dificuldade de se implementar um programa (piscar um LED, por exemplo), é compensada pela pequena quantidade de instruções (35, exatamente).

Para que o processador de qualquer arquitetura computacional realize uma tarefa, é preciso que ele receba um opcode, ou seja, um código de operação. Esse código inclui referências sobre endereços de memória do(s) operando(s) e operador(es) da instrução, número de ciclos (repetições) da operação, etc. O formato dos opcodes varia entre as diferentes arquiteturas de microcontroladores e microprocessadores, sendo que eles são convertidos diretamente em código de máquina.

Entretanto, programar uma máquina, como as ditas acima, através de opcodes é trabalhoso demais para um ser humano, pois não é nada intuitivo (a sintaxe do comando não permite enxergar claramente qual a tarefa realizada pelo mesmo). Para tornar a programação Assembly um tanto mais fácil, são usados os mnemônicos, sendo instruções, as citadas anteriormente, que já permitem ao programador perceber a tarefa realizada pela semântica do comando (assemelha-se à língua inglesa).

Vejamos o seguinte exemplo, que faz o valor 61 em hexadecimal (97 em decimal) ser movido para o registrador AL (da arquitetura x86, Intel).

\*Opcode: IA-32 (B0 61)

\* Mnemônico: MOV AL, 61h

Graças ao Assembler (montador) é possível que uma programação feita unicamente com mnemônicos seja convertida inteiramente para opcodes e, consequentemente, para o código de máquina a ser executado diretamente pelo processador. O montador, portanto, é o principal personagem na interação homem-máquina quando se programa em baixo nível (Assembly).

Cada instrução no PIC16F84A é uma “palavra de 14bits” é dividida num opcode que especifica o tipo da instrução, além de um ou mais argumentos que especificam os operandos e operação. Para cada tipo de instrução é mostrado a seguir a quantidade de bits reservada para cada parte.

* + 1. **Instruções Orientadas por Byte**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 13 |  |  |  |  | 8 | 7 | 6 |  |  |  |  |  | 0 |
| OPCODE | | | | | | d | f | | | | | | |

d (bit do registrador destino do resultado da operação): pode receber 0 para que o destino seja registrador W (acumulador) ou 1 para que o destino seja registrador f.

f: letra para simbolizar qualquer registrador de propósito geral (SRAM) dentre os endereços 0x0C a 0x4F, que pode ser usado na operação.

* + 1. **Instruções Orientadas por Bit**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 13 |  |  | 10 | 9 |  | 7 | 6 |  |  |  |  |  | 0 |
| OPCODE | | | | b | | | f | | | | | | |

b = endereço de 3 bits

f = registrador de propósito geral (como dito em x.2.1.1)

* + 1. **Instruções Orientadas por Literal (Valor numérico)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 13 |  |  |  |  | 8 |  | 6 |  |  |  |  |  | 0 |
| OPCODE | | | | | k (literal) | | | | | | | | |

k: literal constante de 8 bits (0 até 255) usado na operação.

* + 1. **Instruções CALL e GOTO**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 13 |  | 11 | 10 |  |  |  |  |  |  |  |  |  | 0 |
| OPCODE | | | k (literal) | | | | | | | | | | |

k: literal constante de 11 bits (0 até 2047)

* 1. **Conjunto de Instruções do PIC16F84A**

|  |  |  |  |
| --- | --- | --- | --- |
| **INSTRUÇÕES ORIENTADAS POR BYTE** | | **INSTRUÇÕES ORIENTADAS POR BIT** | **INSTRUÇÕES ORIENTADAS POR LITERAIS** |
| **A. Aritméticas** | | | |
| A1) ADDWF | |  | A6) ADDLW |
| A2) DECF | | A7) SUBLW |
| A3) DECFSZ | |  |
| A4) INCF | |
| A5) INCFSZ | |
| A6) SUBWF | |
|  | | **B. Lógicas** |  |
| B1) ANDWF | |  | B5) ANDLW |
| B2) COMF | | B6) IORLW |
| B3) IORWF | | B7) XORLW |
| B4) XORWF | |  |
| **C. Movimentação** | | | |
| C1) MOVF |  | | C3) MOVLW |
| C2) MOVFW |  |
|  |

|  |  |  |
| --- | --- | --- |
| **D. Outras** | | |
| D1) CLRF | D7) BCF | D11) CALL |
| D2) CLRW | D8) BSF | D12) CLRWDT |
| D3) NOP | D9) BTFSC | D13) GOTO |
| D4) RLF | D10) BTFSS | D14) RETFIE |
| D5) RRF |  | D15) RETLW |
| D6) SWAPF | D16) RETURN |
|  | D17) SLEEP |

Tabela 1 – Conjunto Organizado das instruções do PIC16F84A.

A tabela acima mostra todos os 35 mnemônicos do PIC16F84A, organizados de acordo com a função que desempenham e com os tipos de dados que manipulam.

1. **OPERACIONAL**
2. **CONCLUSÃO**
3. **REFERÊNCIAS**

Engineers Garage. Disponível em: <https://www.engineersgarage.com/tutorials/difference-between-microprocessor-and-microcontroller>. Acesso em 27 de Janeiro de 2018.

Microchip. PIC16F84A Data Sheet. 2001.

Wikipedia. Disponível em: <https://pt.wikipedia.org/wiki/Microcontrolador\_PIC>. Acesso em 27 de Janeiro de 2018.