rede e-Tec Brasil

Microcontroladores

Édilus de Carvalho Castro Penido

Ronaldo Silva Trindade

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
MINAS GERAIS
Campus Ouro Preto

Ouro Preto - MG
2013

Presidência da República Federativa do Brasil

Ministério da Educação

Secretaria de Educação Profissional e Tecnológica

Este caderno foi elaborado em parceria entre o Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus Ouro Preto e a Universidade Federal de Santa Maria para a Rede e-Tec Brasil.

Equipe de Elaboração
Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – IFMG-Ouro Preto

Reitor
Caio Mário Bueno Silva/IFMG-Ouro Preto

Direção Geral
Arthur Versiani Machado/IFMG-Ouro Preto

Coordenação Institucional
Sebastião Nepomuceno/IFMG-Ouro Preto

Coordenação de Curso
Cristiano Lúcio Cardoso Rodrigues/IFMG-Ouro Preto

Professor-autor
Édilus de Carvalho Castro Penido/IFMG-Ouro Preto
Ronaldo Silva Trindade/IFMG-Ouro Preto

Equipe de Acompanhamento e Validação
Colégio Técnico Industrial de Santa Maria – CTISM

Coordenação Institucional
Paulo Roberto Colusso/CTISM

Coordenação Técnica
Iza Neuza Teixeira Bohrer/CTISM

Coordenação de Design
Erika Goellner/CTISM

Revisão Pedagógica
Andressa Rosemárie de Menezes Costa/CTISM
Janaína da Silva Marinho/CTISM
Jaqueline Müller/CTISM
Marcia Migliore Freo/CTISM

Revisão Textual
Ana Paula Cantarelli/CTISM
Fabiane Sarmento Oliveira Fruet/CTISM
Tatiana Rehbein/UNOCHAPECÓ

Revisão Técnica
Andrei Piccini Legg/CTISM

Ilustração
Gabriel La Rocca Cóser/CTISM
Marcel Santos Jacques/CTISM
Rafael Cavalli Viapiana/CTISM
Ricardo Antunes Machado/CTISM

Diagramação
Cássio Fernandes Lemos/CTISM
Leandro Felipe Aguilar Freitas/CTISM

Ficha catalográfica elaborada por Maristela Eckhardt – CRB-10/737
Biblioteca Central da UFSM

P411m Penido, Édilus de Carvalho Castro
Microcontroladores / Édilus de Carvalho Castro Penido. Ronaldo Silva Trindade. – Ouro Preto : Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais ; Santa Maria : Universidade Federal de Santa Maria, ColégioTécnico Industrial de Santa Maria ; Rede e-Tec Brasil, 2013.
80 p. : il. ; 28 cm
ISBN 978-85-86473-12-8

1. Engenharia elétrica 2. Eletrônica 3. Computação 4. Microeletrônica 5. Circuitos eletrônicos 6. Microcontroladores I. Trindade, Ronaldo Silva II. Rede e-Tec Brasil III. Título

CDU 621.3.049.77
681.5

Apresentação e-Tec Brasil

Prezado estudante,
Bem-vindo a Rede e-Tec Brasil!

Você faz parte de uma rede nacional de ensino, que por sua vez constitui uma das ações do Pronatec – Programa Nacional de Acesso ao Ensino Técnico e Emprego. O Pronatec, instituído pela Lei nº 12.513/2011, tem como objetivo principal expandir, interiorizar e democratizar a oferta de cursos de Educação Profissional e Tecnológica (EPT) para a população brasileira propiciando caminho de o acesso mais rápido ao emprego.

É neste âmbito que as ações da Rede e-Tec Brasil promovem a parceria entre a Secretaria de Educação Profissional e Tecnológica (SETEC) e as instâncias promotoras de ensino técnico como os Institutos Federais, as Secretarias de Educação dos Estados, as Universidades, as Escolas e Colégios Tecnológicos e o Sistema S.

A educação a distância no nosso país, de dimensões continentais e grande diversidade regional e cultural, longe de distanciar, aproxima as pessoas ao garantir acesso à educação de qualidade, e promover o fortalecimento da formação de jovens moradores de regiões distantes, geograficamente ou economicamente, dos grandes centros.

A Rede e-Tec Brasil leva diversos cursos técnicos a todas as regiões do país, incentivando os estudantes a concluir o ensino médio e realizar uma formação e atualização contínuas. Os cursos são ofertados pelas instituições de educação profissional e o atendimento ao estudante é realizado tanto nas sedes das instituições quanto em suas unidades remotas, os polos.

Os parceiros da Rede e-Tec Brasil acreditam em uma educação profissional qualificada – integradora do ensino médio e educação técnica, – é capaz de promover o cidadão com capacidades para produzir, mas também com autonomia diante das diferentes dimensões da realidade: cultural, social, familiar, esportiva, política e ética.

Nós acreditamos em você!
Desejamos sucesso na sua formação profissional!

Ministério da Educação
Janeiro de 2013

Nosso contato
etecbrasil@mec.gov.br

Indicação de ícones

Os ícones são elementos gráficos utilizados para ampliar as formas de linguagem e facilitar a organização e a leitura hipertextual.

Atenção: indica pontos de maior relevância no texto.

Saiba mais: oferece novas informações que enriquecem o assunto ou "curiosidades" e notícias recentes relacionadas ao tema estudado.

Glossário: indica a definição de um termo, palavra ou expressão utilizada no texto.

Mídias integradas: sempre que se desejar que os estudantes desenvolvam atividades empregando diferentes mídias: vídeos, filmes, jornais, ambiente AVEA e outras.

Atividades de aprendizagem: apresenta atividades em diferentes níveis de aprendizagem para que o estudante possa realizá-las e conferir o seu domínio do tema estudado.

Sumário

Palavra do professor-autor 9

Apresentação da disciplina 11

Projeto instrucional 13

Aula 1 – Conhecendo os microcontroladores 15
1.1 Origem dos microcontroladores 15
1.2 Arquiteturas Harvard e Von Neuman 17

Aula 2 – Os microcontroladores da família PIC 21
2.1 Folha de dados do PIC12F675 21
2.2 Aplicações simples com PIC12F675 23

Aula 3 – *Hardware* do microcontrolador 27
3.1 Descrição dos pinos 27
3.2 Memória 28
3.3 Registradores 28
3.4 Pinos de I/O 31
3.5 Temporizadores 32

Aula 4 – *Hardware* do microcontrolador II 35
4.1 Comparadores 35
4.2 PWM 36
4.3 Conversor A/D 37
4.4 USART 37

Aula 5 – *Set* de instruções 41
5.1 Estrutura das instruções 41
5.2 Grupos de instruções 42

Aula 6 – Interrupções 45
6.1 Interrupções no microcontrolador 45
6.2 Interrupção de *timer* 46
6.3 Interrupção externa 46
6.4 Outras interrupções 47

Aula 7 – Programação Assembly **49**
7.1 Programa de computador 49
7.2 Programando em Assembly MPASM 49
7.3 Exemplos de programas 51
7.4 Pisca LED 51

Aula 8 – Programação Assembly II **59**
8.1 Programando em Assembly com interrupção 59

Aula 9 – Programação C **65**
9.1 Linguagem de alto nível 65
9.2 Principais estruturas da linguagem C 65
9.3 Programas em C 68

Aula 10 – Programação C II **75**
10.1 Conversor A/D do microcontrolador PIC12F675 75
10.2 Programa exemplo A/D 75

Referências **79**

Currículo do professor-autor **80**

Palavra do professor-autor

Prezado estudante,

É com enorme satisfação que lhe apresentamos a disciplina de Microcontroladores do seu curso de Automação Industrial. Este trabalho é fruto de nossa longa experiência no trato com os microcontroladores, seja como professor, seja como desenvolvedor de equipamentos que utilizam essa maravilha da moderna microeletrônica. Os microcontroladores têm enorme aplicação em nosso dia a dia e, muitas vezes, nem nos damos conta de que o equipamento que estamos utilizando possui um microcontrolador. Citamos como exemplo os aparelhos de DVD, televisão, forno de micro-ondas, geladeira, portão eletrônico, sistemas eletro/eletrônicos de automóveis e até o nosso cartão de crédito com *chip*. Se naqueles equipamentos de uso diário os microcontroladores já se fazem presente, imagine como eles estão embutidos em praticamente todas as aplicações eletrônicas dentro de uma indústria.

Ao longo do curso você será capaz de constatar por si só uma enorme gama de aplicações. Para encerrar esta conversa, lembre-se: se você chegou até este ponto, poderá ir muito mais longe, para além deste curso. O mundo precisa de uma pessoa como você, que corre atrás de seus objetivos, que busca aprender e aperfeiçoar-se cada vez mais. Conte com toda a nossa equipe para apoiá-lo e esteja certo de que você irá superar as suas dificuldades. Um forte abraço dos professores.

Ronaldo Trindade e Édilus Penido

Apresentação da disciplina

Nesta disciplina de Microcontroladores vamos percorrer juntos um universo novo de aplicações da microeletrônica. Até aqui, você já teve a oportunidade de conhecer e estudar os diversos componentes eletrônicos, tais como: diodos, transistores e circuitos integrados.

Você também já sabe que os componentes eletrônicos são interligados para formar circuitos, os quais desempenham funções específicas e, na eletrônica digital, pôde estudar como os circuitos executam funções lógicas (AND, OR, XOR, NOT) em suas mais variadas combinações.

Acontece que quando a complexidade de um circuito aumenta muito (em termos das diversas funções a serem executadas), muitas vezes, torna-se inviável a produção de tal circuito.

Nesse ponto é mais fácil e mais barato utilizar um circuito que possa ser programado (o microcontrolador) e, utilizando-se corretamente os seus terminais de entrada e de saída, executar internamente as funções desejadas através de um ***software***.

A proposta desta disciplina é, então, ensiná-lo a programar um microcontrolador de forma que ele possa executar as funções a ele designadas.

Para atingir o nosso objetivo, estudaremos a estrutura interna de um microcontrolador, suas funções internas, seus periféricos associados, seus registradores, temporizadores, pinos de entrada/saída e demais periféricos de interface com o mundo exterior.

Estudaremos também a forma de programar, gravar e testar o *software* dentro de um microcontrolador e, finalmente, apresentaremos algumas aplicações práticas que você poderá implementar.

A-Z

software
Termo inglês que designa o conjunto de instruções capazes de serem interpretadas e executadas por um determinado processador de dados. *Software* é também conhecido como "programa de computador".

Projeto instrucional

Disciplina: Microcontroladores (carga horária: 75h).

Ementa: Introdução aos microcontroladores. Programação Assembler para microcontroladores. Programação C para microcontroladores. Aplicações de microcontroladores.

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA (horas)
1. Conhecendo os microcontroladores	Conhecer os fundamentos e as principais arquiteturas dos microcontroladores.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	07
2. Os microcontroladores da família PIC	Descrever as características do PIC12F675.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	07
3. *Hardware* do microcontrolador	Conhecer o *hardware* interno do microcontrolador PIC12F675.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	07
4. *Hardware* do microcontrolador II	Conhecer o *hardware* interno do microcontrolador PIC12F675.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	08
5. *Set* de instruções	Apresentar o conjunto de instruções Assembly do microcontrolador PIC12F675.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	08
6. Interrupções	Compreender a função das interrupções em um microcontrolador. Conhecer as interrupções disponíveis no microcontrolador PIC12F675.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	08
7. Programação Assembly	Descrever a programação Assembly de um microcontrolador. Compreender a importância da programação Assembly no PIC12F675.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	08

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA (horas)
8. Programação Assembly II	Expandir os conceitos sobre a programação Assembly do microcontrolador PIC12F675, através de exemplos.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	08
9. Programação C	Conhecer a linguagem C aplicada ao microcontrolador. Estudar as aplicações da linguagem C para o microcontrolador PIC12F675.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	07
10. Programação C II	Conhecer o conversor analógico digital (A/D) do microcontrolador PIC12F675. Estudar a utilização prática do conversor A/D do microcontrolador PIC12F675.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	07

Aula 1 – Conhecendo os microcontroladores

Objetivos

Conhecer os fundamentos e as principais arquiteturas dos micro-controladores.

1.1 Origem dos microcontroladores

Um microcontrolador é, em última análise, um computador em um único *chip* (Figuras 1.1 e 1.2). Esse *chip* contém um processador (Unidade Lógica e Aritmética – ULA), memória, periféricos de entrada e de saída, temporizadores, dispositivos de comunicação serial, dentre outros.

Os microcontroladores surgiram como uma evolução natural dos circuitos digitais devido ao aumento da complexidade dos mesmos. Chega um ponto em que é mais simples, mais barato e mais compacto, substituir a lógica das portas digitais por um conjunto de processador e *software*.

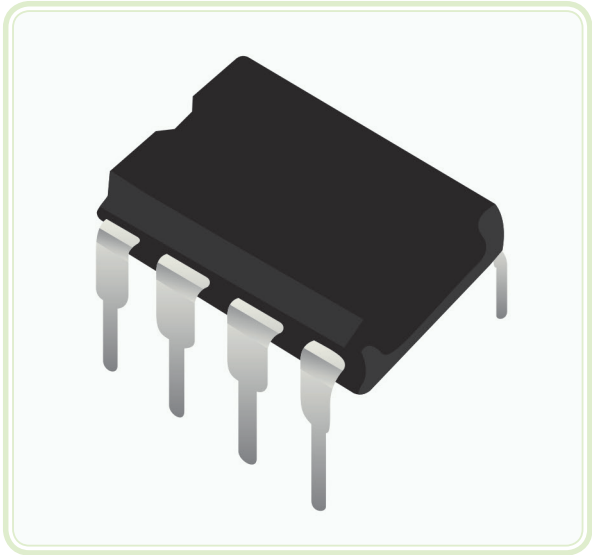


Figura 1.1: Microcontrolador PIC12F675
Fonte: Microchip Technology Inc., 2012

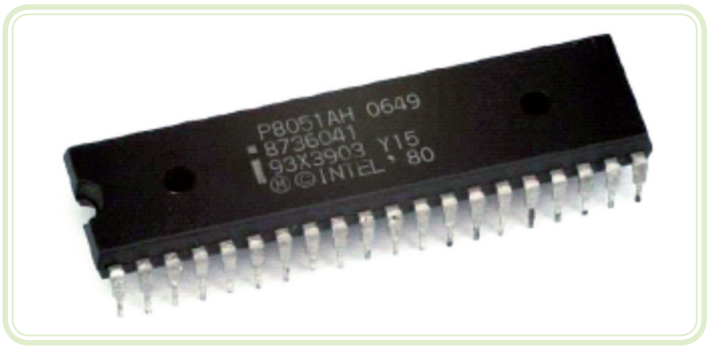


Figura 1.2: Microcontrolador INTEL 8051
Fonte: http://upload.wikimedia.org/wikipedia/commons/thumb/f/f0/KL_Intel_P8051.jpg/220px-KL_Intel_P8051.jpg

O primeiro microcontrolador foi lançado pela empresa Intel em 1977 e recebeu a sigla "8048". Com a sua posterior evolução, deu origem à família "8051". Esse *chip* é programado em linguagem Assembly e possui um poderoso conjunto de instruções.

Por ser um dos precursores, é utilizado em muitas aplicações de automação em diversas áreas do mundo.

O microcontrolador possui internamente os seguintes dispositivos:

a) Uma CPU (*Central Processor Unit* ou Unidade de Processamento Central), cuja finalidade é interpretar as instruções de programa.

b) Uma memória PROM (*Programmable Read Only Memory* ou Memória Programável Somente de Leitura) na qual são gravadas as instruções do programa.

c) Uma memória RAM (*Random Access Memory* ou Memória de Acesso Aleatório) utilizada para memorizar as variáveis utilizadas pelo programa.

d) Um conjunto de LINHAS de I/O para controlar dispositivos externos ou receber impulsos de sensores, interruptores, etc.

e) Um conjunto de dispositivos auxiliares ao funcionamento, ou seja, gerador de *clock*, contadores, UASART para comunicação, etc.

A Figura 1.3 apresenta o diagrama de blocos de um microcontrolador, mostrando os principais elementos descritos até aqui.

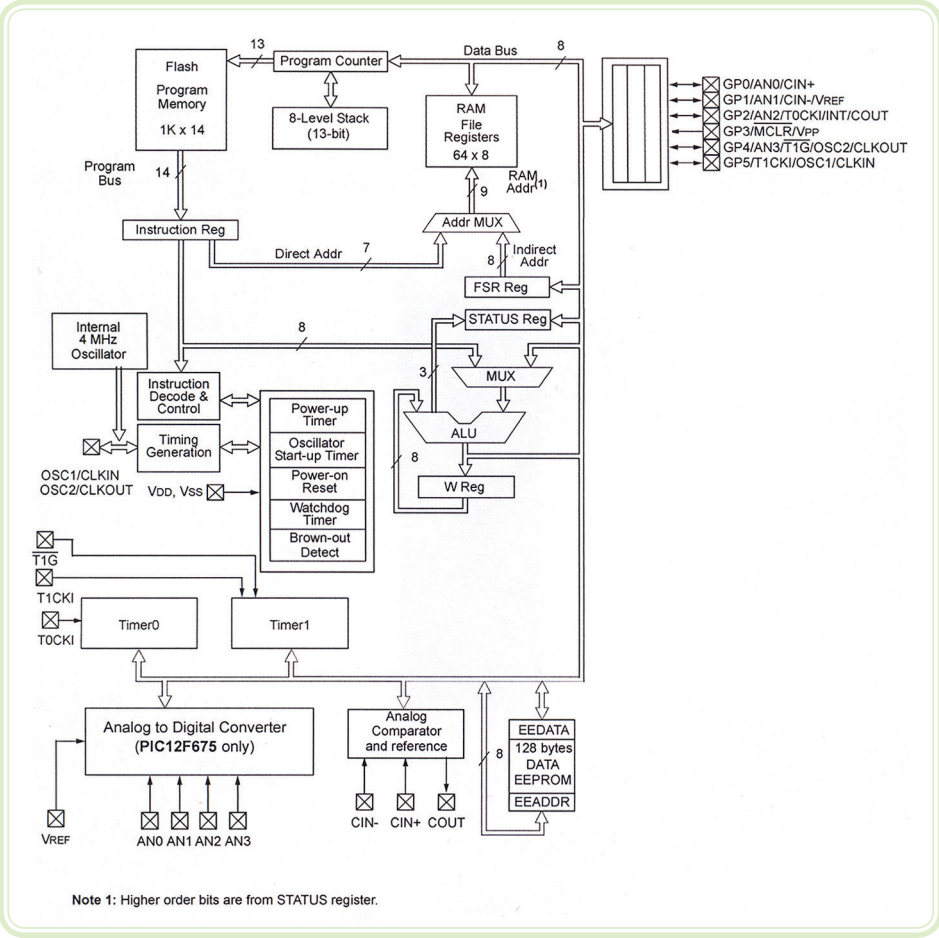


Figura 1.3: Diagrama de blocos PIC12F675
Fonte: Microchip Technology Inc., 2012

Atualmente, os principais fabricantes de microcontroladores são:
Intel
http://www.intel.com

Zilog
http://www.zilog.com

National
http://www.national.com

Microchip
http://www.microchip.com

Motorola
http://www.motorola.com

Analog Devices
http://www.analog.com/microconverter

1.2 Arquiteturas Harvard e Von Neuman

Quando um sistema de processamento de dados (processadores e microcontroladores) possui uma única área de memória na qual ficam armazenados os dados (variáveis) e o programa a ser executado (*software*), dizemos que esse sistema segue a arquitetura de Von Neuman (pronuncia-se "fon noiman").

No caso em que os dados (variáveis) ficam armazenados em uma área de memória e o programa a ser executado (*software*) fica armazenado em outra área de memória, dizemos que esse sistema segue a arquitetura Harvard.

A máquina proposta por Von Neuman é composta pelos seguintes componentes (Figura 1.4):

a) Memória.

b) Unidade de controle.

c) Unidade Lógica e Aritmética (ULA).

d) Registradores.

e) Periféricos de entrada e saída.

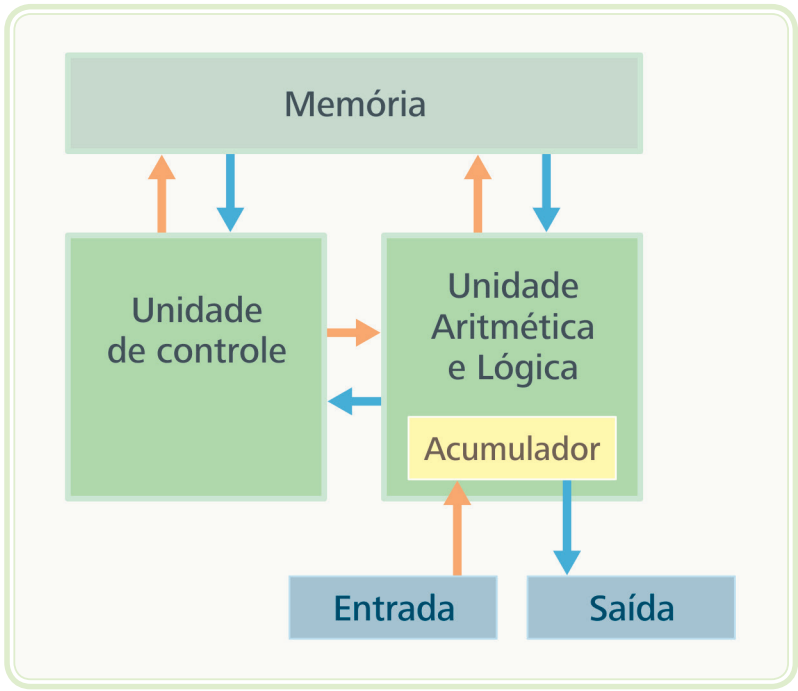


Figura 1.4: Arquitetura Von Neuman
Fonte: Autor

Conforme pode ser observado na Figura 1.4, não existe separação entre dados e programa, uma vez que há uma única área de memória. Dessa forma, o processador deve executar uma única ação por vez: ou acessa os dados ou executa uma instrução.

Na arquitetura Harvard (Figura 1.5) observamos dois barramentos distintos: um para acessar a memória de dados e outro para acessar a memória de programas. Dessa forma, o processador pode buscar e executar uma instrução ao mesmo tempo em que acessa a memória de dados para ler ou para gravar algum valor. Veja a Figura 1.5; nela é possível observar o barramento de dados (*Data Bus*), em vermelho, partindo da memória RAM e seguindo até a

Unidade Lógica e Aritmética (ALU, em inglês). Na mesma figura, observa-se o barramento de programa (*Programm Bus*) em azul, que parte da memória de programa e chega na ALU.

Devido à separação entre dados e programa, um processador da arquitetura Harvard executará um programa em menor tempo do que um processador da arquitetura Von Neuman de mesmo *clock*.

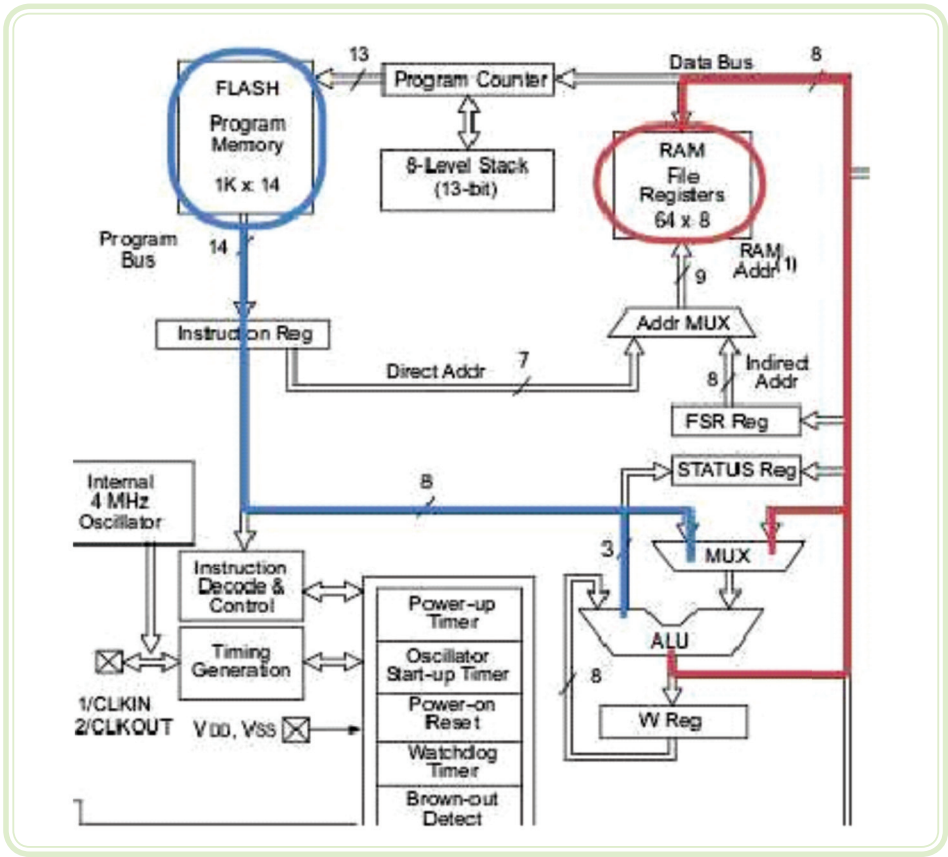


Figura 1.5: Arquitetura Harvard
Fonte: Microchip Technology Inc., 2012

Resumo

Nesta aula, você aprendeu que um microcontrolador é um computador em miniatura, dentro de um único *chip*, capaz de executar funções específicas através de um programa (*software*).

Aprendeu também as duas arquiteturas de construção de microcontroladores e a diferença que existe entre ambas.

Atividades de aprendizagem

1. Como atividade de aprendizagem desta aula, procure identificar nos equipamentos eletroeletrônicos de sua casa aqueles que utilizam microcontroladores. Após, faça uma lista dos mesmos e envie-as para a plataforma.

2. Responda também às seguintes questões:

a) Quais as principais diferenças entre um microcontrolador e um microprocessador?

b) Quais as principais vantagens da arquitetura Harvard?

Dica

Consulte a bibliografia disponível no polo e conteúdos disponibilizados na internet.

Aula 2 – Os microcontroladores da família PIC

Objetivos

Descrever as características do PIC12F675.

2.1 Folha de dados do PIC12F675

Um dos grandes fabricantes mundiais de microcontroladores é a Microchip, também responsável por produzir os microcontroladores PIC (*Peripherical Interface Controller*). Esses microcontroladores possuem tecnologia RISC (*Reduced Instruction Set Computer*) e processadores com conjunto de instruções reduzidas (neste caso, são 35 instruções simples que executam em 1 ou 2 ciclos de máquina). Existem PICs de 14, 16 e 32 *bits*, de 8 a 40 pinos no encapsulamento, o que permite uma ampla gama de opções de aplicação (ZANCO, 2006).

Para que possamos entender o funcionamento dos microcontroladores da família PIC, estudaremos um dos mais simples representantes dessa família: o PIC12F675. Este microcontrolador é encapsulado em um invólucro de 8 pinos e possui um pequeno número de funções externas mantendo toda a arquitetura interna, o que facilita o aprendizado.

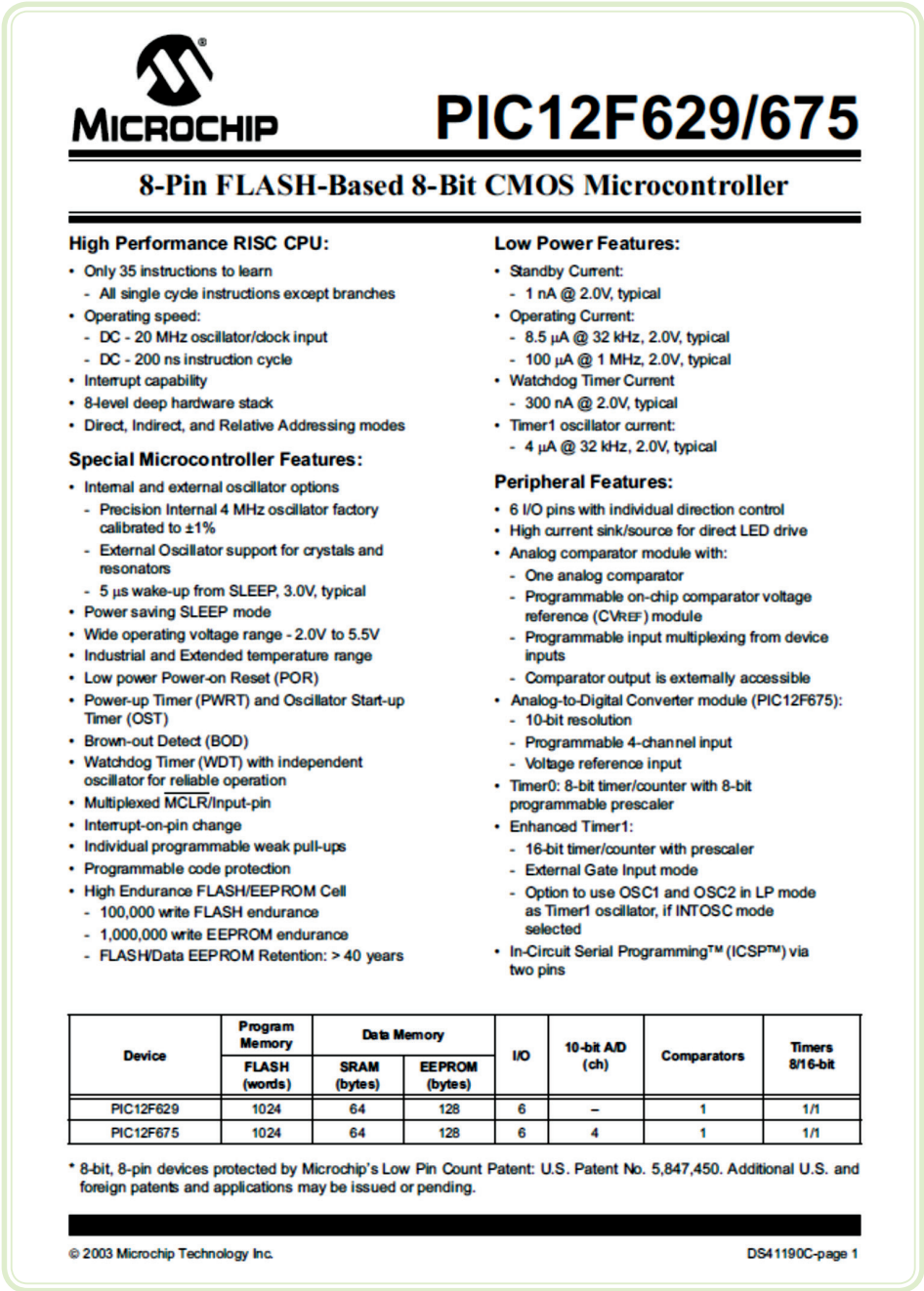
A Figura 2.1 apresenta as principais características do PIC12F675 e o texto, em inglês por se tratar do *data-sheet* (folha de dados), emitido pela Microchip em 2012.

***High performance* RISC CPU** – trata-se de uma CPU de alto desempenho na arquitetura RISC que possui apenas 35 instruções (Assembly), as quais são executadas em um ciclo de *clock*. O *clock* máximo chega a 20 MHz e o ciclo de instrução pode ser tão pequeno quanto 200 ns. Possui ainda a capacidade de interrupção com oito níveis de profundidade da pilha, bem como modos de endereçamento direto, indireto e relativo.

Special microcontroller features – são características especiais do microcontrolador, tais como: opção de oscilador interno ou externo (RC, ressonador ou cristal); modo *sleep* para economia de energia; ampla faixa de tensões de operação (2 V a 5,5 V); *watchdog timer* com temporizador independente;

interrupção por variação de sinal nos pinos; memória *flash* de alta persistência, podendo ser regravado até 100.000 vezes, e ciclo de escrita na memória EEPROM de até um milhão de vezes, com retenção na memória acima de 40 anos.

Low power features – características de baixa potência, tais como: consumo de corrente em modo de espera de 1 nA quando alimentado com 2 V e corrente de operação de 100 uA com *clock* de 1 MHz.



MICROCHIP

PIC12F629/675

8-Pin FLASH-Based 8-Bit CMOS Microcontroller

High Performance RISC CPU:

- Only 35 instructions to learn
  - All single cycle instructions except branches
- Operating speed:
  - DC - 20 MHz oscillator/clock input
  - DC - 200 ns instruction cycle
- Interrupt capability
- 8-level deep hardware stack
- Direct, Indirect, and Relative Addressing modes

Special Microcontroller Features:

- Internal and external oscillator options
  - Precision Internal 4 MHz oscillator factory calibrated to ±1%
  - External Oscillator support for crystals and resonators
  - 5 µs wake-up from SLEEP, 3.0V, typical
- Power saving SLEEP mode
- Wide operating voltage range - 2.0V to 5.5V
- Industrial and Extended temperature range
- Low power Power-on Reset (POR)
- Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Brown-out Detect (BOD)
- Watchdog Timer (WDT) with independent oscillator for reliable operation
- Multiplexed MCLR/Input-pin
- Interrupt-on-pin change
- Individual programmable weak pull-ups
- Programmable code protection
- High Endurance FLASH/EEPROM Cell
  - 100,000 write FLASH endurance
  - 1,000,000 write EEPROM endurance
  - FLASH/Data EEPROM Retention: > 40 years

Low Power Features:

- Standby Current:
  - 1 nA @ 2.0V, typical
- Operating Current:
  - 8.5 µA @ 32 kHz, 2.0V, typical
  - 100 µA @ 1 MHz, 2.0V, typical
- Watchdog Timer Current
  - 300 nA @ 2.0V, typical
- Timer1 oscillator current:
  - 4 µA @ 32 kHz, 2.0V, typical

Peripheral Features:

- 6 I/O pins with individual direction control
- High current sink/source for direct LED drive
- Analog comparator module with:
  - One analog comparator
  - Programmable on-chip comparator voltage reference (C_{VREF}) module
  - Programmable input multiplexing from device inputs
  - Comparator output is externally accessible
- Analog-to-Digital Converter module (PIC12F675):
  - 10-bit resolution
  - Programmable 4-channel input
  - Voltage reference input
- Timer0: 8-bit timer/counter with 8-bit programmable prescaler
- Enhanced Timer1:
  - 16-bit timer/counter with prescaler
  - External Gate Input mode
  - Option to use OSC1 and OSC2 in LP mode as Timer1 oscillator, if INTOSC mode selected
- In-Circuit Serial Programming™ (ICSP™) via two pins

Device	Program Memory	Data Memory		I/O	10-bit A/D (ch)	Comparators	Timers 8/16-bit
	FLASH (words)	SRAM (bytes)	EEPROM (bytes)				
PIC12F629	1024	64	128	6	–	1	1/1
PIC12F675	1024	64	128	6	4	1	1/1

* 8-bit, 8-pin devices protected by Microchip's Low Pin Count Patent: U.S. Patent No. 5,847,450. Additional U.S. and foreign patents and applications may be issued or pending.

Figura 2.1: Principais características do PIC12F675
Fonte: Microchip Technology Inc., 2012

Peripheral features – características de periféricos: 6 pinos disponíveis para entrada/saída; possibilidade de fornecimento de corrente para acionamento de LEDs; módulo comparador analógico com tensão de referência programável; módulo conversor analógico/digital de resolução de 10 *bits* com quatro canais de entrada multiplexáveis e entrada para tensão de referência do A/D; *timer*/contador de 8 *bits* com *prescaler* programável; e *timer* melhorado de 16 *bits* e Programação Serial *In-Circuit* (ICSP) através de dois pinos.

2.2 Aplicações simples com PIC12F675

Para ilustrar a aplicação prática dos microcontroladores, veremos dois circuitos: um contador de eventos e um controlador de acionamento de bombeamento de água.

2.2.1 Contador de eventos

Um contador de eventos pode ser utilizado, por exemplo, para contar o número de peças produzidas em uma linha de montagem num determinado período. O circuito ilustrado na Figura 2.2 possui um sensor ótico (fototransistor T1) que é acionado toda vez que uma peça passa por ele em uma correia transportadora.

Quando o feixe luminoso é interrompido, o fototransistor (T1) vai da saturação para o corte, e a tensão do coletor vai de nível zero para nível alto. O nível alto aciona o pino de interrupção externa (pino 5), cuja rotina de tratamento incrementa um contador e o seu valor é transmitido para a porta serial de um microcomputador através dos pinos 6 e 7 do PIC.

O LED D2 é chamado de *heart beat* (batimento cardíaco), pois fica piscando a cada 500 ms apenas para indicar o funcionamento do circuito. Esse LED é acionado através da interrupção de um temporizador interno (*timer* 0).

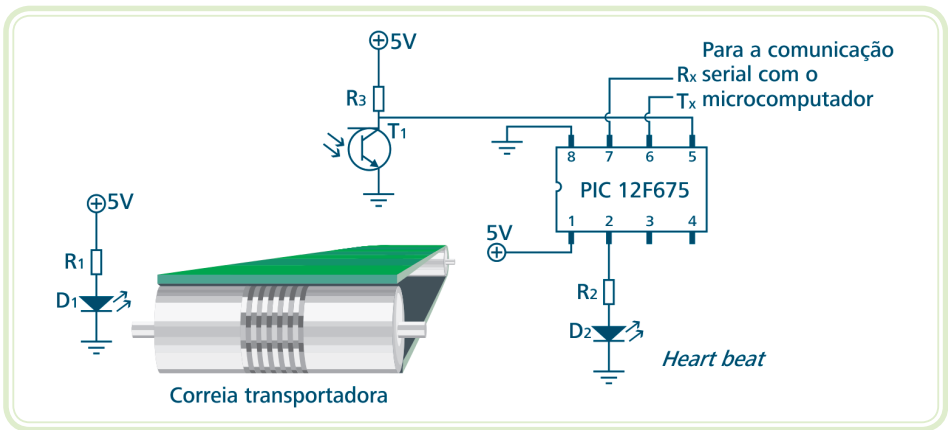


Figura 2.2: Contador de eventos
Fonte: Autor

2.2.2 Acionador de um sistema de bombeamento de água

A Figura 2.3 apresenta o diagrama esquemático de um sistema de bombeamento de água.

Nesse modelo, há duas caixas d'água, uma inferior e outra superior, ambas equipadas com sensores de nível que fornecem um sinal analógico proporcional ao nível da água na caixa.

No circuito, o PIC é programado para acionar a bomba d'água sempre que o nível da caixa inferior estiver acima de 10% do total (para evitar que a bomba d'água seja ligada sem água) e o nível da caixa superior estiver abaixo de 50%.

Uma vez acionada a bomba d'água, a mesma permanece ligada até que a água atinja 90% do nível da caixa superior ou até que a caixa inferior esvazie.

Observe a necessidade de uma interface de potência para ligar o PIC na bomba d'água, uma vez que a corrente máxima de um pino do PIC é da ordem de 20 mA e a corrente de acionamento da bomba d'água é de alguns ampères, alternada e em tensão elevada (127 V ou 220 V).

Os sensores de nível normalmente são ajustados para fornecer uma tensão de 1 V a 5 V, na faixa de vazio a cheio da caixa d'água. Valores abaixo de 1 V indicam falha no sensor de nível, e essa é uma forma de implementar segurança a falha no sistema de sensoriamento.

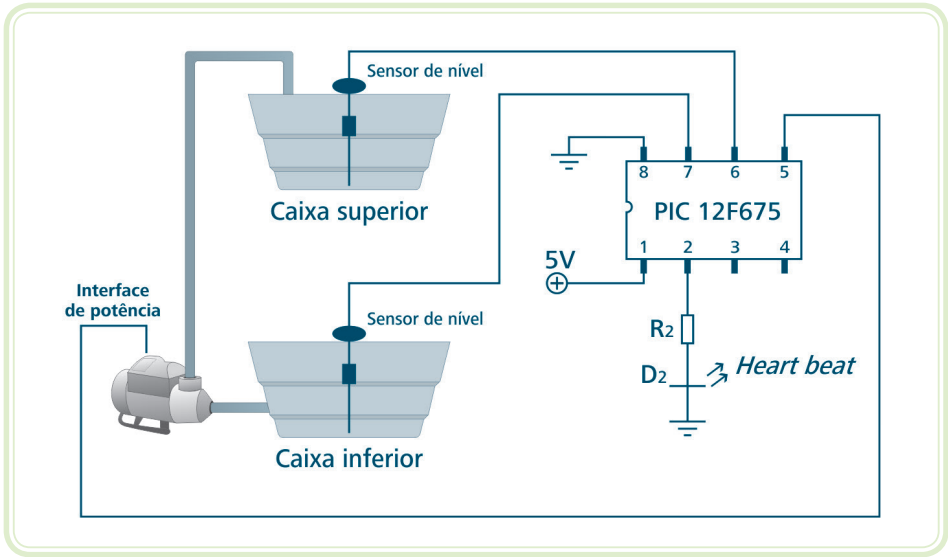


Figura 2.3: Sistema de bombeamento
Fonte: Autor

Resumo

Nesta aula, você conheceu um pouco sobre a família de microcontroladores PIC, em especial a do PIC12F675, bem como duas aplicações práticas envolvendo esse microcontrolador.

Atividades de aprendizagem

1. Quem é o fabricante do PIC12F675?
2. Quantas instruções Assembly o PIC possui?
3. Qual é o consumo de corrente do PIC no *clock* de 1 MHz?
4. Qual é a faixa de tensão de alimentação do PIC?
5. Qual é a resolução (em *bits*) do conversor A/D do PIC?
6. Quantos temporizadores o PIC12F675 possui?
7. Nos circuitos apresentados, qual é a função do LED *heart beat*?
8. Nos circuitos apresentados, qual deles utiliza o conversor A/D?
9. Nos circuitos apresentados, qual deles utiliza a interrupção externa?
10. Descreva, sucintamente, outra aplicação de uso do PIC12F675.

Aula 3 – *Hardware* do microcontrolador

Objetivos

Conhecer o *hardware* interno do microcontrolador PIC12F675.

3.1 Descrição dos pinos

A Figura 3.1 apresenta a pinagem do PIC12F675.

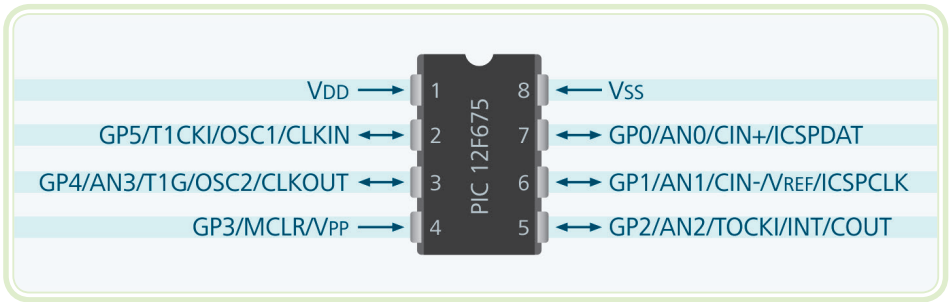


Figura 3.1: Pinagem do PIC12F675
Fonte: Microchip Technology Inc., 2012

- O pino 1 é a entrada de alimentação positiva de +5 V.

- O pino 2 pode assumir as seguintes configurações: entrada e saída (configurável) geral (GP5), *clock* do *timer* 1 (T1CK), entrada1 do circuito externo de oscilador de *clock* (OSC1) ou, ainda, entrada de sinal externo de *clock*.

- O pino 3 pode assumir as seguintes configurações: entrada e saída (configurável) geral (GP4), quarta entrada analógica (AN3), entrada2 do circuito externo de oscilador de *clock* (OSC2) ou, ainda, saída de sinal externo de *clock*.

- O pino 4 pode assumir as seguintes configurações: entrada (configurável) geral (GP3), *master clear* (baixo ativo) ou tensão de programação (VPP).

- O pino 5 pode assumir as seguintes configurações: entrada e saída (configurável) geral (GP2), terceira entrada analógica (AN2), *clock* do *timer* 0, interrupção externa (INT) ou, ainda, saída do comparador interno (COUT).

- O pino 6 pode assumir as seguintes configurações: entrada e saída (configurável) geral (GP1), segunda entrada analógica (AN1), entrada inversora do comparador interno (CIN-), tensão de referência do comparador interno (VREF) ou, ainda, *clock* da programação *in-circuit* (ICSPCLK).

- O pino 7 pode assumir as seguintes configurações: entrada e saída (configurável) geral (GP0), primeira entrada analógica (AN0), entrada não inversora do comparador interno (CIN+) ou, ainda, dados da programação *in-circuit* (ICSPDAT).

- O pino 8 é o terra da alimentação.

3.2 Memória

O PIC12F675 possui dois tipos de memória: memória de programa com 1024 palavras (*words*) de 8 *bits*, e memória de dados, com 64 *bytes* de RAM estática e 128 *bytes* de EEPROM.

A memória de programa é onde fica armazenado o programa gravado no PIC e que será executado tão logo o mesmo seja ligado na alimentação.

Na memória de dados, armazena-se as variáveis do programa, ou até 128 *bytes* de dados na memória EEPROM, que serão mantidos mesmo que o circuito seja desligado da alimentação.

3.3 Registradores

Os registradores (SFR – *Special Function Regiters*) são posições da memória que recebem nomes específicos e têm função bem definida: guardar a configuração e o estado de funcionamento atual do PIC.

Normalmente, cada *bit* do registrador tem uma função específica. Assim, temos um registrador para definir se as portas são de entrada ou de saída, ativar e desativar interrupções, apresentar o estado da CPU, etc.

Os principais registradores do PIC12F675 e seus respectivos endereços, em hexadecimal, são:

a) TMR0 (01H) – armazena a contagem do *timer*. Sempre que este contador chegar a zero e o INTCON estiver ativado, a interrupção de *timer* 0 será ativada.

b) STATUS (03H) – mostra o estado interno da CPU.

c) GPIO (05H) – apresenta o estado dos pinos de entrada/saída.

d) INTCON (0BH) – ativa/desativa o conjunto de todas as interrupções e cada uma delas de forma independente.

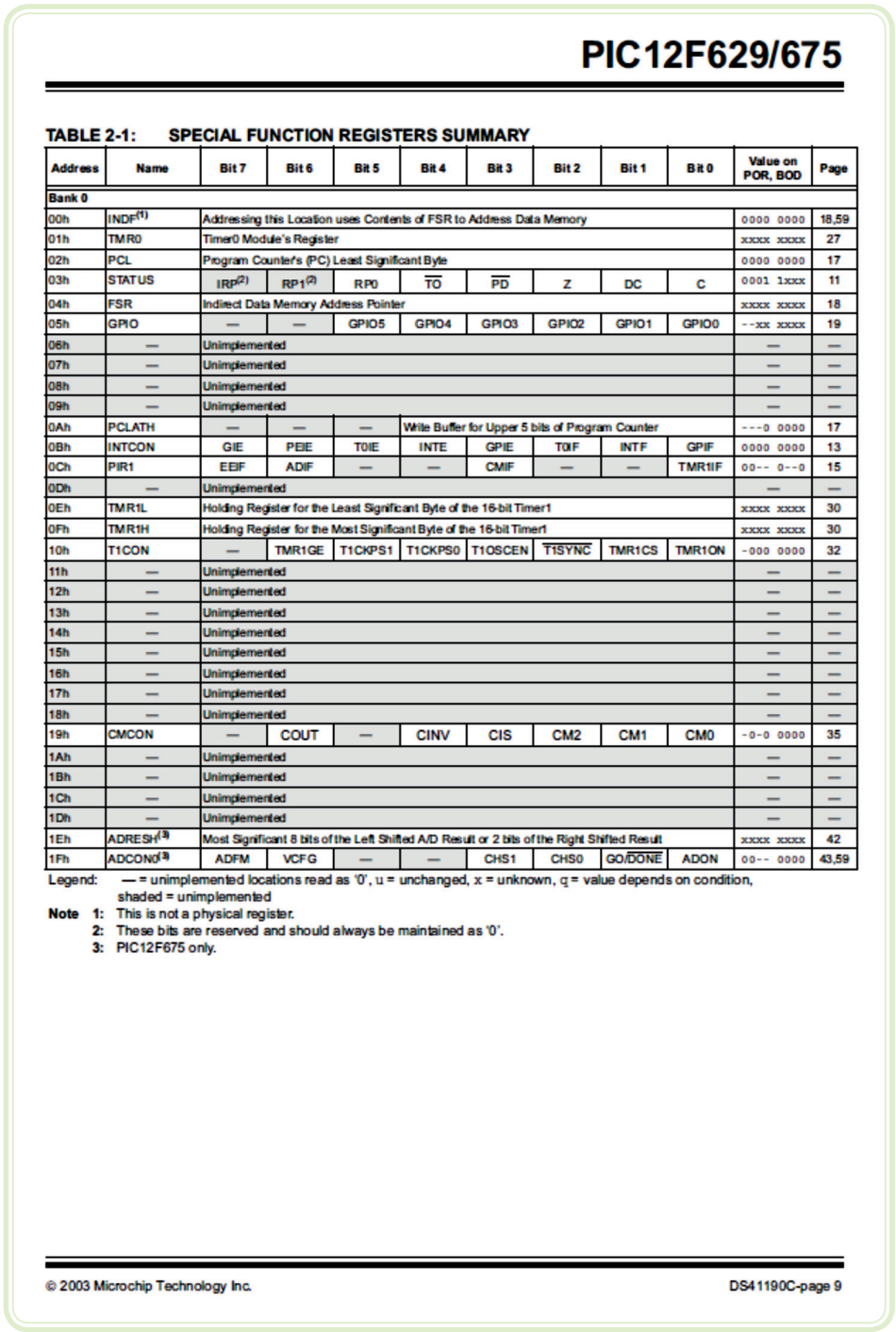
e) CMCON (19H) – apresenta o estado das entradas e da saída do comparador interno.

f) ADCON (1FH) – apresenta o estado do conversor A/D.

g) TRISIO (85H) – define se os pinos de entrada/saída atuarão como entrada ou como saída (individualmente).

h) ANSEL (9FH) – seleciona o estado de cada um dos pinos de entrada/saída quanto à sua operação como pino analógico ou digital.

As Figuras 3.2 e 3.3 apresentam as tabelas do fabricante contendo todos os registradores.



PIC12F629/675

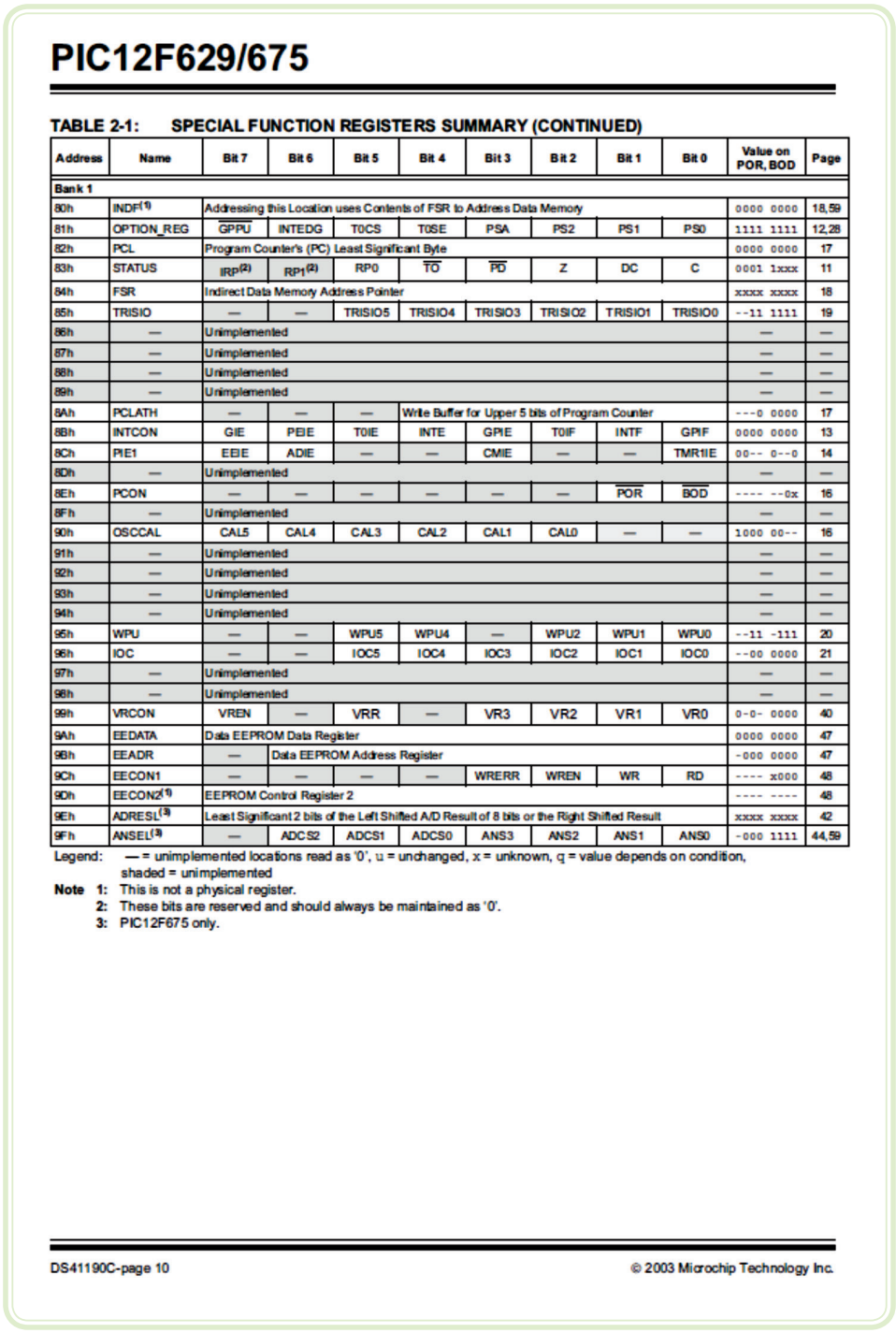
TABLE 2-1: SPECIAL FUNCTION REGISTERS SUMMARY

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOD	Page
Bank 0											
00h	INDF(1)	Addressing this Location uses Contents of FSR to Address Data Memory								0000 0000	18,59
01h	TMR0	Timer0 Module's Register								xxxx xxxx	27
02h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	17
03h	STATUS	IRP(2)	RP1(2)	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	11
04h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	18
05h	GPIO	—	—	GPIO5	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0	--xx xxxx	19
06h	—	Unimplemented								—	—
07h	—	Unimplemented								—	—
08h	—	Unimplemented								—	—
09h	—	Unimplemented								—	—
0Ah	PCLATH	—	—	—	Write Buffer for Upper 5 bits of Program Counter					---0 0000	17
0Bh	INTCON	GIE	PEIE	T0IE	INTE	GPIE	T0IF	INTF	GPIF	0000 0000	13
0Ch	PIR1	EEIF	ADIF	—	—	CMIF	—	—	TMR1IF	00-- 0--0	15
0Dh	—	Unimplemented								—	—
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit Timer1								xxxx xxxx	30
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit Timer1								xxxx xxxx	30
10h	T1CON	—	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON	-000 0000	32
11h	—	Unimplemented								—	—
12h	—	Unimplemented								—	—
13h	—	Unimplemented								—	—
14h	—	Unimplemented								—	—
15h	—	Unimplemented								—	—
16h	—	Unimplemented								—	—
17h	—	Unimplemented								—	—
18h	—	Unimplemented								—	—
19h	CMCON	—	COUT	—	CINV	CIS	CM2	CM1	CM0	-0-0 0000	35
1Ah	—	Unimplemented								—	—
1Bh	—	Unimplemented								—	—
1Ch	—	Unimplemented								—	—
1Dh	—	Unimplemented								—	—
1Eh	ADRESH(3)	Most Significant 8 bits of the Left Shifted A/D Result or 2 bits of the Right Shifted Result								xxxx xxxx	42
1Fh	ADCON0(3)	ADFM	VCFG	—	—	CHS1	CHS0	GO/$\overline{DONE}$	ADON	00-- 0000	43,59

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

Note 1: This is not a physical register.
2: These bits are reserved and should always be maintained as '0'.
3: PIC12F675 only.

Figura 3.2: Registradores do PIC12F675
Fonte: Microchip Technology Inc., 2012



PIC12F629/675

TABLE 2-1: SPECIAL FUNCTION REGISTERS SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOD	Page
Bank 1											
80h	INDF(1)	Addressing this Location uses Contents of FSR to Address Data Memory								0000 0000	18,59
81h	OPTION_REG	GPPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	12,28
82h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	17
83h	STATUS	IRP(2)	RP1(2)	RP0	TO	PD	Z	DC	C	0001 1xxx	11
84h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	18
85h	TRISIO	—	—	TRISIO5	TRISIO4	TRISIO3	TRISIO2	TRISIO1	TRISIO0	--11 1111	19
86h	—	Unimplemented								—	—
87h	—	Unimplemented								—	—
88h	—	Unimplemented								—	—
89h	—	Unimplemented								—	—
8Ah	PCLATH	—	—	—	Write Buffer for Upper 5 bits of Program Counter					---0 0000	17
8Bh	INTCON	GIE	PEIE	T0IE	INTE	GPIE	T0IF	INTF	GPIF	0000 0000	13
8Ch	PIE1	EEIE	ADIE	—	—	CMIE	—	—	TMR1IE	00-- 0--0	14
8Dh	—	Unimplemented								—	—
8Eh	PCON	—	—	—	—	—	—	POR	BOD	---- --0x	16
8Fh	—	Unimplemented								—	—
90h	OSCCAL	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	—	—	1000 00--	16
91h	—	Unimplemented								—	—
92h	—	Unimplemented								—	—
93h	—	Unimplemented								—	—
94h	—	Unimplemented								—	—
95h	WPU	—	—	WPU5	WPU4	—	WPU2	WPU1	WPU0	--11 -111	20
96h	IOC	—	—	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0	--00 0000	21
97h	—	Unimplemented								—	—
98h	—	Unimplemented								—	—
99h	VRCON	VREN	—	VRR	—	VR3	VR2	VR1	VR0	0-0- 0000	40
9Ah	EEDATA	Data EEPROM Data Register								0000 0000	47
9Bh	EEADR	—	Data EEPROM Address Register							-000 0000	47
9Ch	EECON1	—	—	—	—	WRERR	WREN	WR	RD	---- x000	48
9Dh	EECON2(1)	EEPROM Control Register 2								---- ----	48
9Eh	ADRESL(3)	Least Significant 2 bits of the Left Shifted A/D Result of 8 bits or the Right Shifted Result								xxxx xxxx	42
9Fh	ANSEL(3)	—	ADCS2	ADCS1	ADCS0	ANS3	ANS2	ANS1	ANS0	-000 1111	44,59

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

Note 1: This is not a physical register.
2: These bits are reserved and should always be maintained as '0'.
3: PIC12F675 only.

DS41190C-page 10 © 2003 Microchip Technology Inc.

Figura 3.3: Registradores do PIC12F675 (continuação)
Fonte: Microchip Technology Inc., 2012

3.4 Pinos de I/O

O PIC12F675 possui seis pinos de I/O (conforme Figura 3.1):

a) Pino 7 – GP0 – entrada ou saída, analógica ou digital, configurado nos registradores TRISIO e ANSEL, respectivamente.

b) Pino 6 – GP1 – entrada ou saída, analógica ou digital, configurado nos registradores TRISIO e ANSEL, respectivamente.

c) Pino 5 – GP2 – entrada ou saída, analógica ou digital, configurado nos registradores TRISIO e ANSEL, respectivamente.

d) Pino 4 – GP3 – apenas entrada, analógica ou digital, configurado nos registradores TRISIO e ANSEL, respectivamente.

e) Pino 3 – GP4 – entrada ou saída, analógica ou digital, configurado nos registradores TRISIO e ANSEL, respectivamente.

f) Pino 2 – GP5 – entrada ou saída, analógica ou digital, configurado nos registradores TRISIO e ANSEL, respectivamente.

Os pinos 1 e 8 são utilizados para alimentação positiva e terra, respectivamente.

3.5 Temporizadores

Os temporizadores têm ampla aplicação, pois permitem a marcação precisa de intervalos de tempo. O PIC12F675 possui dois temporizadores: um de 8 *bits* (*timer* 0) e um de 16 *bits* (*timer* 1).

O *timer* 0 conta de 0 a 255 (8 *bits*) e o *timer* 1 conta de 0 a 65535 (16 *bits*). Sempre que o *timer* atingir sua contagem máxima e nós adicionarmos mais uma unidade, ele retornará a zero; quando isso acontece, falamos que houve um estouro ou transbordamento do *timer*. Neste momento, a interrupção associada ao *timer* é acionada, caso a mesma esteja habilitada.

Como cada incremento do *timer* gasta, exatamente, um ciclo de máquina, é possível inicializar o *timer* com o valor adequado a fim de produzir a contagem de tempo que se deseja.

Por exemplo, se desejarmos contar 100 µs, podemos inicializar o *timer* 0 com 156 (256 - 100) para um ciclo de máquina de 1 µs. Quando o *timer* 0 atingir 255 e tentar passar para 256, ele retornará a zero e terá se passado exatamente 100 incrementos de 1 µs, totalizando 100 µs.

Resumo

Nesta aula, você iniciou o aprendizado sobre o *hardware* interno do PIC12F675, conheceu a pinagem do mesmo e aprendeu sobre a memória, os temporizadores e os registradores internos desse microcontrolador.

Atividades de aprendizagem

1. Quantos pinos de I/O o PIC12F675 tem disponível?

2. Em qual pino do PIC12F675 devemos ligar um sinal que produza interrupção externa?

3. Quantos conversores A/D existem no PIC12F675?

4. Quais pinos do PIC12F675 podem ser utilizados como entrada de conversão A/D?

5. Qual é o registrador utilizado para definir se um pino é de entrada ou de saída no PIC12F675?

6. Qual é o registrador utilizado para ativar/desativar as interrupções no PIC12F675?

7. Quantos e quais os temporizadores do PIC12F675?

8. O que acontece quando a contagem de um *timer* atinge o seu limite?

9. Qual deve ser o valor inicial de um *timer* de 16 *bits* para contar o intervalo de tempo de 1 ms com ciclo de máquina de 1 µs?

Aula 4 – *Hardware* do microcontrolador II

Objetivos

Conhecer o *hardware* interno do microcontrolador PIC12F675.

4.1 Comparadores

O módulo comparador consiste em um circuito de comparador analógico (Figura 4.1) que pode ter suas entradas e sua saída acessadas pelos pinos do PIC. É controlado pelo registrador CMCON que permite desligar ou ligar os pinos do comparador aos pinos externos do PIC (5, 6 e 7).

A saída do comparador vai ao nível alto sempre que o valor da entrada não inversora for maior que o valor da entrada inversora.

Através do registrador VRCON podemos ajustar o nível de tensão de referência a ser aplicada à entrada inversora do comparador.

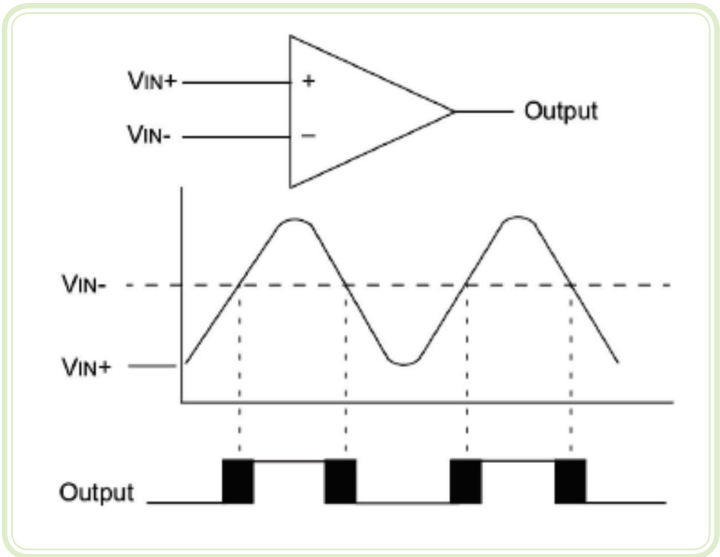


Figura 4.1: Comparador
Fonte: Microchip Technology Inc., 2012

4.2 PWM

O módulo PWM (*Pulse-Width Modulation* – Modulação por Largura de Pulso) não existe no PIC12F675, entretanto, devido à sua importância e considerando que são encontrados nos PICs de maior número de pinos, nós iremos estudá-lo.

O módulo PWM consiste em um oscilador de onda retangular onde se fixa a frequência e se alterna o ciclo ativo (*duty cycle*), conforme ilustrado na Figura 4.2. Normalmente, os PICs possuem o PWM com ajuste da largura de pulso de 10 *bits* (210 = 1.023), ou seja, podemos ajustar o nível alto, desde zero (saída desligada) até 1.023 que representa o máximo do sinal (saída ligada continuamente).

Através do PWM podemos gerar um sinal contínuo (por meio da filtragem ou da integração) a partir de um sinal digital pulsado.

O valor médio de saída vale:

$$VS = VP \times \frac{TON}{T}$$

Onde: VS – Tensão de saída
VP – Tensão máxima de saída
TON – Tempo de nível alto (*duty cycle* – valor variável)
T – Período do sinal (valor fixo)

O PWM é muito utilizado para o controle de velocidade de motores de corrente contínua.

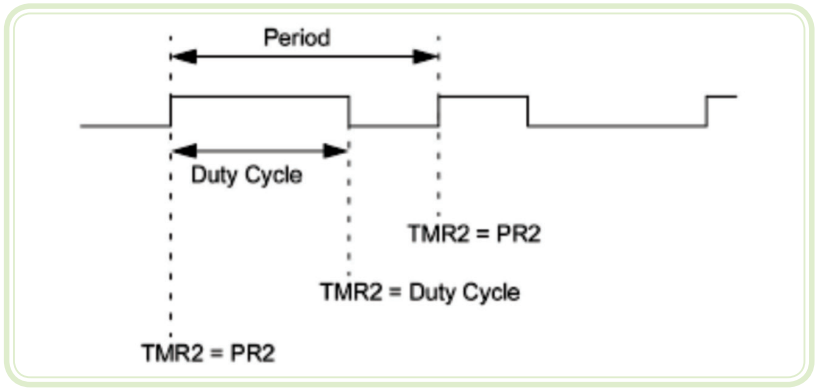


Figura 4.2: Modulação por largura de pulso PWM
Fonte: Microchip Technology Inc., 2012

4.3 Conversor A/D

O conversor analógico-digital (A/D) efetua a conversão de um sinal analógico para a sua representação digital de 10 *bits*. O PIC12F675 possui quatro entradas analógicas (GP0 a GP3) que são multiplexadas para um circuito de amostragem e retenção conforme ilustrado na Figura 4.3. A saída do circuito de amostragem e retenção é ligado à entrada do conversor A/D de 10 *bits*.

O conversor A/D gera um resultado binário através de um processo de aproximação sucessiva e armazena o resultado em um registrador de 10 *bits*. A tensão de referência utilizada pelo conversor pode ser selecionada por *software*, entre a tensão de alimentação ou a tensão aplicada ao pino "Vref".

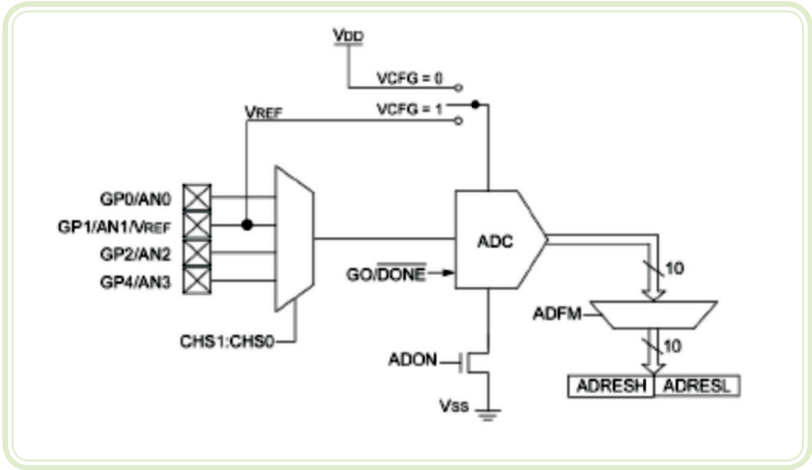


Figura 4.3: Conversor analógico digital
Fonte: Microchip Technology Inc., 2012

4.4 USART

O módulo USART (*Universal Synchronous Asynchronous Receiver Transmitter* – Transmissor/Receptor Universal Síncrono e Assíncrono), utilizado para a comunicação serial, também não está presente no PIC12F675 (neste caso, devemos implementar a comunicação serial por *software*), porém, o assunto será abordado devido à sua importância e à sua existência em vários microcontroladores.

Esse módulo implementa todo o protocolo lógico de comunicação pela porta serial RS-232 com o microcomputador. Para o protocolo físico devemos utilizar um conversor de níveis (como o MAX232), uma vez que o PIC fornecerá níveis de tensão de 0 V a 5 V e a RS-232 trabalha com níveis de +15 V a -15 V. A Figura 4.4 ilustra alguns desses elementos.

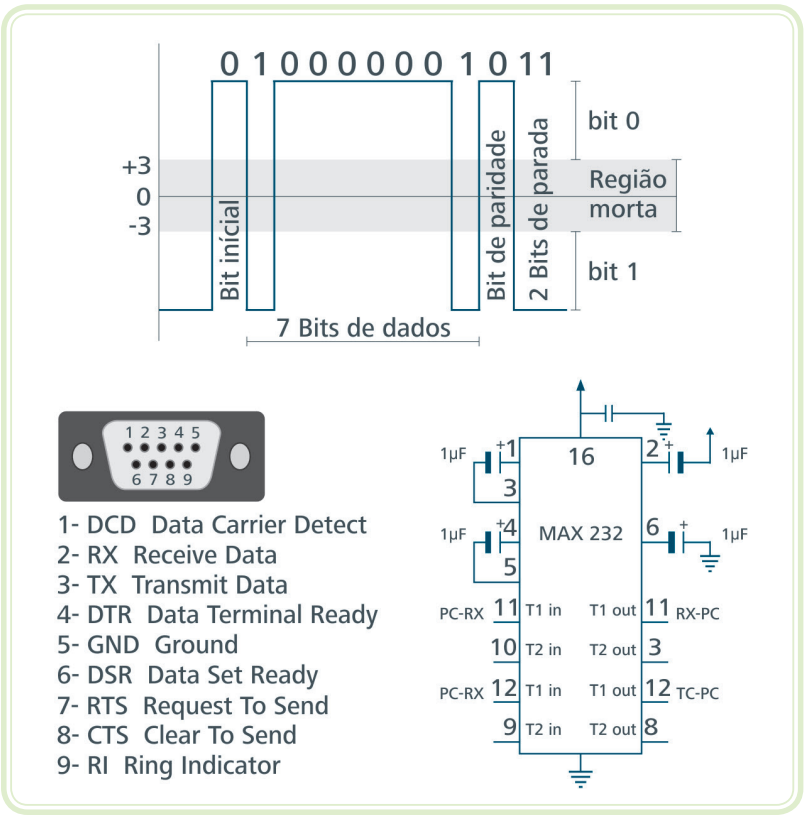


Figura 4.4: Comunicação serial
Fonte: Microchip Technology Inc., 2012

Resumo

Com esta aula, você concluiu o aprendizado sobre o *hardware* interno do PIC12F675. Conheceu a utilização do comparador interno, da modulação por largura de pulso (PWM), do conversor A/D e da comunicação serial.

Atividades de aprendizagem

1. Quando é que a saída do comparador vai ao nível alto?

2. Qual é o registrador responsável pelo ajuste da tensão de referência do comparador?

3. Descreva, sucintamente, o funcionamento do PWM.

4. Cite uma aplicação do PWM.

5. Quantos conversores A/D o PIC12F675 possui?

6. Qual é o processo de conversão utilizado pelo PIC12F675?

7. Qual é a função do módulo USART?

8. Qual é a função do *chip* MAX232?

Aula 5 – *Set* de instruções

Objetivos

Apresentar o conjunto de instruções Assembly do microcontrolador PIC12F675.

5.1 Estrutura das instruções

Todas as operações lógicas e aritméticas do PIC12F675 realizadas pela ULA (Unidade Lógica e Aritmética) do processador estão relacionadas com o registrador de trabalho *working register*, chamado de W.

Essas operações lógicas e aritméticas podem ser realizadas entre o conteúdo do registrador W e o conteúdo de outro registrador ou uma constante, chamada de literal pela Microchip.

Este microcontrolador possui vários tipos de periféricos, cujo funcionamento é gerenciado através de registradores específicos chamados SFRs (*Special Function Registers* – Registradores com Função Especial).

Como os SFRs estão presentes fisicamente na memória de dados do microcontrolador, devem ser ajustados sempre antes da execução repetitiva da rotina principal, de acordo com as funções pretendidas para o circuito.

Os SFRs são organizados em dois blocos, chamados de Banco de Memória 0 e Banco de Memória 1, nos quais a seleção é feita através do *bit* 5 do registrador STATUS. Sendo assim, antes de ler ou escrever em um SFR, devemos nos assegurar que estamos no Banco de Memória correto através do ajuste em 0 ou 1 do *bit* 5 do registrador STATUS.

Alguns modos e funções de operação do PIC12F675 não são selecionáveis diretamente pelo *firmware*, devendo ser configurados durante o processo de programação. Essa configuração se dá através de diretivas incluídas no programa fonte, as quais são usadas apenas na programação, não fazendo parte diretamente do *firmware* propriamente dito (não são instruções passíveis de serem executadas pelo processador do microcontrolador).

5.2 Grupos de instruções

Didaticamente, as instruções do microcontrolador PIC são organizadas em três grupos: instruções orientadas a registradores com tamanho de um *byte*, instruções orientadas aos *bits* de registradores e instruções orientadas a operações com constantes literais e controle de fluxo do programa. Detalhando as instruções temos, segundo Lavínia (2002):

5.2.1 Instruções orientadas a registradores com tamanho de um *byte*

- ADDW F – (*add file* em *work*) adiciona os conteúdos dos registradores W e F.

- ANDWF – (*and file* e *work*) executa a operação E (*and*) lógica entre os conteúdos dos registradores W e F.

- CLRF – (*clear file*) carrega o valor 0 no registrador F.

- CLRW – (*clear work*) carrega o valor 0 no registrador W.

- COMF – (*complement file*) complementa (inverte *bit* a *bit*) o conteúdo do registrador F.

- DECF – (*dec file*) decrementa (subtrai um) do conteúdo do registrador F.

- DECFSZ – (*dec file*, *skip* se zero) decrementa (subtrai um) do conteúdo do registrador F e não executa (pula) a próxima instrução se o resultado do incremento for igual 0.

- INCF – (*inc file*) incrementa (soma um) o conteúdo do registrador F.

- INCFSZ – (*inc file*, *skip* se zero) incrementa (soma um) ao conteúdo do registrador F e não executa (pula) a próxima instrução se o resultado do incremento for igual 0.

- IORWF – (W *or file*) executa a operação OU (*or*) lógica entre os conteúdos dos registradores W e F.

- MOVF – (*move file* para W) move o valor do registrador F para o registrador W.

- MOVWF – (*move* W para F) copia o conteúdo do registrador W para o registrador F.

- NOP – (*no operation*) nenhuma operação é executada, apenas gasta um ciclo de máquina.

- RLF – (*rotate left file*) rotaciona à esquerda o conteúdo do registrador F.

- RRF – (*rotate rigth file*) rotaciona à direita o conteúdo do registrador F.

- SUBWF – (*sub* f de W) subtrai o conteúdo do registradores F do registrador W e armazena o resultado em W.

- SWAPF – (troca *file*) troca os 4 *bits* mais significativos com os 4 menos significativos de F.

- XORWF – (WXORF) executa a operação "ou exclusivo" (*xor*) lógica entre os conteúdos dos registradores W e F e armazena em W.

5.2.2 Instruções orientadas aos *bits* de registradores

- BCF – (*bit clear file*) ajusta o *bit* b do registrador f para nível baixo (0).

- BSF – (*bit set file*) ajusta o *bit* b do registrador f para nível alto (1).

- BTFSC – (*bit* teste *file*, *skip* se *clear*) testa o *bit* b do registrador f e não executa a próxima instrução se ele estiver em nível baixo (0).

- BTFSS – (*bit* teste *file*, *skip* se *set*) testa o *bit* b do registrador f e não executa a próxima instrução se ele estiver em nível baixo (1).

5.2.3 Instruções orientadas a operações com constantes literais e controle de fluxo do programa

- ADDLW – (*add* literal a W) o conteúdo do registrador W é somado a uma constante k de 8 *bits* e o resultado é guardado no registrador W.

- ANDLW – (*and* literal e W) executa a operação E (*and*) lógica entre o conteúdo do registrador W e uma constante k de 8 *bits* e armazena o resultado em W.

- CALL – chamada uma sub-rotina.

- CLRWDT – (*clear wdt*) carrega o valor 0 no temporizador do *watchdog timer*.

- GOTO – salto incondicional para outro local no programa.
- IORLW – (*or* literal ou W) executa a operação OU (*or*) lógica entre o conteúdo do registrador W e uma constante k de 8 *bits* e armazena o resultado em W.
- MOVLW – (*move* literal para W) carrega uma constante k de 8 *bits* no registrador W.
- RETFIE – retorno de uma rotina de interrupção.
- RETLW – retorno de uma sub-rotina com o carregamento de uma constante k de 8 *bits* (literal) no registrador W.
- RETURN – retorno de uma sub-rotina.
- SLEEP – coloca o processador no modo de baixo consumo.
- SUBLW – subtrai o literal de 8 *bits* do conteúdo do registrador W e armazena o resultado no registrador W.
- XORLW – executa a operação "ou exclusivo" (*xor*) lógica entre o conteúdo do registrador W e uma constante k de 8 *bits* (literal).

Resumo

Nesta aula, você conheceu o conjunto de instruções Assembly do PIC12F675, bem como a forma que elas são estruturadas e agrupadas para facilitar a memorização.

Atividades de aprendizagem

1. Onde são realizadas as operações lógicas e aritméticas do PIC12F675?
2. Qual é o principal registrador do PIC12F675?
3. Quem gerencia os periféricos do microcontrolador?
4. Quais são os grupos de instruções Assembly do PIC12F675?
5. Quantas são as instruções Assembly do PIC12F675?

Aula 6 – Interrupções

Objetivos

Compreender a função das interrupções em um microcontrolador.

Conhecer as interrupções disponíveis no microcontrolador PIC12F675.

6.1 Interrupções no microcontrolador

O método mais utilizado para a verificação do estado de um pino de entrada é a leitura frequente do nível nele presente (técnica de *polling* – sondagem) por ser o método de fácil implementação.

No entanto, a despeito desta simplicidade, esse método não se mostra adequado em situações onde é preciso uma resposta imediata do processador assim que houver uma mudança no nível de um pino.

Em tais casos, é recomendado que seja utilizado uma interrupção, ou seja, a chamada de uma função auxiliar que só é executada se houver ocorrido um evento externo específico, como por exemplo, a mudança do estado de um pino, conforme Pereira (2002).

Nesse caso, após a chamada da função auxiliar, o fluxo original do programa principal só será retomado quando a interrupção for concluída.

Além da citada interrupção por alteração de nível em pino, o PIC12F675 possui várias outras fontes de interrupção, que podem ser configuradas e usadas a partir de informações obtidas em seu *datasheet*.

Para utilizar as interrupções, o registrador INTCON.GIE deve ser ativado (INTCON – endereço 0BH, GIE – *bit* 7 do INTCON). O GIE funciona como uma espécie de chave geral de todas as interrupções; se colocarmos zero no INTCON.GIE, desabilitamos, simultaneamente, todas as interrupções.

Como o próprio nome indica, uma interrupção serve para interromper a execução normal do programa principal e, imediatamente, tratar do evento que a gerou.

6.2 Interrupção de *timer*

A interrupção de *timer* ocorre sempre que o contador do *timer* estoura, isto é, quando atinge o valor máximo e é incrementado de uma unidade. Por exemplo, o *timer* 0 (endereço 01H) é um temporizador de 8 *bits* (conta de 0 a 255). Quando o contador atingir 255, no próximo incremento ele estourará (tentará passar para 256), retornando a zero e disparando a interrupção de *timer* 0.

Para que essa interrupção funcione, deve-se ter INTCON.GIE = 1 (liga chave geral); INTCON.T0IE = 1 (liga *timer* 0). Sempre que ocorrer o estouro do contador, o *bit* INTCON.T0IF estará em 1 e a rotina de tratamento dessa interrupção será acionada.

A interrupção de *timer* é muito útil quando desejamos medir intervalos de tempo de forma precisa. Por exemplo, considerando um *clock* interno de 1 MHz, obteremos um ciclo de máquina de 1 µs. Carregando o *timer* 0 com o valor 250, após 5 ciclos de máquina (5 µs) teremos a ocorrência da interrupção de *timer*, a qual poderá ser utilizada para incrementar um contador que contará de forma precisa, de 5 µs em 5 µs.

6.3 Interrupção externa

A interrupção externa é muito utilizada, pois permite a detecção exata do instante em que os eventos externos acontecem, tais como: quando algum objeto passou em frente a um sensor de presença, quando um eixo que gira completou uma volta, quando a tensão da rede passou por zero, etc.

Essa interrupção pode ser disparada pela borda de subida ou pela borda de descida do sinal. Tal seleção é feita no registrador OPTION_REG.INTDEG (81H.6).

A interrupção é ativada fazendo-se INTCON.GIE = 1 (liga chave geral); INTCON.INTE = 1 (liga INT EXT). Sempre que ocorrer uma transição de sinal no pino correspondente, INTCON.INTF (0BH.1) estará em 1 e a rotina de tratamento dessa interrupção será acionada.

Antes de sair da rotina de interrupção, esse *flag* deverá ser colocado em zero.

6.4 Outras interrupções

O microcontrolador PIC possui outras interrupções, como:

a) Interrupção de mudança de estado – ocorre quando algum pino muda de nível de sinal.

b) Interrupção de escrita na EEPROM – ocorre quando se conclui a escrita na memória EEPROM.

c) Interrupção de fim de conversão A/D – ocorre quando se conclui a conversão de analógica para digital do conversor A/D.

d) Interrupção de WDT – ocorre quando estoura o *Watchdog timer* e o sistema é reinicializado.

Resumo

Nesta aula, você estudou a função das interrupções em microcontroladores e conheceu também quais são as interrupções disponíveis no PIC12F675.

Atividades de aprendizagem

1. Para que utilizamos as interrupções em um microcontrolador?

2. Qual registrador é utilizado como chave geral de interrupção no PIC12F675?

3. O que acontece quando o *flag* INTCON.INTF estiver em 1 no PIC12F675?

4. Como se ativa a interrupção externa no PIC12F675?

5. Qual interrupção é utilizada para se verificar o término de conversão do conversor A/D no PIC12F675?

Aula 7 – Programação Assembly

Objetivos

Descrever a programação Assembly de um microcontrolador.

Compreender a importância da programação Assembly no PIC12F675.

7.1 Programa de computador

Um programa de computador é um conjunto ordenado de instruções capazes de executar uma ação útil em um sistema computacional, o qual pode ser representado de diversas formas (linguagens). Em última análise, o processador só entende uma linguagem: a da máquina, constituída de "zeros" e "uns".

Com o objetivo de facilitar o entendimento, surgiu uma série de linguagens de programação que são, basicamente, formas mais elaboradas de se desenvolver o raciocínio computacional de maneira mais inteligível.

Todo programa de computador, em qualquer linguagem, deverá ser traduzido (através de um compilador ou de um interpretador) para linguagem de máquina.

Uma das linguagens de mais baixo nível (mais próxima da máquina) é o Assembly. Essa linguagem constitui-se de instruções simples (mnemônicos), que são a representação simbólica da linguagem de máquina. No caso do PIC, há 35 instruções.

7.2 Programando em Assembly MPASM

Programar se aprende programando. Para que você possa entender melhor os aspectos de programação do PIC, descreveremos alguns programas simples em MPASM, o Assembly da Microchip. Para isso, precisamos de um programa que os traduza para a linguagem de máquina.

A Microchip disponibiliza um programa gratuito que faz o papel de ambiente gráfico de desenvolvimento e vem com ferramentas de assemblador e linker, o MPLAB. O MPLAB pode ser obtido gratuitamente no *site* da Microchip

(www.microchip.com) para a plataforma Windows. Instale-o em seu computador e experimente-o. As Figuras 7.1 e 7.2 apresentam o ambiente de desenvolvimento (IDE) descrito anteriormente.

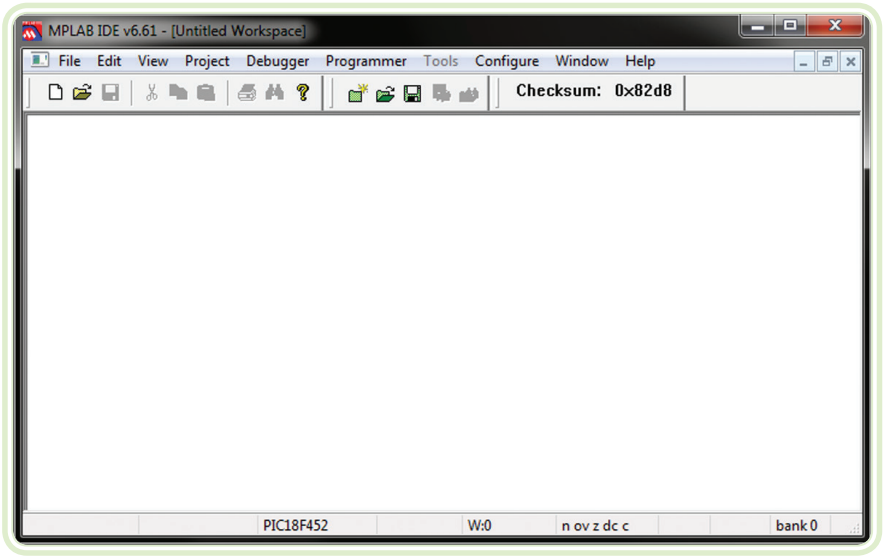


Figura 7.1: MPLAB – Tela de abertura
Fonte: Microchip Technology Inc., 2012

Um bom tutorial de MPLAB pode ser encontrado em: ftp://ftp.cefetes.br/Cursos/Eletrotecnica/Microprocessadores/PIC/MiniCursoMplab5.pdf

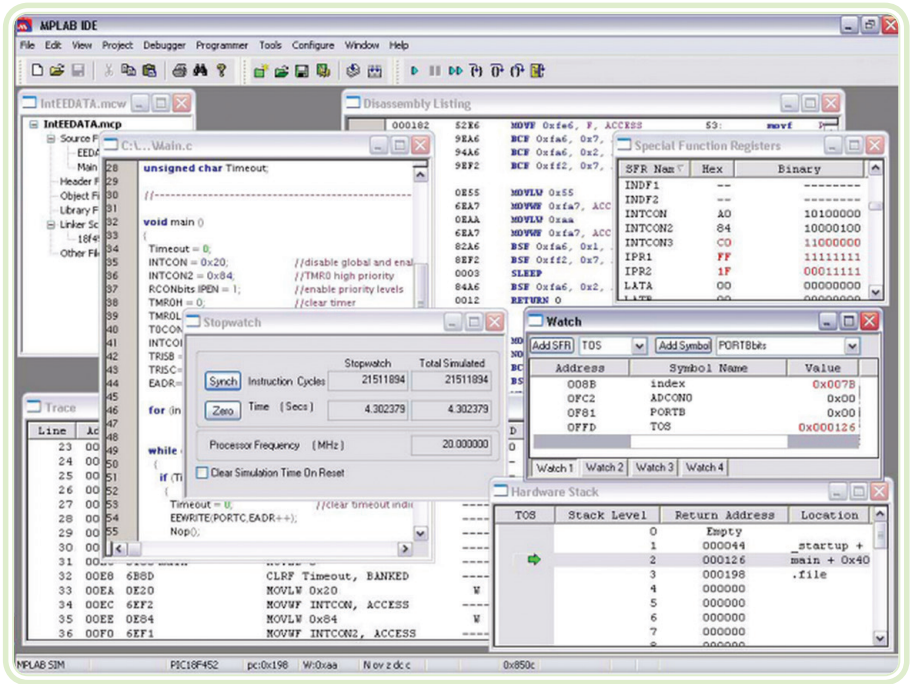


Figura 7.2: MPLAB – Tela de desenvolvimento
Fonte: Microchip Technology Inc., 2012

Devido à exiguidade de tempo, não trataremos do uso do MPLAB neste curso, mas você encontrará vários tutoriais na internet ou nas referências bibliográficas deste material.

7.3 Exemplos de programas

Para entender melhor os conceitos de programação, vamos analisar uma série de programas escritos para o PIC12F675, todos desenvolvidos considerando as ligações físicas de chaves, resistores, potenciômetros e LEDs, mostradas na Figura 7.3.

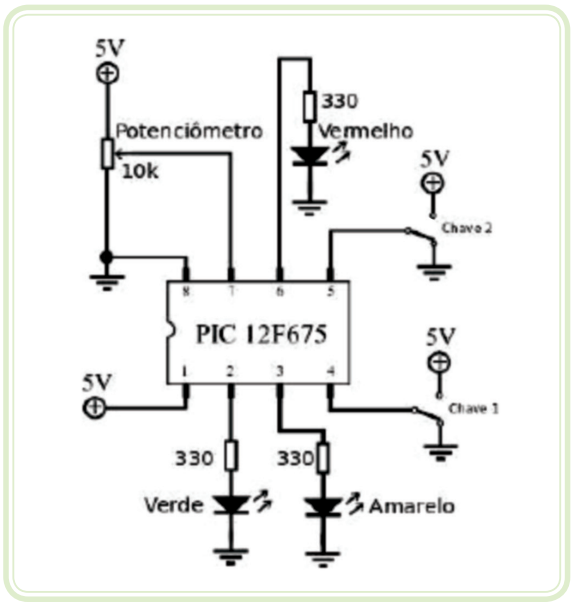


Figura 7.3: Circuito base
Fonte: Autor

7.4 Pisca LED

O programa a seguir faz com que o LED verde (ligado no pino 2) pisque em intervalos de 500 ms. A função desse programa pode ser implementada com o seguinte algoritmo:

a) Configurar os SFRs pertinentes.

b) Colocar o pino 2 em nível alto.

c) Pausar a execução do programa por um tempo.

d) Colocar o pino 2 em nível baixo.

e) Pausar a execução do programa por um tempo.

f) Repetir as operações de "b" a "e" indefinidamente.

Analisando o *datasheet* do PIC12F675, é possível descobrir quais SFRs devem ser configurados e com quais valores, de acordo com as especificações do sistema. Ao fazer isso, para este primeiro exemplo, constatamos que é preciso configurar os registradores a seguir para que o programa, que será escrito, funcione adequadamente com o circuito já apresentado:

a) ANSEL – seleção da função analógica ou digital para os pinos de entrada e saída que podem operar nesses dois modos, conforme descrito a seguir. Ao escrevermos "1" no *bit* associado ao pino, estaremos indicando que o seu funcionamento será no modo analógico; já se escrevermos "0", estaremos configurando o pino para trabalhar no modo digital.

- Pino 7 – modo digital (GPIO 0) ou modo analógico (AN 0).
- Pino 6 – modo digital (GPIO 1) ou modo analógico (AN 1).
- Pino 5 – modo digital (GPIO 2) ou modo analógico (AN 2).
- Pino 3 – modo digital (GPIO 4) ou modo analógico (AN 3).

b) TRISIO – definição da direção do sinal (entrada ou saída) em um pino. Ao escrevermos "0" no *bit* associado ao pino, estaremos configurando o pino como saída (*output*); já se escrevermos "1", estaremos configurando o pino como entrada (*input*).

c) CMCON – controle das funções do comparador de tensão interno do microcontrolador.

d) GPIO – seleção do nível lógico de um pino quando ele for configurado como saída digital, ou leitura do nível presente em um pino quando ele for configurado como entrada digital.

Para fazer a configuração correta desses SFRs, é preciso estar atento para o fato de que os registradores ANSEL e TRISIO estão presentes fisicamente no Banco de Memória 1, enquanto os registradores CMCON e GPIO estão presentes fisicamente no Banco de Memória 0. Sendo assim, antes de utilizá-los, deve-se configurar adequadamente o *bit* 5 (RP0) do registrado STATUS.

Um possível modo de escrever em Assembly um programa que se comporte dessa maneira é mostrado a seguir. Tudo o que está após o ";" é comentário e não é executado pelo PIC.

```
;INCLUSAO DE ARQUIVOS
#INCLUDE <p12f675.inc> ; definições referentes ao PIC12F675
;CONFIGURAÇÃO DO MODO DE FUNCIONAMENTO
__CONFIG _INTRC_OSC_NOCLKOUT & _WDT_OFF & _PWRTE_OFF & _MCLR_
OFF & _CPD_OFF & _CP_OFF &
;DEFINIÇÃO DOS NOMES E ENDEREÇOS DAS VARIÁVEIS UTILIZADAS NO
PROGRAMA
CBLOCK 0x0C ; endereço inicial da memória de usuário
TEMP1 ; contador para o delay de 1 ms
TEMP500 ; contador para o delay de 500 ms
ENDC ; fim do bloco de memória de variáveis
;DEFINIÇÃO DAS ENTRADAS DIGITAIS
#DEFINE Chave_1 GPIO, 2; Pino 5
#DEFINE Chave_2 GPIO, 3; Pino 4
;DEFINIÇÃO DAS SAÍDAS DIGITAIS
#DEFINE LED_VD GPIO, 5 ; Pino 2
#DEFINE LED_AM GPIO, 4 ; Pino 3
#DEFINE LED_VM GPIO, 1 ; Pino 6
;DEFINIÇÃO DO SELETOR DO BANCO DE MEMÓRIA
#DEFINE Banco STATUS, RP0 ; bit RP0 do registrador STATUS (SFR)
;CONFIGURAÇÃO DOS SFRs
BSF Banco ; Seleção do Banco 1 da Memória
MOVLW B'00000001' ; modo analógico/digital dos pinos de I/O
MOVWF ANSEL
MOVLW B'00001101' ; direção dos pinos de I/O digitais
MOVWF TRISIO
BCF Banco ; seleção do Banco 0 da Memória
MOVLW B'00000111' ; desativação do comparador de tensão
MOVWF CMCON
;INICIALIZAÇÃO DO PROGRAMA
BCF LED_VD ; apaga o LED verde
BCF LED_AM ; apaga o LED amarelo
BCF LED_VM ; apaga o LED vermelho
```

```
;ROTINA PRINCIPAL, implementa um loop infinito
Principal:
BSF LED_VD ; acende o LED verde
CALL DELAY_500MS ; pausa de 500 ms
BCF LED_VD ; apaga o LED verde
CALL DELAY_500MS ; pausa de 500 ms
GOTO Principal ; salto para o início da rotina principal
; Rotina de delay de 500 ms. Repete 200 x a rotina de 2,5 ms
DELAY_500MS:
MOVLW .200
MOVWF TEMP500
DL_50
CALL DELAY_2MS ; pausa de 10 ms
DECFSZ TEMP500,F ; decrementa TEMP500. Zerou?
GOTO DL_50 ; não, repete o ciclo.
RETURN ; sim, finaliza a rotina.
; Rotina de delay de 2,5 ms. Repete 250 x a rotina de 10 µs
DELAY_2MS:
MOVLW .250
MOVWF TEMP1
DL_10 ; cada ciclo gasta 10 microsegundos
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
DECFSZ TEMP1,F ; decrementa TEMP1. Zerou?
GOTO DL_10 ; não, repete o ciclo
RETURN ; sim, finaliza a rotina
;FIM DO PROGRAMA
END
```

7.4.1 Detalhando o programa "Pisca LED"

7.4.1.1 Inclusão

```
;INCLUSAO DE ARQUIVOS
#INCLUDE <p12f675.inc> ; definições referentes ao PIC12F675
```

Neste bloco, incluímos um arquivo externo, neste caso, o arquivo "p12f 675. inc" que contém os nomes dos registradores do microcontrolador em uso.

Configuração

```
;CONFIGURAÇÃO DO MODO DE FUNCIONAMENTO
    CONFIG _INTRC_OSC_NOCLKOUT & _WDT_OFF & _PWRTE_OFF &
MCLR_OFF & _CPD_OFF & _CP_OFF &
```

Neste bloco, configuramos os *fuses* internos do microcontrolador que funcionam como chaves internas, as quais podem ser ligadas ou desligadas, como por exemplo:

- _INTRC_OSC_NOCLKOUT – o PIC utilizará o oscilador interno de 4 MHz, o que garante que cada ciclo de máquina tenha 1 µs de duração.
- _WDT_OFF – desliga o *Watchdog timer*.
- _PWRTE_OFF – desliga o *reset* de *power on*.
- _MCLRE_OFF – desliga o *masterclear*.

7.4.1.2 Variáveis

```
;DEFINIÇÃO DOS NOMES E ENDEREÇOS DAS VARIÁVEIS UTILIZADAS NO
PROGRAMA
CBLOCK 0x0C ; endereço inicial da memória de usuário
TEMP1 ; contador para o delay de 1 ms;
TEMP500 ; contador para o delay de 500 ms
ENDC ; fim do bloco de memória de variáveis
```

Aqui, declaramos endereço inicial para a criação das variáveis, bem como a definição das mesmas.

7.4.1.3 Entradas e saídas

```
DEFINIÇÃO DAS ENTRADAS DIGITAIS
#DEFINE Chave_1 GPIO, 2; Pino 5
#DEFINE Chave_2 GPIO, 3; Pino 4
; DEFINIÇÃO DAS SAÍDAS DIGITAIS
#DEFINE LED_VD GPIO, 5 ; Pino 2
#DEFINE LED_AM GPIO, 4 ; Pino 3
#DEFINE LED_VM GPIO, 1 ; Pino 6
```

Neste bloco, associamos nomes significativos aos pinos do microcontrolador.

7.4.1.4 Configuração dos registradores

```
;DEFINIÇÃO DO SELETOR DO BANCO DE MEMÓRIA
#DEFINE Banco STATUS, RP0 ; bit RP0 do registrador STATUS (SFR)
;CONFIGURAÇÃO DOS SFRs
BSF Banco ; seleção do Banco 1 da Memória
MOVLW B'00000001'; modo analógico/digital dos pinos de I/O
MOVWF ANSEL
MOVLW B'00001101'; direção dos pinos de I/O digitais
MOVWF TRISIO
BCF Banco ; seleção do Banco 0 da Memória
MOVLW B'00000111'; desativação do comparador de tensão
MOVWF CMCON
```

Neste bloco, configuramos alguns registradores internos que serão utilizados no programa.

Perceba que, inicialmente, colocamos o valor numérico (literal) para o registrador W e, na sequência, transferimos o valor de W para o registrador alvo.

7.4.1.5 Inicialização

```
;INICIALIZAÇÃO DO PROGRAMA
BCF LED_VD ; apaga o LED verde
BCF LED_AM ; apaga o LED amarelo
BCF LED_VM ; apaga o LED vermelho
```

Aqui, inicializamos os pinos de saída, apagando todos os LEDs pela imposição de valor zero aos pinos correspondentes.

7.4.1.6 *Loop* principal

```
;ROTINA PRINCIPAL, implementa um loop infinito
PRINCIPAL:
BSF LED_VD ; acende o LED verde
CALL DELAY_500MS ; pausa de 500 ms
BCF LED_VD ; apaga o LED verde
CALL DELAY_500MS ; pausa de 500 ms
GOTO PRINCIPAL ; salto para o início da rotina principal
```

Agora, implementamos um *loop* infinito criando o *label* PRINCIPAL, executando algumas instruções e, no final, efetuando um salto incondicional (GOTO) para o mesmo *label*.

Iniciamos ligando o LED verde pela imposição de nível 1 no pino correspondente (chamamos uma sub-rotina que provoca um *delay* de 500 ms) colocamos nível "0' no LED (chamamos novamente a sub-rotina de *delay* de 500 ms) e recomeçamos a mesma sequência.

7.4.1.7 Rotinas de *delay*

Rotina de *delay* de 500 ms. Repete 200 × a rotina de 2,5 ms

```
DELAY_500MS:
MOVLW .200
MOVWF TEMP500
DL_50
CALL DELAY_2MS ; pausa de 2,5 ms
DECFSZ TEMP500,F ; decrementa TEMP500. Zerou?
GOTO DL_50 ; não, repete o ciclo.
RETURN ; sim, finaliza a rotina.
; Rotina de delay de 2,5 ms. Repete 250 × a rotina de 10 us
DELAY_2MS:
MOVLW .250
MOVWF TEMP1
DL_10 ; cada ciclo gasta 10 microssegundos
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
DECFSZ TEMP1,F ; decrementa TEMP1. Zerou?
GOTO DL_10 ; não, repete o ciclo.
RETURN ; sim, finaliza a rotina.
```

O *delay* de 500 ms é obtido através da repetição (200 vezes) da rotina de atraso de 2,5 ms. Inicia-se colocando o valor 200 na variável TEMP500 (através do registrador W) e efetua-se sucessivas chamadas à rotina de atraso de 2,5 ms. Após, deve-se ir decrementando a variável TEMP500 até que ela chegue a zero, momento em que haverá o retorno (RETURN) da sub-rotina.

Analogamente, há a rotina de atraso de 2,5 ms, obtida pela repetição (250 vezes) de um bloco que consome 10 microsegundos. Esse tempo é alcançado através de sete instruções NOP (que não possuem função alguma e consomem

1 ciclo de máquina), uma instrução DECFSZ (que consome mais 1 ciclo de máquina) e a instrução GOTO (que consome 2 ciclos de máquina).

7.4.1.8 Fim do programa

```
;FIM DO PROGRAMA
END
```

É imprescindível que o programa se encerre com a instrução END.

Resumo

Nesta aula, você aprendeu sobre a programação Assembly do PIC12F675 através de um exemplo comentado.

Atividades de aprendizagem

1. Qual é a linguagem que o microprocessador entende?
2. O que é Assembly?
3. O que faz o comando NOP?
4. Como se implementa um *loop* infinito em Assembly?
5. Como se efetua a chamada de uma sub-rotina em Assembly?
6. Como se efetua o retorno de uma sub-rotina em Assembly?

Aula 8 – Programação Assembly II

Objetivos

Expandir os conceitos sobre a programação Assembly do microcontrolador PIC12F675 através de exemplos.

8.1 Programando em Assembly com interrupção

Assim que ocorrer um evento de interrupção e o *bit* GIE, bem como a habilitação individual da interrupção estiverem ativados, haverá o desvio do programa para o vetor de interrupção 0×0004.

A fim de tratar adequadamente a interrupção, deve-se observar o seguinte:

a) **Salvamento do contexto atual** – procedimento necessário para se preservar o conteúdo dos registradores que estejam sendo utilizados no programa principal e que possam ser alterados pela sub-rotina de tratamento da interrupção. Normamente, devem-se salvar os registradores W e STATUS.

b) **Verificação do tipo de interrupção que ocorreu** – utiliza-se uma sequência de teste BTFSC para verificar os *flags* das possíveis interrupções, desviando-se para o tratamento individual de cada uma.

c) **Tratamento da interrupção** – apaga-se o *flag* da interrupção em questão e procede-se a execução das ações necessárias ao atendimento da interrupção.

d) **Recuperação do contexto e retorno** – restaura-se o conteúdo dos registradores STATUS e W, e executa-se o RETFIE.

8.1.1 Pisca LED com interrupção

Retomamos o programa anterior, em que o LED verde pisca a cada 500 ms, e acrescentamos um tratamento de interrupção externa para acender o LED vermelho sempre que a chave 1 passar do nível baixo para o nível alto.

```
;INCLUSAO DE ARQUIVOS
#INCLUDE <p12f675.inc> ; definições referentes ao PIC12F675
```

```
;CONFIGURAÇÃO DO MODO DE FUNCIONAMENTO
__CONFIG _INTRC_OSC_NOCLKOUT & _WDT_OFF & _PWRTE_OFF & _MCLR_
OFF & _CPD_OFF & _CP_OFF &

DEFINIÇÃO DOS NOMES E ENDEREÇOS DAS VARIÁVEIS UTILIZADAS NO
PROGRAMA
CBLOCK 0x0C ; endereço inicial da memória de usuário
TEMP1 ; contador para o delay de 1 ms;
TEMP500 ; contador para o delay de 500 ms
W_TEMP ; para salvar W
STATUS_TEMP ; para salvar STATUS
ENDC ; fim do bloco de memória de variáveis
Endereço de início do programa
ORG 0x0000
GOTO PRINCIPAL
;Vetor de interrupção
ORG 0x0004
MOVWF W_TEMP ; salva o conteúdo de W em W_TEMP
SWAPF STATUS, W ; salva o conteúdo de STATUS
MOVWF STATUS_TEMP ; em STATUS_TEMP
BTFSC INTCON,T0IF ; testa se ocorreu interrupção de timer 0
GOTO TRATA_TMR0 ; se ocorreu, trata a interrupção
BTFSC INTCON,T0IF ; testa se ocorreu interrupção de timer 0
GOTO TRATA_TMR0 ; se ocorreu, trata a interrupção
BTFSC PIR1,TMR1IF ; testa se ocorreu interrupção de timer 1
GOTO TRATA_TMR1 ; se ocorreu, trata a interrupção
BTFSC INTCON,INTF ; testa se ocorreu interrupção externa
GOTO TRATA_INT ; se ocorreu, trata a interrupção
FIM_INT:
SWAPF STATUS_TEMP, W ; recupera o valor do registrador STATUS
MOVWF STATUS ; restaura o conteúdo do registrador STATUS
SWAPF W_TEMP, W ; restaura o conteúdo do registrador W
RETFIE ; retorna da interrupção

TRATA_TMR0:
BCF INTCON,T0IF
; executa comandos para tratamento
; da interrupção de timer 0
GOTO FIM_INT
```

```
TRATA_TMR1:
BCF PIR1,T1IF
; executa comandos para tratamento
; da interrupção de timer 1
GOTO FIM_INT

TRATA_INT: ; tratamento da interrupção externa
BCF INTCON,INTIF
BSF LED_VM ; acende o LED vermelho
GOTO FIM_INT

;DEFINIÇÃO DAS ENTRADAS DIGITAIS
#DEFINE Chave_1 GPIO, 2; Pino 5
#DEFINE Chave_2 GPIO, 3; Pino 4

;DEFINIÇÃO DAS SAÍDAS DIGITAIS
#DEFINE LED_VD GPIO, 5 ; Pino 2
#DEFINE LED_AM GPIO, 4 ; Pino 3
#DEFINE LED_VM GPIO, 1 ; Pino 6

;DEFINIÇÃO DO SELETOR DO BANCO DE MEMÓRIA
#DEFINE Banco STATUS, RP0 ; bit RP0 do registrador STATUS (SFR)

;CONFIGURAÇÃO DOS SFRs
BSF Banco ; seleção do Banco 1 da Memória
MOVLW B'00000001' ; modo analógico/digital dos pinos de I/O
MOVWF ANSEL
MOVLW B'00001101' ; direção dos pinos de I/O digitais
MOVWF TRISIO
BCF Banco ; seleção do Banco 0 da Memória
MOVLW B'00000111' ; desativação do comparador de tensão
MOVWF CMCON
MOVLW B'10010000' ; ativa interrupções GIE e interrupção externa INTE
MOVWF INTCON

;INICIALIZAÇÃO DO PROGRAMA
BCF LED_VD ; apaga o LED verde
BCF LED_AM ; apaga o LED amarelo
BCF LED_VM ; apaga o LED vermelho
```

```
;ROTINA PRINCIPAL, implementa um loop infinito
PRINCIPAL:
BSF LED_VD ; acende o LED verde
CALL DELAY_500MS ; pausa de 500 ms
BCF LED_VD ; apaga o LED verde
CALL DELAY_500MS ; pausa de 500 ms
GOTO Principal ; salto para o início da rotina principal

; Rotina de delay de 500ms. Repete 200 x a rotina de 2,5 ms
DELAY_500MS:
MOVLW .200
MOVWF TEMP500
DL_50
CALL DELAY_2MS ; pausa de 10 ms
DECFSZ TEMP500,F ; decrementa TEMP500. Zerou?
GOTO DL_50 ; não, repete o ciclo.
RETURN ; sim, finaliza a rotina.

; Rotina de delay de 2,5 ms. Repete 250 x a rotina de 10 µs
DELAY_2MS:
MOVLW .250
MOVWF TEMP1
DL_10 ; cada ciclo gasta 10 microssegundos
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
NOP ; gasta um ciclo de tempo
DECFSZ TEMP1,F ; decrementa TEMP1. Zerou?
GOTO DL_10 ; não, repete o ciclo.
RETURN ; sim, finaliza a rotina.
;FIM DO PROGRAMA
END
```

8.1.2 Detalhando o programa "pisca LED com interrupção"

Agora, iremos detalhar, passo a passo, os blocos do programa que foram alterados/acrescentados no programa anterior:

8.1.2.1 Variáveis

```
;DEFINIÇÃO DOS NOMES E ENDEREÇOS DAS VARIÁVEIS UTILIZADAS NO
PROGRAMA
CBLOCK 0x0C ; endereço inicial da memória de usuário
TEMP1 ; contador para o delay de 1ms;
TEMP500 ; contador para o delay de 500ms
W_TEMP ; para salvar W
STATUS_TEMP ; para salvar STATUS
ENDC ; fim do bloco de memória de variáveis
```

Acrescentamos as variáveis W_TEMP e STATUS_TEMP, necessárias ao salvamento de contexto ao se tratar interrupções.

8.1.2.2 Início do programa

```
;Endereço de início do programa
ORG 0x0000
GOTO PRINCIPAL
```

8.1.2.3 Vetor de interrupção

```
;Vetor de interrupção
ORG 0x0004
MOVWF W_TEMP ; salva o conteúdo de W em W_TEMP
SWAPF STATUS, W ; salva o conteúdo de STATUS
MOVWF STATUS_TEMP ; em STATUS_TEMP
BTFSC INTCON,T0IF ; testa se ocorreu interrupção de timer 0
GOTO TRATA_TMR0 ; se ocorreu, trata a interrupção
BTFSC INTCON,T0IF ; testa se ocorreu interrupção de timer 0
GOTO TRATA_TMR0 ; se ocorreu, trata a interrupção
BTFSC PIR1,TMR1IF ; testa se ocorreu interrupção de timer 1
GOTO TRATA_TMR1 ; se ocorreu, trata a interrupção
BTFSC INTCON,INTF ; testa se ocorreu interrupção externa
GOTO TRATA_INT ; se ocorreu, trata a interrupção
FIM_INT:
SWAPF STATUS_TEMP, W ; restaura o conteúdo do
MOVWF STATUS ; registrador STATUS
SWAPF W_TEMP, W ; restaura o conteúdo do registrador W
RETFIE ; retorna da interrupção

TRATA_TMR0:
BCF INTCON,T0IF ; executa comandos para tratamento da interrupção de timer 0
GOTO FIM_INT
```

```
TRATA_TMR1:
BCF PIR1,T1IF ; executa comandos para tratamento da interrupção de
timer 1
GOTO FIM_INT
TRATA_INT: ; tratamento da interrupção externa
BCF INTCON,INTIF
BSF LED_VM ; acende o LED vermelho
GOTO FIM_INT
```

Definimos o endereço do vetor de interrupção, salvamos o contexto (W e STATUS) e verificamos os *flags* de interrupção para descobrir qual delas ocorreu (BTFSC). Uma vez detectada a interrupção, desviamos para a rotina específica (GOTO).

Neste caso, tratamos a interrupção externa (INTCON, INTIF) desviando para "TRATA_INT:". Inicialmente, limpamos o *flag* correspondente, setamos o LED vermelho e chamamos a rotina de retorno de interrupção (FIM_INT). Essa rotina efetua a restauração de contexto (valores de W e de STATUS) e retorna da interrupção (RETFIE).

O restante do programa fica inalterado e o LED verde continua piscando a cada 500 ms.

Resumo

Nesta aula, você expandiu os seus conhecimentos sobre programação Assembly através da utilização da interrupção externa do PIC12F675.

Atividades de aprendizagem

1. Qual é a importância do uso das interrupções em um microcontrolador?

2. Qual registrador é utilizado como chave de interrupção externa no PIC12F675?

3. Por que é necessário salvar o contexto do programa antes de atender à uma interrupção?

4. O que se deve fazer ao término do atendimento de uma interrupção?

Aula 9 – Programação C

Objetivos

Conhecer a linguagem C aplicada ao microcontrolador.

Estudar as aplicações da linguagem C para o microcontrolador PIC12F675.

9.1 Linguagem de alto nível

"C" é uma linguagem de programação de alto nível largamente utilizada e com uma extensa e diversificada literatura sobre a mesma.

Uma das principais características da linguagem C é a sua operação através de funções que podem ser chamadas a partir de uma função principal, denominada **main()**. Ela deve existir em qualquer programa escrito nessa linguagem (uso obrigatório).

Em termos de funcionamento, os programas mostrados a seguir, são idênticos aos respectivos programas escritos em Assembly mostrados anteriormente. As observações feitas servirão para realçar as diferenças entre as variadas formas de escrever os *firmwares* de controle.

O uso de linguagens de alto nível na programação de microcontroladores se deve à grande complexidade dos programas escritos em Assembly; o volume de código é muito maior em Assembly do que em linguagens de alto nível, o que torna muito difícil a depuração do mesmo quando escrito em Assembly.

De qualquer forma, é muito importante o conhecimento do *hardware* interno do PIC, bem como dos seus registradores e funções especiais, para que se tire o máximo proveito dos mesmos com a programação em linguagem C.

9.2 Principais estruturas da linguagem C

A linguagem C, como a maioria das linguagens de programação, possui algumas estruturas necessárias para se escrever programas. Estudaremos as principais.

9.2.1 Variáveis, atribuições e comparações

Em C, há os seguintes tipos básicos de variáveis:

- **Char** – guarda um caractere.
- **Int** – guarda um número inteiro.
- **Float** – guarda um número real com precisão simples.
- **Double** – guarda um número real com precisão dupla.
- **Void** – tipo vazio.

A declaração de variáveis e atribuição de valores para as mesmas é feita como em:

```
int evento;
char corrida;
float tempo;
evento = 5;
corrida = 'C';
tempo = 27.25;
```

Para efetuar comparações, utilizam-se os seguintes operadores:

- > maior que.
- >= maior ou igual.
- < menor que.
- <= menor ou igual.
- == igual.
- != diferente.

9.2.2 Estruturas de controle

Segundo Pereira (2003), a principal estrutura de controle é o comando "**if**", o qual testa se uma condição é verdadeira e, então, executa um bloco de comandos. Caso a condição do "**if**" não seja verdadeira, é possível executar um outro bloco de comandos com o comando "**else**", como no exemplo a seguir:

```
void main(void){
int a;
a = 5;
if (a > 0){
        printf("'a' é positivo");
    } else if (a == 0){
        printf("'a' é nulo");
    } else {
        printf("'a' é negativo");
}}
```

9.2.3 Estruturas de repetição

Em C, há as seguintes estruturas de repetição:

9.2.3.1 For

For (variável = valor_inicial; condicao_envolvendo_variavel; incremento/decremento de var)

```
{
comandos...
}
```

Exemplo

```
for(int i=0; i <= 10; i++) {
    printf("i=",i);
}
```

9.2.3.2 While (condição)

While (condição)

```
{
 comandos...
 /*Lembre-se que os comandos devem, de alguma forma, alterar condicao*/
}
```

Exemplo

```
int i=1;
while(i <= 10){
    printf("i=",i);
    i++;
}
```

9.2.3.3 Do while

```
do
        {
         comandos...
         /*Lembre-se que os comandos devem, de alguma forma, alterar condição*/
} while (condição);
```

Exemplo

```
int i=1;
do{
    printf("i=",i);
    i++;
}while(i <= 10);
```

9.3 Programas em C

Conforme fizemos quando estudamos Assembly, desenvolveremos alguns programas, agora em linguagem C, com o intuito de ilustrar e de melhor entender as estruturas da linguagem.

9.3.1 Pisca LED

O programa a seguir faz com que o LED verde (ligado no pino 2) pisque em intervalos de 500 ms.

A função deste programa pode ser implementada com o seguinte algoritmo:

a) Configurar os parâmetros de inicialização.

b) Colocar o pino 2 em nível baixo.

c) Iniciar um *loop* infinito.

d) Inverter o estado do pino 2.

e) Pausar a execução do programa por um tempo.

```
#include <12f675.h>
#include <regs_12f6xx.h>
#use delay(clock=4000000)
#fuses INTRC_IO, WDT, NOPUT, MCLR, NOBROWNOUT
```

```
// definição das entradas e saídas
#bit Chave_1 = GPIO.2 // Pino 5
#bit Chave_2 = GPIO.3 // Pino 4
#bit LED_VD = GPIO.5 // Pino 2
#bit LED_AM = GPIO.4 // Pino 3
#bit LED_VM = GPIO.1 // Pino 6
// variáveis para definição da direção dos pinos
#bit Tris_Chave_1 = TRISIO.2
#bit Tris_Chave_2 = TRISIO.3
#bit Tris_LED_VD = TRISIO.5
#bit Tris_LED_AM = TRISIO.4
#bit Tris_LED_VM = TRISIO.1
void main(){
setup_ADC_ports(NO_ANALOGS);
Tris_LED_VD = 0; // saída
Tris_LED_AM = 0; //saída
Tris_LED_VM = 0; // saída
Tris_Chave_1 = 1; // entrada
Tris_Chave_2 = 1; // entrada
LED_VD = 0; // apagado
LED_AM = 0; // apagado
LED_VM = 0; // apagado
while (true){
LED_VD = !LED_VD;
delay_ms(500);
```

9.3.2 Detalhando o programa "pisca LED em C"

9.3.2.1 Inclusão e configuração

```
#include <12f675.h>
#include <regs_12f6xx.h>
#use delay(clock=4000000)
#fuses INTRC_IO, WDT, NOPUT, MCLR, NOBROWNOUT
```

Neste bloco, temos a inclusão dos arquivos contendo as definições dos registradores do PIC12F675. Temos, também, a definição da frequência de *clock* (4 MHz) e a definição dos *fuses* internos do PIC.

9.3.2.2 Entradas e saídas

```
// definição das entradas e saídas
#bit Chave_1 = GPIO.2 // Pino 5
```

```
#bit Chave_2 = GPIO.3 // Pino 4
#bit LED_VD = GPIO.5 // Pino 2
#bit LED_AM = GPIO.4 // Pino 3
#bit LED_VM = GPIO.1 // Pino 6
```

Neste bloco, definimos os nomes dos pinos de entrada e de saída.

9.3.2.3 Direção das entradas e saídas

```
// variáveis para definição da direção dos pinos
#bit Tris_Chave_1 = TRISIO.2
#bit Tris_Chave_2 = TRISIO.3
#bit Tris_LED_VD = TRISIO.5
#bit Tris_LED_AM = TRISIO.4
#bit Tris_LED_VM = TRISIO.1
```

Neste bloco, definimos os nomes das variáveis que serão usados na definição da direção dos pinos de entrada e de saída.

9.3.2.4 *Loop* principal

```
void main(){
setup_ADC_ports(NO_ANALOGS);
Tris_LED_VD = 0; // saída
Tris_LED_AM = 0; //saída
Tris_LED_VM = 0; // saída
Tris_Chave_1 = 1; // entrada
Tris_Chave_2 = 1; // entrada
LED_VD = 0; // apagado
LED_AM = 0; // apagado
LED_VM = 0; // apagado
while (true){
LED_VD = !LED_VD;
delay_ms(500);
```

Aqui, temos a função **main()**, que é onde o programa roda efetivamente. Inicialmente, definimos todas as portas como entradas e saídas digitais. A seguir, definimos a direção dos pinos (entradas e saídas). Inicializamos todos os LEDs como "apagados" e entramos no *loop* infinito, o qual inverte o valor do LED e aguarda 500 ms.

Note que o código ficou bem mais simples e mais legível em C do que em Assembly.

9.3.3 Pisca LED com interrupção

Retomamos o programa anterior, em que o LED verde pisca a cada 500 ms, e acrescentamos um tratamento de interrupção externa para acender o LED vermelho sempre que a chave 1 passar do nível baixo para o nível alto.

```
#include <12f675.h>
#include <regs_12f6xx.h>
#use delay(clock=4000000)
#fuses INTRC_IO, WDT, NOPUT, MCLR, NOBROWNOUT
// definição das entradas e saídas
#bit Chave_1 = GPIO.2 // Pino 5
#bit Chave_2 = GPIO.3 // Pino 4
#bit LED_VD = GPIO.5 // Pino 2
#bit LED_AM = GPIO.4 // Pino 3
#bit LED_VM = GPIO.1 // Pino 6
// variáveis para definição da direção dos pinos
#bit Tris_Chave_1 = TRISIO.2
#bit Tris_Chave_2 = TRISIO.3
#bit Tris_LED_VD = TRISIO.5
#bit Tris_LED_AM = TRISIO.4
#bit Tris_LED_VM = TRISIO.1
#int_ext
void trata_int_ext() // trata a interrupção externa
{
LED_VM = 1
void main(){
setup_ADC_ports(NO_ANALOGS);
enable_interrupts(global | int_ext);
Tris_LED_VD = 0; // saída
Tris_LED_AM = 0; //saída
Tris_LED_VM = 0; // saída
Tris_Chave_1 = 1; // entrada
Tris_Chave_2 = 1; // entrada
LED_VD = 0; // apagado
LED_AM = 0; // apagado
LED_VM = 0; // apagado
while (true){
LED_VD = !LED_VD;
delay_ms(500);
```

9.3.4 Detalhando o programa "pisca LED com interrupção"

```
#int_ext
void trata_int_ext() // trata a interrupção externa
LED_VM = 1;
```

Acrescentamos a rotina que trata a interrupção externa, cuja função é ligar o LED vermelho.

```
void main(){
setup_ADC_ports(NO_ANALOGS);
enable_interrupts(global | int_ext);
Tris_LED_VD = 0; // saída
Tris_LED_AM = 0; //saída
Tris_LED_VM = 0; // saída
Tris_Chave_1 = 1; // entrada
Tris_Chave_2 = 1; // entrada
LED_VD = 0; // apagado
LED_AM = 0; // apagado
LED_VM = 0; // apagado
while (true){
LED_VD = !LED_VD;
delay_ms(500);
```

Habilitamos a interrupção global e a interrupção externa (**enable_interrupts(global | int_ext**).

Note como é muito mais simples lidar com interrupções em C quando se compara com o Assembly.

Resumo

Nesta aula, você aprendeu sobre a linguagem C aplicada aos microcontroladores e conheceu também as aplicações específicas da linguagem C para o PIC12F675.

Atividades de aprendizagem

1. Como associamos nomes aos pinos do PIC em C?

2. Em C, como definimos se um pino do PIC é entrada ou se é saída?

3. Como implementamos um *loop* infinito em C?

4. Em C, como se ativa a interrupção externa no PIC12F675?

5. Em C, como é feito o tratamento da interrupção externa no PIC12F675?

Aula 10 – Programação C II

Objetivos

Conhecer o conversor analógico digital (A/D) do microcontrolador PIC12F675.

Estudar a utilização prática do conversor A/D do microcontrolador PIC12F675.

10.1 Conversor A/D do microcontrolador PIC12F675

O PIC12F675 possui um conversor A/D (analógico/digital) interno que, quando utilizado no modo oito *bits* de resolução, gera um resultado entre 0 e 255, correspondente a uma tensão entre 0 e 5 V aplicada a um de seus pinos, os quais podem operar no modo analógico.

É possível configurá-lo também para operar em dez *bits* de resolução; isso aumenta a precisão da conversão A/D, mas torna a conversão mais lenta do que em oito *bits*.

Para uma conversão A/D de dez *bits*, o resultado estará ente 0 e 1023, correspondendo à tensão de 0 a 5 V, linearmente.

Para utilizar esse conversor, inicialmente é necessário ligá-lo e configurar o pino de entrada como entrada analógica. A partir daí, basta efetuar a seleção do canal de entrada (0 a 3), aguardar um tempo da ordem de 15 μs para que a conversão se efetive e, finalmente, efetuar a leitura do "**adc**".

A-Z

adc
Designação de conversor analógico digital, obtida das iniciais da expressão em inglês "analog-to-digital converter".

10.2 Programa exemplo A/D

O programa a seguir implementa um conversor A/D no pino 7 e, através de um potenciômetro, possibilita variar a tensão entre 0 e 5 V (veja o circuito da Aula 7).

Os LEDs serão acionados em função do nível de tensão na entrada. Abaixo de 2 V, o LED verde é acendido. Entre 2 e 4 V, o LED amarelo é acendido. Acima de 4 V, o LED vermelho é acendido.

Utilizando o conversor de 10 *bits*, a tensão de 0 a 5 V será mapeada em números de 0 a 1.023. A conversão será efetuada em degraus de 5 V/1023 = 4,888 mV.

Efetuando interpolação linear, os valores, após a conversão para as tensões de referência serão: 2 V → 409 e 4 V → 818. É necessário realizar o arredondamento para valor inteiro.

```
#include <12f675.h>
#include <regs_12f6xx.h>
#device ADC=10
#use delay(clock=4000000)
#fuses INTRC_IO, WDT, NOPUT, MCLR, NOBROWNOUT
// definição das entradas e saídas
#bit Chave_1 = GPIO.2 // Pino 5
#bit Chave_2 = GPIO.3 // Pino 4
#bit LED_VD = GPIO.5 // Pino 2
#bit LED_AM = GPIO.4 // Pino 3
#bit LED_VM = GPIO.1 // Pino 6
// variáveis para definição da direção dos pinos
#bit Tris_Chave_1 = TRISIO.2
#bit Tris_Chave_2 = TRISIO.3
#bit Tris_LED_VD = TRISIO.5
#bit Tris_LED_AM = TRISIO.4
#bit Tris_LED_VM = TRISIO.1
void main(){
setup_ADC_ports(AN0_ANALOG); // AN0 como entrada analógica, as demais como digital
setup_ADC(ADC_CLOCK_INTERNAL); // utiliza o clock interno para o conversor A/D
Tris_LED_VD = 0; // saída
Tris_LED_AM = 0; // saída
Tris_LED_VM = 0; // saída
Tris_Chave_1 = 1; // entrada
Tris_Chave_2 = 1; // entrada
LED_VD = 0; // apagado
LED_AM = 0; // apagado
LED_VM = 0; // apagado
```

```
int16 V; // declara variável
while (true){
set_adc_channel(0); // AN0 -> pino 7
delay_us(15); // tempo para a conversão A/D
V = read_adc(); // 0 a 1023
// comparação para determinar qual LED deve ser ligado
if (V < 409){ // < 2V
    LED_VD = 1;
    LED_AM = 0;
    LED_VM = 0;
} else if (V < 818){ < 4V
    LED_VD = 0;
    LED_AM = 1;
    LED_VM = 0;
} else {
    LED_VD = 0;
    LED_AM = 0;
    LED_VM = 1;
}
}
}
```

Os pontos relevantes desse código são:

- **#deviceADC = 10** – define que a conversão será de 10 *bits*.
- **setup_ADC_ports(AN 0_AN ALOG)** – define o pino 7 como entrada analógica.
- **setup_ADC(ADC_CLOCK_INTERNAL)** – utiliza o *clock* interno para acionar o conversor A/D.
- **set_adc_channel (0)** – inicia a conversão A/D no AN0.
- **delay_us (15)** – aguarda um tempo para garantir o término da conversão.
- **V = read_adc()** – transfere o valor do conversor para a variável V.
- **if (V < 409)** – compara valores para determinar o LED que será ativado.

Resumo

Nesta aula você aprendeu a utilizar o conversor A/D do PIC12F675.

Atividades de aprendizagem

1. Qual a diferença prática entre conversão A/D de 8 *bits* e de 10 *bits*?

2. No PIC12F675, qual o comando utilizado para definir se um pino é entrada analógica?

3. No PIC12F675, qual o comando utilizado para selecionar o canal de entrada para o conversor A/D?

4. Qual é o comando utilizado para efetuar a leitura do valor armazenado no conversor?

Referências

LAVÍNIA, D. J. de S.; NICOLÁS, C. **Conectando o PIC** – Recursos avançados. Érica: São Paulo, 2002.

MICROCHIP TECHNOLOGY INC. Disponível em: <http://www.microchip.com>. Acesso em: 20 out. 2011.

PEREIRA, F. **Microcontroladores PIC** – Técnicas avançadas. Érica: São Paulo, 2002.

PEREIRA, F. **Microcontroladores PIC** – Programação em C. Érica: São Paulo, 2003.

ZANCO, Wagner da Silva. **Microcontroladores PIC** – Técnicas de software e hardware para projetos de circuitos eletrônicos. Érica: São Paulo, 2006.

WIKIPEDIA. Disponível em: <http://pt.wikipedia.org/wiki/Intel_8051>. Acesso em: 20 out. 2011.

Currículo do professor-autor

Édilus de Carvalho Castro Penido é natural de Belo Horizonte - MG, iniciou a sua formação técnica com o curso de Eletrônica no Colégio Técnico do Centro Pedagógico (COLTEC) da Universidade Federal de Minas Gerais (UFMG), em 1989. Graduou-se em Engenharia Elétrica pela Pontifícia Universidade Católica de Minas Gerais (PUC-MG), em 1996 e concluiu, em 2008, o Curso de Especialização em Projeto de Circuitos Integrados pela Universidade Federal de Minas Gerais (UFMG). Atualmente está concluindo o Curso de Mestrado em Sustentabilidade Socioeconômica e Ambiental pela Universidade Federal de Ouro Preto (UFOP).

Desde 1997 desenvolve suas atividades profissionais como docente em várias instituições de formação técnica e superior, com atuação constante na capacitação e na qualificação de profissionais nas áreas de Eletrotécnica, Eletrônica e Automação. A partir de 2008, passou a atuar como professor do Curso Técnico em Automação Industrial do Campus Ouro Preto do Instituto Federal de Minas Gerais (IFMG).

Ronaldo Silva Trindade é engenheiro civil formado pela Universidade Federal de Ouro Preto (UFOP), com mestrado em estruturas metálicas pela REDEMAT/UFOP. É professor do IFMG, Campus Ouro Preto (antiga Escola Técnica Federal de Ouro Preto) desde 1987. É autodidata em eletrônica e em computação, possuindo grande experiência em laboratório de eletroeletrônica e no desenvolvimento de sistemas microcontrolados.