

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI - UFSJ
DEPARTAMENTO DE ENGENHARIA ELÉTRICA - DEPEL

INTRODUÇÃO AOS MICROCONTROLADORES PIC

THIAGO VELOSO GOMES
JOÃO MATHEUS DE OLIVEIRA ARANTES

São João del-Rei
Março de 2007.

Sumário

1	Introdução	2
1.1	Estrutura do microcontrolador PIC 16F628A	2
1.2	Arquitetura Havard e Código RISC	3
1.3	Ciclos de máquina	3
2	O PIC 16F628A	5
3	Mapas de memória	7
4	Registradores Especiais	8
4.1	STATUS	8
4.2	PCON	10
4.3	TRISA e TRISB	10
4.4	PORTA e PORTB	10
5	MPlab	10
5.1	Compilando o Projeto	10
5.2	Gravando o projeto	11
6	Criando um Programa	13
6.1	Arquivos de definição (<i>Include</i>)	13
6.2	Constantes	13
6.3	Define	14
6.4	ORG	14
6.5	END	14
6.6	CBLOCK e ENDC	14
6.7	Registrador WORK (W)	14
6.8	Estruturação básica	14
7	Conjunto de instruções para o PIC 16F628A	15
7.1	Instruções e Argumentos	15
7.2	Lista de Instruções	16
8	Rotinas	17
8.1	Acessando o banco de dados	17
8.2	Lendo e escrevendo nas portas	18
8.3	Fazendo contagem de eventos	18
8.4	Fazendo comparações	18
8.5	Fazendo contagem de tempo	19
9	Exercícios Propostos	19

1 Introdução

O desenvolvimento acelerado e o surgimento de novas tecnologias têm trazido nos últimos anos mais facilidades a vida cotidiana dos seres humanos. Dentre elas estão os dispositivos “inteligentes” presentes em grande parte dos lares, estabelecimentos comerciais, hospitais, indústrias, etc. Esses dispositivos são ditos inteligentes pois conseguem realizar alguma ação autônoma, ou seja, sem a necessidade de um acionamento humano ou semi-autônoma interagindo assim com o homem. Essas ações estão normalmente relacionadas a um estado de um dispositivo de entrada ou saída (periférico), a alguma ação previamente determinada ou a um comando externo. Esses estados são então enviados a um “cérebro”, ou seja, a um *controlador* que irá processar as informações vindas dos periféricos e tomar alguma decisão. Dispositivos inteligentes, comumente chamados de sistemas autônomos, podem ser utilizados para diversos fins, tais como sinalização, controle, segurança, transmissão de dados, etc. Estes sistemas são representados de uma forma genérica na Figura 1.

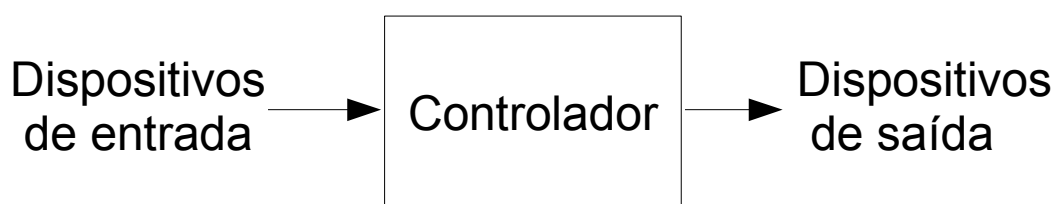


Figura 1: Representação genérica de um sistema autônomo.

Existem diversos dispositivos que podem ser utilizados para dar autonomia a um sistema, como por exemplo: um computador, um Controlador Lógico Programável (CLP) ou um microcontrolador, sendo este último o tema de trabalho desta apostila. **O microcontrolador pode ser definido como um dispositivo eletrônico de tamanho reduzido dotado de uma “inteligência” programável utilizado no controle de processos lógicos.** Existem diversos tipos e fabricantes de microcontroladores. No entanto, no decorrer desse curso, será utilizado o microcontrolador PIC 16F628A da Microship.

1.1 Estrutura do microcontrolador PIC 16F628A

A Figura 2 mostra o diagrama interno do PIC 16F628A. No diagrama de blocos retirado do datasheet da Microchip podem ser visualizadas as diversas partes que compõem o microcontrolador 16F628A. O mesmo é composto pela ULA (Unidade Lógica Aritmética - do inglês ALU), que está diretamente ligado ao registrador W (Work). No canto superior esquerdo temos a memória de programa e saindo desse bloco temos um barramento de 14 bits (Program Bus 14). Mais ao centro está a memória de dados (RAM). Ela já possui o barramento de 8 bits (Data Bus 8). Do lado direito podemos visualizar as portas com todos os seus pinos de I/O. Na parte inferior se encontram os periféricos, tais como a EEPROM, os Timers, o comparador interno, o módulo CCP e a porta serial USART. Um pouco mais ao centro temos o registrador STATUS. Na parte superior temos ainda o contador de linha de programa (Program Counter) e a pilha de oito níveis (Stack). Temos ainda os circuitos internos de reset, osciladores, Watchdog Timer (WDT), Power-up e Brown-out internos.

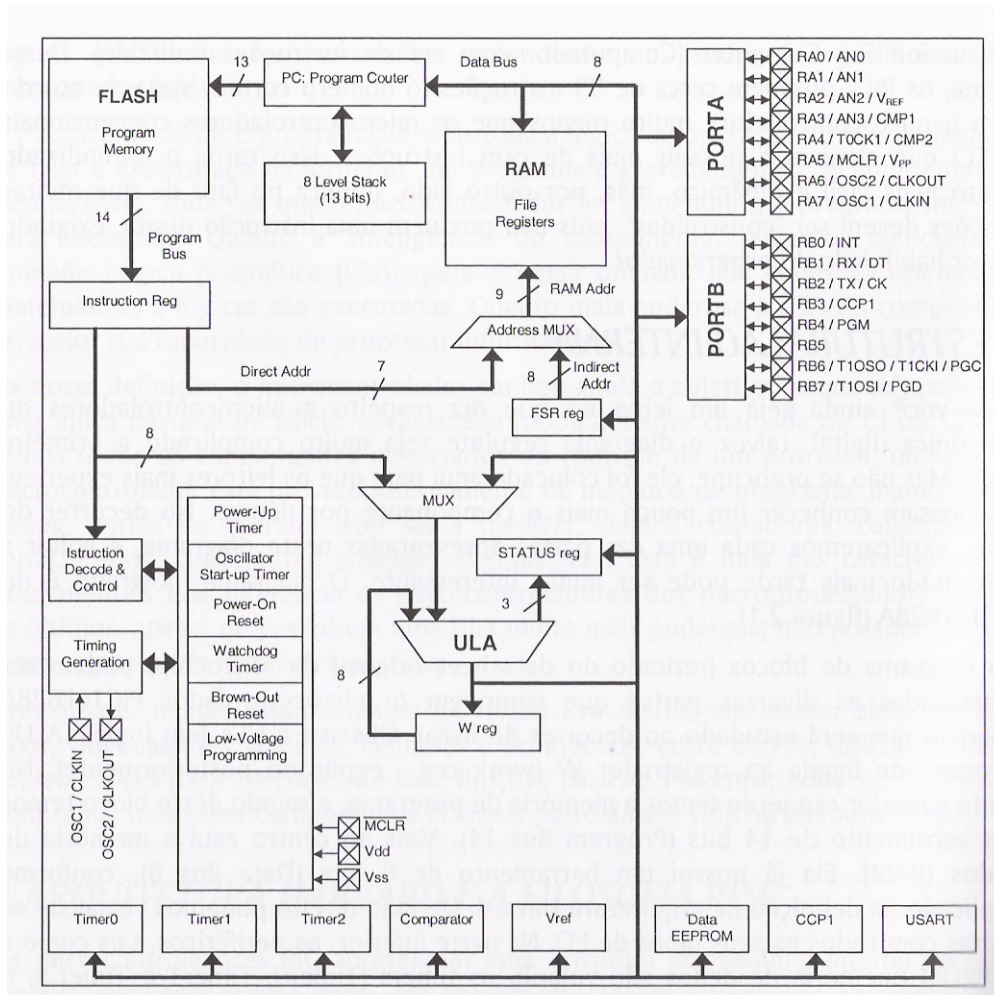


Figura 2: Diagrama interno do PIC16F628A.

1.2 Arquitetura Havard e Código RISC

Na arquitetura tradicional tipo Von Neuman, existe apenas um barramento interno utilizado para instruções e dados. Na arquitetura Havard existe um barramento para instruções e outro para dados. O barramento dos microcontroladores PIC é do tipo Havard sendo que o barramento de dados é de 8 bits e o barramento de instruções pode ser de 12, 14 ou 16.

Os PIC's utilizam a tecnologia RISC (Reduced Instruction Set Computer) e possuem em torno de 35 instruções, o que torna o aprendizado mais fácil e dinâmico.

1.3 Ciclos de máquina

O sinal do clock é internamente dividido por quatro nos microcontroladores PIC. Assim, para um clock externo de 4 MHz, temos um clock interno de 1 MHz e, conseqüentemente, cada ciclo de máquina dura 1 μ s.

A divisão do clock por quatro forma as fases Q1, Q2, Q3, e Q4. O program counter é incrementado automaticamente na fase Q1 do ciclo de máquina e a instrução seguinte é buscada da memória de programa e armazenada no registrador de instruções no ciclo Q4. Ela é decodificada e executada no próximo ciclo, no intervalo de Q1 até Q4. Essa característica de buscar a informação num

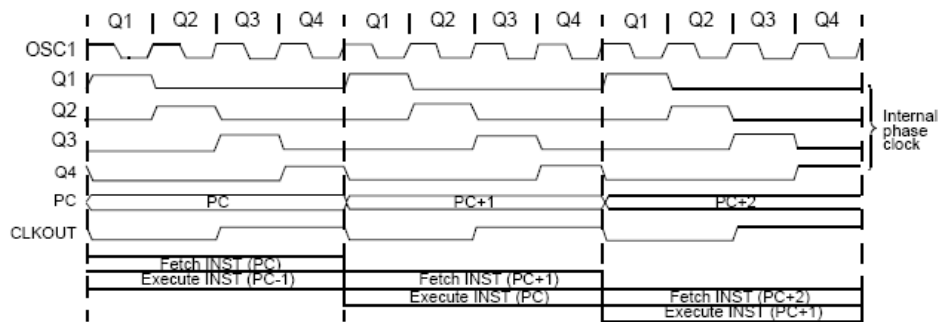
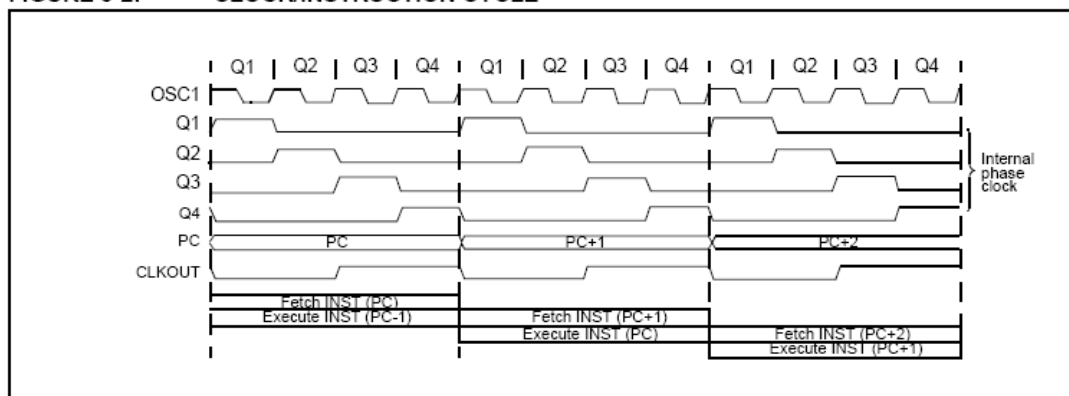


Figura 3: Ciclos de máquina do PIC16F628A.

FIGURE 3-2: CLOCK/INSTRUCTION CYCLE



EXAMPLE 3-1: INSTRUCTION PIPELINE FLOW

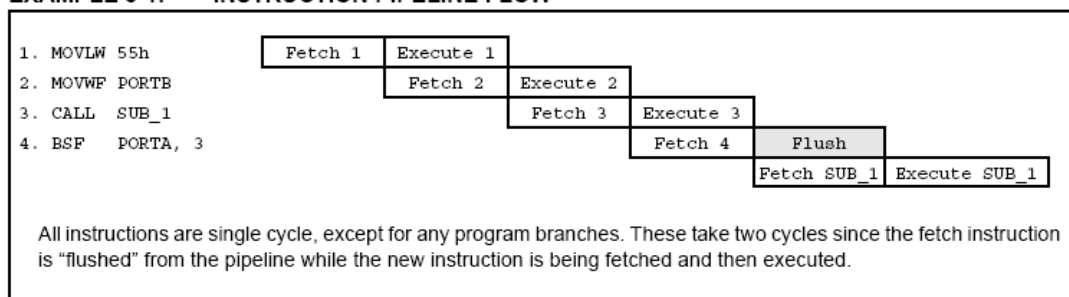


Figura 4: Exemplo de ciclo de máquina do PIC16F628A.

ciclo de máquina e executá-la no próximo é conhecida como PIPELINE. Ela permite que quase todas as instruções sejam executadas em apenas um ciclo, gastando assim $1 \mu s$ (para um clock de 4 MHz) e tornando o sistema muito mais rápido. As únicas exceções referem-se às instruções que geram "saltos" no program counter, como chamadas de rotinas e retornos. Ao executar essas in-

struções, o PIPELINE deve ser primeiramente limpo para depois poder ser carregado novamente com o endereço correto, consumindo para isso dois ciclos de máquina. Esse PIPELINE é facilmente implementado devido arquitetura Havard.

Os diagramas das Figuras 3 e 4 foram retirados do manual da Microchip e demonstram claramente as divisões do ciclo nas quatro fases (Q1 a Q4) e o conceito de PIPELINE.

2 O PIC 16F628A

O PIC 16F628A possui as seguintes características:

- Microcontrolador de 8 bits;
- 2 osciladores internos 37 KHz e 4 MHz;
- 16 portas de I/O configuráveis como entrada ou saída;
- 10 interrupções disponíveis;
- Memória de programação Flash de 2048 Bytes;
- Memória EEPRON interna de 128 Bytes;
- Programação com 14 bits e 35 instruções.

Legenda:

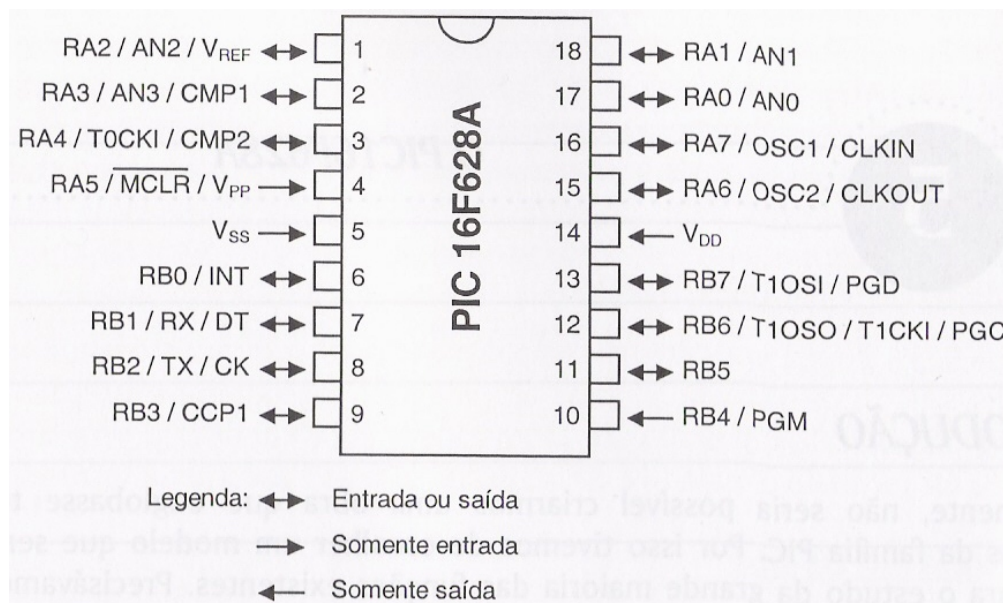


Figura 5: Pinagem do PIC16F628A.

A função de cada pino do microcontrolado é mostrada na Tabelas 1 e 2. A posição dos mesmos é mostrada na Figura 5.

Tabela 1: Nomenclatura de identificação dos pinos do PIC 16F628A.

Número do Pino	Função	Tipo Entrada	Tipo Saída	Descrição
17	RA0	ST	CMOS	I/O digital bidirecional.
	AN0	AN		Entrada analógica para os comparadores.
18	RA1	ST	CMOS	I/O digital bidirecional.
	AN1	AN		Entrada analógica para os comparadores.
1	RA2	ST	CMOS	I/O digital bidirecional.
	AN2	AN		Entrada analógica para os comparadores.
	V_{REF}		AN	Saída da tensão de referência programável.
2	RA3	ST	CMOS	I/O digital bidirecional.
	AN3	AN		Entrada analógica para os comparadores.
	CMP1		CMOS	Saída do comparador 1.
3	RA4	ST	OD	I/O digital bidirecional.
	T0CKI	ST		Entrada externa do contador TMR0.
	CMP2		OD	Saída do comparador 2.
4	RA5	ST		entrada digital.
	MCLR	ST		Master Clear (reset externo. O PIC só funciona quando este pino encontra-se em nível alto.
	V_{PP}			Entrada para tensão de programação (13V).
15	RA6	ST	CMOS	I/O digital bidirecional.
	OSC2		XTAL	Saída para cristal externo.
	CLKOUT		CMOS	Saída com onda quadrada em $\frac{1}{4}$ da frequência imposta em OSC1 quando em modo RC. Essa frequência equivale aos ciclos de máquina internos.
16	RA7	ST	CMOS	I/O digital bidirecional.
	OSC1	XTAL		Entrada para cristal externo.
	CLKIN	ST		Entrada para osciladores externos (híbridos ou RC).
6	RB0	TTL	CMOS	I/O digital bidirecional com pull-up interno.
	INT	ST		Entrada para interrupção externa.
7	RB1	TTL	CMOS	I/O digital bidirecional com pull-up interno.
	RX	ST		Recepção para comunicação USART assíncrona.
	DT	ST	CMOS	Via de dados para comunicação USART síncrona.

Tabela 2: Continuação: Nomenclatura de identificação dos pinos do PIC 16F628A.

8	RB2	TTL	CMOS	I/O digital bidirecional com pull-up interno.
	TX		CMOS	Transmissão para comunicação USART assíncrona.
	CK	ST	CMOS	Via de clock para comunicação USART síncrona.
9	RB3	TTR	CMOS	I/O digital bidirecional com pull-up interno.
	CCP1	ST	CMOS	I/O para o Capture, Compare e PWM.
10	RB4	TTL	CMOS	I/O digital bidirecional com pull-up interno. Interrupção por mudança de estado.
	PGM	ST		Entrada para programação em baixa tensão (5 V).
11	RB5	TTL	CMOS	I/O digital bidirecional com pull-up interno. Interrupção por mudança de estado.
12	RB6	TTL	CMOS	I/O digital bidirecional com pull-up interno. Interrupção por mudança de estado.
	T1OSO		XTAL	Saída para cristal externo para TMR1.
	PGC	ST		Clock da programação serial (ICSP).
13	RB7	TTL	CMOS	I/O digital bidirecional com pull-up interno. Interrupção por mudança de estado.
	T1OSI	XTAL		Entrada para cristal externo para TMR1.
	PGD	ST	CMOS	Data da programação Serial (ICSP).
5	V_{SS}	P		GND.
14	V_{DD}	P		Alimentação positiva.

Tabela 3: Continuação: Nomenclatura de identificação dos pinos do PIC 16F628A.

P	Power (alimentação)
-	Não utilizado
TTL	Entrada tipo TTL
ST	Entrada tipo Schmitt Trigger
CMOS	Saída do tipo CMOS
OD	Saída tipo dreno aberto (open drain)
NA	Entrada/Saída analógica

3 Mapas de memória

Como mencionado anteriormente, o PIC 16F628A possui três memórias. Na memória de Programa fica armazenado o código desenvolvido pelo usuário. Na memória de dados ficam armazenadas

nados os valores dos registradores especiais e das variáveis definidas pelo usuário. Na memória EEPROM podem ser armazenados resultados de cálculos e medidas de determinadas grandezas feitas pelo PIC. A Figura 6 e a Figura 7 mostram como são organizadas a memória de programa e a memória de dados, respectivamente.

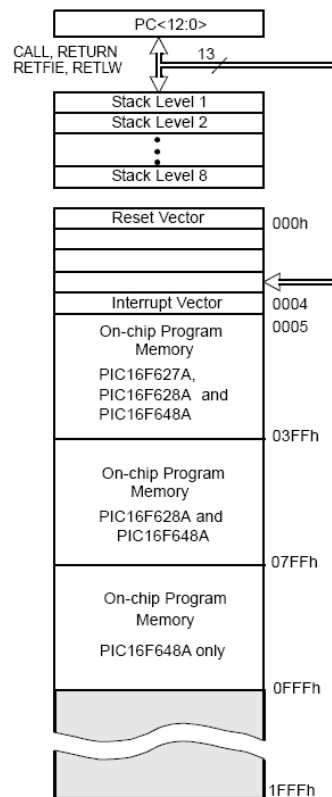


Figura 6: Mapa da memória de programa.

4 Registradores Especiais

O PIC possui registradores especiais denominados SFR (*Special Function Registers*). Estes registradores tem a função de armazenar a **configuração** e o **estado** de funcionamento atual da máquina. A seguir serão apresentados os principais registradores.

4.1 STATUS

As principais funções deste registrador estão relacionadas às operações matemáticas realizadas na ULA do PIC. Ele indica estouros de registradores (C-Carry e DC-Digt Carry) e os resultados iguais a zero. É importante ressaltar que para o caso da subtração, o Carry trabalha com lógica invertida.

[illegible]

Figura 7: Mapa da memória de dados.

4.2 PCON

A principal função do PCON é a escolha da frequência do oscilador interno (37 KHz ou 4 MHz).

4.3 TRISA e TRISB

Os registradores TRISA e TRISB servem para configurar como entrada ou saída os pinos das portas PORTA e PORTB, respectivamente. Para configurar como entrada basta escrever 1 no bit correspondente e como saída basta escrever 0 no bit correspondente. Os bits podem ser selecionados como entrada e saída da forma que se desejar. Por exemplo, para o PORTB, se a sequência 10010111 for utilizada para configurar o registrador TRISB, isto significa que as portas RB0, RB1, RB2, RB4 e RB7 foram configuradas como entrada (1) e que RB3, RB5 e RB6 foram configuradas como saída (0). Lembrado que RB0 representa o bit menos significativo (LSB - à direita) e RB7 o bit mais significativo (MSB - à esquerda).

4.4 PORTA e PORTB

Os registradores PORTA e PORTB são utilizados para modificar diretamente o estado (0 ou 1) das portas. Quando o bit é configurado como entrada, o estado da porta é armazenado diretamente no bit correspondente a porta nestes registradores. Quando o bit é configurado como saída o seu estado pode ser modificado escrevendo-se diretamente no bit correspondente destes registradores. Por exemplo se o bit 3 do PORTA estiver configurado como entrada (bit 3 do TRISA em 1) basta ler o estado do bit 3 do registrador PORTA para conhecer o estado da entrada.

5 MPLab

O MPLab é o software desenvolvido pela Microchip, distribuído gratuitamente, utilizado como ambiente de desenvolvimento de projetos para PIC's. Com ele é possível desenvolver, compilar e debbugar o código desenvolvido. A Figura 8 mostra o ambiente de trabalho do Mplab.

Ao iniciar o desenvolvimento de uma aplicação utilizando o MPLab é necessário primeiramente salvar o novo *Workspace* criado quando inicia-se o MPLab. O *Workspace* armazena todas as informações necessárias ao desenvolvimento da aplicação. Para salvá-lo clique no menu *File > save workspace*.

O MPLab trabalha também com o conceito de projeto. O projeto abrange o código-fonte e muitas outras informações necessárias para a compilação e execução da aplicação. Para criar um novo projeto clique em *Project > new*, a tela mostrada na Figura 9 aparecerá. Basta então escolher o nome para o projeto e o local onde o mesmo será armazenado.

Caso já existe um arquivo fonte em seu HD basta então adicioná-lo ao projeto clicando com o botão direito em *Source Files* e em seguida em *Add Files...* conforme mostrado na Figura 10. Este arquivos são do tipo **asm** que se refere a um arquivo Assembly, sendo esta a linguagem de programação adotada neste curso. Caso você não possua um arquivo fonte basta criá-lo clicando no botão *New* abaixo do menu *File*.

5.1 Compilando o Projeto

As configurações que serão mostradas a seguir são necessárias para que o projeto seja compilado de forma correta. Os passos são os seguintes:

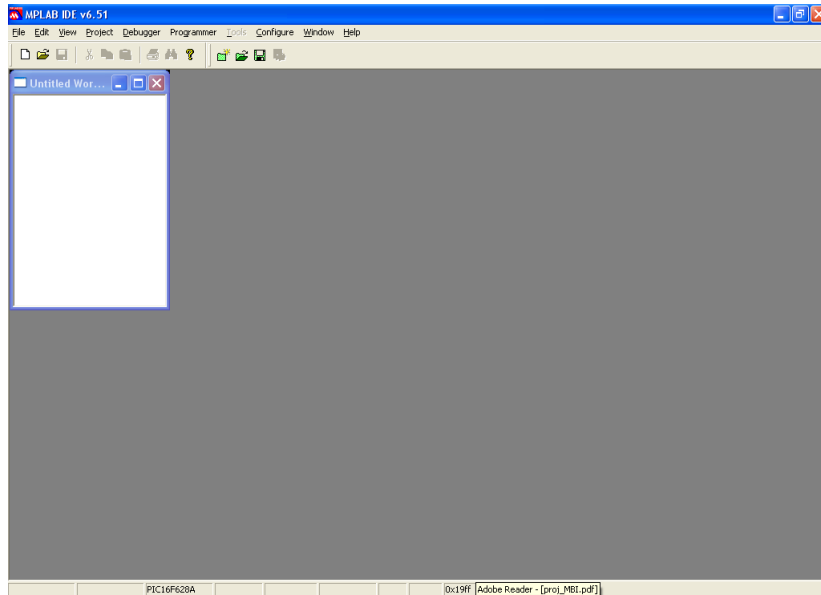


Figura 8: Ambiente de trabalho do MPlab.

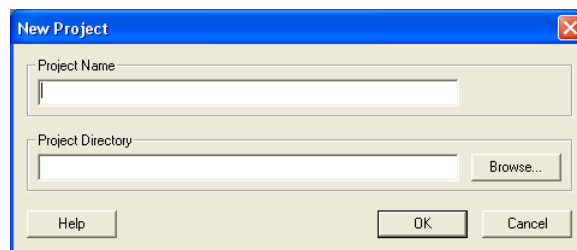


Figura 9: Criando um novo projeto.

- Selecionar o microcontrolador a ser utilizado. Para isto, clique em *Configure > Select device*. Em seguida basta selecionar o modelo do PIC na opção *Device*.
- Compilar o projeto clicando no menu *Project > Make* ou na tecla *F10*. Em seguida a tela *Output* aparecerá mostrando o resultado da compilação.

5.2 Gravando o projeto

Antes de gravar o projeto é importante configurar algumas opções do PIC. Para realizar estas configurações clique em *Configure > Configuration Bits*, em seguida a tela mostrada na Figura 11 aparecerá. A única opção que **deve** ser escolhida obrigatoriamente é a opção *Oscillator*. Pois esta opção define o tipo de oscilador a ser utilizado pelo PIC. Os principais tipos de oscilador e suas montagens elétricas são mostrada a seguir:

- Oscilador interno - Oscilador de 37 KHz ou 4 MHz.
- Oscilador a cristal - Diversas frequências disponíveis no mercado.
- Oscilador RC - a frequência depende dos valores do resistor e do capacitor utilizados.

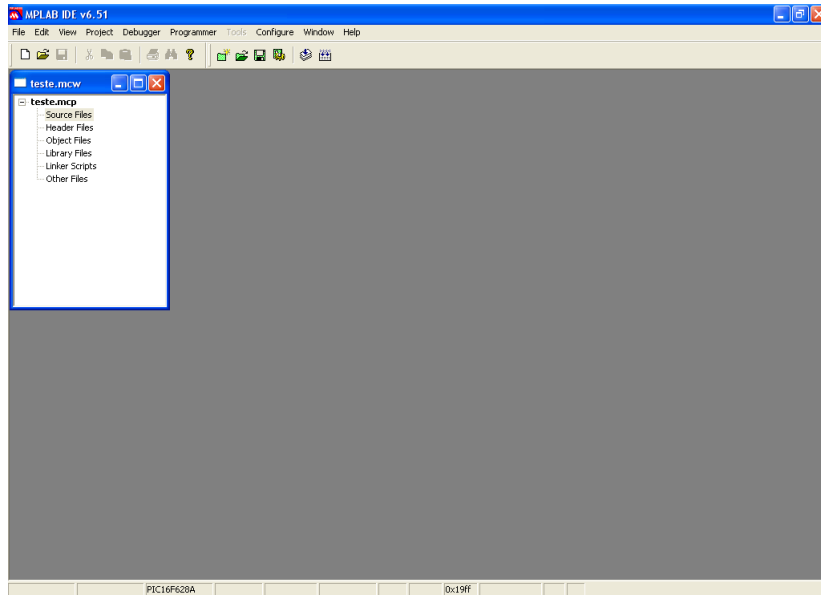


Figura 10: Adicionando um arquivo fonte.

Configuration Bits			
Address	Value	Category	Setting
2007	3FFF	Oscillator	RC: CLKOUT on RA6/OSC2/CLKOUT, RC on RA7/OSC1/CLKIN
		Watchdog Timer	On
		Power Up Timer	Disabled
		Brown Out Detect	Enabled
		Master Clear Enable	Enabled
		Low Voltage Program	Enabled
		Data EE Read Protect	Disabled
		Code Protect	Off

Figura 11: Configurando as opções do PIC.

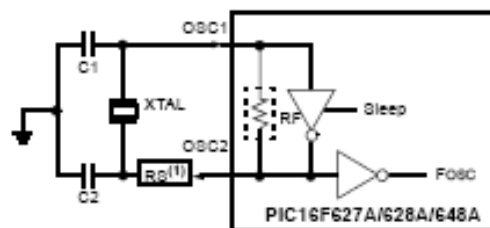


Figura 12: Montagem elétrica do PIC com oscilador a cristal.

Para gravar o programa no PIC basta então clicar no menu *Programmer > Select Programmer > Picstar Plus*. Em seguida será aberta a janela *Output*. Com o gravador ligado e conectado a porta serial selecione o menu *Programmer > Enable Programmer*. Após o gravador ser habilitado, as opções de gravação também serão habilitadas no menu *Programmer*. São elas:

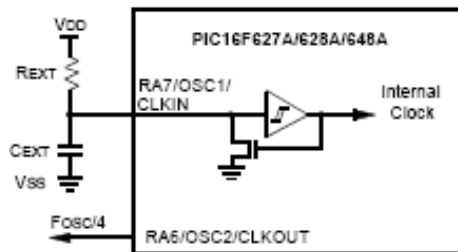


Figura 13: Montagem elétrica do PIC com oscilador RC.

- *Verify* - permite verificar se o programa que está na memória do PIC corresponde ao programa que está aberto no computador, ou seja, procura divergência entre os programas;
- *Read* - permite que o código contido no PIC seja lido se o mesmo não estiver protegido;
- *Blank* - verifica se o PIC está vazio;
- *Program* - grava o programa no PIC.

6 Criando um Programa

Em uma programação Assembly como em todas as outras linguagens de programação é necessário que as instruções sejam colocadas na ordem correta para garantir que o programa funcione corretamente. Nesta seção será mostrada os principais elementos que compõem um código-fonte em Assembly para a programação do PIC.

6.1 Arquivos de definição (*Include*)

Arquivos de definições podem ser utilizados para padronizar e agilizar a programação. Para isto podemos utilizar os arquivos denominados pela Microchip de “Includes”. A Microchip desenvolveu arquivos “Includes” para cada tipo de microcontrolador, sendo que neles estão definidos os nomes e endereços dos SFR’s e outras definições que são necessárias para o uso do microcontrolador. Estes arquivos já estão disponíveis após a instalação do MPLab e para o PIC 16F628A o arquivo é o *P16F628A.INC*. O nome da instrução para incluir o arquivo é: `#INCLUDE <P16F628A.INC>`. Este arquivo encontra-se geralmente na pasta *C: > Arquivos de programas > MPLAB IDE > MCHIP Tools*

6.2 Constantes

Para associar um nome ou variável a um número utiliza-se a diretriz **EQU**. Por exemplo, em alguns projetos são necessários definir-se valores de referência para alarmes ou pontos de operação de uma determinada variável, o denominado *Set Point*. Isto pode ser feito da seguinte forma:

Set_Point EQU .100

Neste caso o valor decimal (.X) 100 foi atribuído a variável *Set Point* atribuindo a mesma um valor constante. Isto também pode ser feito quando deseja atribuir um nome a uma posição de memória. Por exemplo:

Idade EQU H'0005'

Isto significa que a posição de memória H'0005' pode ser chamada durante o código pela *string* Idade.

6.3 Define

A diretiva #DEFINE também é utilizada para este tipo de aplicação. Porém ela não substitui nomes somente por números mas por expressões inteiras. Esta diretiva é muito utilizada para definir nomes para os pinos de I/O do PIC. Por exemplo:

#DEFINE BOTAO PORTB,1

Esta diretiva significa que toda vez que for necessário acessar o pino 1 da PORTB basta utilizar o nome BOTAO.

6.4 ORG

A diretiva ORG define o direcionamento da posição de memória de programação. Ela deve ser colocada no início do programa e refere-se ao vetor RESET e ao vetor de interrupções. O vetor de RESET é iniciado na posição 0x00 e o vetor de interrupção na posição 0x04. Esta instrução colocada no início do programa faz com que ele comece neste ponto.

6.5 END

Esta diretiva deve ser colocada no final do programa.

6.6 CBLOCK e ENDC

Estas diretivas servem para definir um grupo sequencial de EQU's, precisando informar apenas o endereço inicial da primeira variável. Isto facilita a conversão de um programa de um modelo de PIC para outro quando a RAM inicia em um endereço diferente.

6.7 Registrador WORK (W)

O registrador WORK é um registrador temporário utilizado pela ULA. Ele é extremamente utilizado pois não é possível ler e escrever diretamente na memória sem utilizá-lo.

6.8 Estruturação básica

A sequência básica do programa não segue uma estrutura totalmente rígida. Uma sequência sugerida é:

1. Cabeçalho - feito com comentário utilizando ;
2. Inserir arquivo *Include*.

3. Declarar variáveis.
4. Definir constantes.
5. Definir Entradas e saídas.
6. Definir endereço inicial de processamento - Vetor RESET.
7. Definir endereço inicial do vetor de interrupções.
8. Definir rotinas de interrupção.
9. Definir rotinas e subrotinas.
10. Desenvolver programa.

7 Conjunto de instruções para o PIC 16F628A

Na Tabela 4 são apresentadas as instruções disponíveis para o PIC 16F628A. Porém para facilitar o entendimento serão apresentadas as terminologias utilizadas para a formação das instruções e nas próximas subseções serão apresentadas as principais instruções divididas em subconjuntos.

7.1 Instruções e Argumentos

A seguir será apresentada a forma de construção dos nomes das instruções para facilitar o entendimento da mesma e seus argumentos. São eles:

- **Work (W)** - Registrador temporário para as operações da ULA;
- **File (F)** - Referência a um registrador, ou seja, uma posição de memória. É representado por F nos nomes das instruções e por f nos argumentos;
- **Literal (L)** - Um número qualquer pode ser escrito na forma decimal (.), hexadecimal (H) e binária (B). É representado por L nos nomes das instruções e por k nos argumentos;
- **Destino** - O local onde deve ser armazenado o resultado da operação. Este destino pode ser somente: F, que guardará o resultado no próprio registrador passado como argumento ou W;
- **Bit** - refere-se a um bit específico dentro de um Byte. É representado por B nos nomes das instruções e por b nos argumentos;
- **Skip** - É utilizado para criar desvios para a próxima linha;
- **Set** - Ato de setar um bit (nível lógico 1);
- **Clear** - Ato de limpar um bit (nível lógico 0);
- **Zero** - Gera desvios em algumas instruções quando o resultado da operação é zero.

7.2 Lista de Instruções

A seguir é apresentado o conjunto de instruções para o PIC 16F628A.

Tabela 4: Operações com registradores

Instrução	Argumento	Descrição
ADDWF	f,d	Soma W e f, guardando o resultado em d.
ANDWF	f,d	Lógica "E" entre W e f, guardando o resultado em d.
CLRF	f	Limpa f.
COMF	f,d	Calcula o complemento de f, guardando o resultado em d.
DECF	f,d	Decrementa f, guardando o resultado em d.
DECFSZ	f,d	Decrementa f, guardando o resultado em d e pula a próxima linha se o resultado for zero.
INCF	f,d	Incrementa f, guardando o resultado em d.
INCFSZ	f,d	Incrementa f, guardando o resultado em d, e pula a próxima linha se o resultado for zero.
IORWF	f,d	Lógica "OU" entre W e f, guardando o resultado em d.
MOVF	f,d	Move f para d (cópia).
MOVWF	f	Move W para f (cópia).
RLF	f,d	Rotaciona f 1 bit para a esquerda.
RRF	f,d	Rotaciona f 1 bit para a direita.
SUBWF	f,d	Subtrai W de f ($f - W$), guardando o resultado em d.
SWAPF	f,d	Executa uma inversão entre as partes alta e baixa de f, guardando o resultado em d.
XORWF	f,d	Lógica "OU exclusivo" entre W e f, guardando o resultado em d.

Tabela 5: Operações com literais

Instrução	Argumentos	Descrição
ADDLW	k	Soma k com W, guardando o resultado em W.
ANDLW	k	Lógica "E" entre k e W, guardando o resultado em W.
IORLW	k	Lógica "OU" entre k e W, guardando o resultado em W.
MOVLW	k	Move k para W.
SUBLW	k	Subtrai W de k ($k - W$), guardando o resultado em W.
XORLW	k	Lógica "OU exclusivo" entre k e W, guardando o resultado em W.

Tabela 6: Operações com bits

Instrução	Argumentos	Descrição
BCF	f,b	Impõe 0 (zero) ao bit b do registrador f.
BSF	f,b	Impõe 1 (um) ao bit b do registrador f.
BTFSC	f,b	Testa o bit b do registrador f, e pula a próxima linha se ele for 0 (zero).
BTFSS	f,b	Testa o bit b do registrador f, e pula a próxima linha se ele for 1 (um).

Tabela 7: Controles

Instrução	Argumentos	Descrição
CLRW	-	Limpa W.
NOP	-	Gasta um ciclo de máquina sem fazer absolutamente nada.
CALL	R	Executa a rotina R.
CLRWDT	-	Limpa o registrador WDT para não acontecer o reset.
GOTO	R	Desvia para o ponto R, mudando o PC.
RETFIE	-	Retorna de uma interrupção.
RETLW	k	Retorna de uma rotina, com k em W.
RETURN	-	Retorna de uma rotina, sem afetar W.
SLEEP	-	Coloca o PIC em modo sleep (dormindo) para economia de energia.

8 Rotinas

Nesta seção serão apresentadas algumas rotinas que serão bastante utilizadas para o desenvolvimento de programas mais complexos.

8.1 Acessando o banco de dados

A memória do PIC possui quatro banco de dados de memória para os registradores especiais (SFR) e para a memória de variáveis do sistema. Para acessar o banco de dados que se deseja utilizar é necessário alterar os valores dos bits RP0 e RP1 do registrador STATUS. A Tabela 7 mostra a combinação dos bits RP0 e RP1 necessária para acessar os diferentes bancos de memória.

Tabela 8: Banco

Banco	RP1	RP0
0	0	0
1	0	1
2	1	0
3	1	1

Para acessar um banco basta setar os valores do bits conforme a tabela. Por exemplo, a rotina utilizada para acessar o banco 1 é:

```
BSF STATUS,RP0
```

Coloca o estado do bit RP0 do registrador STATUS em 1.

8.2 Lendo e escrevendo nas portas

Uma porta pode ser lida ou escrita de duas maneiras: em conjunto, ou seja, todos os 8 bits simultaneamente ou bit a bit. A leitura de uma porta em conjunto pode ser feita da seguinte maneira:

```
MOVF PORTB,W  
MOVWF DADO
```

A rotina acima escreve o valor contido na PORTB em W (work) e depois escreve valor de W em DADO (variável criada para armazenar valor de PORTB).

A escrita de uma porta em conjunto pode ser feita utilizando a seguinte instrução:

```
MOVLW B'00010100'  
MOVWF PORTA
```

A rotina acima escreve o valor literal em W (work) e depois escreve valor de W em PORTA.

A escrita bit a bit de uma porta pode ser feita da seguinte maneira:

```
BSF PORTB,2
```

A rotina acima escreve 1 no bit 2 do PORTB.

8.3 Fazendo contagem de eventos

AS instruções DECFSZ e INCFSZ podem ser utilizadas para efetuar contagens. A seguir é apresentada uma rotina de contagem decrescente implementada com a instrução DECFSZ.

```
MOVLW .20  
MOVWF CONTAGEM  
LOOP  
DECFSZ CONTAGEM,F  
GOTO LOOP  
FIM
```

Na rotina acima o contador é iniciado com o valor 20 que é decrementado de 1 unidade a cada loop do sistema até chegar ao valor 0 quando a instrução GOTO é saltada finalizando o processo.

8.4 Fazendo comparações

A instrução SUB afeta diretamente o flag (carry) do registrador STATUS . Quando o resultado da subtração é positivo o carry (C) assume nível lógico 1. Assim, é possível durante uma subtração saber se um número é maior ou menor que outro. Quando o resultado da subtração é negativo, o carry (C) assume nível lógico 0. Por exemplo se subtrairmos Num1 = 20 de Num2 = 10 o resultado da subtração é - 10 (negativo) e o carry estará em nível lógico 0. Se subtrairmos Num1 = 20 de Num2 = 30 o resultado será 10 (positivo) e o carry estará em nível lógico 1.

A seguir é mostrado a rotina para implementação de uma comparação:

```
COMPARA1
    MOVF NUM1,W
    SUBWF NUM2,W
    BTFSS STATUS,C
    GOTO RESP1
    GOTO RESP2
```

FIM

Primeiramente, o valor de NUM1 é movido para W. Em seguida o valor de W (NUM1) é subtraído de NUM2 guardando o resultado em W. Logo em seguida é feito um teste no carry. Se o resultado é negativo $NUM2 < NUM1$. Se o resultado é positivo $NUM2 > NUM1$.

8.5 Fazendo contagem de tempo

Existem três formas de contar tempo no PIC:

- contar os ciclos de máquina por meio de loopings;
- contando os ciclos de máquina por meio do contador TIMER (0, 1 ,2);
- contando pulsos externos utilizando a entrada T0CKI e do TMR0 ou da entrada T1CKI e do TMR1.

A seguir será mostrada a rotina de implementação da contagem de ciclos por meio de loopings.

```
DELAY
    MOVLW .250
    MOVWF TEMP1
DL1
    NOP
    DECFSZ TEMP1,F
    GOTO DL1
    RETURN
```

Assim, a rotina dura 1ms visto que, além de o contador TEMP1 ser decrementado 250 vezes, cada looping (DL1) gasta quatro ciclos o que equivale a $1 \mu s$.

9 Exercícios Propostos

Na figura 14 está representada a bancada utilizada para realização de testes dos programas desenvolvidos, bem como a pinagem utilizada pelos seus leds, push-bottons e botoeiras. Sendo assim, nessa seção serão propostos alguns exercícios, tendo em vista as primeiras noções de programação apresentadas até aqui.

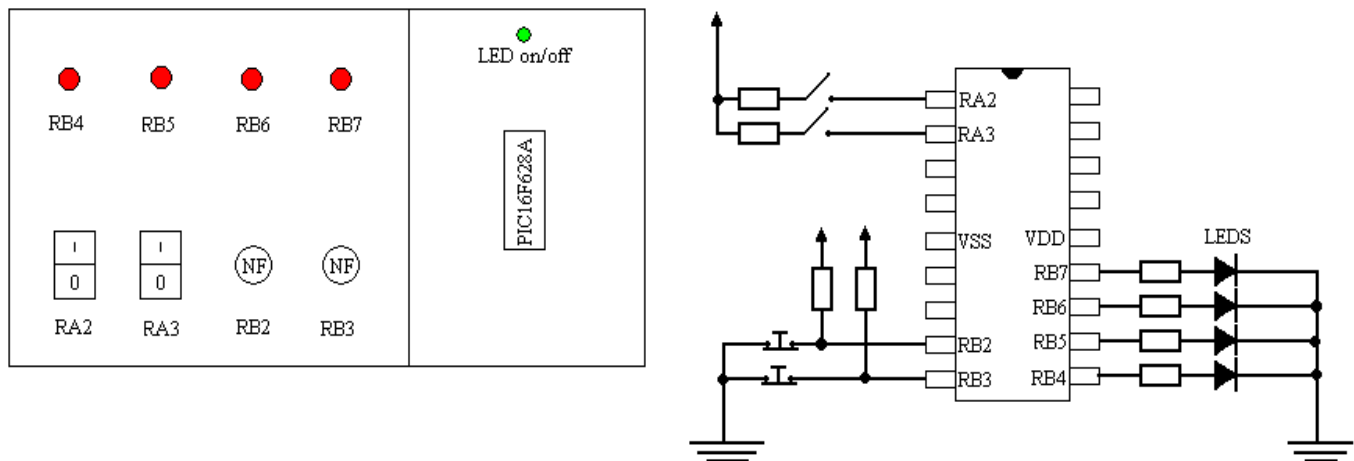


Figura 14: Representação da bancada didática e pinagem utilizada

1. Elaborar um programa utilizando uma chave que:
 - com a chave desligada, acenda os dois leds centrais;
 - com a chave ligada, apague os leds centrais e acenda os outros dois das laterais, realizando assim uma alternância entre pares de leds.
2. Elaborar um programa que controle o nível de líquido dentro de um tanque como o representado na figura 15

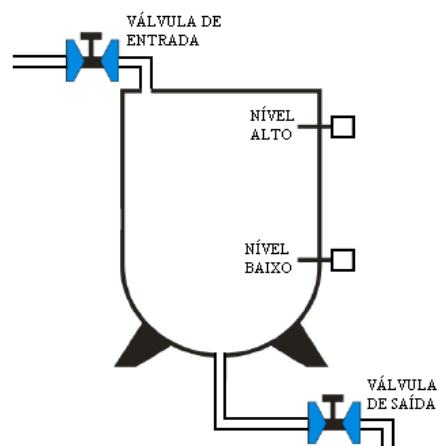


Figura 15: Planta de um reservatório

3. Suponha que há um estacionamento para automóveis com vagas para 15 carros. Deseja-se fazer o controle de quantas vagas estão ocupadas. Elabore um programa que faça isso automaticamente.
4. Na indústria há a necessidade de não ligar vários motores simultaneamente devido à alta potência requerida na partida de cada motor. Através de uma botoeira, ligue 4 motores com um atraso no acionamento de um para o outro.

5. Elaborar um programa para dar partida em um motor trifásico estrela-triângulo, utilizando três contatores.