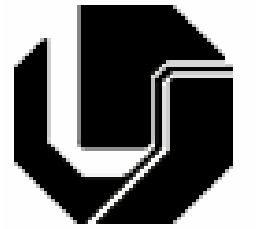
UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA ELÉTRICA

PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA



RELÉ DIGITAL PARA PROTEÇÃO CONTRA SOBRECORRENTE E MONITORAMENTO DA OPERAÇAO DE MOTORES DE INDUÇÃO

LEONARDO COSTA DE PAULA

UBERLÂNDIA, JULHO DE 2005

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA ELÉTRICA

PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

RELÉ DIGITAL PARA PROTEÇÃO CONTRA SOBRECORRENTE E MONITORAMENTO DA OPERAÇÃO DE MOTORES DE INDUÇÃO

Dissertação apresentada por **Leonardo Costa de Paula** à Universidade Federal de Uberlândia para a **obtenção do título de Mestre em Engenharia Elétrica** aprovada sem restrições em 15/07/2005 pela Banca Examinadora:

Prof. **Darizon Alves de Andrade**, Phd (UFU) – Orientador

Prof. Edilberto Pereira Teixeira, Dr. (UNIUBE)

Prof. Carlos Augusto Bissochi Jr., Dr. (UFU)

RELÉ DIGITAL PARA PROTEÇÃO CONTRA SOBRECORRENTE E MONITORAMENTO DA OPERAÇÃO DE MOTORES DE INDUÇÃO

LEONARDO COSTA DE PAULA

Dissertação apresentada por Leonardo Costa de Paula à Universidade Federal de Uberlândia como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica.

_____

Prof. Darizon Alves de Andrade

Phd. Eng. Elétrica – Orientador

DEDICATÓRIA

Dedico este trabalho aos meus pais e familiares pelos incentivos e por terem estado sempre ao meu lado, dedico ainda aos meus amigos pela ajuda e motivação. Dedico também ao meu orientador, Prof. Darizon, que tornou possível a realização deste sonho.

AGRADECIMENTOS

Agradeço ao Prof. Darizon pela confiança, amizade, orientação e grande disposição diante das inúmeras dificuldades que surgiram durante esta nossa caminhada.

Agradeço aos meus pais por estarem sempre dispostos em me apoiar acreditando mais do que eu que este sonho seria possível.

Aos meus irmãos e sobrinho por sempre me motivarem a concluir este trabalho e por acreditarem que este objetivo pessoal seria alcançado.

Aos meus amigos e sócios Charles e Wagner por me motivarem e ainda por me cobrirem durante as minhas ausências, facilitando assim a conclusão deste mestrado.

Aos meus amigos Felipe, Luis Marcelo e Renato pela força e pela pronta disposição em me ajudar em tudo que foi preciso.

Aos meus colegas e amigos do laboratório pelas ajudas e dicas que sempre facilitaram o desenrolar deste trabalho, por isto serei eternamente grato.

Ao SENAI CFP-FAM e à UFU pela parceria que facilitou a elaboração do protótipo aqui desenvolvido e principalmente ao Emerson que teve grande participação na confecção do protótipo final.

Aos meus grandes amigos que colaboraram para a conclusão deste trabalho direta ou indiretamente simplesmente por serem meus verdadeiros amigos.

RESUMO

Este trabalho apresenta uma solução para o monitoramento em tempo real das correntes de um motor de indução juntamente com proteção contra sobre correntes, com curva corrente x tempo programável. A idéia principal é mostrar um equipamento baseado em tecnologia digital para proteção contra sobrecorrente para motores de indução capaz de substituir com vantagens os relés bimetálicos.

A curva típica de operação de qualquer relé bimetálico pode ser facilmente programada neste dispositivo inteligente inclusive em tempo real e com o motor em funcionamento.

O equipamento incorpora uma interface para comunicação em rede que possibilita o acionamento remoto do motor via PC e ainda a monitoração das condições de carga. Com a reprogramação deste relé inteligente, facilmente um mesmo dispositivo pode ser utilizado para controlar motores de diferentes potências ou carga nominal. Assim pode-se substituir o motor sem haver a necessidade de se substituir o sistema de proteção.

O trabalho descreve ainda os detalhes do projeto do equipamento e apresenta resultados experimentais obtidos com um protótipo desenvolvido em laboratório.

ABSTRACT

This work describes a micro controlled-based device for over current protection and online monitoring of induction motor operation, able to replace with some advantages the traditional over current thermal relays.

It allows real time monitoring of induction machine currents and over current protection, with programmable current x time characteristic. The typical current x time thermal relay curve can be easily programmed in this device and both the curve and setting points of operation can be altered online with the motor in operation.

The device has a network communication interface for remote control and operation monitoring in a remote PC. Due to the programmable characteristic, the hardware can be used for different ranges of motor sizes and rated powers.

The work shows details of design and programming, and presents experimental results obtained with a prototype developed in laboratory.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

CI: Circuito Integrado (Plural: CIs)

CPL: Controlador Lógico Programável (Plural: CLPs)

PC: *Personal Computer* (computador pessoal)

GND: *Ground* (conexão terra ou massa de um circuito)

SCR: *Semi-Conductor Retificator* ou Tiristores (Plural: SCRs)

TTL: *Transistor Transistor Logic* (tecnologia de transistor bipolar para CIs)

FF: *Flip-Flop* (*lacth* capaz de armazenar um bit de memória)

LCD: *Liquid Crystal Display* (display de cristal líquido)

LED: *Ligth Emission by Diod* (diodo emissor de luz)

ADC: *Analogic Digital Converter* (conversor analógico digital)

I/O: *Input and Output* (dispositivos ou pinos de entrada e saída de dados)

F.E.M.: Força eletromotriz

F.M.M.: Força magneto motriz

RISC: *Reduced Instruction Set Computers* (conjunto reduzido de instruções)

CISC: *Complex Instruction Set Computers* (conjunto complexo de instruções)

USART: *Universal Synchronous Asynchronous Receiver Transmitter* (canal serial universal)

EEPROM: *Electrical Eraser Programmable Read Only Memory* (memória não volátil programável e apagável eletricamente byte a byte)

CAN: *Controller Area Network* (tipo de camada física de rede)

Capítulo 1 - Introdução

1.1 Motivações

Desde o início da automação industrial, época em que só era possível controlar a indústria através da utilização das CLPs não microcontrolados de bancadas, procurava-se controlar todos os equipamentos industriais a partir de uma sala única, comumente chamada de "sala de controle".

O motivo principal para se desejar que o controle de dispositivos industriais seja feito a partir de uma única sala pode ser resumido em poucas palavras: basta pensar nos benefícios trazidos para uma empresa caso fosse possível que um único funcionário pudesse ser responsável pela monitoração e controle de vários equipamentos fabris ao mesmo tempo sem precisar sair de sua poltrona, ou seja, este trabalhador seria capaz de ligar ou desligar equipamentos remotamente, verificar medições instantâneas e até mesmo descobrir falhas, tudo isto de um único ponto da empresa, da "sala de controle".

A constante evolução da microeletrônica possibilitou implementações de controles que, além de serem mais "inteligentes" (agora microcontrolados), ocupassem menos espaço nas salas de controle. Estas novas técnicas de controle podem ser instaladas próximo ao equipamento que se deseja controlar e sua comunicação com a sala de controle pode ser feita através de uma rede local, facilitando assim a manutenção e instalação dos mesmos.

Com a inclusão de microcontroladores, em alguns casos microprocessadores, no mercado da microeletrônica os antigos CLPs passaram por uma profunda modificação sob o ponto de vista de *hardware*, isto é, deixaram de ser apenas controladores lógicos

programáveis e passaram a ser também microprocessados, o que possibilita uma melhor programação de suas funções específicas.

Assim, os CLPs tiveram uma enorme diminuição do seu tamanho físico e deixaram de ser equipamentos de bancadas que ocupavam toda uma sala. Outra vantagem seria a comunicação com o computador central que agora pode gerenciar vários CLPs ao mesmo tempo pela mesma rede de comunicação.

Cada vez mais se utilizam microcontroladores como dispositivos controladores, isto se deve a vários motivos, dentre os quais se podem ressaltar o fato de que o microcontrolador possibilita tanto o controle quanto a leitura de dados dos equipamentos controlados (permitindo assim uma comunicação bidirecional entre o computador central da sala de controle e todos os equipamentos) e o baixo custo dos microcontroladores atuais. Assim, utilizando-se dos microcontroladores, torna-se viável praticamente todo tipo de atuação e monitoramento.

Os engenheiros eletricistas com ênfase em eletrônica costumam querer dar soluções eletrônicas para todo tipo de problema, mesmo os mais simples. Com este intuito, foi analisada a necessidade atual da automação industrial. Depois de uma breve pesquisa sobre os dispositivos controladores existentes no mercado para este tipo de automação, verificou-se que um relé microcontrolado, que atue e proteja ao mesmo tempo os motores trifásicos, está em falta ou está presente com um preço ainda muito elevado para aplicação no mercado nacional.

Como atualmente a maioria das proteções para motores utiliza um relé bimetálico, relé este que possibilita apenas uma proteção do equipamento e não uma boa noção de como está o histórico de funcionamento deste motor. Esta idéia se aplica a

acionamentos de motores que ainda é, em sua maioria, realizado com contatores, mas é igualmente adequada ao uso de acionamentos utilizando componentes de estado sólido.

Assim a idéia de se desenvolver este trabalho surgiu, visando um produto nacional novo e barato, que seja capaz de não apenas monitorar e proteger, mas também atuar sobre um motor de indução trifásico. O produto final também deverá ser hábil de trabalhar com programações locais e remotas.

Toda esta análise e estudo ajudaram para que a idéia ganhasse as motivações e importâncias necessárias para ser implementada aqui, visto que os outros dispositivos equivalentes existentes no mercado atual, se comparado ao que está sendo proposto, são equipamentos que utilizam tecnologias distintas ou possuem custos mais elevados.

1.2 Objetivos

O objetivo deste trabalho é desenvolver um relé inteligente de baixo custo para o mercado nacional, assim como as informações de como este protótipo pôde ser implementado. O protótipo visa a atender às necessidades das empresas que desejam supervisionar motores de indução e levantar possíveis históricos de operação.

As características básicas deste protótipo são:

- Dispositivo microprocessado de 8 bits;
- Teclas para programação local;
- Display de cristal líquido para facilitar a interface com o usuário;
- No mínimo três entradas analógicas para a medição (monitoramento em tempo real) da corrente trifásica, indicando assim o estado de carga;
- Pelo menos um canal de comunicação RS-232 ou RS-485;

- No mínimo uma saída TTL para ligar / desligar um motor.

Outras características podem ser agregadas, mas deverão estar em placas separadas para simplificar e modular o projeto como um todo.

Para controlar este dispositivo remotamente será desenvolvido um software com a linguagem de programação orientada a objeto Delphi, este software será desenvolvido para os sistemas operacionais Windows 9x e XP. As características fundamentais deste software são:

- Interface amigável para atender também aos usuários mais leigos;
- Acesso remoto simplificado, a partir da porta serial do computador pessoal;
- Possibilita o monitoramento e acionamento de motores remotamente;
- Informa em tempo real ao usuário caso haja uma sobrecarga no motor ou qualquer outra anomalia ligada às correntes de carga;
- Informa ao usuário a corrente RMS que o motor está consumindo, podendo assim gerar gráficos e tabelas de históricos.

1.3 O Estado da Arte

Os relés inteligentes recebem este nome porque são capazes de ligar ou desligar um equipamento remotamente, mas, na maioria dos equipamentos com este nome, não passa de uma simples chave que pode ou não informar seu estado atual, isto é, se ele está ou não ligado.

Atualmente existem vários equipamentos no mercado que são chamados de relés inteligentes, porém poucos possuem todas as características a que se propõe este

trabalho, além do que, os que as possuem, são produtos muito mais caros do que se deseja conseguir com este protótipo.

A maioria dos produtos existentes no mercado limita-se ao controle de ligar e desligar, sem se preocupar em responder qual é a corrente atual que o equipamento ligado está consumindo ou qualquer histórico relacionado a isto. Por não haver o monitoramento da corrente, o relé utilizado para acionar a carga normalmente é um relé bimetálico, que não passa de uma proteção limitada, pois não há como se saber, de uma sala de controle, se o motor protegido por este relé está ou não com problemas, pois a única informação que se consegue tirar de lá seria se o mesmo está aberto ou não.

Já os produtos que atendem à questão do monitoramento da corrente normalmente não se aproveitam deste dado para o monitoramento de sobrecarga, deixando este encargo para um fusível rearmável ou até mesmo o tradicional relé bimetálico. Assim, estas linhas de produtos apenas informam qual o histórico da corrente de carga, mas não atuam no motor de acordo com o que estão monitorando, em resumo, trata-se de um monitoramento também limitado.

Um trabalho foi publicado recentemente, julho de 2004, apresentando um relé digital multifuncional, trata-se de um dispositivo capaz de monitorar as tensões e correntes desenvolvido para criar um dispositivo capaz de ser programado através de entradas com níveis lógicos de 0 e 1. Com relação ao protótipo proposto aqui este relé tem a vantagem da monitoração das tensões, mas não é capaz de ser programado para reproduzir uma curva sobre corrente x tempo como o deste trabalho [17].

O LOGO! da Siemens é um bom exemplo de como está o mercado atual deste tipo de controlador, este equipamento possui saídas de até 10A e entradas para informações

lógicas 0 ou 1, isto é, pode-se executar uma programação para que ele ligue caso uma determinada entrada estiver em nível lógico ligado (1) ou para ele desligar se a entrada mudar de desligada (0) para ligada (1). Várias combinações são possíveis e este equipamento já substitui muitos outros dispositivos programáveis para o controle e automação de iluminações, esteiras rolantes entre outros. Mas para o controle de motores não é como o proposto aqui, pois não há uma entrada de um conversor analógico digital para monitorar as correntes e, portanto, não é capaz de proteger um motor contra sobre correntes [18].

A própria Siemens tem ainda o SIMATIC S7-200, trata-se de um micro CLP utilizado para controlar motores e máquinas em geral na indústria, como novamente ele não possui características para o monitoramento de corrente, por isto não possibilita a proteção contra sobre corrente [18].

Saindo dos dispositivos controladores e entrando nos dispositivos de proteção, ainda da marca Siemens, dois outros relés valem a pena serem apresentados. O primeiro é apenas um relé térmico para proteção contra sobrecarga, já o segundo é um relé inteligente, bastante parecido com a proposta deste trabalho salvo pelo fato dele não possuir um display ou comando para programação local. Apesar disto o SIMOCODE-DP, como é chamado este relé inteligente, é capaz de se comunicar com um PC e monitorar as correntes das três fases do motor. Por não possuir um display LCD este equipamento possui um tamanho bastante reduzido e parecido com relés térmicos comuns [18].

Todos estes dados pesquisados determinaram a motivação da implementação deste protótipo, visto que se trata de um produto caro em um mercado de poucos fabricantes. Ainda por se tratar de um produto cujo mercado é favorável e poucas empresas estão investindo na fabricação destes tipos de dispositivos.

1.4 Contribuições

As contribuições deste trabalho estão voltadas à automação industrial, que é possibilitar um melhor controle e monitoramento da operação de motores trifásicos utilizando-se de um computador pessoal a um custo abaixo do existente hoje no mercado.

Este protótipo será desenvolvido para o mercado voltado a motores, porém pode-se utilizá-lo para o controle e monitoramento de qualquer equipamento trifásico que esteja dentro das especificações de carga do equipamento. Por este motivo pode ser empregado em inúmeras aplicações, onde o monitoramento de sobre correntes em tempo real se fizer necessário.

Outra contribuição que merece ser ressaltada aqui seria o fato de esta proposta criar um relé inteligente de tal modo que seja possível programá-lo em tempo real, isto é, enquanto o processo estiver em andamento (o relé estiver monitorando uma carga) pode-se alterar sua curva de resposta sem que haja qualquer interferência na operação da carga monitorada.

1.5 Disposição do trabalho

Este trabalho está disposto de uma maneira simples, direta e de fácil entendimento, para possibilitar que futuros trabalhos sejam implementados a partir dele. A estrutura básica desta dissertação está resumida a seguir:

- Capítulo 1 - Introdução: relata a motivação do trabalho, descreve os objetivos, fala sobre o mercado e produtos similares e por fim as contribuições da dissertação;
- Capítulo 2 - Máquinas Elétricas Trifásicas: fornece uma introdução básica sobre máquinas de indução para possibilitar o seu controle de partida. Por isto este capítulo

engloba também tópicos como tipos de partidas, operação normal e de sobrecarga, métodos mais utilizados para proteção de sobrecarga e finalmente uma proposta para solução que será implementada;

- Capítulo 3 - Microcontroladores: parte do trabalho dedicada à apresentação teórica da engenharia eletrônica dando-se mais ênfase aos microcontroladores. Neste capítulo serão apresentadas as noções mínimas necessárias para o bom entendimento do protótipo do relé inteligente que será montado no decorrer deste estudo;
- Capítulo 4 - Projeto do Equipamento (*Hardware* e *Firmware*): nesta parte do trabalho estão descritos todas as características físicas do protótipo e também como o equipamento foi programado, contendo itens importantes como os diagrama de blocos, o projeto do hardware proposto, os módulos de programação do firmware, os fluxogramas dos módulos mais importantes e ainda uma análise sobre o equipamento proposto;
- Capítulo 5 - Desenvolvimento do *Software*: parte do trabalho que apresenta as necessidades do software de controle bem como uma modelagem necessária para supri-las, este capítulo é divido em fluxograma e avaliação;
- Capítulo 6 - Resultados Práticos: aqui são discutidos e apresentados todos os resultados dos testes feitos em laboratório. Este item apresenta uma discussão sobre a avaliação final do protótipo de seus resultados, uma comparação com os produtos similares e por fim uma análise técnica de tudo;
- Capítulo 7 - Considerações Finais: neste capítulo se encontra a conclusão do trabalho bem como algumas propostas para futuros trabalhos relacionados;

- Referências Bibliográficas: parte que contém toda a bibliografia utilizada que possibilitou a criação deste trabalho;
- Anexo A : impressão dos arquivos fontes tanto de firmware quanto de software.

Capítulo 2 - Máquinas Elétricas Trifásicas

Serão abordadas neste capítulo informações como: a operação do motor de indução; o tipo de partida utilizado durante os testes práticos; as curvas de sobrecarga que o motor pode suportar bem como sua corrente nominal; os métodos de proteção existentes e recomendados pela norma brasileira NBR 5410.

Como o foco deste trabalho é projetar um relé inteligente com microcontrolador interno, que seja capaz de monitorar as correntes de fase do motor e ainda atuar de acordo uma programação específica para cada motor. A programação deste relé inteligente irá proteger o motor contra sobrecarga. O estudo sobre motores elétricos que será visto neste capítulo será bastante objetivo, para que se possa entender o funcionamento básico dos mesmos e conhecer as necessidades básicas para sua proteção.

2.1 O motor de indução trifásico

Os motores de indução trifásicos, também chamados de motores assíncronos, são sem dúvida alguma os mais utilizados na indústria atualmente. Embora o motor de indução seja, talvez, o mais simples de todos os motores sob o ponto de vista de operação e trabalho, a teoria de sua operação é bastante sofisticada [1].

Durante os testes práticos do protótipo do relé inteligente, que é o objeto deste trabalho, foi utilizado um motor de indução trifásico de baixa potência, por isto será visto aqui apenas alguns conceitos básicos deste tipo de motor.

2.1.1 Aspectos construtivos

Existem dois tipos de máquinas de indução. O primeiro tipo seria as máquinas de rotor bobinado têm um rotor semelhante ao do estator, isto é, possuem três enrolamentos

isolados distribuídos ao longo da periferia do rotor cujas fases estão deslocadas de 120º no espaço. A ligação com o exterior da máquina faz-se através de três anéis e escovas aos quais se pode ligar um circuito exterior, normalmente resistências de partida ou sistemas de regulação de velocidade.

Já as máquinas de rotor em gaiola têm um rotor constituído por um núcleo de ferro no qual se encontram condutores ligados na periferia do rotor através de dois anéis que fecham um curto entre si. Esta construção tem um elevado nível de robustez, um baixo peso bem como um reduzido momento de inércia e, no conjunto, é talvez a máquina mais barata.

De todos os motores elétricos existentes, o motor de gaiola de esquilo é o mais simples no aspecto construtivo. Outra característica que faz com que o motor de gaiola de esquilo seja largamente utilizado é o fato de possuir uma operação quase isenta de manutenção, o que o indica para aplicações em localizações remotas ou de severas condições de trabalho – ambientes agressivos etc.

Por tudo isto é provavelmente a máquina mais utilizada em acionamentos de velocidade fixa e é cada vez mais utilizada também em acionamentos de velocidade ajustável, apesar das dificuldades de controle que apresenta. A Figura 2.1 representa o corte esquemático de uma máquina de indução, onde abc são as correntes das fases entrando e.a'b'c' são as mesmas correntes saindo pelo outro lado.

A armadura do estator não é diferente da de uma máquina síncrona de corrente alternada e por isto não requer nenhuma elaboração adicional, já o núcleo do rotor de um motor de indução é um cilindro de aço laminado, no qual condutores de cobre ou de alumínio são fundidos ou são enrolados paralelamente ao eixo em ranhuras existentes no núcleo. Nota-se que os condutores não precisam ser isolados do núcleo, pois as correntes induzidas no rotor

seguem o “caminho” de menor resistência, ou seja, os condutores de cobre ou de alumínio fundido [1].

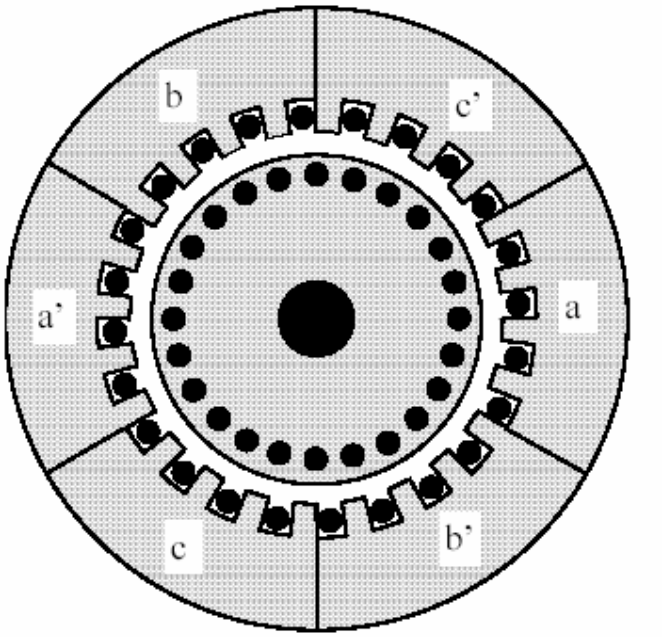


Figura 2.1 – Corte esquemático de uma máquina de indução

Em motores cujo rotor é do tipo gaiola de esquilo, os condutores do rotor estão curto-circuitados em cada terminal por anéis terminais contínuos – daí o nome de “gaiola de esquilo”. Caso os rotores sejam maiores, os anéis são soldados aos condutores em vez de serem moldados na construção. Um detalhe construtivo deste tipo de motor é que as barras nem sempre são paralelas ao eixo do rotor, mas podem ser deslocadas ou colocadas segundo um pequeno ângulo em relação a ele para produzir um conjugado mais uniforme e diminuir o “zumbido” magnético durante a operação do motor [1].

Motores de rotor bobinado têm seus condutores de cobre distribuídos nas diversas ranhuras, usualmente isolados do núcleo de ferro. Estes motores podem ser ligados tanto em delta quanto em estrela.

2.1.2 Campo magnético girante

Ainda observando a Figura 2.1, a bobina aa' representa todas as bobinas associadas à fase A para um par de pólos. De modo similar, a bobina bb' representa as bobinas da fase B e a bobina cc' representa as bobinas da fase C.

Quando uma das extremidades de cada fase é ligada entre si o enrolamento do estator trifásico é dito como conectado em estrela, por exemplo, as extremidades abc em curto e a'b'c' em cada ponto da "estrela". Em contrapartida, quando ligados independentemente, isto é, fase A ligada em a e a', fase B ligada em b e b' e fase C ligada em c e c', é dito que o motor está conectado em triângulo. O desenho esquemático dos tipos de ligações estrela e triângulo é ilustrado na Figura 2.2.

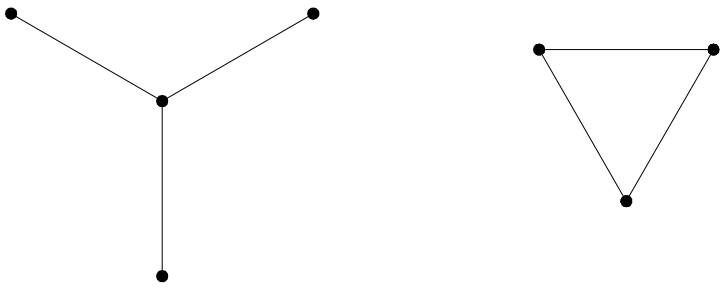


Figura 2.2 – Tipos de conexões: Estrela e Triângulo

Um campo magnético girante, de amplitude constante e girando à velocidade síncrona pode ser produzido por qualquer grupo polifásico de enrolamentos cujos eixos magnéticos estejam igualmente espaçados ao longo da periferia do estator se tiverem o mesmo número de fases e se as correntes que circulam através dos enrolamentos também estiverem uniformemente defasadas no tempo. Um exemplo bastante simples seria de um enrolamento bifásico, imagine este enrolamento disposto fisicamente no estator com um deslocamento de 90°, assim seria produzido um campo girante constante desde que as correntes das fases também estivessem deslocadas em quadratura no tempo.

Em virtude do que foi dito é que todas as máquinas de indução trifásicas necessitam de três enrolamentos individuais e idênticos com eixos magnéticos deslocados de 120° elétricos no espaço e pelos quais circulem correntes defasadas também de 120° no tempo. Somente com este pré-requisito será possível produzir um campo magnético de amplitude constante, que gira à velocidade síncrona.

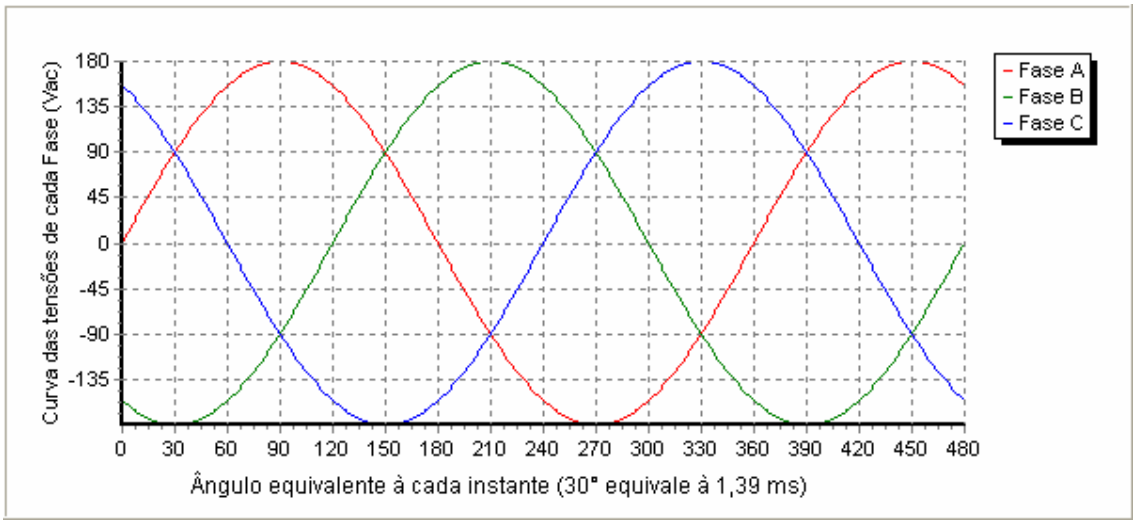


Figura 2.3 – Relação entre as tensões defasadas em 120° no tempo

A Figura 2.3 ilustra as tensões senoidais trifásicas equilibradas aplicadas à armadura de uma máquina trifásica, assim pode-se considerar a mesma curva para as correntes trifásicas que circulam pela armadura, se considerarmos a mesma impedância presente nos três enrolamentos. Para facilitar a análise deste gráfico, o deslocamento no tempo das correntes, no eixo horizontal, está em graus proporcional a cada instante, isto facilita a análise da curva senoidal para cada fase e também simplifica a verificação da defasagem entre as fases, que é de 120°.

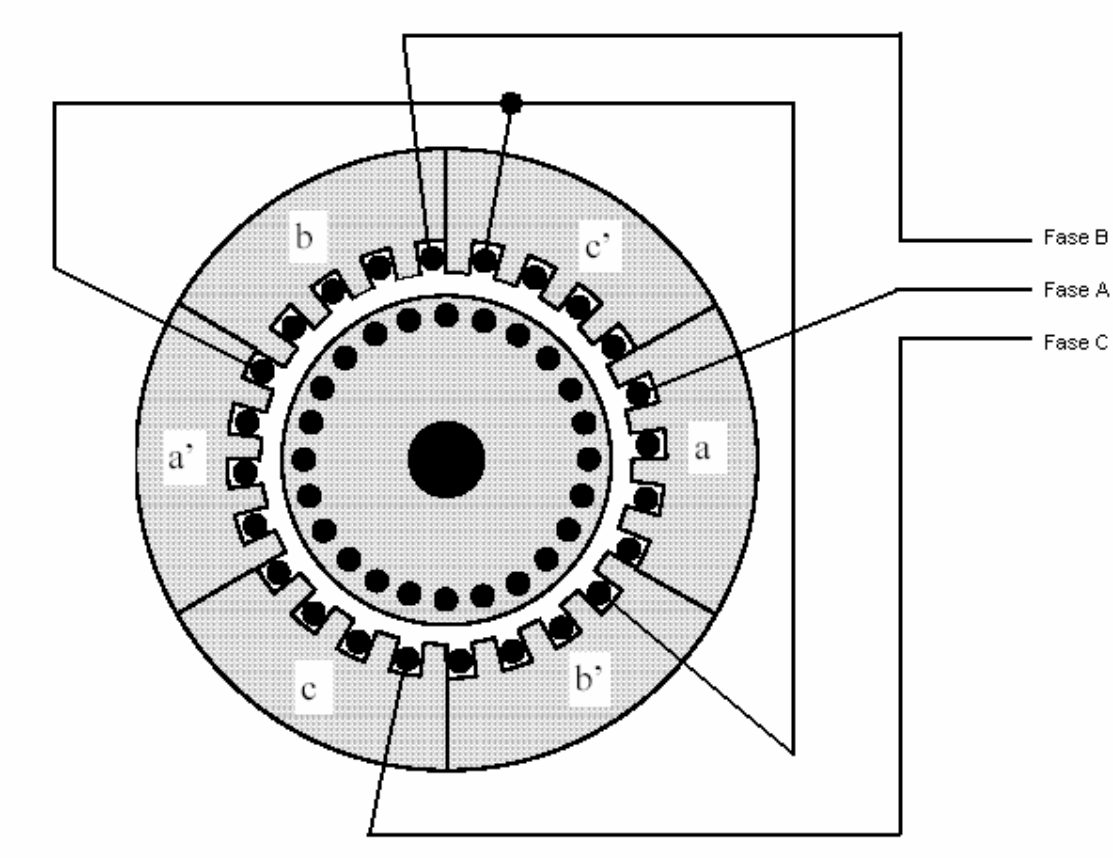


Figura 2.4 – Relações entre os enrolamentos defasados de 120°

Ligando-se as fases A, B e C nos terminais a, b e c do enrolamento de um estator de um motor de indução trifásico e ligando-se os terminais a', b' e c' em curto, conforme é mostrado na Figura 2.4, obtém-se um campo magnético resultante girante de amplitude constante e velocidade síncrona.

Para entender melhor como é formado este campo magnético resultante girante deve-se calcular o campo magnético gerado por cada fase e posteriormente calcular-se o campo resultante. Então para verificar o giro deste campo resultante deve-se calcular novamente os campos magnéticos de cada fase só que para um instante diferente do primeiro, conforme o gráfico da Figura 2.3.

Agora será explicado qualitativamente como o fluxo magnético é induzido por cada fase, para isto será mostrado o que ocorre com o fluxo de cada fase em quatro diferentes instantes do gráfico da Figura 2.3: 90°; 210°; 330°; 450°. Observe que o instante de 450° fecha um período completo em relação ao instante de 90° (450°-90°=360°), ou seja, é a segunda vez que a fase A passa pelo seu ponto de máximo. Este três instantes distintos (90°, 210° e 330°) foram escolhidos devido ao fato de serem os instantes em que cada fase encontra-se no seu valor máximo.

Uma análise simplificada de como o fluxo magnético é gerado por cada uma das fases pode ser obtida através da regra da mão direita [1], utilizada para a determinação da direção e do sentido do fluxo, e através de uma analogia entre fases, para a determinação da amplitude do fluxo.

A Figura 2.5 mostra o sentido da corrente elétrica gerado por cada uma das fases nos quatro instantes escolhidos. Pode-se observar na Figura 2.3 que no instante de 90° (4,17ms) as fases B e C encontram-se com polaridade invertida com relação à fase A, por isto

que na Figura 2.5 o sentido da corrente elétrica devido às fases B e C estão invertidas, isto é, entrando nos pontos b' e c' e saindo nos pontos b e c.

O fluxo magnético gerado pela corrente circulante devido à fase A no instante de 90° é mostrado na Figura 2.5 conforme a regra da mão direita, já a amplitude devido a esta fase é considerada a máxima amplitude de um fluxo magnético gerado por uma única fase cujo valor é de $F_a=F_{max}$, pois neste instante a fase A encontra-se em sua amplitude máxima (veja Figura 2.3).

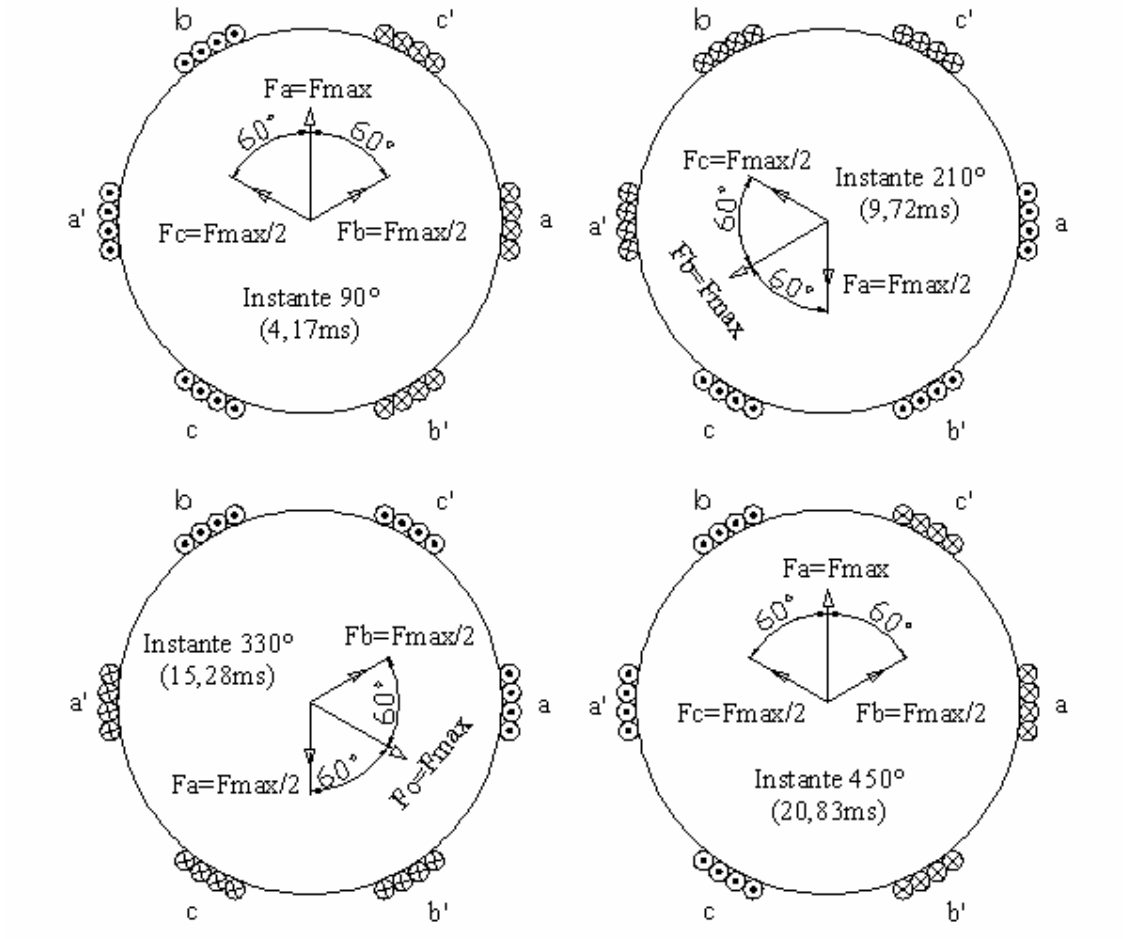


Figura 2.5 –Fluxo magnético resultante de cada fase em instante distintos

As direções e os sentidos dos fluxos magnéticos gerados pelas fases B e C também estão indicados na figura, assim como suas amplitudes que serão iguais entre si em módulo, cujo módulo vale $F_b=F_c=F_{max}/2$, pois no instante de 90° é exatamente o instante em que os valores de suas fases encontram-se, em módulo, na metade do seu valor máximo.

Como o ângulo entre os dois fluxos magnéticos gerados pelas fases B e C é de 120° e suas amplitudes são iguais, o fluxo resultante destas duas componentes será um com mesma direção e sentido do fluxo magnético gerado pela fase A, cuja amplitude será a metade da amplitude do fluxo magnético gerado pela fase A. Logo o fluxo magnético resultante das três fases no instante de 90° terá a mesma direção e sentido do fluxo gerado pela fase A e amplitude 50% maior.

Nos demais instantes as análises são as mesmas vistas anteriormente com uma única diferença: o fluxo magnético resultante das três fases muda de direção. Um detalhe importante para finalizar a análise do instante de 90° da Figura 2.5 é que independente do instante em que se fizer a análise do fluxo resultante, seu módulo sempre será o mesmo e irá valer 50% a mais do valor máximo de um fluxo gerado por uma única fase.

Outro detalhe bastante importante é que a velocidade síncrona depende apenas do número de pares de pólos para o qual o enrolamento trifásico foi projetado e da freqüência da fonte de alimentação trifásica, conforme mostra a equação abaixo:

$$N_S = \frac{120 \cdot f}{P} \qquad \text{Equação 2.1}$$

Onde:

- N_s é a velocidade síncrona em rpm;
- f é a freqüência da fonte de alimentação CA em hertz;
- P é o número de pólos;

2.1.3 Princípio de funcionamento do motor de indução

O princípio de funcionamento do motor de indução está baseado no fato da força magneto motriz girante estabelecer um campo magnético girante de mesma velocidade, isto é, velocidade síncrona. O caminho do campo magnético é estabelecido no estator e no rotor, de tal forma que, na presença do campo magnético, se o rotor estiver parado ou girando com velocidade diferente da síncrona ocorre uma velocidade relativa entre o campo girante e o rotor. Esta é a velocidade de escorregamento e é imprescindível para a operação do motor.

Com a diferença de velocidade os condutores do rotor são cortados pelas linhas de fluxo do campo magnético, o que dá origem a tensões e correntes induzidas no rotor. Essas correntes por sua vez estabelecem o campo magnético do rotor, cuja interação com o campo produzido pelas correntes estatóricas resulta em conjugado e conseqüente movimento.

A diferença entre as velocidades do campo magnético produzido pelos enrolamentos e o giro do rotor é uma característica dos motores de indução e, por isto, eles são classificados como motores assíncronos ou não síncronos, a Figura 2.6 mostra como seria o circuito equivalente dos motores trifásicos.

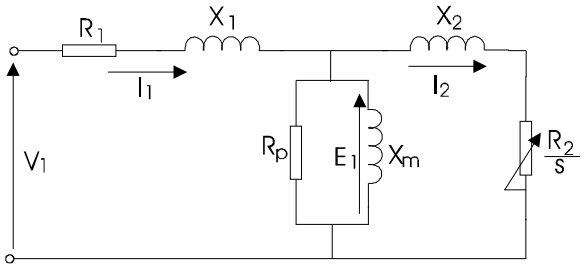


Figura 2.6 –Circuito equivalente do motor de indução

Conforme foi visto, a velocidade do motor nunca poderá ser igual à do campo magnético girante, pois se isto ocorresse a corrente induzida seria zero e não se produziriam

fluxo magnético nem conjugado. Por isto se diz que ele deve "escorregar" em velocidade a fim de que se produza conjugado. O que resulta numa diferença de velocidades produzidas entre a velocidade síncrona do campo magnético girante (N_s) e a velocidade na qual o rotor do motor gira (N_r) como resultado do conjugado produzido pela interação entre seu campo e o campo magnético girante.

Esta diferença na velocidade é chamada **velocidade de escorregamento** ou **rotação de escorregamento** e é normalmente expressa como uma fração ou porcentagem da velocidade síncrona como é mostrada na equação abaixo:

$$s = \frac{N_s - N_r}{N_s} \text{ ou } N_r = N_s \cdot (1 - s) = 120 \cdot \frac{f}{P} \cdot (1 - s) \qquad \text{Equação 2.2}$$

Onde:

- s é o escorregamento em fração da velocidade síncrona;
- N_r é a velocidade do rotor em rpm;
- N_s é a velocidade síncrona em rpm;
- f é a freqüência da fonte de alimentação CA em hertz;
- P é o número de pólos;

Quando o motor operar a vazio, o escorregamento é muito baixo e sua impedância rotórica é muito elevada. Assim a corrente no rotor é reduzida à corrente suficiente para produzir o conjugado necessário a vazio. O fator de potência é muito baixo e em atraso (tipicamente menor que 0,3), pois a corrente que circula pelo estator é utilizada apenas para a magnetização da máquina.

Quanto maior a carga do motor maior terá que ser o conjugado necessário para acioná-la, já para se obter um maior conjugado terá que ser maior a diferença de velocidades

entre o rotor e o campo girante no entreferro o que, conseqüentemente, aumentará o escorregamento.

Entretanto, quando uma carga mecânica é aplicada ao rotor, isto leva a um aumento no escorregamento e conseqüente a um aumentando na freqüência da corrente induzida do rotor, na reatância e na sua f.e.m. induzida. O aumento da corrente induzida no rotor reflete-se num aumento da corrente primária do estator, em decorrência disto uma corrente maior será requerida no estator melhorando assim o fator de potência, que resultará na produção de mais potência mecânica (trabalho).

2.1.4 Partida do motor de indução

Lembrando que este trabalho visa desenvolver um protótipo para proteção contra sobrecarga para motores trifásicos e não um estudo das diferentes formas de partida bem como os métodos para se limitar a corrente de partida, não farão parte desta dissertação informações detalhadas sobre o assunto, mas sim apenas uma pequena ilustração do tipo de partida que foi utilizada durante os testes práticos [1] e [2].

Na maioria das suas aplicações os motores de indução do tipo gaiola, de pequena potência, podem arrancar por ligação direta à rede elétrica sem que se verifiquem quedas bruscas na tensão de suprimento e sem que se verifique ainda um tempo de partida prolongado. Foi este o tipo de motor utilizado nos testes relacionados a este trabalho.

O motor utilizado durante os testes práticos foi acionado através de uma botoeira que por sua vez aciona um contator que está ligado ao objeto deste trabalho: o relé inteligente. O acionamento desta botoeira deu-se através de um comando TTL vindo do relé inteligente. Este motor foi conectado em triângulo e partida direta, pois como é de pequeno

porte não causa nenhum estresse na rede elétrica e por isto não se faz necessário nenhum tipo de controle de partida.

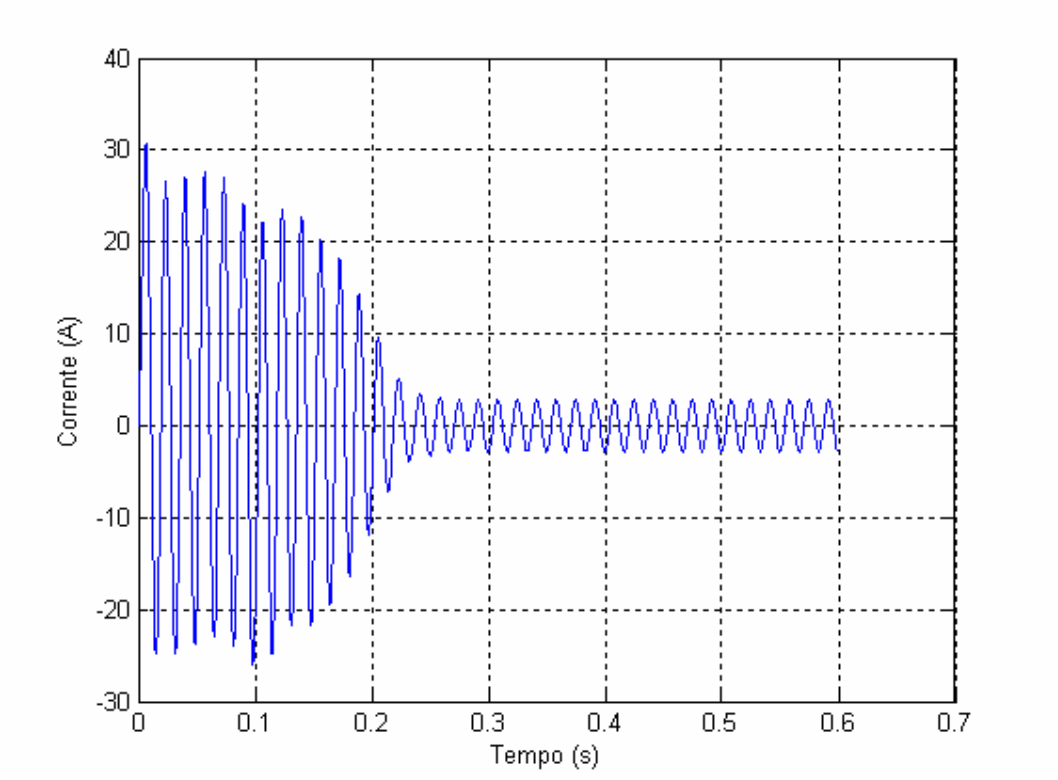


Figura 2.7 – Simulação da partida a vazio de um motor de indução

Apesar de não haver nenhum tipo de controle de partida neste trabalho, é necessário informar o tempo de partida do motor a ser monitorado e controlado, pois o relé inteligente precisará desta informação para poder ignorar a corrente lida pelo seu transdutor para efeito de monitoramento da corrente em regime permanente, já que a corrente de partida é muitas vezes maior que a nominal. A Figura 2.7 mostra a simulação do tempo de partida típico de um motor a vazio. Note que o tempo de partida é menor do que 250 ms.

Já a Figura 2.8 mostra o mesmo motor da Figura 2.7 partindo com carga nominal, note que o tempo de partida sobe para aproximadamente 350 ms. Mesmo assim, os tempos de partida típicos de um motor de indução normalmente são menores do que 1 s.

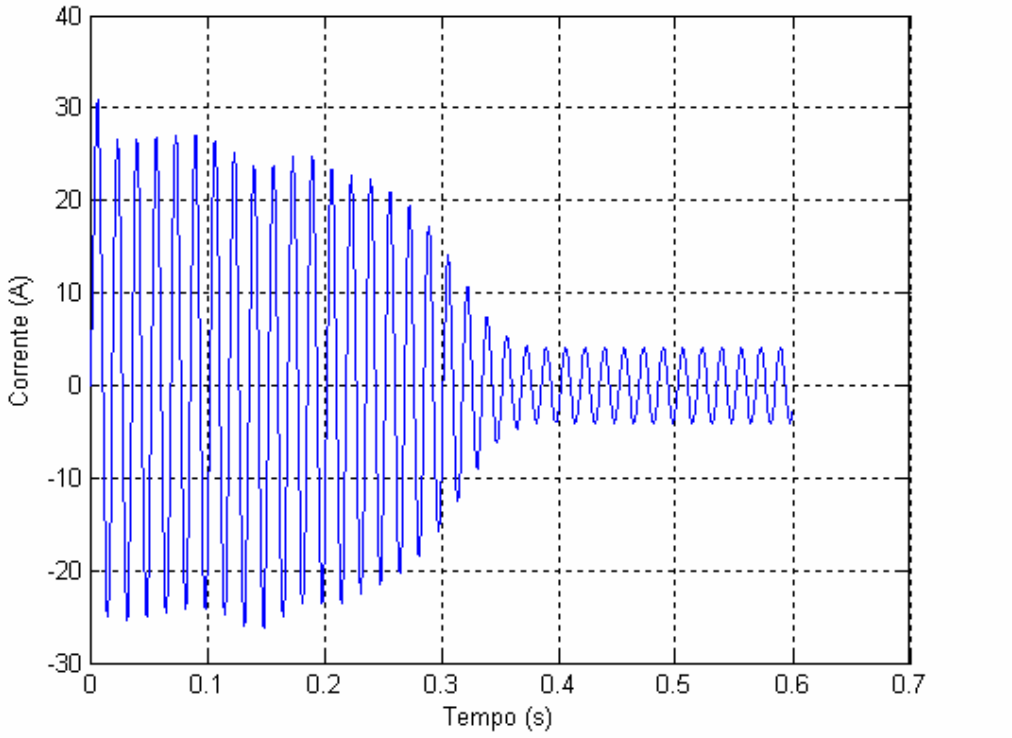


Figura 2.8 – Simulação da partida um motor de indução à plena carga

2.2 Corrente de partida e corrente nominal de um motor

Os motores elétricos são cargas que apresentam características bastante peculiares se comparadas às demais cargas elétricas existentes. Por isto, os equipamentos a motor (denominação que a NBR 5410 sugere para este tipo de dispositivo) devem possuir um circuito de ligação e proteção diferenciado.

Durante a partida, todo motor absorve uma corrente bastante superior à de funcionamento normal à plena carga, como foram mostradas nas Figura 2.7 e Figura 2.8. Por isto, o projeto do circuito de ligação deverá ser dimensionado para suportar as condições de partida e não só as condições de regime permanente. A corrente de partida I_p de um motor trifásico de indução tipo gaiola de esquilo, que são utilizados em mais de 90% das aplicações, apresenta os seguintes valores típicos [3]:

- Motores de dois pólos: $4{,}2 \cdot I_n \leq I_p \leq 9 \cdot I_n$;
- Motores de mais de dois pólos: $4{,}2 \cdot I_n \leq I_p \leq 7 \cdot I_n$.

Onde:

- I_n a corrente nominal;
- I_p a corrente de partida do motor de indução.

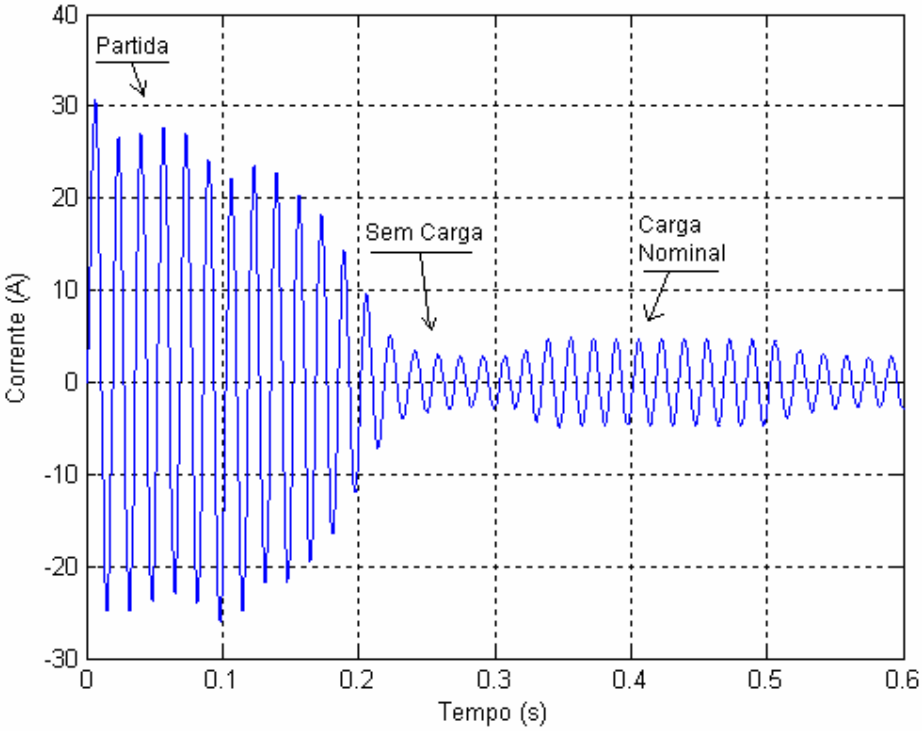


Figura 2.9 – Simulação da partida a vazio e inclusão da carga nominal temporária

Observando a simulação da Figura 2.9 pode-se ter uma nítida visão da diferença entre as correntes de partida, até 250 ms, com carga nominal, entre 350 ms e 500 ms, e sem carga, a partir de 500 ms.

Outra peculiaridade dos equipamentos a motor é que a sua potência absorvida em funcionamento é determinada pela potência mecânica no eixo do motor, potência que é solicitada pela carga acionada, o que pode resultar em sobrecarga no circuito de alimentação caso não haja proteção adequada.

Observe que o dispositivo de proteção contra sobre corrente deverá suportar, sem atuar, a corrente de partida, sendo capaz, portanto, de distinguir o momento de partida do motor da sua condição de operação normal, pois se a corrente de funcionamento normal ultrapassar 15% da corrente nominal o dispositivo de proteção deverá ser capaz de atuar desligando o motor [3].

2.3 Métodos de proteção contra sobrecarga

Alguns dispositivos foram surgindo basicamente para tratar do importante assunto de proteção dos circuitos de motores, dentre eles o relé térmico de sobrecarga é sem dúvida nenhuma o mais empregado quando o assunto é a proteção contra sobre corrente. Normalmente os relés térmicos são conectados juntamente com os contatores. Os relés térmicos de sobrecarga são divididos em classes de disparo, que permitem adaptá-los às características dos motores, em especial às condições de partida.

Outros dispositivos de proteção podem ser utilizados, mas como o objeto principal deste trabalho é desenvolver um relé inteligente que consiga atuar de forma semelhante a um relé térmico, então apenas este tipo de dispositivo será apresentado aqui.

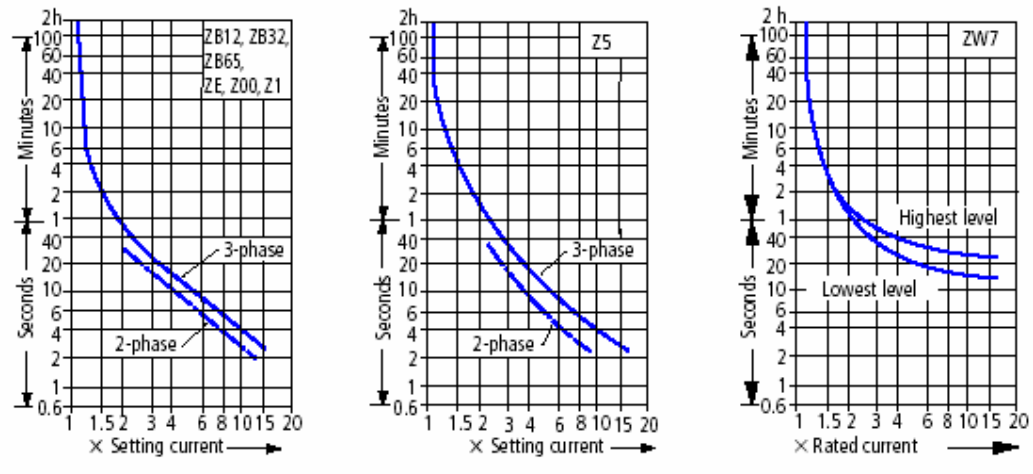


Figura 2.10 – Curvas de disparo de alguns relés térmicos

Conforme pode ser observado na Figura 2.10, os relés bi metálicos irão desligar o motor conforme as curvas apresentadas na figura. Estes relés são constituídos basicamente por um conjunto de lâminas composta de dois metais de diferentes coeficientes de dilatação térmica – um conjunto destes por fase – e um mecanismo de disparo. O seu funcionamento está baseado no aquecimento destes dois metais, pois como eles estão soldados um sobre o outro ao esquentarem devido à sobre corrente na fase um dos metais irá dilatar mais que o outro, fazendo com que estas lâminas se afastem do contato abrindo o circuito. Pode-se notar que se trata de um sistema de proteção relativamente simples.

Para tentar eliminar ou pelo menos atenuar os efeitos de temperaturas ambientes superiores às indicadas pela curva de disparo de cada relé bi metálico, recorre-se à

compensação do relé, obtida através da alteração na conformação das lâminas bimetálicas ou pela utilização de uma lâmina bimetálica auxiliar [3].

A proteção contra curto-circuito deve ficar por conta de outro dispositivo, tal como um fusível ou um disjuntor magnético. Isto porque o relé bi metálico pode levar um tempo considerado alto para atuar sobre uma corrente de curto-circuito.

2.4 Proposta de solução para o problema de sobrecarga

Uma proposta de solução seria a substituição de um relé bi metálico por um relé inteligente. A diferença maior entre estes dois dispositivos está associada ao fato de que o relé inteligente possibilitará ao técnico responsável pela manutenção do motor visualizar um breve histórico de sobre correntes que causaram o desligamento automático do motor.

O protótipo proposto para o relé inteligente seria, em termos funcionais, parecido com o diagrama ilustrado pela Figura 2.11. Observe que a idéia seria possibilitar em primeiro lugar uma interface gráfica e bastante amigável com um PC remoto através de uma comunicação serial, em seguida pensa-se em utilizar sensores de corrente para monitorar as correntes de fase do motor e por fim deseja-se atuar sobre a bobina de um contator utilizando uma saída de nível lógico digital.

A placa microcontrolada deste protótipo necessitaria possuir um canal serial para comunicação remota, pelo menos uma saída digital para comandar o acionamento do motor e ainda um conversor analógico digital (ADC) para poder tratar as correntes lidas.

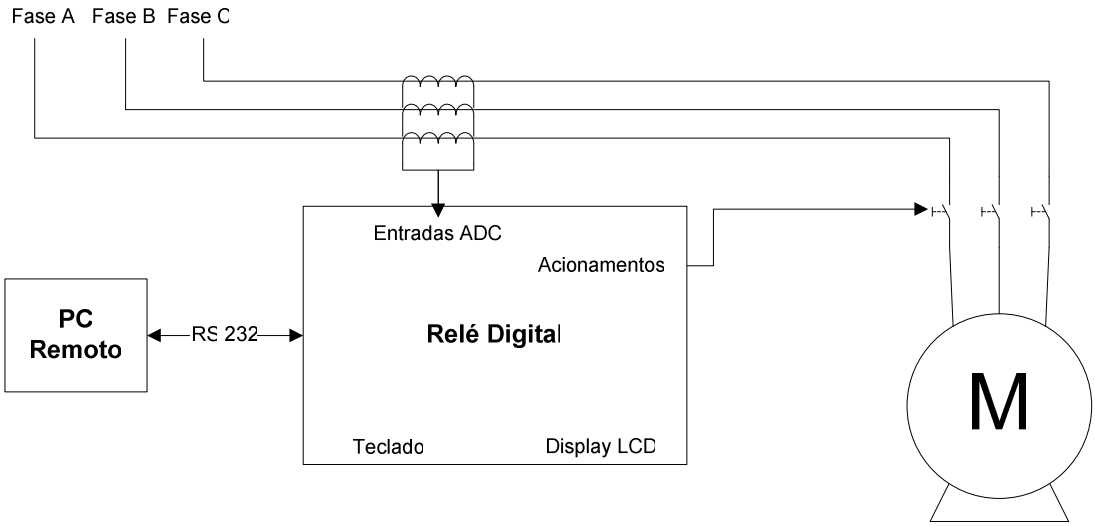


Figura 2.11 – Diagrama de uma proposta para o relé digital

As vantagens deste protótipo sobre a proteção tradicional que utiliza um relé térmico são:

- Possibilita o monitoramento on-line da corrente do motor por um PC remoto;
- Histórico das sobre correntes as quais o motor foi submetido;
- Possibilidade de acionamento remoto do motor;
- Praticamente as mesmas funções disponíveis remotamente através do PC podem ser comandadas localmente através de um teclado com quatro botões e um display LCD;
- Alteração da curva de resposta (Sobrecorrente x Tempo) com o sistema em operação.

Com posse dos conhecimentos básicos necessários para o bom entendimento do funciomento do objeto de controle, o motor de indução, pode-se agora partir para o estudo teórico da eletrônica necessária para o desenvolvimento de uma placa que consiga atender aos requisitos deste trabalho. No próximo capítulo será visto um estudo sobre eletrônica e microcontroladores visando o desenvolvimento do *hardware* do relé digital.

Capítulo 3 - Microcontroladores

Neste capítulo serão apresentados os conceitos básicos sobre microcontroladores e eletrônica, o enfoque visa um melhor entendimento de como foi possível desenvolver o protótipo do relé inteligente.

Ao se analisar este trabalho de uma maneira macro pode-se notar que se trata de um trabalho mais voltado à engenharia eletrônica do que à engenharia de potência, apesar de sua aplicação estar totalmente direcionada à engenharia de potência. Este fato faz com que o prévio estudo da engenharia eletrônica e uma ênfase um pouco maior no estudo de microcontroladores seja fundamental para o bom entendimento.

Este capítulo basicamente se divide da seguinte forma: noções sobre a eletrônica analógica, sobre eletrônica digital, sobre microcontroladores e um detalhe maior sobre o microcontrolador PIC utilizado. Todo este conteúdo foi utilizado no desenvolvimento do protótipo.

3.1 Noções de eletrônica analógica

A eletrônica vista aqui foi dividida em duas partes: eletrônica analógica e eletrônica digital. A eletrônica analógica é a base de toda a engenharia eletrônica e está presente direta ou indiretamente em praticamente todos os ramos da ciência atual.

A eletrônica básica envolve circuitos simples livres de circuitos integrados com apenas componentes chamados de analógicos tais como: resistores, indutores, capacitores, diodos e transistores. Já a eletrônica digital abrange circuitos com os chamados circuitos integrados (CI), os CIs são componentes eletrônicos que possuem dentro deles inúmeros resistores, capacitores, diodos e transistores agrupados de tal maneira que são capazes de executar determinadas tarefas previamente programadas.

Neste projeto foram utilizados resistores, capacitores e transistores juntamente com alguns circuitos integrados, por isto algumas noções básicas do funcionamento de circuitos que utilizem estes componentes serão mostradas aqui.

3.1.1 Noções de circuitos com resistores

O resistor é conhecido como um dos componentes passivos de um circuito, pois ele não interfere de forma ativa no comportamento da corrente elétrica ou da tensão. Trata-se apenas de um componente capaz de limitar a corrente elétrica devido a sua resistência à corrente elétrica. A lei de Ohm ilustrada na equação abaixo mostra bem o funcionamento de um resistor num circuito.

$$V = R \cdot i \Leftrightarrow i = \frac{V}{R} \Leftrightarrow R = \frac{V}{i} \qquad \text{Equação 3.1}$$

Onde:

- R: é a resistência elétrica de um resistor;
- V: é a diferença de potencial elétrico entre dois pontos;
- i: é a corrente elétrica que passa pelo resistor devido esta diferença de potencial.

A Figura 3.1 ilustra como é a representação esquemática de um resistor em um circuito.

Figura 3.1 – Representação esquemática de um resistor

É importante entender o funcionamento de um resistor em um circuito, pois o seu conceito foi amplamente utilizado no decorrer deste trabalho. Primeiro com resistores de

Pull-up e *Pull-down*, em seguida com resistores para limitar a corrente de base de um transistor, dispositivo que será visto no decorrer deste capítulo.

3.1.2 Noções de circuitos com diodos

O diodo já é um semicondutor, isto é, um dispositivo normalmente de silício que é trabalhado quimicamente para se implantar nele características especiais [4]. Aqui será visto apenas o funcionamento básico do diodo sem explicar como ou porque ele consegue se comportar desta maneira.

O diodo "D1", cuja representação em um circuito está ilustrado na Figura 3.2, funciona como um controlador do sentido convencional da corrente elétrica, isto é, ele "aponta" para o sentido que ele permite a corrente passar e se por algum motivo uma corrente reversa tentar passar por ele, esta corrente não conseguirá passar no sentido reverso e o diodo irá reter toda a tensão que estiver tentando produzir esta corrente reversa.

Figura 3.2 – Representações esquemáticas de diodos

São vários os tipos de diodos existentes no mercado, mas neste projeto foram utilizados apenas dois tipos: o diodo simples e o diodo zenner. O zenner é conectado ao circuito de maneira diferente do diodo simples, ele é utilizado normalmente quando se deseja limitar uma tensão para evitar que algum outro componente não se queime. Este diodo é capaz de funcionar, quando polarizado com uma corrente reversa, como um circuito aberto até a sua tensão nominal, depois que atinge esta tensão ele passa a conduzir corrente para evitar

que sua tensão aumente além da sua tensão nominal. Este tipo de diodo é utilizado neste projeto como proteção das entradas analógicas do microcontrolador.

3.1.3 Noções de circuitos com transistores

Os transistores foram a grande revolução na eletrônica do século XX, foi graças a ele que hoje existem poderosos computadores e os mais variados circuitos integrados. O transistor possibilitou a evolução da eletrônica digital como um todo.

Neste projeto o transistor foi utilizado como uma chave liga desliga e por isto será apenas apresentada aqui a sua função como chave [4] e [5]. O funcionamento de um transistor, cuja representação pode ser vista na Figura 3.3, como chave é basicamente ligando-se os pinos 2 e 3 dele como se fossem um diodo, ou seja, o sentido convencional da corrente elétrica que deverá passar por ele deverá ser do pino 2 para o pino 3 e utilizar o pino 1 como sendo a chave ou *gate*.

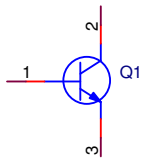


Figura 3.3 – Representação esquemática de um transistor

Basicamente o transistor funciona da seguinte forma, se uma tensão for aplicada na base (pino 1) de tal forma que uma corrente de base passe pelo transistor então o transistor irá “abrir” uma passagem de corrente entre o coletor (pino 2) e o emissor (pino 3). Quando se retirar a corrente de base o transistor irá fechar a passagem de corrente entre o coletor e o emissor, funcionando assim como uma chave de liga e desliga.

Deve-se cuidar para que as correntes de base e as correntes que passam entre o coletor e o emissão não ultrapassem as correntes máximas especificadas pelo fabricante do transistor em questão, pois isto iria danificar o transistor e, portanto, iria fazer com que o circuito não funcionasse corretamente.

3.2 Noções sobre eletrônica digital

Eletrônica digital seria um pré-requisito para microcontroladores, aqui foi utilizado apenas o conhecimento sobre *flip-flops* e sobre conversor analógico digital além de uma boa noção de circuitos TTL e CMOS.

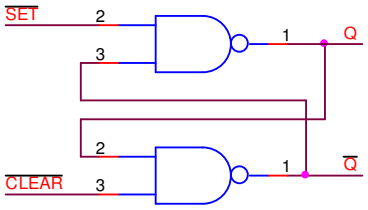
A eletrônica digital foi a ciência que possibilitou o surgimento de computadores e microcontroladores, foi graças a esta ciência que hoje é possível realizar o controle preciso de vários processos. Na eletrônica digital o que se faz é representar todos os sinais analógicos e que se deseja controlar em números binários ("0" e "1"). Depois de se conseguir equacionar o que se deseja dentro da álgebra booleana fica relativamente fácil encontrar um circuito que consiga executar tal controle [6].

É importante ter uma boa noção sobre o *latch* (circuito base para um *Flip-Flop*) e ainda mostrar o funcionamento básico de um conversor analógico digital, pois o *lacth* é a estrutura básica dos dispositivos de memória atuais e o conversor foi utilizado neste trabalho para poder monitorar a corrente do motor.

3.2.1 *Flip-Flop*, estrutura de memória digital

O *lacth nand* é um *Flip-Flop* (FF) simples com apenas duas portas *NAND*. A característica básica de um *lacth* seria possuir dois pinos de saída, uma sendo o complemento da outra em termos de álgebra booleana, isto é, quando uma tiver o valor "1" a outra terá o valor "0" e vice-versa. Estas saídas são normalmente denominadas como sendo Q e $\overline{Q}$.

As entradas deste *lacth* são $\overline{SET}$ e $\overline{CLEAR}$, estão normalmente em repouso no estado alto e por isto possuem a barra em cima do nome, assim se alguém desejar colocar a saída Q em nível lógico alto deverá colocar a entrada $\overline{SET}$ em nível lógico baixo mantendo a entrada $\overline{CLEAR}$ em nível lógico alto, caso deseje-se colocar a saída em nível lógico baixo deve-se fazer o oposto. As duas entradas não podem estar ao mesmo tempo no nível lógico baixo, pois isto seria um comando inválido para este circuito relativamente simples. O circuito e a tabela-verdade de um *lacth nand* pode ser visto na Figura 3.4.



$\overline{SET}$	$\overline{CLEAR}$	Q	$\overline{Q}$
1	1	Q	$\overline{Q}$
0	1	0	1
1	0	1	0
0	0	inválido	

Figura 3.4 – *Lacth Nand* com portas *NAND*: esquemático e tabela-verdade

Facilmente pode-se conseguir variações deste circuito como por exemplo colocando uma porta *NAND* a mais em cada entrada de comando conseguindo assim um circuito síncrono, isto é, introduzindo um comando de *Enable* ou *Clock*. Mas o mais importante era mostrar aqui como é possível armazenar um bit de memória.

Agrupando-se quatro *lacht* destes pode-se armazenar uma palavra de 4 bits, que é chamada de *nibble*. Então se agrupando dois *nibbles* obtém-se uma palavra de 8 bits denominada byte (observe que seria necessário oito circuitos de *lacth* como o da Figura 3.4). E assim por diante pode-se conseguir o circuito com o número de bits que se necessite simplesmente agrupando-se vários *lacths* em paralelo.

3.2.2 Conversor analógico digital

O estudo de um conversor analógico digital (ADC) não pode ocorrer sem primeiro ilustrar como é o funcionamento de um conversor digital analógico (DAC), isto porque um ADC é muito mais complexo do que um DAC. Normalmente o ADC utiliza um DAC interno para saber quando a conversão chegou ao fim.

O circuito mais tradicional de uma conversão digital analógica seria uma rede R / 2R, esta é a melhor maneira de se explicar o funcionamento de um DAC porque além de trabalhar basicamente com resistores é fácil de montar na prática pelo fato de possuir apenas dois valores de resistências: R e 2R. A Figura 3.5 mostra como seria um DAC com esta rede de apenas 4 bits.

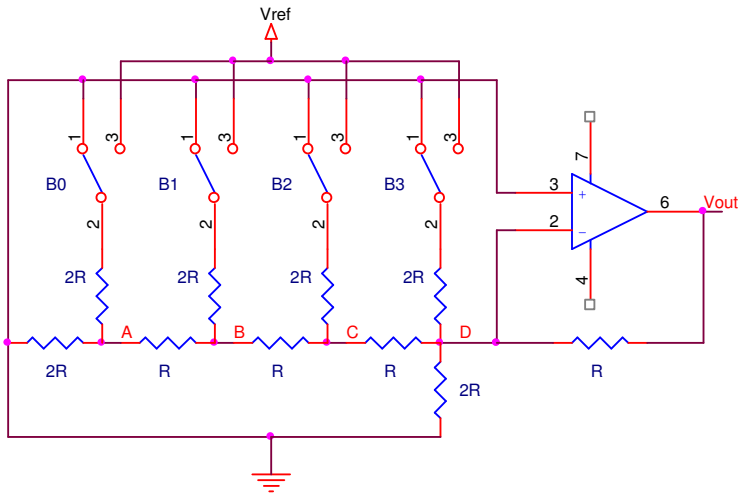


Figura 3.5 – DAC de 4 bits com rede R / 2R

A entrada deste DAC seria a palavra B de 4 bits, representada na figura por B3, B2, B1 e B0. O número indicado de cada bit indica a potência que a base dois do número binário deverá ser o expoente da potência com base 2, isto é, o bit B2 tem valor 2^2 e o bit B3

tem valor 2^3. Assim a palavra B pode variar de 0, quanto todos os bits estiverem com o valor zero, até 7, quando todos os bits tiverem o valor um.

Para explicar o funcionamento deste circuito pode-se utilizar o teorema da superposição de fontes, isto é, pode-se analisar separadamente cada bit do circuito e depois somar a contribuição de cada bit para então chegar ao valor de V_{out}. Também será necessário utilizar o teorema de Thévenin, veja referência bibliográfica [7], para encontrar a tensão e a resistência equivalente vista do ponto "D".

A análise deste circuito fica mais fácil se for feito do bit mais significativo primeiro, assim primeiramente será vista a contribuição apenas deste bit ligado à V_{ref} e os demais ligados ao terra.

O circuito então teria no ponto "A" o resistor de 2R do B0, que estaria conectado à massa (terra), em paralelo com o 2R do ponto "A" para a massa, logo a resistência equivalente do ponto "A" para o terra seria 2R//2R=R. Então poderia se somaria esta resistência com a resistência existente entre os pontos "A" e "B" o que resultaria novamente em uma resistência de 2R.

Então se repetiria esta mesma análise até chegar ao ponto "D", que seria um pouco diferente. No ponto "D" ter-se-ia uma resistência de 2R vinda do ponto "C", a resistência de 2R do próprio ponto "D" à massa, o que difere o ponto "D" dos demais pontos, e ainda o resistor 2R do bit B0 que nesta consideração estaria ligado à V_{ref}. Logo as duas resistências de 2R que estão ligadas à massa resultariam em uma resistência equivalente de R e a tensão no ponto "D" seria a do divisor de tensão entre os resistores 2R, ligado à V_{ref}, e R ligado à massa. A tensão calculada por este divisor de tensão seria então $\frac{1}{3} \cdot V_{ref}$.

Já a resistência equivalente vista pelo ponto “D” seria de 2R/3. O circuito equivalente de Thévenin visto do ponto “D” está ilustrado na Figura 3.6, observe que o V_{th} é a tensão calculada para cada bit, no caso do bit B3 será $\frac{1}{3}\cdot V_{ref}$.

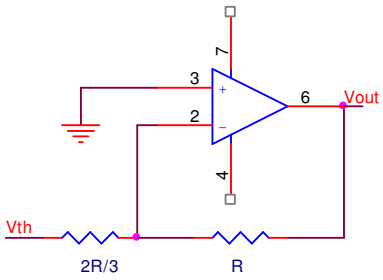


Figura 3.6 – Circuito equivalente de Thévenin do DAC de 4 bits

Como há uma massa virtual entre os terminais 2 e 3 do amplificador operacional, a equação para determinar a tensão de saída é dada pela equação .

$$V_{out} = -\frac{R}{2R/3}\cdot V_{th} = -\frac{3}{2}\cdot V_{th} \qquad \text{Equação 3.2}$$

Como a tensão de Thévenin para o caso do bit B3 é $\frac{1}{3}\cdot V_{ref}$, a contribuição para a tensão de saída do bit B3 será de $-0{,}5\cdot V_{ref}$. Fazendo uma análise análoga para encontrar a tensão de Thévenin para o bit B2 será encontrada uma tensão de $\frac{1}{6}\cdot V_{ref}$ e conseqüentemente a contribuição deste bit para a tensão de saída será de $-0{,}25\cdot V_{ref}$.

As demais contribuições respeitarão a regra de ser a metade da anterior até o bit menos significativo. Como o teorema da superposição diz que a tensão resultante será a

soma das contribuições parciais das tensões encontra-se para o DAC de 4 bits a seguinte equação de tensão de saída:

$$V_{out} = -\frac{B}{8} \cdot V_{ref} \qquad \text{Equação 3.3}$$

Onde:

- V_{out} e V_{ref}: são as tensões de saída e referência, respectivamente;
- B: é o valor da palavra binária de entrada do DAC.

Observe que o maior valor de um DAC normalmente atinge ao valor da fonte de referência utilizada, neste exemplo pode-se notar que o máximo valor seria, em módulo, 7/8 da tensão de referência.

No ADC acontece a mesma coisa, se for necessário ler uma tensão de um potenciômetro ligado a uma tensão V_{ref} jamais seria possível ler o valor exato de V_{ref}, mas apenas $2^{(n-1)}$ valores com em passos discretos de $2^{-(n-1)}$ iniciados em 0 e máximo valor de $\frac{2^{(n-1)}-1}{2^{(n-1)}} V_{ref}$, onde n seria o número de bits do seu conversor ADC.

Neste trabalho inicialmente foi utilizado um ADC de 8 bits, pois um ADC de 8 bits seria capaz de ler uma valor aproximado de 99,22% do valor máximo de referência e um passo aproximado de 0,78% do valor de referência, ou seja, em termos de metrologia, este "instrumento" de medição seria da classe de 1% de exatidão, o que é mais do que suficiente para este protótipo.

Existem vários tipos de ADC no mercado atual e a característica mais importante que varia entre eles é a velocidade de conversão, esta velocidade de conversão varia porque varia o algoritmo utilizado para a implementação do ADC.

Como o intuito deste capítulo é apenas explicar como funciona basicamente a ciência que será utilizada no trabalho, será visto apenas um ADC típico e seu funcionamento para explicar como funciona esta conversão, os demais tipos de ADC serão apenas citados e podem ser vistos com mais detalhes na referência bibliográfica [6].

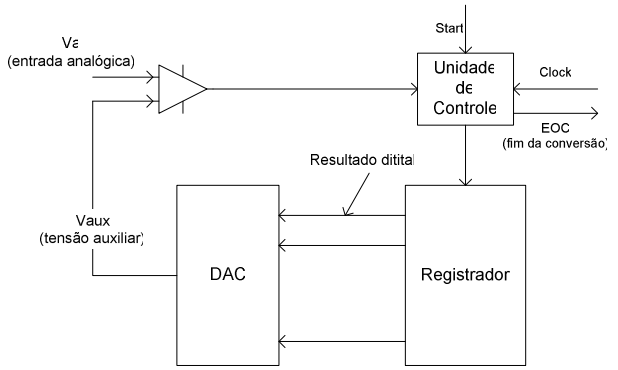


Figura 3.7 – Diagrama geral de uma classe de ADC

A Figura 3.7 mostra como seria um diagrama de blocos de uma ADC, o funcionamento é relativamente simples: primeiramente um comando de *START* deve ser dado à unidade de controle quando a entrada analógica V_a estiver pronta para a conversão, a unidade de controle irá precisar de um sinal de *Clock* externo para poder sincronizar o envio da primeira palavra digital ao registrador. Então a cada passo do *Clock* o ADC verifica se a nova tensão gerada pelo DAC interno é maior do que a da entrada analógica, quando esta resposta for verdadeira então o ADC interrompe sua conversão habilitando o sinal *EOC*.

Alguns tipos de ADC são: ADC de rampa digital, há um contador interno que irá contar desde 0 até um valor um pouco maior que V_a encontrando assim o seu valor; ADC de aproximações sucessivas, é o mais utilizado por reduzir o seu tempo de conversão de acordo com o número de bits, pois este verifica do bit mais significativo para o menos

significativo se cada um destes bits devem ou não estar ligados; ADC flash, é o de maior velocidade disponível atualmente, porém requer muito mais circuitos e conseqüentemente mais caro.

3.3 Microcontroladores

Resumidamente pode-se dizer que um microcontrolador nada mais é do que um circuito integrado da eletrônica digital com uma tabela-verdade muito maior que a maioria dos circuitos integrados e cuja tabela-verdade interna possibilita a execução de várias tarefas diferentes.

Seu funcionamento básico seria a tradução de um valor de uma palavra binária na sua entrada, que são chamados de *opcode*, em função de sua tabela-verdade interna. Os *opcodes* são os valores de entrada da unidade central de processamento (CPU) que foram previamente programados neste CI e que representam algum tipo de operação, que pode ser lógica, aritmética ou de transferência de dados, tudo respeitando sua tabela-verdade.

Na verdade é bem mais complexo do que isto, mas esta é uma analogia bastante interessante de se fazer. Existem vários livros abordando este assunto e alguns deles estão na referência bibliográfica deste trabalho [6], [8] e [9]. A melhor maneira para verificar o funcionamento de um microcontrolador é através de seu diagrama de blocos como o da Figura 3.8.

Observe que a Figura 3.8 possui vários circuitos independentes em um único chip conectados de tal forma que tudo execute exatamente os *opcodes* gravados na memória de programa, ROM neste caso. No caso dos microcontroladores utilizados ainda há mais um tipo de memória de dados além da memória RAM, a memória EEPROM. As portas paralelas e seriais são para realizar a interface com o mundo externo ao microcontrolador, as

interrupções são para melhorar algumas funções, por exemplo, a da comunicação serial. Os temporizadores podem ser utilizados para a contagem de tempo e assim realizar alguns atrasos ou esperas por algum evento de um determinado tempo. Os conversores ADC e DAC também são utilizados para interagir com o mundo externo.

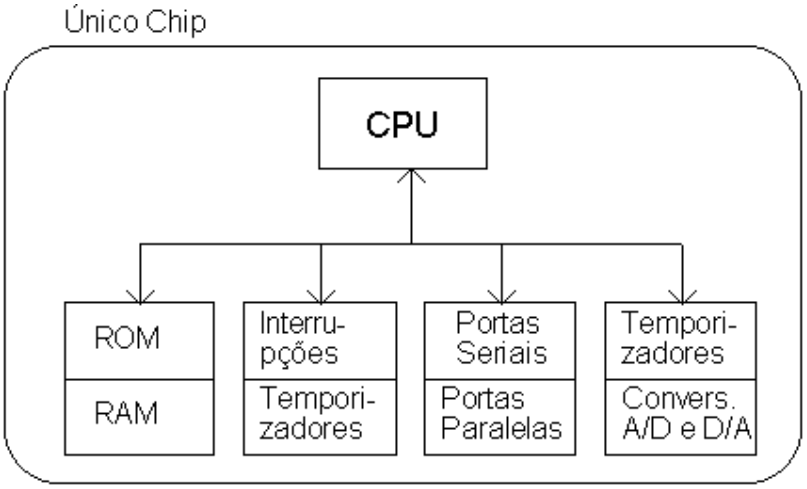
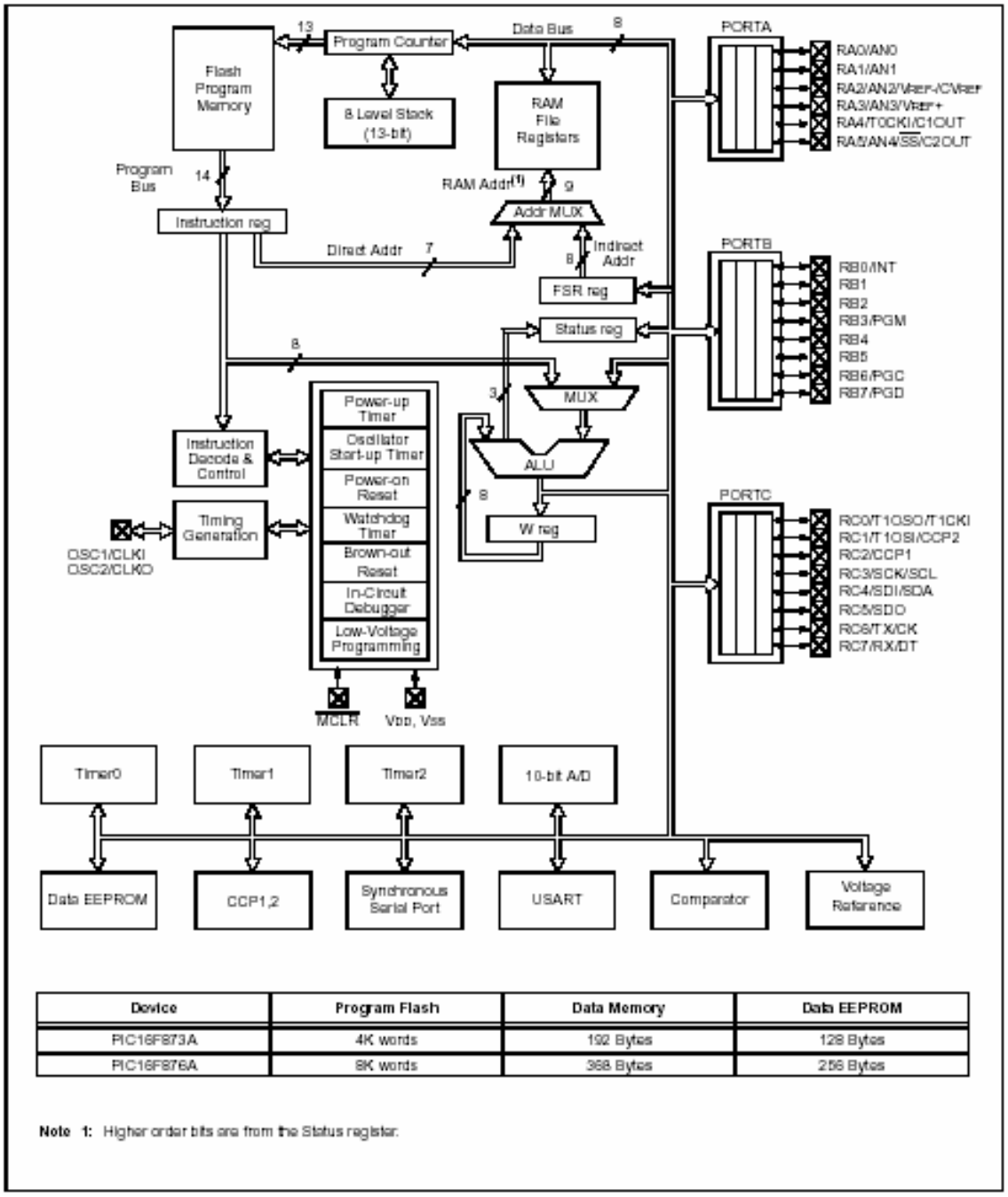


Figura 3.8 – Diagrama de blocos de um típico microcontrolador

A explicação de como estes vários periféricos internos ao microcontrolador funcionam e interagem entre si pode ser vista na referência bibliográfica de cada microcontrolador em particular. Detalhes sobre o funcionamento de microcontroladores em geral também podem ser encontrados nas bibliografias deste trabalho [10] e [11].

3.3.1 Microcontrolador PIC16F876A

O microcontrolador utilizado aqui será apresentado de uma maneira resumida e direta para ilustrar todo o funcionamento básico do PIC adotado e ainda dar uma noção de como funciona um microcontrolador de tecnologia RISC [8].



Device	Program Flash	Data Memory	Data EEPROM
PIC16F873A	4K words	192 Bytes	128 Bytes
PIC16F876A	8K words	368 Bytes	256 Bytes

Note 1: Higher order bits are from the Status register.

Figura 3.9 – Diagrama de blocos do PIC16F87XA

Na Figura 3.9 está o diagrama de blocos do microcontrolador utilizado para a implementação do protótipo do relé inteligente. Observe a quantidade de funções disponíveis

e imagine a densidade deste tipo de circuito integrado, conseqüentemente a complexidade deste microcontrolador.

Este PIC possui 28 pinos, dos quais apenas 22 estão disponíveis para I/O, foram utilizados 4 pinos da porta RB como barramento de dados, isto é, funcionando como I/O, e outros 3 pinos desta porta foram utilizados como barramento de controle para o controle do LCD e um pino ficou reservado para habilitar e desabilitar o teclado.

A porta RA foi toda praticamente reservada para o conversor analógico digital, exceto pelo pino que habilita o teclado. Enquanto que a maioria dos pinos da porta RC ficaram disponíveis para futuras implementações. Esta porta é a da comunicação serial, 2 pinos foram utilizados para a comunicação em si e, se for o caso, 1 pino poderá ser utilizado para o controle do fluxo da comunicação se o protocolo RS-485 for utilizado.

Todas as portas possuem saídas compatíveis com a tecnologia TTL e são capazes de fornecer correntes de até 25mA cada, caso seja necessária mais corrente um *drive* de corrente deverá ser conectado à saída da porta em questão.

Apresentados os conhecimentos que possibilitem o projeto e desenvolvimento da placa de circuito impresso que viabilize este projeto será dado início, no próximo capítulo, ao desenvolvimento e projeto em si. Primeiramente, a placa de circuito impresso será desenvolvida e posteriormente o *firmware* que consiga atender às necessidades de programação deste protótipo será implementado.

Capítulo 4 - Projeto do Equipamento (*Hardware* e *Firmware*)

O enfoque deste capítulo será mostrar todos os passos para a escolha dos componentes para a confecção do protótipo e também como foi elaborado o sistema operacional que irá rodar na placa microcontrolada, também chamado de *firmware*.

É importante ressaltar a sutil diferença existente entre *firmware* e *software*. O *firmware* seria o programa desenvolvido especialmente para uma placa, por exemplo, a BIOS de um computador pessoal, que é desenvolvido especialmente para realizar a interface com a placa mãe. Já o *software* é um programa desenvolvido para uma aplicação final, normalmente o *software* roda sobre um sistema operacional previamente escolhido enquanto que o *firmware* não.

Este capítulo está dividido em itens que obedecem a uma ordem cronológica da criação do protótipo. Inicialmente será apresentado um diagrama de blocos que mostra os componentes básicos de todo o projeto. Posteriormente cada bloco do diagrama será estudado a parte visando a determinação de todos os componentes necessários para o seu funcionamento. Por fim será feita a integração entre todos estes blocos criando assim um protótipo para este projeto.

4.1 Modelagem por Diagrama de Blocos

O melhor diagrama de blocos para mostrar as diferentes características necessárias ao protótipo é ilustrado na Figura 4.1.

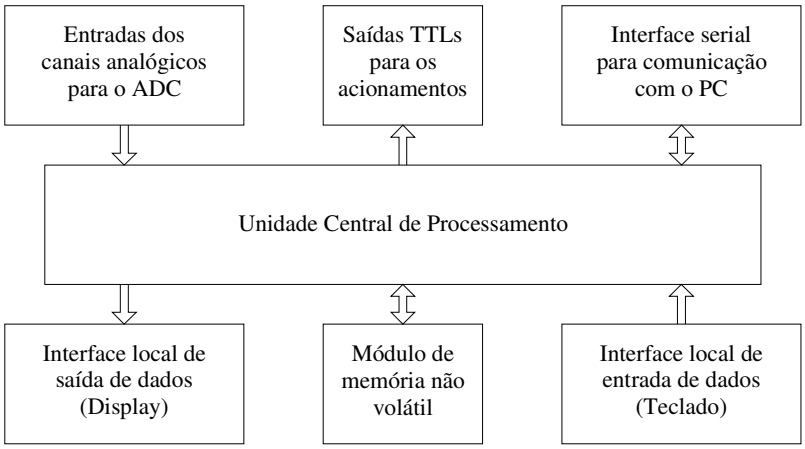


Figura 4.1 – Diagrama de blocos do relé inteligente

Para atender aos requisitos exigidos neste diagrama de blocos foi necessário fazer um estudo das inúmeras relações entre os custos e os benefícios dos componentes compatíveis existentes no mercado. Também foi levada em conta a estrutura do laboratório da Universidade Federal de Uberlândia, onde o protótipo foi desenvolvimento.

4.1.1 Unidade central de processamento

Por se tratar de um protótipo bastante simples em termos de processamento, a unidade central de processamento (CPU) escolhida pôde ser a de um microcontrolador básico de 8 bits, que é a mais barata.

Além disto o microcontrolador deve possuir uma interface serial compatível com a existente nos computadores pessoais, assim foi escolhida a interface USART que é bastante comum entre os microcontroladores atuais. Outra consideração é o número de pinos I/O, pois o dispositivo deverá ler um teclado, comandar um LCD, atuar em pelo menos um acionamento e ainda portas para comandar um ADC externo, conforme a proposta inicial, ou possuir entradas analógicas para um ADC interno.

Observe que a escolha do microcontrolador está relacionada com todas as funções que se deseja obter do equipamento. Inicialmente pensou-se em utilizar a família do 80C51 da Intel, por ser mais fácil de programar devido à sua arquitetura CISC. Mas por conveniência da estrutura laboratorial existente, optou-se inicialmente pelo PIC16F628 e posteriormente pelo PIC16F876A, por este último possuir um conversor analógico digital interno além de pinos de I/O.

Outro fator que auxiliou na escolha da família PIC foi o fato de que este tipo de microcontrolador possui uma memória flash interna suficientemente grande para armazenar todo o desenvolvimento do *firmware* o que simplifica o projeto de *hardware*, enquanto que a tecnologia da família 80C51, na maioria das opções existentes, necessitaria de um barramento de dados externo para acessar a memória de programa externa.

4.1.2 Módulo de memória não volátil

Alguns bytes de memória EEPROM são necessários para implementar a memória não volátil do equipamento, isto porque se deseja que o mesmo “lembre” da sua última configuração ao religar.

A EEPROM foi escolhida no lugar da memória FLASH porque a aplicação necessitava de pouca memória e seria melhor utilizar uma memória que possuísse escrita e leitura por byte, ou seja, byte a byte a EEPROM pode ser alterada e lida enquanto que a FLASH normalmente precisa ser apagada por bloco.

Normalmente estes dois tipo de memória são utilizadas como memória de programa, mas neste caso a idéia é utilizá-las como memória de dados não volátil, assim sua gravação é feita em tempo de execução e o PIC já possui instruções que facilitam este tipo de programação.

O PIC16F876A possui 256 bytes de memória EEPROM, então para o desenvolvimento deste firmware optou-se por dividir estes 256 bytes em 8 blocos de 32 bytes cada, no primeiro bloco de memória foram colocados os dados referentes à configuração do equipamento, tais como identificação do equipamento (ID) e os dados do motor a ser controlado (corrente nominal, tempo de partida e suas respectivas unidades). No segundo bloco foram colocados os dados referentes à curva que o relé inteligente deverá obedecer durante a monitoração do motor configurado. Nada foi implementado nos demais blocos, mas o desenvolvimento do *firmware* foi de tal maneira que facilmente pode-se aumentar o número máximo de pontos da curva de controle do equipamento ou utilizar este vários bytes para outras aplicações.

4.1.3 Entradas dos canais analógicos para o ADC

Foi exatamente nesta fase do projeto que se desistiu do PIC16F628, pois como ele não possui um ADC interno seria necessário um externo, até aí tudo estava dentro do esperado. Mas quando se percebeu que seria necessário um ADC serial externo e quando se estudou o datasheet de um periférico destes tudo mudou, pois esta programação seria bastante complicada de se desenvolver.

Em virtude desta dificuldade optou-se migrar para o PIC16F876A, porque este possui a mesma estrutura de memória interna, o que possibilitou migrar todo o código de programa desenvolvido para o PIC16F628 sem grandes modificações, e ainda possui um ADC interno de 10 bits com até 5 canais. Em virtude desta troca o projeto ficou com pinos de I/O sobrando, já que o PIC16F876A possui 22 pinos de I/O [11].

Já quanto às características do ADC interno, como o que se deseja monitorar é uma corrente senoidal cuja freqüência de rede é de 60 Hz, o tempo de aquisição de dados

deste ADC é mais do que o suficiente, aproximadamente 20 μs [11]. Quanto à configuração do ADC interno, optou-se por utilizar apenas três canais e utilizar as outras duas entradas analógicas como entrada de referência para melhorar a qualidade do sinal medido.

4.1.4 Interface serial para comunicação com o PC

A interface serial escolhida para comunicação com o PC foi a RS-232 para o desenvolvimento do protótipo, mas pretende-se utilizar a RS-485/422 também caso seja possível. A idéia inicial era utilizar uma rede CAN para esta comunicação, porém seria muito mais difícil e caro.

A vantagem da rede CAN para a rede RS-485 é que pode-se conectar um novo equipamento na rede em qualquer topologia e sem se preocupar com a fiação, enquanto que a rede RS-485/422 necessita ser conectada ponto a ponto, não permitindo ramificações. Então a rede CAN ficará como idéia de trabalhos futuros.

Como o protocolo entre a RS-232 e a RS-485 são os mesmos e a única diferença é na camada física da rede, então tudo que for feito para um poderá ser aproveitado para o outro padrão bastando substituir a camada física, isto é, a interface elétrica da comunicação. Ambos os padrões utilizam normalmente 8 bits de dados, um bit de sincronismo chamado *start bit* e um bit de parada denominado de *stop bit*. Outras variações podem ser utilizadas, por exemplo, a utilização de um bit de paridade para fazer o controle de redes com mais de um escravo no caso do padrão RS-485 ou RS-422. A Figura 4.2 ilustra bem como está distribuído estes bits ao longo do tempo.

Figura 4.2 – Protocolo de comunicação típico para RS-232 e RS-485

A diferença mais significativa em termos funcionais entre os padrões RS-232 e RS-485 é que o primeiro necessita de no mínimo três fios (um para transmissão, um para recepção e um comum de referência) enquanto que no segundo podem ser utilizados apenas dois fios (ambos são utilizados para a transmissão e recepção) ou RS-422 com quatro fios (dois para transmissão e dois para recepção de dados).

Se forem utilizados apenas dois fios para o padrão RS-485 a comunicação deverá ser *half duplex*, isto é, apenas um dispositivo de cada vez poderá transmitir seus dados, isto porque existe apenas um canal de dados e não pode ser usado para transmitir e receber dados ao mesmo tempo.

Já no padrão RS-232 a comunicação pode ser sempre *full duplex*, ou seja, ambos os dispositivos podem transmitir simultaneamente, pois existe um canal para transmissão e outro para recepção.

Para evitar que o PC transmita dados ao mesmo tempo que o protótipo e que ocorra colisão de dados na rede caso ela seja *half duplex* desenvolveu-se toda a programação da serial com um protocolo mestre escravo, isto é, quando um equipamento somente transmite depois de receber algum dado e o PC é o único que inicia uma tranmissão..

A diferença entre as camadas físicas utilizadas está basicamente nas tensões colocadas na rede e seus significados. Para o RS-232 o nível lógico zero é transmitido quando uma tensão positiva entre 3V e 12V estiver entre os terminais de transmissão e o comum, analogamente um nível lógico um é quando esta tensão for negativa entre -12V e -3V. Na recepção de dados os mesmos níveis de tensão devem ser respeitados mudando apenas que a diferença de tensão estará entre os terminais de recepção e o comum. O padrão RS-485, por possuir apenas dois fios, trabalha com níveis lógico correspondentes às diferenças de tensão entres estes dois fios. Se a tensão for positiva então um zero lógico foi transmitido e se for negativa um nível lógico um foi transmitido ou recebido. Normalmente a amplitude desta tensão varia de 2V à 6V [16].

A utilização do padrão RS-232 é mais simples e facilita a interface com os computadores pessoais atuais, pois eles possuem, normalmente, pelo menos uma porta serial RS-232. Se for utilizar o RS-485 – que tem a vantagem de atingir maiores distâncias devido ao fato de transmitir tensões balanceadas sem uma tensão de referência comum – será necessário instalar um conversor de RS-485 para RS-232 na porta serial do PC [16].

Os componentes adquiridos para realizar a comunicação serial foram o MAX232 e o HIN232 para o padrão RS-232 e o DS485 para o padrão RS-485. A escolha por estes dispositivos foi baseada no baixo custo e por serem fáceis de encontrar.

4.1.5 Interface local de saída de dados (display)

Para fazer uma interface visual com o usuário local optou-se pela utilização de um display de cristal líquido de 2 linhas por 16 colunas, pois este LCD é bastante comum e possui caracteres suficientes para realizar uma boa interface. Visando melhorar ainda a

interface de saída de dados foi ligado um LED vermelho indicando se o motor está ligado ou não.

O LCD escolhido foi um com comunicação paralela pela facilidade de sua programação, porém foi preciso utilizar 4 pinos de dados e 3 pinos de controle para realizar esta comunicação. Gastou-se apenas 4 pinos de dados porque este tipo de LCD pode ser conectado com um barramento de 4 ou de 8 bits de dados. Optou-se pelo barramento de 4 bits para "economizar" portas de I/O do PIC e porque já havia 4 pinos no barramento de dados destinados ao teclado, assim foi compartilhar estes pinos. Fazer funcionar esta interface foi a parte mais complicada de todo o projeto, isto porque foram desenvolvidas aqui todas as rotinas de comunicação com o LCD em linguagem de máquina.

4.1.6 Interface local de entrada de dados (teclado)

A entrada de dados não foi difícil de implementar e a idéia inicial foi satisfatória para atender ao que se esperava do projeto. Foi criado um teclado de apenas quatro teclas: uma para acessar o menu do dispositivo (tecla "Menu"); uma para sinalizar uma confirmação de dados (Tecla "Ok"); uma para sinalizar um incremento (tecla "Up"); e finalmente uma para sinalizar um decremento (tecla "Down").

Com estas quatro teclas foi possível criar uma boa interface de entrada de dados tanto para perguntas e confirmações simples quanto para entrada de valores numéricos pelo usuário. Para "ler" estas teclas foram necessários utilizar 5 pinos do PIC, os mesmos 4 pinos do barramento de dados utilizados na comunicação com o LCD mais 1 pino de controle para habilitar o teclado – o que possibilitou a economia de 3 pinos de I/O do PIC.

4.1.7 Saídas para os acionamentos

Foi separado um pino de I/O do PIC para poder controlar o acionamento do motor, caso no final do projeto sobre mais pinos outras aplicações poderão surgir, mas por hora o objetivo é apenas controlar o acionamento de um único motor através da monitoração de suas correntes.

Uma pequena placa adicional foi utilizada para elevar a corrente do pino do PIC para uma corrente capaz de acionar um contator, por exemplo. Esta placa utilizará um circuito bastante simples com um transistor e um relé de 5V comum.

4.2 *Hardware* proposto

Conforme foi visto anteriormente, inicialmente o projeto foi baseado no PIC16F628, uma vez que este PIC atendia ao requisitos de memória RAM e EEPROM para os dados e memória FLASH para o *firmware* e aos requisitos de pinos de I/O. Então foram desenvolvidos tanto o protocolo de rede da camada aplicação com este microcontrolador como toda a interface com o LCD e com o teclado.

Somente quando se concluiu esta parte do trabalho foi que se iniciou o estudo de como seria a interface com o ADC externo. Como já estava previsto neste projeto, reservou-se dois pinos de I/O (RA6 e RA7) para a comunicação serial com o ADC externo. A Figura 4.3 mostra como estava o projeto que utilizava o PIC16F628.

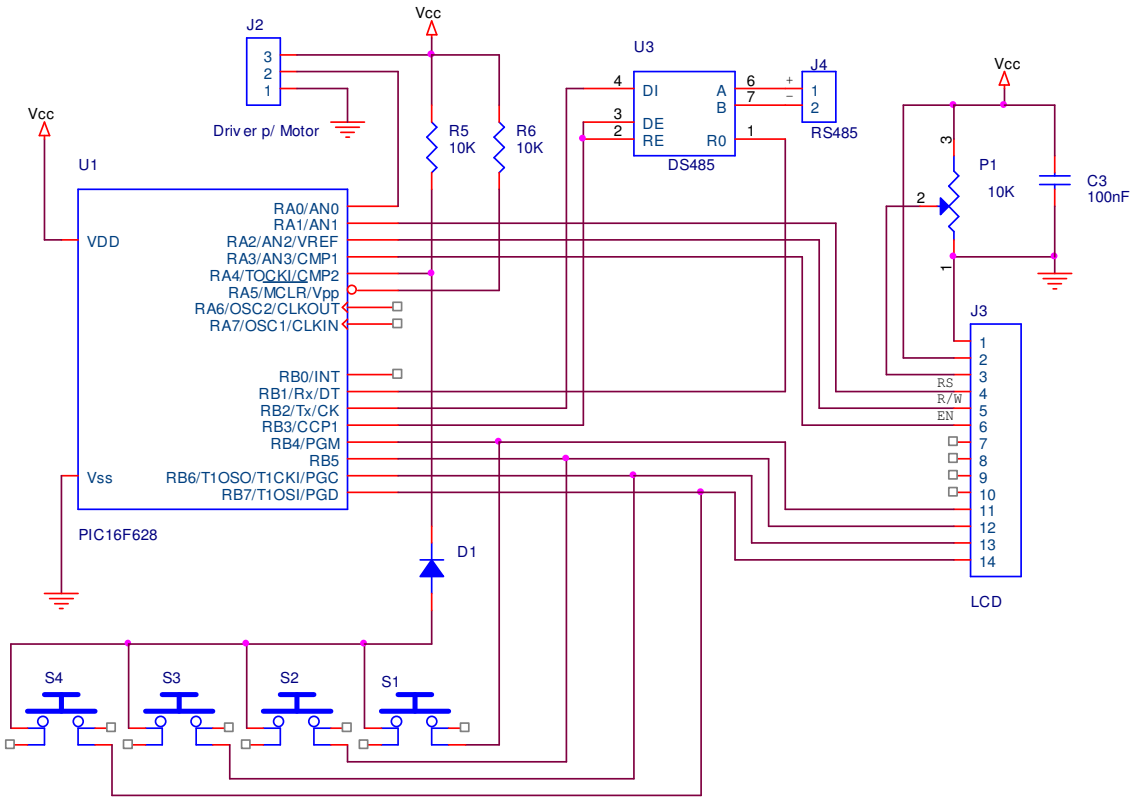


Figura 4.3 – Esquemático do relé inteligente utilizando o PIC16F628

Depois de um estudo completo sobre como seria a comunicação com o ADC externo, o tempo necessário para se programá-lo e o fato de o projeto com o PIC16F628 estar caminhando para o seu limite em termos de *hardware*, decidiu-se então trocar este microcontrolador por outro que já possuísse um conversor analógico digital interno, ou seja, o PIC16F876A.

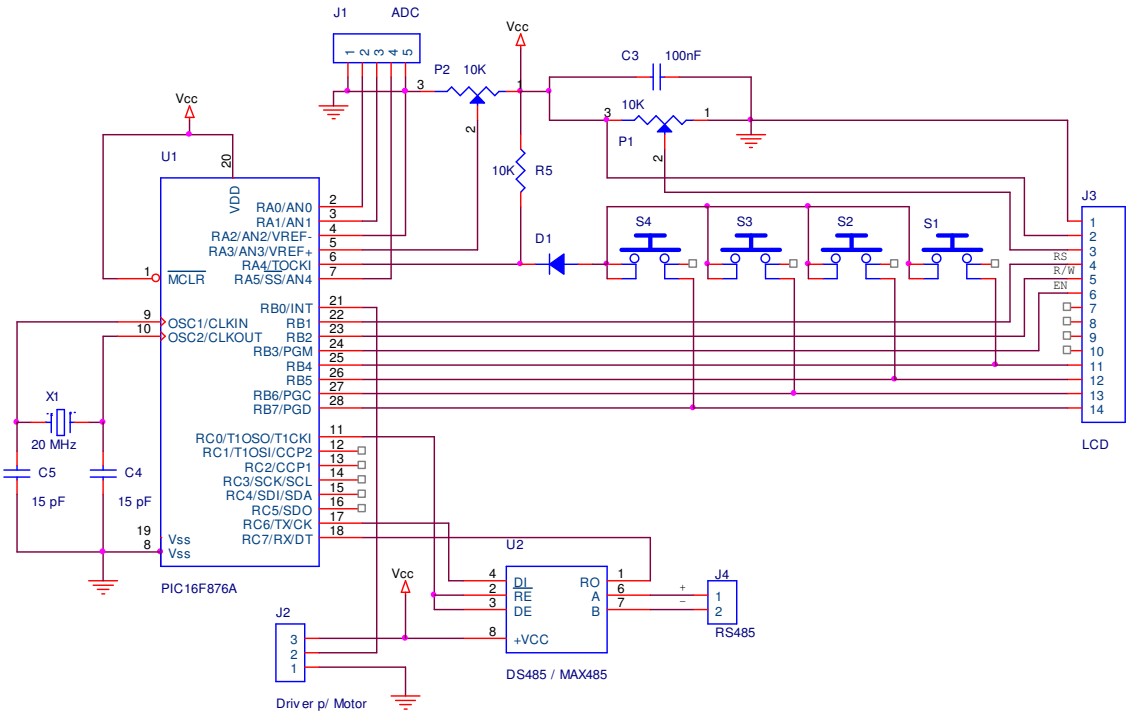


Figura 4.4 – Esquemático do relé inteligente utilizando o PIC16F876A

A escolha do novo microcontrolador não foi difícil, pois bastou procurar pelo PIC imediatamente superior que possuísse as mesmas características existentes no PIC16F628 e que ainda dispusesse de um ADC interno e mais alguns pinos de I/O para eventuais novas necessidades.

O primeiro microcontrolador que atendia a estas exigências foi o PIC16F873A, mas ao utilizá-lo descobriu-se que a disposição da sua memória RAM tornava impossível utilizar a mesma camada de aplicação da rede serial já desenvolvida, foi assim que se optou finalmente pelo PIC16F876A que é compatível com o endereçamento de memória RAM do PIC16F628, pois todo o *firmware* foi desenvolvido utilizando-se de ponteiros na declaração de variáveis.

A compatibilidade entre o PIC16F628 e o PIC16F876A foi possível principalmente devido ao fato de possuírem ponteiros para acessar os 16 últimos bytes do banco 0 da memória RAM, o que possibilita o acesso aos valores das variáveis do banco 0 mesmo estando com os outros bancos selecionados. Isto não ocorre no PIC16F873A.

O esquemático mostrado na Figura 4.4 foi o utilizado na confecção da placa do protótipo do relé inteligente salvo apenas a substituição do componente U2, o DS485, por um HIN232, o que simplificou consideravelmente os testes no laboratório, visto que o laboratório não contava com um conversor RS-485 para RS-232 que seria necessário instalar no PC. Outra mudança causada pele troca do DS485 pelo HIN232 foi o cabo de comunicação, que passou de 2 fios para um cabo de 3 fios, incluindo o fio de referência "GND".

Agora uma breve explicação deste esquemático dará uma idéia de como a placa funciona. Iniciando com U1, que é o PIC16F876A, ele foi conectado a uma alimentação de 5V (Vcc) e a um cristal de 20MHz – com este cristal o PIC executa uma instrução em apenas 200ns.

O conector J1 é utilizado para a entrada analógica dos três canais analógicos disponibilizados pelo PIC. O *hardware* ligado a este conector deverá limitar a tensão em até 5V para que não queime estas entradas analógicas. Este conector também fornece a tensão de referência para que o ADC interno identifique esta de tensão de entrada.

O conector J2 irá fornecer um sinal digital para o acionamento do motor, lembrando que será necessário um amplificador de corrente conhecido como *driver* para que o sinal digital seja capaz de acionar um contator, que por sua vez irá acionar o motor.

O U2 e o conector J4 foram substituídos conforme já foi dito aqui, mas eles têm a função da camada física da rede de comunicação serial, isto é, é através deles que os dados realmente são transmitidos entre o relé inteligente e o PC remoto.

O conector J3 será ligado ao LCD, observe que serão necessários 4 pinos de dados e 3 pinos de controle para que o PIC consiga comunicar-se com o LCD. Ainda estão disponíveis neste conector dois pinos para a alimentação (Vcc e Gnd) e ainda um pino para o ajuste do contraste do display.

Os botões S1, S2, S3 e S4 terão as funções das teclas Down, Up, Ok e Menu, respectivamente. Observe que o pino 6 do PIC – o RA4 – está ligado neste botões e enquanto ele estiver com um nível lógico "1" (5V) o teclado estará desabilitado e os 4 pinos de dados – de RB4 até RB7 – poderão ser utilizados na comunicação com o LCD.

4.3 *Firmware* proposto

Como foi explicado anteriormente, o esquema da placa do relé inteligente é relativamente simples, mas isto é graças ao fato do microcontrolador utilizado possuir internamente vários periféricos. Se não fosse o fato do microcontrolador possuir uma memória flash para armazenar o *firmware*, uma memória EEPROM para armazenar os dados de configuração do usuário, um ADC e um canal serial USART então o esquemático seria bem mais complicado do que este apresentado aqui.

Por tudo isto, a grande dificuldade está mais relacionada ao projeto e desenvolvimento do *firmware* do que ao projeto do *hardware* necessário para atender aos requisitos do relé inteligente.

O *firmware* foi desenvolvido de maneira modular, assim o código necessário para controlar o teclado ficou totalmente independente daquele necessário para controlar o LCD. A grande vantagem de programar o microcontrolador em módulos é que a manutenção fica mais fácil.

O módulo que ficou responsável pelo teclado é acionado via varredura pelo programa principal no seu *loop* principal, isto é, o teclado não funciona com interrupção, mas sim por varredura. Já o módulo desenvolvido para controlar o LCD disponibilizou várias funções que simplificaram a escrita de caracteres no display.

Foi criado um módulo contador de tempo que funciona com interrupção, este módulo é utilizado para gerar o relógio interno do relé inteligente. Infelizmente para se conseguir um relógio preciso seria necessário outro cristal conectado diretamente a um pino específico, mas como não há necessidade de tanta exatidão foi utilizado o próprio oscilador de 20 MHz do microcontrolador para a contagem do tempo. Este contador será utilizado para medir o tempo em que o motor estará trabalhando em sobre corrente.

Um módulo de comunicação serial foi desenvolvido totalmente independente, ele é controlado remotamente via interrupção serial. Trata-se de um módulo que simplesmente responde aos comandos remotos de maneira rápida e eficaz. Este módulo utiliza as funções do módulo de acesso à memória EEPROM para gravar e verificar os dados configurados.

Por fim, um módulo capaz de ler os dados do ADC foi desenvolvido de modo a imprimir no display a corrente lida naquele instante. Este módulo também é executado via varredura pelo programa principal, assim como o teclado.

4.3.1 O programa principal do *firmware*

Nesta parte do trabalho serão apresentados todos os fluxogramas dos módulos desenvolvidos para o *firmware* do protótipo de relé inteligente, também será explicado o funcionamento de cada um deles para que todo o projeto do equipamento possa ser entendido.

A Figura 4.5 mostra o fluxograma completo do programa principal, observe que assim que o PIC é inicializado, o primeiro passo é iniciar os valores das variáveis globais

utilizadas no controle do relé inteligente. Em seguida o microcontrolador deve preparar as interrupções (serial e do relógio interno) para que este protótipo possa contar o tempo e ainda comunicar-se com o PC remoto. A última fase da inicialização do equipamento é a do display, isto é, sincronizar a comunicação entre o PIC e o LCD, sem dúvida nenhuma é nesta etapa que a programação é mais trabalhosa porque o PIC trabalha em tempos de "µs" enquanto que o LCD trabalha em "ms".

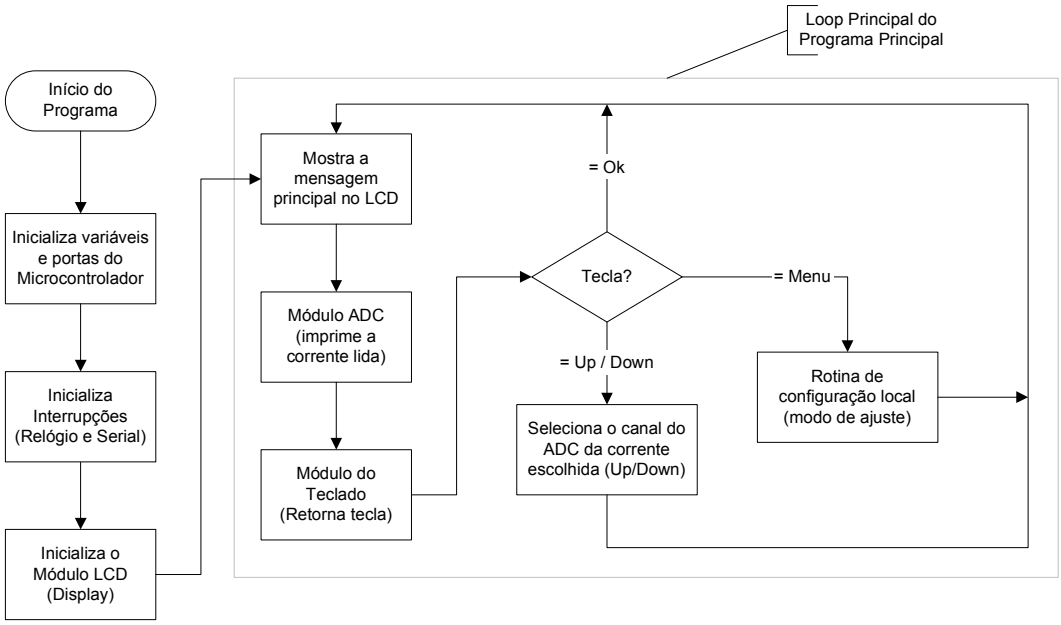


Figura 4.5 – Fluxograma do programa principal do *firmware*

O chamado "*Loop* Principal do Programa Principal" é a parte do programa que terá uma repetição eterna e que fará com que o relé inteligente sempre esteja monitorando as correntes, o teclado e ainda a comunicação serial. O *loop* principal começa mostrando uma mensagem padrão na primeira linha do LCD e em seguida faz a leitura de um dos canais do ADC e mostra o valor da corrente lida de acordo com o que foi configurado na sua memória

EEPROM. Então o microcontrolador irá monitorar o teclado para verificar se há algum comando local a ser atendido.

Observe que a comunicação serial e o controle do relógio juntamente com o controle do tempo em que a corrente monitorada está acima da corrente nominal configurada para o motor em questão não foram tratadas no programa principal e muito menos no *loop* principal. Na verdade a comunicação serial funciona através da interrupção que foi inicializada no programa principal enquanto que o controle de sobre corrente foi implementado na interrupção do relógio interno.

A Figura 4.6 mostra como foi implementada a rotina de configuração local, neste modo de ajuste o teclado é lido até que uma tecla seja pressionada, caso passe muito tempo sem que a tecla seja pressionada então a rotina do teclado retorna que nada foi pressionado. Durante o modo de ajuste o equipamento também não responde a nenhum comando serial, afinal ele está em ajuste, mas recebe os dados normalmente e pode executar o comando assim que sair deste modo, porém o PC remoto pode achar que o comando não foi recebido corretamente caso ocorra um erro de *time out*.

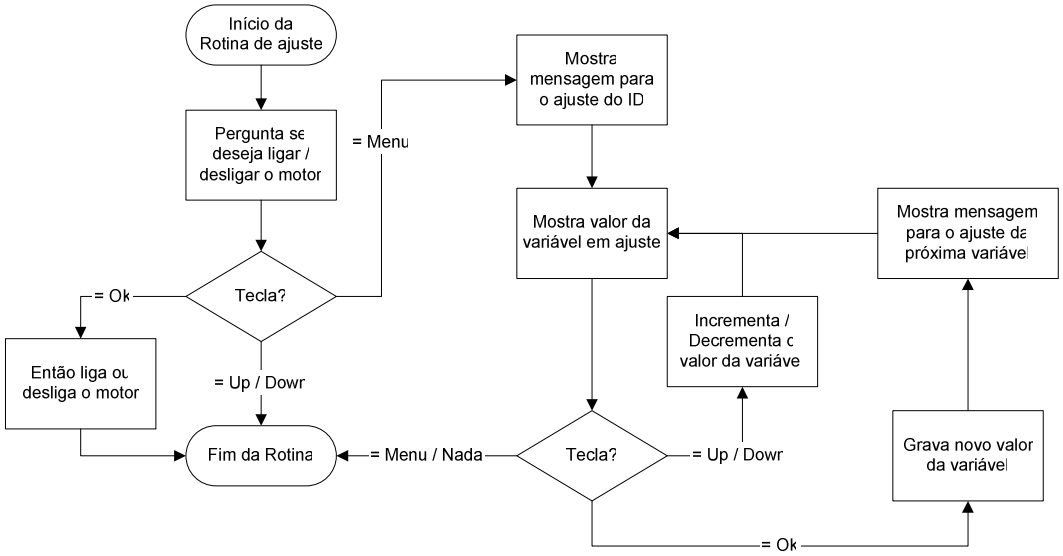


Figura 4.6 – Fluxograma da rotina para configuração local (modo ajuste)

Note que para simplificar o fluxograma não foram colocadas todas as variáveis capazes de serem alteradas localmente, mas sim apenas o nome da primeira que é o ID do equipamento a ser ajustado. Este ID é utilizado para saber a qual equipamento da rede está direcionado esta informação, isto foi desenvolvido no *firmware* já prevendo a instalação de um conversor RS-232 para RS-485 no PC e a troca do chip MAX232/HIN232 por um DS485, por exemplo. Esta mudança já permitiria a confirugação de uma rede, pois tanto o *firmware* quanto o *software* já foram desenvolvidos prevendo esta alteração.

Outras variáveis podem ser configuradas localmente via teclado, mas a interface para isto não é tão amigável quanto a configuração remota. Exemplos de outras variáveis são: máxima variação da corrente lida no ADC (chamado de delta); menor valor da corrente lida no ADC (corrente equivalente se o valor lido no ADC for zero); corrente nominal do motor; tempo de partida do motor; unidade em que o tempo de partida do motor está expresso; unidade em que a corrente do motor está expressa.

4.3.2 Módulo de controle do LCD (display)

O funcionamento do módulo do display é totalmente independente dos demais e é utilizado apenas para mostrar os resultados de uma maneira local. Depois de se inicializar o LCD e criar algumas rotinas que facilitam a impressão de caracteres no display ficou muito mais fácil de se utilizar este módulo, contudo foi aqui empregada a maior parte do tempo em programação. Inclusive, ao se trocar o PIC utilizado, foi necessário revisar todo este módulo porque o novo PIC – o PIC16F876A – necessitava de um cristal externo de 20MHz, assim o sincronismo entre o microcontrolador e o LCD necessitou de atrasos maiores.

O módulo do display conta com uma rotina de inicialização, uma de escrita e uma de leitura de dados. Outras variações destas rotinas básicas foram desenvolvidas para facilitar a escrita de frases completas gravadas na memória de programa – memória flash do microcontrolador – e para simplificar a programação do LCD.

Tabela 4.1 – Descrição dos pinos de um LCD padrão

Pinos	Função	Descrição
1	Alimentação	Terra ou GND
2	Alimentação	Vcc ou +5V
3	V0	Tensão de ajuste do contraste
4	RS	1 = Dado; 0 = Instrução
5	R/W	1 = Leitura; 0 = Escrita
6	E	1 = Habilitado; 0 = Desabilitado
7	B0	Barramento de Dados
8	B1	
9	B2	
10	B3	
11	B4	
12	B5	
13	B6	
14	B7	

A Tabela 4.1 traz uma descrição básica de cada pino do LCD. Dos 7 pinos utilizados para a comunicação entre o display e o microcontrolador, 4 pinos fazem parte do

barramento de dados – do pino 11 até o pino 14 – e é compartilhado com o teclado e 3 pinos são utilizados para o controle do LCD, chamados de barramento de controle.

Tabela 4.2 – Instruções básicas do LCD

Descrição	Modo	RS	R/W	Dados (Hexa)
Display	Liga (sem cursor)	0	0	0C
	Desliga	0	0	0A / 08
Limpa display com *home* p/ cursor		0	0	01
Controle do Cursor	Liga	0	0	0E
	Desliga	0	0	0C
	Desloca p/ Esquerda	0	0	10
	Desloca p/ Direita	0	0	14
	Cursor Home	0	0	02
	Cursor piscante	0	0	0D
	Cursor com alternância	0	0	0F
Sentido de deslocamento do cursor ao entrar caractere	Para a esquerda	0	0	04
	Para a direita	0	0	06
Deslocamento da mensagem ao entrar caractere	Para a esquerda	0	0	07
	Para a direita	0	0	05
Deslocamento da mensagem sem entrada de caractere	Para a esquerda	0	0	18
	Para a direita	0	0	1C
Endereço da primeira posição	Primeira linha	0	0	80
	Segunda linha	0	0	C0

A programação do display foi feita utilizando os comandos mais comuns descritos na A Tabela 4.1 traz uma descrição básica de cada pino do LCD. Dos 7 pinos utilizados para a comunicação entre o display e o microcontrolador, 4 pinos fazem parte do barramento de dados – do pino 11 até o pino 14 – e é compartilhado com o teclado e 3 pinos são utilizados para o controle do LCD, chamados de barramento de controle.

Tabela 4.2, para enviar comandos com apenas 4 bits (*nibble*) de dados é necessário enviar primeiro o *nibble* mais significativo depois o menos significativo. Por

exemplo, para enviar o comando liga display sem cursor (0x0C) deve-se enviar primeiro o *nibble* "0" ("0000" em binário) e depois o *nibble* "C" ("1100" em binário).

Para maiores detalhes de como este módulo ou os próximos foram implementados na prática basta consultar o Anexo A . Neste anexo encontra-se o código fonte compilado de todo o *firmware*, lembrando que a linguagem utilizada para esta programação foi a linguagem de máquina do PIC.

4.3.3 Módulo de controle do teclado

A Figura 4.7 mostra como o circuito (*hardware*) do teclado foi implementado, é necessário entender bem este circuito para compreender como ele foi programado. Primeiramente será apresentado os componentes do circuito:

- Vcc é a tensão de alimentação de 5V;
- R5 é o resistor de *pull-up*;
- D1 é o diodo de isolação dos pinos de dados;
- S1 é o botão do teclado, um *push bottom*;
- O pino de controle é para habilitar e desabilitar o teclado;
- O pino de dados é a representação de um único botão, lembrando que o relé inteligente conta com quatro botões idênticos a este.

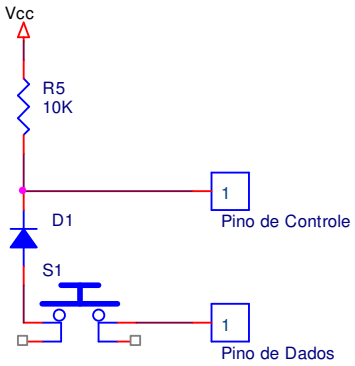


Figura 4.7 – Esquemático do circuito do teclado

Feitas as apresentações dos componentes deste circuito, parte-se agora para o entendimento de como foi possível a programação do teclado. É importante lembrar que o teclado foi programado pelo sistema de varredura, isto é, o *loop* principal fica monitorando o teclado de tempos em tempos e não há nenhuma interrupção do teclado como nos computadores pessoais.

O teclado fica normalmente desabilitado, ou seja, o pino de controle está sempre com nível lógico “1” (+5V). Nesta condição pode-se notar que o diodo jamais irá conduzir, pois a tensão de 5V é a mais alta tensão presente no circuito e, por isto, será impossível que o diodo seja polarizado diretamente a ponto de haver uma diferença de tensão direta de +0,7V. Como jamais passará corrente pelo diodo, o botão poderá ser pressionado que não alterará a tensão que estará presente nos pinos de dados, ou barramento de dados. Assim, os pinos de dados poderão estar sendo utilizados para acessar outros dispositivos, no caso o LCD.

Para se realizar uma leitura do teclado deve-se inicialmente habilitá-lo, isto é, colocar o pino de controle em nível lógico “0” (0V). Quando o pino de controle estiver com nível lógico “0” o diodo poderá ser polarizado diretamente e, conseqüentemente, as

"informações" contidas nos pinos de dados poderão sofrer alterações vindas desta conexão física.

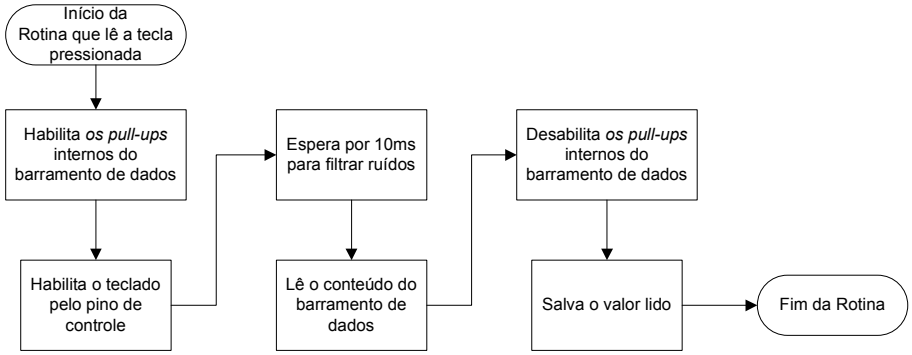


Figura 4.8 – Fluxograma da rotina que lê o teclado

Para ler a tecla propriamente dita deve-se colocar nível lógico "1" no pino de dados que está conectado à tecla que será lida, isto é feito através da habilitação dos resistores de *pull-up* internos da própria porta (veja a Figura 4.8). Em seguida aguarda-se algum tempo para estabilizar as tensões e eliminar os ruídos e então é feita a leitura da porta para verificar as teclas que estão pressionadas. Caso seja lido um nível lógico "1" a tecla não foi pressionada, mas caso seja lido um nível lógico "0" a tecla está pressionada.

Para finalizar a leitura do teclado no momento em que se salva o valor lido é feito um filtro dos bits do barramento de dados zerando os valores dos outros pinos da porta (a porta do PIC possui 8 pinos ou 8 bits) e posteriormente é negado o valor destes pinos, para que o bit fique com o valor "1" somente se a tecla for pressionada, o que facilita a utilização lógica no meio do programa, pois "1" está associado à verdadeiro e "0" à falso.

Observe que se o pino lido estiver em nível lógico "0" significa que está diferente do que foi colocado nele através dos resistores de *pull-up*, isto indica que houve uma

interferência do *hardware* externo, ou seja, o botão foi pressionado. Esta interferência ocorre devido à passagem de corrente pelo botão pressionado e pelo diodo diretamente polarizado, como a tensão de condução do diodo é de aproximadamente 0,7V, então a tensão no pino de dados será de aproximadamente 0,7V, tensão esta que indica um nível lógico "0" e não "1".

A regra implementada para a leitura do teclado neste *firmware* foi a mais simples possível, observe a Figura 4.9 para entendê-la melhor. Se não houver tecla pressionada a varredura passa pelo módulo do teclado somente verificando se deve responder a algum comando serial. Já se houver tecla pressionada, o relé irá guardar apenas a última tecla pressionada, isto é, mesmo que sejam pressionadas mais de uma tecla simultaneamente apenas a última ficará gravada na variável que indica a tecla pressionada. É praticamente impossível soltar as teclas no mesmo instante, pois o tempo que o PIC executa uma instrução é menor que 1µs.

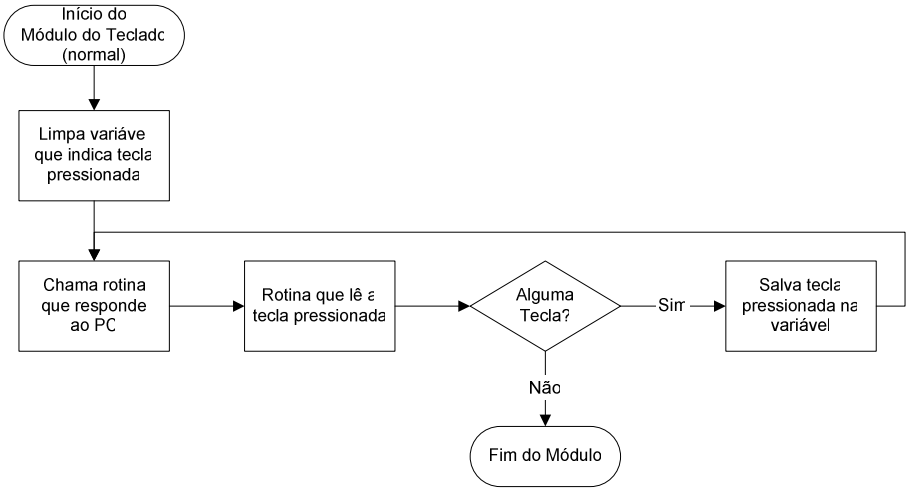


Figura 4.9 – Fluxograma do módulo do teclado em modo normal

Outro fluxograma de leitura do teclado foi implementado para quando o relé inteligente estiver sendo ajustado no módulo local, isto é, caso alguém esteja configurando o relé inteligente localmente. A Figura 4.10 mostra a outra forma de leitura do teclado implementada para quando o equipamento estiver em modo de ajuste local.

Note que de acordo com este segundo fluxograma o relé inteligente não espera que a tecla pressionada seja solta, isto faz com que se a tecla for mantida pressionada pelo usuário do equipamento por muito tempo o módulo irá acreditar que a tecla foi pressionada várias vezes, por isto no modo de ajuste deve-se ter bastante cuidado ao se pressionar o teclado. Também é importante notar que se nenhuma tecla for pressionada por muito tempo, aqui configurado 7s, o relé inteligente irá acreditar que o usuário não mais irá configurar o equipamento e, conseqüentemente, irá sair do modo de ajuste, o que evita que este protótipo fique no modo de ajuste por esquecimento.

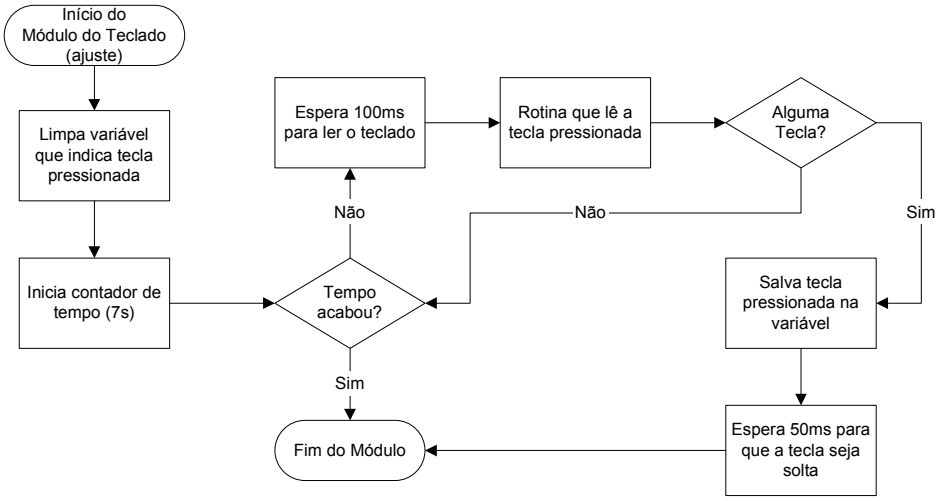


Figura 4.10 – Fluxograma do módulo do teclado em modo de ajuste

4.3.4 Módulo de controle do relógio

O contador de tempo ou relógio interno do relé inteligente foi implementado no *Timer2* do microcontrolador. Este contador de tempo interno foi programado para gerar interrupções de 1ms, isto é, ele foi programado para gerar mil interrupções por segundo.

Esta contagem de tempo possui um erro, como toda medição, mas para esta aplicação o erro é desprezível, porém para criar um relógio não, pois teria um atraso por dia que poderia ser significativo. Este erro não foi levantado porque não é devido ao cálculo, que está exatamente correto, mas devido às variações do cristal de 20 MHz utilizado, por isto quando se deseja uma precisão em um dispositivo destes utilizam-se um cristal de 32 kHz e um capacitor variável para realizar um ajuste fino do relógio.

Nesta rotina de interrupção foi implementado o controle responsável para verificar o tempo de sobre corrente do motor. Este controle simplesmente conta o tempo que a sobre corrente está no motor e verifica se este tempo está dentro do tempo programado para este motor. Caso este tempo seja ultrapassado o relé inteligente irá desligar o motor e gerar um alarme visual localmente e outro no programa instalado no PC remoto.

A grande vantagem de se implementar o controle de sobre corrente na interrupção do relógio interno do protótipo é que, com toda a certeza, o motor será desligado no instante em que foi programado para desligar.

4.3.5 Módulo de controle da comunicação serial

Este módulo é sem dúvida nenhuma o módulo mais complexo que foi desenvolvido neste protótipo, apesar do módulo do LCD ter sido mais trabalhoso e mais problemático, aqui foi elaborado os mais complexos códigos do *firmware*.

A complexidade deste código está relacionada à criação de um protocolo de comunicação e ao sincronismo da comunicação entre o *software* e o *firmware*. Mas, graças ao conhecimento adiquirido durante a elaboração de outros trabalhos que utilizam a comunicação serial [16], não houveram dificuldades na elaboração desta parte do código.

O recebimento de dados via canal serial foi todo implementado através da interrupção do PIC, mas a resposta a esta interrupção foi implementada na varredura do módulo do teclado, pois é o único lugar do *firmware* que sempre está em execução, já que o *loop* principal pode ficar sem ser chamado se alguém ficar segurando uma tecla pressionada.

A rotina responsável pela recepção dos dados vindos do canal serial simplesmente copia estes dados para um *buffer* na memória RAM do microcontrolador, uma vez que esta cópia tenha sido completada um *flag* irá indicar que os dados poderão ser tratados. Então no módulo do teclado uma rotina que verifica este *flag* e trata os dados, se for o caso, é continuamente chamada por varredura.

Já rotina que trata os dados é responsável por executar o comando enviado pelo PC remoto e ainda responder que este comando foi executado corretamente. No instante em que o relé está tratando os dados recebidos ele poderá estar: copiando dados para a EEPROM ou da EEPROM; ligando ou desligando o motor; atualizando o seu relógio interno.

A estrutura da camada de aplicação do protocolo de rede desenvolvido para a comunicação entre o PC e o relé digital é de 48 bytes assim distribuídos: 4 bytes de controle; 3 bytes com as 3 correntes de fase; 3 bytes com o horário do relógio interno (segundo, minuto e hora); 5 bytes disponíveis para futuras implementações; 32 bytes a serem escritos na EEPROM ou lidos; 1 byte com o número de bytes do pacote (48 ou 0x30). Caso haja o recebimento de um byte a mais ou a menos ou caso algum byte de controle seja diferente dos

valores esperados um erro de comunicação será reportado ao PC e será armazenado no seu histórico.

4.3.6 Módulo de controle do ADC

A Figura 4.11 mostra o fluxograma do módulo do ADC que é chamado durante a execução do *loop* principal. Sempre que o programa passa pelo *loop* principal as correntes são lidas pelo ADC do microcontrolador, mas se por algum motivo o *loop* principal for interrompido – pode ser porque alguém travou o teclado ou porque alguém esteja acessando o modo de ajuste – o equipamento não estará atualizando o valor da corrente, o que pode resultar numa tomada de decisão errada diante de uma possível corrente antiga.

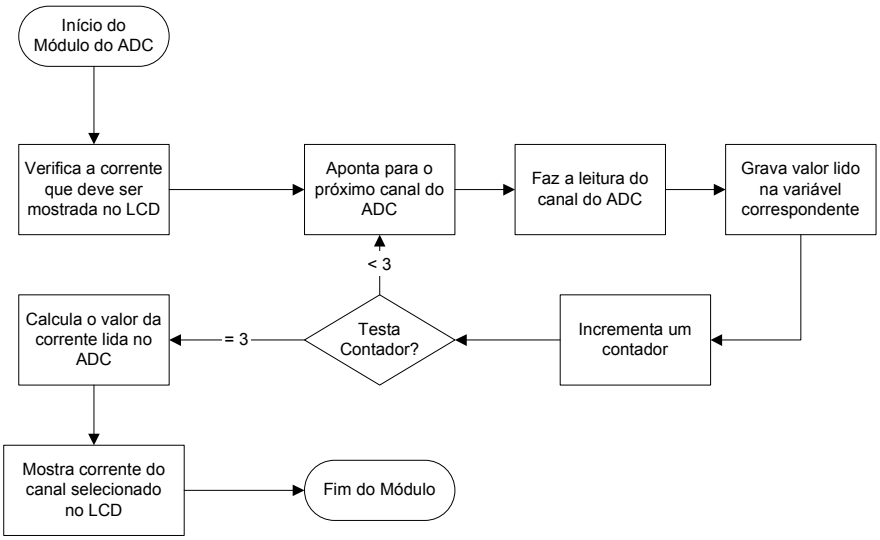


Figura 4.11 – Fluxograma do módulo do ADC

O fluxograma mostra que inicialmente este módulo irá verificar qual dos três canais está selecionado para ser mostrado no LCD, em seguida irá ler as correntes do ADC deixando por último a corrente do canal que deverá ser mostrado no display. Todas as

correntes lidas serão gravadas na memória RAM do equipamento e nenhum histórico é gravado, isto é, apenas a corrente instantânea é utilizada no monitoramento.

O byte lido no ADC representa o valor lido de cada corrente, mas para que este valor seja mostrado no LCD será necessário realizar um cálculo de acordo com uma reta que poderá ser configurada pelo usuário. Esta configuração possibilitará a conexão deste equipamento à diferentes medidores de corrente sem que haja problemas no seu funcionamento, desde que estes diferentes medidores de corrente sejam lineares.

O protótipo foi desenvolvido independnete do sensor utilizado, assim o sensor em si não faz parte do relé digital, mas sim é um periférico necessário e pode ser de qualquer natureza desde que seja uma função linear. O protótipo aceita a configuração desta reta conforme mostra a equação abaixo:

$$I_{Calc} = \frac{\Delta \cdot I_{lido} + I_{zero}}{256} \qquad \text{Equação 4.1}$$

Onde:

- I_{Calc} é a corrente calculada que será mostrada no LCD;
- I_{lido} é o valor proporcional da corrente lida no ADC (byte);
- I_{zero} é o valor proporcional da corrente mínima, quando o valor lido no ADC é zero (byte);
- Δ é a diferença entre a corrente máxima e a corrente mínima do transdutor utilizado.

Com esta função de transferência qualquer transdutor linear pode ser conectado na entrada do ADC do relé inteligente sem problema algum, isto faz com que o equipamento possa trocar de transdutor sem que seja necessário fazer qualquer tipo de adaptação. A única limitação é a da entrada analógica que não pode receber tensões acima de 5 V e, por ser usado

um ADC de 8 bits, maiores do que 0,02 V (5 V dividido por 256) para que seja perceptível ao conversor analógico.

Como o sensor deverá fornecer tensões entre 0,02 V e 5 V para o conversor analógico ser capaz de medir, o ganho do transdutor, ou sensor, utilizado deverá ser implementado antes da entrada do canal analógico.

4.4 Avaliação do protótipo montado

O protótipo desenvolvido para o relé inteligente foi capaz de atender a todos os requisitos iniciais deste trabalho. Com a placa e os componentes eletrônicos utilizados aqui pode se montar o relé inteligente e programá-lo.

O equipamento funcionando não só atendeu como também mostrou que com pequenas modificações, principalmente no seu *firmware*, outras funções bastante interessantes poderiam ser agregadas neste relé inteligente e serão citadas no capítulo final deste trabalho como trabalhos futuros.

Como o tempo de aquisição do ADC é inferior a 50µs, com um ADC tão rápido pode-se desenvolver um algoritmo capaz de medir o sinal do transdutor em AC e não em DC como foi implementado, isto possibilita uma resposta mais rápida a qualquer mudança na corrente monitorada, pois o retificador de precisão que converte o sinal AC em DC causa um pequeno atraso devido ao capacitor utilizado para encontrar o valor médio da onda AC. A Figura 4.12 mostra como deve ser o circuito conectado ao canal analógico do PIC.

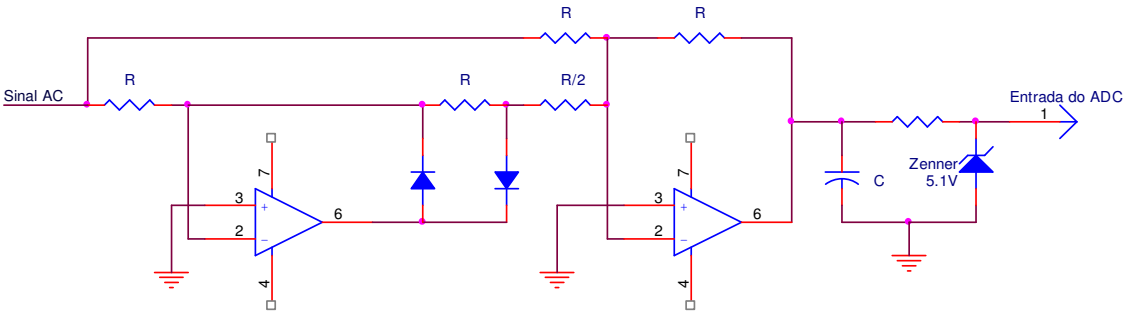


Figura 4.12 – Esquemático do retificador de precisão que converte o sinal AC em DC

Este circuito é capaz de retificar o sinal vindo do transdutor sem as perdas na tensão de 0,7V dos diodos tradicionais, pois este circuito é um retificador de precisão. Na seqüência, o sinal é convertido em DC para simplificar o algoritmo utilizado, que agora não precisará calcular o valor da tensão média que é proporcional à corrente média do motor.

Agora que o *hardware* e *firmware* foram desenvolvidos e testados resta apenas desenvolver uma interface amigável que possibilite a programação e monitoramento remoto de todas as funcionalidades disponíveis no relé digital. No próximo capítulo será visto o desenvolvimento do *software* de controle e monitoramento desenvolvido especificamente para este protótipo. Esta próxima etapa é bastante enriquecedora, pois a partir dela que se alcançará o caráter de automação em tempo real desejada desde o início do trabalho.

Capítulo 5 - Desenvolvimento do *Software*

Este capítulo fará uma explicação de como foi desenvolvido o *software* de controle do protótipo do relé inteligente proposto neste trabalho e mostrará também como este programa é capaz de monitorar e atuar sobre o protótipo. Também será visto aqui todas as dificuldades encontradas nesta etapa e os resultados obtidos nos testes realizados.

A partir de agora será apresentada uma introdução sobre o projeto do *software* e como ele foi implementado seguido por alguns fluxogramas e explicação funcional. Para finalizar o capítulo haverá uma apresentação dos resultados e uma discussão sobre a eficiência do sistema desenvolvido para este trabalho.

5.1 Introdução ao *software* desenvolvido

Este *software* foi desenvolvido visando apenas atender às necessidades de implementação do protótipo, por isto ele ficou bastante simples e sem uma visão de software final, isto é, sem os devidos cuidados que se deve ter com um software comercial.

As características básicas do programa implementado, que foi todo programado em Delphi, é acessar uma das portas serial do computador pessoal e comunicar-se com o protótipo do relé inteligente através do padrão serial RS-232 e, com isto, ser capaz de programar este equipamento de uma maneira bem mais simples do que a programação local, pois o PC conta com vários recursos que o equipamento desenvolvido não possui.

Optou-se que a programação da curva de resposta do relé digital seria programada somente via PC, pois seria muito complicado desenvolver uma interface para esta programação local utilizando-se apenas das 4 teclas disponíveis no protótipo, além ainda da dificuldade que seria de se entrar os dados corretamente com apenas estas teclas.

Com isto o protótipo necessita do PC remoto para sua programação, uma limitação que não é problema devido ao fato de que a idéia inicial deste trabalho é desenvolver um relé inteligente o bastante para possibilitar a monitoração e o controle de um motor através de um único computador central.

5.2 A implementação do programa proposto

A implementação deste *software* precisou ser feita em paralelo com boa parte do desenvolvimento do *firmware* do protótipo, principalmente porque com os recursos disponíveis no PC ficaria mais fácil conferir e depurar os dados gravados na EEPROM do relé digital bem como o seu correto funcionamento.

Contudo, para que a comunicação fosse possível o primeiro grande desafio foi elaborar um protocolo de comunicação que funcionasse e fosse de certa forma seguro quanto a perda de dados ou bytes. O protocolo segue a idéia mostrada no fluxograma da Figura 5.1.

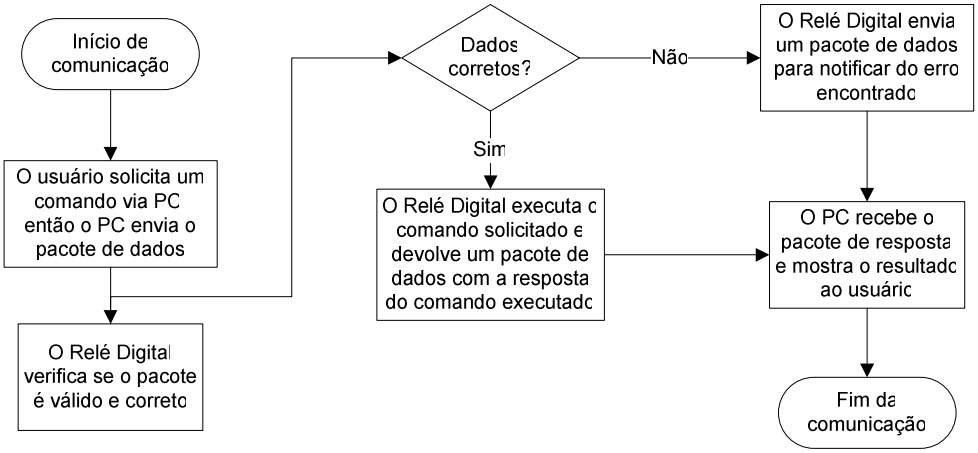


Figura 5.1 – Fluxograma da comunicação serial entre o PC e o Relé

Mas para que a comunicação ficasse segura, garantindo que o PC e o Relé conseguissem "falar" a mesma língua, o pacote de comunicação deveria seguir uma regra, esta regra é mostrada na Tabela 5.1.

Tabela 5.1 – Pacote de dados da comunicação serial

Nome	Descrição	Tamanho
Relé	Número de identificação do Relé	1 Byte
Comando	Comando a ser executado	1 Byte
Erro	Informação do erro ocorrido segundo o Relé	1 Byte
EEAdd	Endereço base da EEPROM	1 Byte
Fases	Valores medidos pelo ADC das 3 fases do motor	3 Bytes
Horário	Informação sobre o horário configurado no Relé (Seg, Min, Horas)	3 Bytes
Reservados	Bytes ainda sem função para possíveis melhoramentos	5 Bytes
EEPROM	Buffer de dados da EEPROM iniciados pelo endereço EEAdd	32 Bytes
Tamanho	Byte de controle com o tamanho do pacote de dados (fixo em 48)	1 Byte

Alguns detalhes devem ser levados em conta para que o pacote seja válido, o byte "Relé" foi previsto para o caso de se desejar plugar o relé digital a uma rede que utilize o padrão RS-485 na comunicação. Assim, com a simples aplicação de um conversor RS-232 para RS-485, o protótipo já poderá se comunicar em uma rede RS-485.

Já o byte de "Comando" foi projetado reservando-se alguns bits, isto é, este byte deve conter os bits 1, 2, 4 e 5 sempre zerados para que o comando seja válido e o bit 0 informa se o motor está ou não ligado naquele momento ou, dependendo do bit 3, se é para ligar ou desligar o motor. Maiores detalhes sobre o funcionamento e valores válidos para cada byte da Tabela 5.1 podem ser vistos nos anexos deste trabalho.

Assim o desenvolvimento do *software* inicialmente foi em função da depuração das funções do equipamento. Foi feita uma tela no programa capaz de mostrar todos os bytes gravados no bloco da EEPROM que se desejava acessar, com isto era possível verificar se os dados configurados no programa foram corretamente gravados no equipamento. O que

possibilitou validar simultaneamente o módulo de comunicação serial e o módulo de escrita e leitura da EEPROM do protótipo.

Com o correto funcionamento da leitura e da escrita na EEPROM e ainda com o correto funcionamento da comunicação serial, partiu-se para o desenvolvimento do módulo capaz de gravar e verificar as configurações principais do relé. Com este módulo pronto, passou a ficar mais fácil a configuração da função de transferência do ADC e também as características do motor a ser monitorado. Agora, utilizando-se desta configuração remota, conseguiu-se depurar e ajustar todo o módulo do conversor analógico digital do equipamento.

Um comando para ligar e desligar o motor foi desenvolvido somente depois que tudo já estava funcionando. Para finalizar o projeto funcional do *software* foi implementada uma tela no programa que possibilita a entrada dos dados da curva de operação do relé inteligente. Esta interface também viabiliza a visualização simultaneamente desta curva, podendo ainda salvá-la no PC e enviá-la para o equipamento.

5.3 Funcionamento e características do programa de monitoração

O programa desenvolvido para o monitoramento remoto do relé inteligente é relativamente pequeno e simples. Ao se executar o programa, ele abre na janela de configurações e traz alguns valores padrões, tais como ID do relé a ser comunicado 255, bloco de memória da EEPROM igual ao bloco 1, corrente nominal do motor igual à 5A entre outros.

A Figura 5.2 mostra a janela principal do programa desenvolvido neste trabalho para o monitoramento do relé inteligente, observe que praticamente todos os dados configuráveis variam de 0 até 255, pois são os valores que cabem em um byte da EEPROM do PIC. Note também que os blocos da memória EEPROM variam de 1 até 8, conforme já foi visto anteriormente. Observe que o bloco 1 está relacionado com a configuração do relé, por

isto o bloco 1 é selecionado automaticamente sempre que a pessoa selecionar a pasta "1.Configurações" do programa.

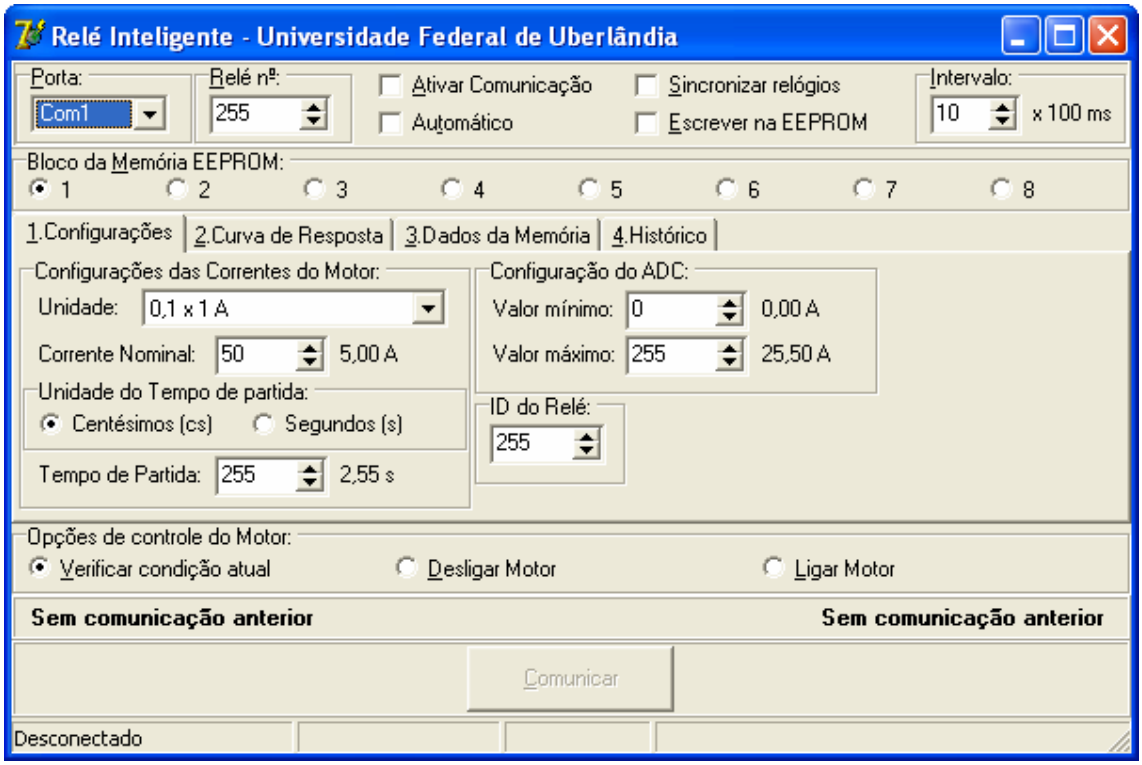


Figura 5.2 – Programa de monitoração do relé inteligente: configurações

A Figura 5.3 já mostra um exemplo de curva que pode ser programada no relé inteligente utilizando este programa, note ainda que o bloco de memória seja automaticamente selecionado para o bloco 2, local onde fica armazenada a curva na memória EEPROM do PIC.

Na coluna "Corrente (A)" é onde se deve programar os valores das correntes enquanto que na mesma linha na coluna do "Tempo (s)" deve-se programar o tempo máximo que a corrente citada nesta linha pode ser fornecida ao motor. Assim um ponto é definido no

gráfico ao lado desta tabela, a interpolação linear de todos os pontos programados na tabela gera uma curva correspondente conforme mostra a própria Figura 5.3.

O número máximo de pontos que se pode configurar para esta curva de resposta do relé é de 16, este valor é suficiente para se montar uma boa curva de resposta se comparada à curva do relé bi metálico, mas com pequenas alterações no *firmware* e no *software* pode-se aumentar o número de pontos desta curva consideravelmente.

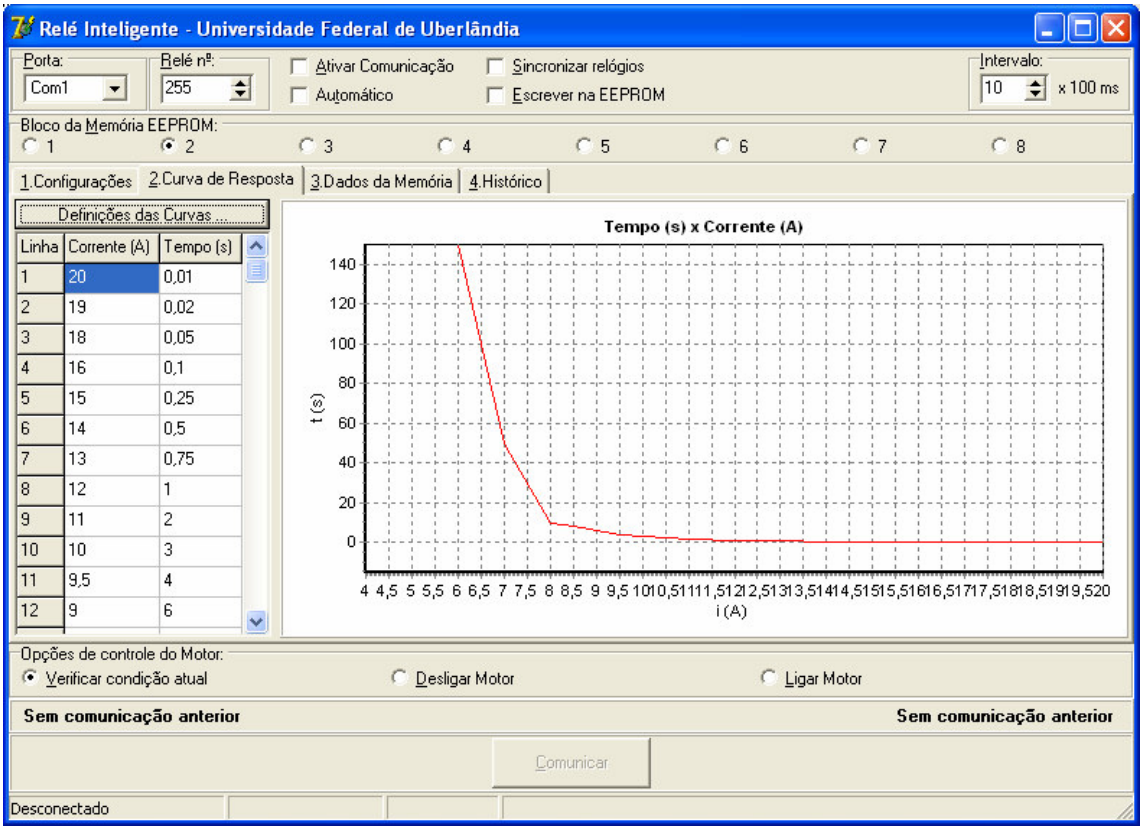


Figura 5.3 – Programa de monitoração do relé inteligente: curva do relé

Estes 16 pontos estão subdivididos em 2 grupos de 8 pontos cada. No primeiro grupo a unidade dos tempos é de centésimos de segundos e por isto pode varia de 0,00s até

2,55s (limite determinado pelo valor máximo de um byte). Já no segundo grupo a unidade dos tempos é de segundos, logo podem variar de 0s até 255s, somente valores inteiro.

Além das limitações nos preenchimentos dos tempos máximos que cada sobre corrente pode passar pelo motor antes do relé inteligente desligar há também uma limitação para os valores das sobre correntes. A sobrecorrente cadastrada na curva de operação não deve ser maior do que a corrente máxima configurada para o ADC, pois assim o relé não será capaz de monitorá-la. O *software* também não permite que se cadastre uma corrente menor do que a corrente nominal do motor, pois senão o motor mal partiria e já seria desligado.

Os valores das sobrecorrentes configuradas na curva podem sofrer pequenas alterações devido à conversão do valor para um valor inteiro compatível no ADC de 8 bits, por isto muitas vezes os valores lidos no protótipo não são os mesmos que foram gravados ou configurados no programa, mas esta diferença é sempre inferior a 1%.

Figura 5.4 – Interface para salvar ou carregar uma curva de resposta

A Figura 5.4 mostra a janela que se abre caso o botão “Definições das Curvas...” seja pressionado. Nesta janela pode-se selecionar uma curva já gravada no PC para enviá-la ao equipamento ou salvar a curva que acabou de ser digitada na pasta “2.Curva de Resposta”. Com esta facilidade consegue-se gerar várias curvas que simulem qualquer relé bimetálico e programar o relé inteligente para obedecer a uma delas.

Um histórico de todas as anomalias ocorridas no relé inteligente durante uma comunicação é gerado na pasta “4.Histórico”, conforme pode ser observado na Figura 5.5.

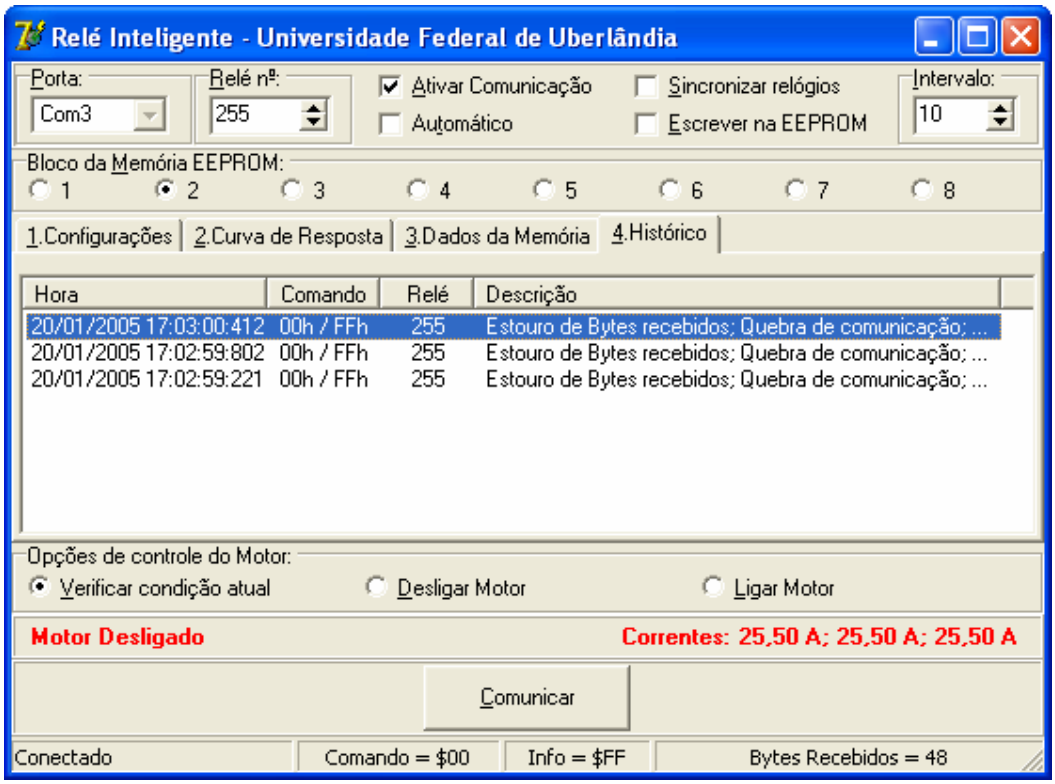


Figura 5.5 – Programa de monitoração do relé inteligente: histórico

Este histórico pode ser apagado sempre que o usuário desejar e informa se houve algum erro de comunicação, erro ao gravar a EEPROM interna ou ainda alerta de sobre corrente. Porém este histórico não foi implementado para ser gerado pelo protótipo do *hardware*, mas sim apenas pelo *software*. Por este motivo o histórico não é descarregado via serial do equipamento, mas sim gerado através do constante monitoramento do PC da situação atual do protótipo.

Observe que no histórico do exemplo utilizado todas as anomalias possíveis foram detectadas pelo equipamento, então para visualizar melhor a descrição dos erros ocorridos pode-se clicar com o botão direito e selecionar a opção "Visualizar toda descrição..." sobre a linha que se deseja verificar ou então dar um duplo clique nesta linha. A Figura 5.6 mostra uma mensagem com toda a descrição para que se possa ler toda a mensagem, observe que neste exemplo todos os possíveis erros de uma comunicação foram apresentados, além do desligamento do motor devido a uma sobre corrente.

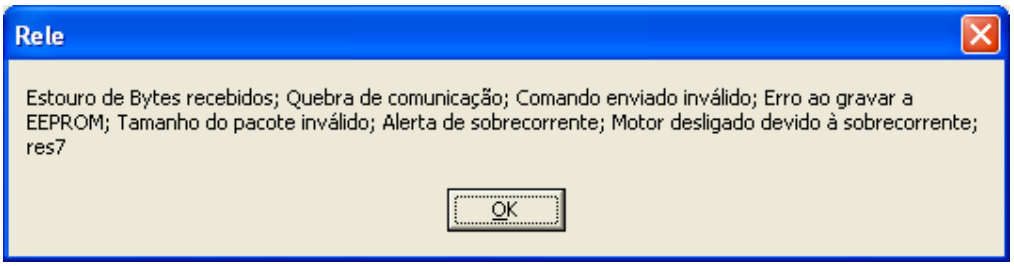


Figura 5.6 – Visualização da descrição das anomalias ocorridas no protótipo

Observando as figuras do programa de monitoração do relé inteligente pode-se notar que algumas das configurações são globais, por exemplo, o bloco de memória, a opção de "sincronizar relógios" de "Escrever na EEPROM" ou até mesmo ligar e desligar o motor.

Estes comandos são globais e podem ser enviados ao equipamento juntamente com a opção descrita na pasta de comandos específicos selecionada.

Caso deseje-se manter o programa monitorando o relé inteligente basta marcar a opção "Automático" e configurar a opção de intervalo de acordo com o computador que está sendo utilizado. Com isto o programa irá ficar comunicando com o relé e levantando um histórico automaticamente de tempos em tempos em função do intervalo selecionado.

Outro detalhe que poder ser ressaltado é o fato de o programa utilizar as cores para chamar a atenção do usuário dependendo do tipo de alerta. Por exemplo, se o motor estiver ligado a cor azul irá aparecer no escrito "Motor Ligado" se estiver apenas desligado a cor é um preto, mas caso tenha sido desligado devido a uma sobre corrente a cor é o vermelho mostrado na Figura 5.5.

5.4 Avaliação e resultados do *software*

O *software* desenvolvido para comunicar-se com o protótipo apresentou um ótimo desempenho, pois ficou pequeno (menor que 1MB) e roda muito bem em qualquer máquina que tenha o sistema operacional Windows instalado.

Nos testes de comunicação percebeu-se que o tempo para gerar um erro de *Time Out* estava pequeno demais para máquinas mais lentas – algo em torno de um segundo. Por este motivo foi necessário aumentar este tempo em até 5 vezes.

A programação da transmissão serial é totalmente compatível com o padrão RS-485, assim caso resolva-se instalar um conversor de RS-232 para RS-485 na saída da porta do PC não será necessário realizar nenhum tipo de alteração no software, mas no protótipo sim já que ele foi montado com o MAX232 ou HIN232. A menos que se instale no protótipo também um conversor RS-232 para RS-485. Um detalhe é que este conversor

deverá possuir o chamado SDC (*Send Data Control*), um circuito capaz de detectar automaticamente se a porta em questão está iniciando uma transmissão de dados ou não.

De uma maneira geral o *software* atendeu a todas as necessidades deste trabalho que, além de possuir uma interface amigável, facilita o aprendizado e o manuseio do equipamento.

Depois de todo estes desenvolvimento partiu-se para os testes práticos visando a aprovação do projeto como um todo, alguns dos resultados encontrados nos testes podem ser verificados no próximo capítulo e discutidos no capítulo final.

Capítulo 6 - Resultados Práticos

Aqui serão vistos os resultados experimentais do protótipo monitorando as correntes de um motor de indução trifásico de tal maneira que as eventuais sobre-correntes deste motor obedeçam às curvas programadas pelo *software* de configuração, o qual foi descrito no capítulo anterior.

Os testes e os ensaios realizados com este protótipo estão voltados ao controle da sobre corrente de acordo com uma curva previamente definida, assim o que foi feito nesta parte do trabalho resumiu-se em apenas configurar algumas curvas e verificar como foi o comportamento do relé inteligente durante os testes de sobre corrente e se o mesmo conseguia ou não desligar o motor conforme a curva nele definida.

6.1 Tempo de partida

A programação do tempo de partida do motor faz com que o relé inteligente permita durante a partida que uma sobre corrente qualquer seja absorvida pelo motor. Graças a esta programação é que o relé inteligente permite que o motor parta, pois se não houvesse esta configuração o relé iria desligar o motor logo que o ele fosse ligado, pois a corrente de partida é muitas vezes a corrente nominal.

O protótipo permite a programação de tempos de partida de 1 centésimo de segundo até 255 segundos, que é o limite de um byte. Esta faixa de valores permite atender à todos os motores de indução, pois dificilmente um motor levará mais do que 10 segundos para entrar no seu regime permanente de funcionamento.

A Figura 6.1 mostra o comportamento do motor utilizado nos testes em laboratório durante sua partida sem carga, note que o tempo de partida nesta condição ficou próximo aos 250 ms.

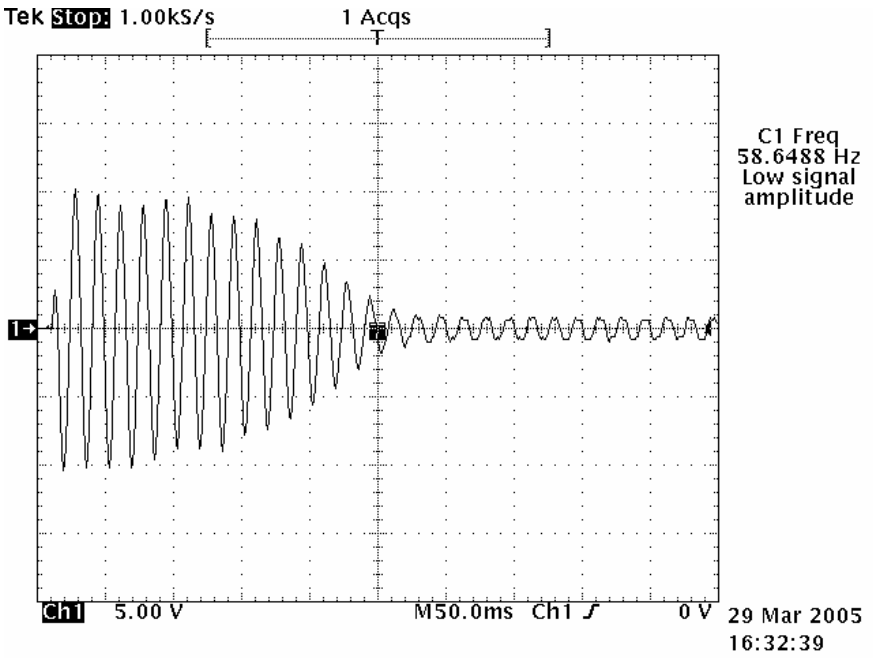


Figura 6.1 – Corrente de partida sem carga

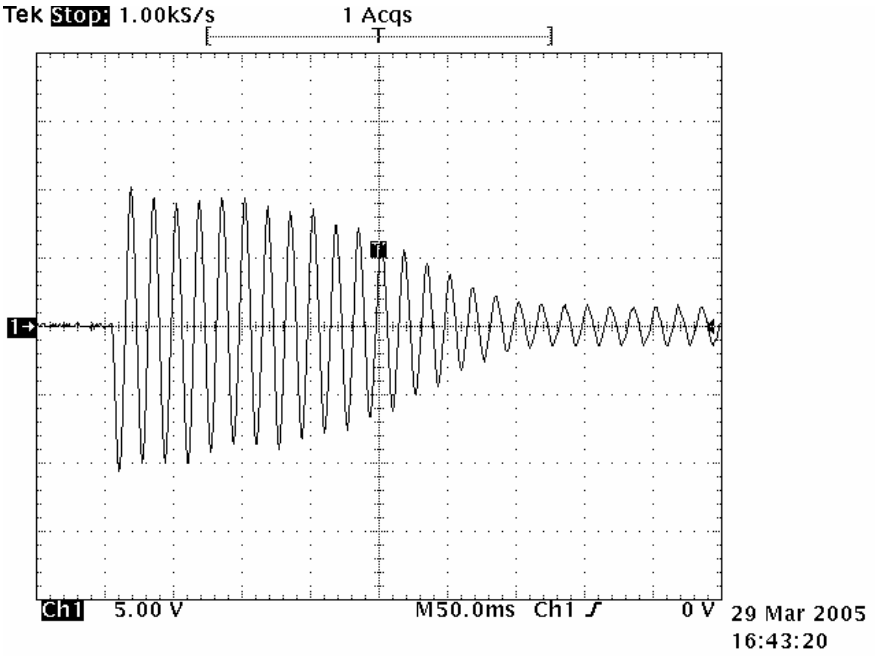


Figura 6.2 – Corrente de partida com carga nominal

O mesmo motor da Figura 6.1 leva agora cerca de 300 ms para partir com carga nominal. É importante notar que o tempo de partida depende também da condição de carga de partida e por isto deve-se levar em conta a carga de partida para se programar o tempo de partida do motor, neste exemplo o tempo de partida aumentou em 20%.

Durante o primeiro teste do equipamento com relação ao tempo de partida o funcionamento não estava correto, então foi necessário realizar algumas alterações no *firmware* do protótipo para que o tempo de partida passasse a ser respeitado. A implementação para o correto funcionamento do relé inteligente foi relativamente simples, bastou colocar uma condição que ignorava a sobre corrente se o tempo de partida ainda não tivesse se esgotado.

6.2 Funcionamento normal

Depois de programado o tempo de partida do motor, iniciaram-se os testes para verificar como o relé se comportava durante o seu funcionamento normal, isto é, deixou-se o motor funcionando com sua corrente nominal por uma hora. Este teste visou verificar se o relé conseguiria monitorar a corrente e manter o motor ligado até que recebesse um comando para desligá-lo. Para tanto foi necessário mantê-lo a uma corrente de até 15% da corrente nominal, porque a curva programada no relé somente iniciava o desligamento do motor a partir deste valor.

Este teste provou que o protótipo funcionou como se esperava enquanto a corrente do motor ficasse abaixo da curva programada, assim o equipamento foi validado por ter conseguido ligar e desligar o motor através de um comando remoto, via PC, ou um comando local, via teclado.

Na Figura 6.3 pode-se verificar um momento de transição entre o funcionamento do motor a vazio e seu funcionamento com carga nominal. Este momento foi utilizado várias vezes para confirmação do funcionamento do relé digital. No primeiro ensaio buscava-se verificar se o relé funcinaria corretamente durante uma transição brusca do valor da corrente do motor. Assim, observou-se que o protótipo comportou-se totalmente imparcial diante desta variação da corrente, exatamente como era esperado.

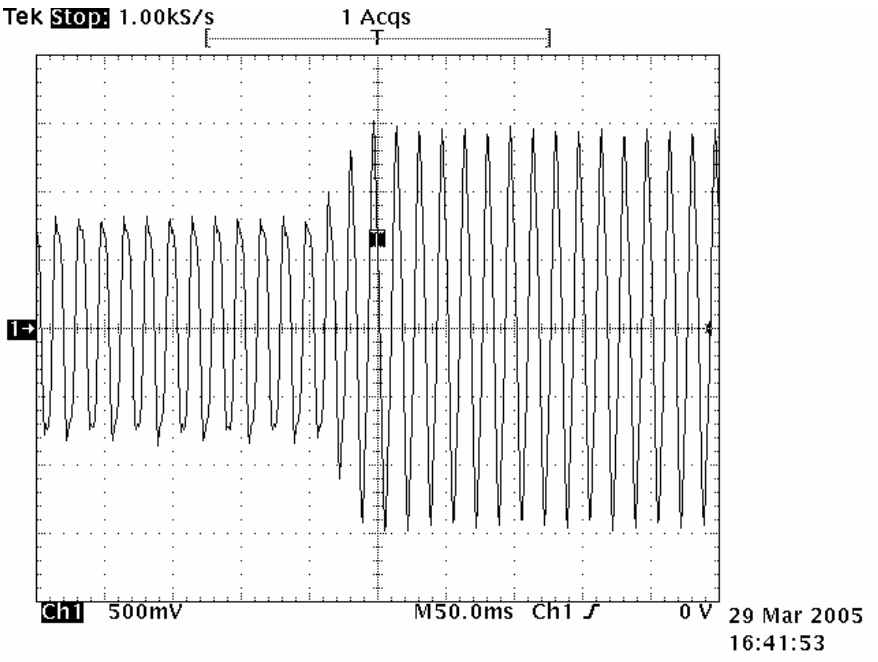


Figura 6.3 – Corrente do motor no momento da aplicação de sua carga nominal

6.3 Proteção contra sobre corrente

Agora, para finalizar os testes deste equipamento, foram programados diferentes correntes como sendo uma sobre corrente que deveria ser desligada em um determinado tempo.

Para facilitar os testes foi programado no relé que a sua corrente nominal era a da sua operação a vazio e a sua corrente nominal como sendo a primeira sobrecorrente que deveria o relé deveria desligar com tempo de programação de 5 segundos. Agora a Figura 6.3 passou a ser “entendida” pelo relé inteligente como momento de transição de sua operação com corrente nominal para uma sobre corrente.

O resultado também foi positivo, isto é, o relé desligava o motor quando se passava os tempos programados para esta sobre corrente, vários tempos diferentes foram experimentados.

Vários tempos de partida diferentes também foram colocados e testados de uma maneira simulada porque existe no laboratório meios para monitorar tempos abaixo de 1 segundo com precisão. Novamente para realizar os testes no laboratório foram feitas algumas programações fora da realidade, visando apenas a validação do equipamento.

Um exemplo do tipo de teste que foi feito seria a programação de um tempo de partida de 2 s, uma corrente igual á corrente à vazio da máquina como sendo uma sobre corrente que deve ser cortada em 3 s e então foi verificado que 5 s após ter se ligado o motor o mesmo foi desligado pelo relé inteligente automaticamente, conforme o esperado.

Após os testes com tempos de partida acima de 1 segundo acreditou-se que o funcionamento em tempos de partida abaixo de 1 segundo também estaria correto, já que a rotina programada no *firmware* é a mesma. O relógio interno do protótipo mostrou-se satisfatório para a contagem de curtos intervalos de tempo.

Com estes testes notou-se que o relé inteligente respeitou o tempo de partida e passou a verificar as sobre correntes somente após este tempo ter se passado, assim o equipamento atendeu a uma das suas principais características: monitorar a sobre corrente quando a máquina estiver em regime permanente e não no momento de sua partida.

Durante todos os ensaios foram utilizados sensores de efeito Hall para o monitoramento das correntes. Como o foco deste trabalho esteve sempre voltado ao desenvolvimento do relé digital o sensor a ser empregado poderia ser qualquer um, pois o relé foi projetado para se adaptar ao sensor como poderá ver observado a seguir no item que se refere à configuração do ADC.

6.4 Configuração do ADC

Como já foi citado anteriormente, o conversor analógico digital interno do PIC foi programado para ser totalmente configurável pelo usuário em uma função de transferência linear qualquer. Assim o equipamento tornou-se extremamente aberto e pronto para atender a todos os casos.

O ADC interno do equipamento foi programado para ler uma tensão já retificada que representasse uma corrente equivalente a que estava passando pelo motor naquele instante. Esta corrente pôde ser monitorada pelo PC remoto sem maiores problemas.

Um erro na medição era visível e era devido às aproximações e arredondamentos dos cálculos envolvidos bem como o fato do conversor analógico digital utilizado ter sido de apenas 8 bits, apesar de o PIC possuir capacidade para um conversor de até 10 bits. Este erro não representou um problema porque ele era inferior a 1,0% do fundo de escala, isto é, do valor máximo configurado para o ADC ler.

O erro citado acima não foi significativo para o controle da sobre corrente do motor já que normalmente a curva de resposta programada no equipamento possuía valores de correntes vizinhos que variavam entre si em aproximadamente 10%.

Apesar da idéia proposta aqui ter deixado o equipamento bastante aberto a diversos motores com diversas amplitudes de correntes, um problema foi gerado por se dar

esta solução de medição. O problema que surge é que um circuito de retificação com precisão e um circuito de proteção para garantir que a tensão de entrada na porta do canal do ADC do PIC não ultrapassasse o valor máximo de 5V teve de ser implementado.

Outro problema, mas não muito crítico é que o tempo de resposta do relé inteligente agora está limitado ao tempo de resposta da tensão retificada que indica o valor da corrente medida. Como esta tensão teve que ser retificada, um capacitor foi colocado no circuito e este capacitor atrasa a resposta da tensão. Se um capacitor de capacitância muito elevada for utilizado haverá a vantagem de se conseguir mais estabilidade no valor medido da corrente, mas em contrapartida o tempo de resposta do relé inteligente será menor porque a corrente medida sempre será uma corrente de alguns instantes atrás.

Estes dois problemas não afetaram os testes do protótipo porque o equipamento foi desenvolvido para executar respostas a partir de 1 centésimo de segundo. Como o período de um ciclo da rede elétrica leva aproximadamente 16,7 milésimos de segundos ou 1,7 centésimos de segundos, o protótipo deverá monitorar pelo menos um período completo da forma de onda da corrente elétrica medida antes de se tomar uma decisão, o que é bastante aceitável. Devido aos atrasos no próprio circuito eletrônico desenvolvido, aconselha-se utilizar tempos de respostas acima dos 20 centésimos de segundos, o que já é bastante satisfatório para este equipamento.

6.5 Curva de resposta do relé inteligente

Qualquer curva de resposta pode ser programada pelo usuário a qualquer momento mesmo com o motor ligado, graças a esta facilidade este protótipo poderá atender a diversos motores de indução existentes no mercado com a facilidade de se trocar o motor de indução sem que se precise trocar a proteção, mas sim apenas a sua programação.

Algumas curvas de resposta foram programadas e testadas no equipamento para verificar se o protótipo era capaz de desligar o motor caso o tempo de sobre corrente medido por ele fosse maior que o tempo nele configurado, porém todas as situações testadas aqui foram simulações e não o funcionamento real em campo, mas foi suficiente para validar o relé inteligente.

Um exemplo das curvas programadas no equipamento nesta fase de testes foi a curva da Figura 6.4. Observe que para correntes acima de 10A, o que representa 200% da corrente nominal, foi programado para o relé inteligente desligar em 3 segundos. Os valores de corrente acima de 200% praticamente têm os tempos de resposta próximos, pois não há como abaixar muito mais do que 0,5s o tempo de resposta devido ao que foi citado anteriormente.

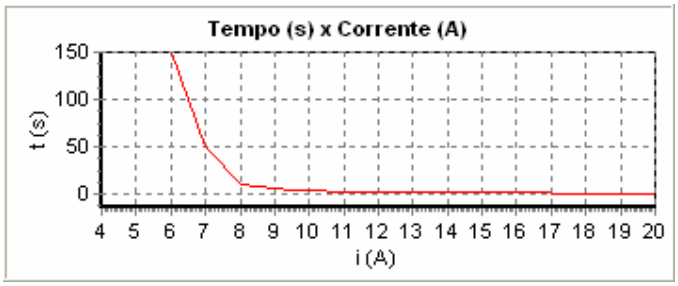


Figura 6.4 – Curva de resposta contra sobre corrente programada no protótipo

Neste exemplo o comportamento do relé inteligente foi bastante satisfatório, pois ele conseguiu desligar o equipamento com precisão para todos os tempos pedidos acima de 1 segundo. Para tempos abaixo deste valor já não foi possível avaliá-lo, pois´era impossível cronometrar o tempo de resposta do relé digital.

Pode-se dizer que o relé inteligente desenvolvido neste trabalho pode ser capaz de simular as diversas curvas dos relés bimetálicos existentes no mercado atual sem maiores problemas, no capítulo seguinte será feita uma análise destes resultados e propostos alguns trabalhos futuros baseados nesta dissertação.

Capítulo 7 - Considerações Finais

Pode-se notar que este trabalho abrange diversas áreas da engenharia elétrica, das quais vale citar: eletrônica básica; eletrônica de potência; eletrônica digital; máquinas elétricas; microcomputadores e microcontroladores; sistemas de tempo real.

O curioso é que este trabalho iniciou-se na linha de pesquisa de potência e máquinas elétricas, mas como o relé inteligente não está preocupado em controlar a partida do motor e muito menos com o controle da velocidade da máquina, acabou-se investindo mais tempo no estudo de microcontroladores e microcomputadores do que no próprio motor elétrico.

Além de ter sido um trabalho bastante completo do ponto de vista da engenharia elétrica, ele também possibilitou unir as diversas áreas citadas acima. Fato este que provou que a integração interdisciplinar das matérias estudadas em um curso de engenharia pode e deve ser mais explorada em seus cursos, pois se não fosse por esta integração entre as disciplinas este trabalho não teria sido desenvolvido.

7.1 Conclusões

O trabalho conseguiu atingir todos os objetivos iniciais e mostrou um nicho de mercado pouco explorado pela indústria brasileira. Este nicho precisaria receber mais atenção e mais investimentos visando diminuir os custos destes equipamentos, normalmente importados, utilizados na indústria nacional.

O custo do protótipo, desconsiderando custo do projeto de engenharia envolvido, não ficou muito alto comparado com o que existe no mercado para a proteção de motores contra a sobre corrente. Este seria outro fator que poderia atrair a atenção de alguns

empresários nacionais para que iniciassem um trabalho para desenvolver um produto nacional.

Outro ponto favorável ao desenvolvimento de um produto comercial com base neste protótipo seria o fato de que não existem muitos equipamentos nacionais com as mesmas características e mesmo custo deste protótipo.

7.2 Trabalhos Futuros

Vários trabalhos futuros podem ser desenvolvidos a partir desta idéia, principalmente trabalhos que necessitem de um processador local para desenvolver alguma função como controle de partida ou de velocidade, pois este protótipo possui um microcontrolador de grande capacidade com muita memória disponível.

Pode-se aproveitar a referência [3] para complementar este trabalho com um ótimo controle de partida para os motores de indução, isto pode ser feito utilizando o próprio PIC deste trabalho sem que seja necessário realizar grandes mudanças no *hardware* ou *firmware*.

Pode-se ainda utilizar os outros 8 pinos de I/O que sobraram para controlar vários motores em paralelo, visto que este microcontrolador trabalha em 200ns de ciclo de instrução e o seu ADC interno consegue realizar uma conversão em menos de 100µs. Para tanto bastaria incluir um circuito integrado multiplexador na entrada dos pinos do ADC, por exemplo, utilizando-se 1 pino de I/O no multiplexador já seria possível saltar de 3 para 6 canais analógicos e ainda sobrariam 7 pinos, dos quais apenas um seria utilizado para acionar um outro motor.

Outra sugestão seria reduzir o custo deste trabalho retirando o display de cristal líquido, pois a interface com o PC é muito mais simples e pode ser utilizada em seu lugar,

assim constrói-se um equipamento mais barato (menos da metade do preço de um com LCD) com mesmo potencial.

Diversos trabalhos podem ser originados a partir deste estudo, cada variação na aplicação ou novidade geraria um trabalho novo, utilizar uma rede CAN ou Ethernet (TCP/IP) no lugar da serial, uma rede USB ou ainda uma conexão bluetooth para manter o equipamento compatível com as novas tecnologias.

Referências Bibliográficas

[1] Kosow, Irving I. – Máquinas Elétricas e Transformadores. 2ª. Edição, 1979. Editora Globo. Porto Alegre, RS.

[2] Lima, Marco Rogério Calheira. Desenvolvimento de um Sistema Eletrônico para Partida de Motores de Indução Trifásicos com Controle Programável da Amplitude da Corrente de Partida. 1998. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal de Uberlândia.

[3] Guia EM da NBR 5410. Capítulo 5 – Proteção contra sobre correntes e Capítulo 7 – Circuito de Motores. São Paulo, dezembro de 2001.

[4] Millman, Jacob e Halkias, Christos C. Eletrônica – Volume 1. 2ª. Edição, 1981. Editora MacGraw-Hill. São Paulo, SP.

[5] Millman, Jacob e Halkias, Christos C. Eletrônica – Volume 2. 2ª. Edição, 1981. Editora MacGraw-Hill. São Paulo, SP.

[6] Tocci, Ronald J. e Widmer, Neal S. Sistemas Digitais – Princípios e Aplicações. 8ª. Edição, 2003. Editora Pearson Education do Brasil. São Paulo, SP.

[7] Hayt Jr., William H. e Kemmerly, Jack E. Análise de Circuitos em Engenharia. 1ª. Edição, 1971. Editora McGraw-Hill. São Paulo, SP.

[8] Gimenez, Salvador R. – Microcontroladores 8051. 1ª. Edição, 2002. Editora Pearson Education do Brasil. São Paulo, SP.

[9] Silva Júnior, Vidal Pereira da – Aplicações Práticas do Microcontrolador 8051. 5ª. Edição, 1996. Editora Érica Ltda. São Paulo, SP.

[10] DATASHEET: PIC16F62X datasheet.

[11] DATASHEET: PIC16F87XA datasheet.

[12] DATASHEET: *Interfacing PICmicros to an LCD Module*.

[13] DATASHEET: Display LCD.

[14] DATASHEET: DS485.

[15] DATASHEET: HIN232.

[16] ARTIGO: Fernandes, José Manuel e Paula, Leonardo Costa – *On-line Control Systems Using RS-485*. ISA 2001, Houston, Texas

[17] ARTIGO: Mozina, C; Young, M; Bailey, B; Baker, B; Dalke, G – Commissioning and maintenance testing of multifuncition digital relays. Plup and Paper Industry Technical Conference, 2004.

[18] SITE: Site oficial da Siemens no Brasil (www.siemens.com.br)

Bibliográficas recomendadas

[19] Toro, Vincent Del – Fundamentos de Máquinas Elétricas, LTC Livros Técnicos e Científicos Editora SA, Rio de Janeiro, RJ.

[20] Halliday, David e Resnick, Robert. Fundamentos de Física 3 – Eletromagnetismo. 2ª. Edição, 1994. Editora LTC – Livros Técnicos e Científicos. Rio de Janeiro, RJ.

[21] Silva, Sérgio Batista da. Contribuição ao uso de microcontroladores na proteção de circuitos em baixa tensão. 2003. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal de Uberlândia.

Anexo A Código Fonte do Firmware

```
;*************************************************************************************************
; PROJETO DE UM RELÉ INTELIGENTE
; CARACTERÍSTICAS:
;       ESTE PROTÓTIPO POSSUI UM DISPLAY LCD 16x2, 4 BOTÕES PARA UMA PROGRAMAÇÃO LOCAL, 3 ENTRADAS
; ANALÓGICAS PARA LEITURA DAS CORRENTES DE UM MOTOR DE INDUÇÃO TRIFÁSICO, 1 SAÍDA TTL PARA
; LIGAR OU DESLIGAR O MESMO MOTOR, 1 PORTA SERIAL RS232 PARA PROGRAMAÇÃO REMOTA.
; ELABORADO POR:
;       ENG. LEONARDO COSTA DE PAULA
; ORIENTADO POR:
;       DR. DARIZON ALVES DE ANDRADE - UFU
; INICIADO EM:   29/07/03
; Arquivo: Main.asm
;*************************************************************************************************

        PROCESSOR       16F876A
;       RADIX           HEX

; Configura o bit de controle

;       __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC

#INCLUDE <P16F876A.INC>
#INCLUDE <D:\PROJETOS\PIC\UTILS\MACROS.INC>
#INCLUDE <MAIN.INC>

        ORG                     0X00
;       GOTO            R_INICIO_TST
        GOTO            R_INICIO


;=================================================================================================
; Tratamento das Interrupções
;=================================================================================================

        ORG                     0X04
        GOTO            R_INTERRUPCOES

;=================================================================================================
; ATENÇÃO!!! ÁREA RESERVADA PARA AS MENSAGENS (POIS O ENDEREÇO MÁXIMO É 255!!!)

MSG_INICIO
        RETLW   D'5'    ; TAMANHO DA MENSAGEM
        RETLW   'R'
        RETLW   'E'
        RETLW   'L'
        RETLW   'E'
        RETLW   ' '

MSG_MOTOR
        RETLW   D'6'
        RETLW   'L'
        RETLW   'I'
        RETLW   'G'
        RETLW   'A'
        RETLW   'R'
        RETLW   '?'

MSG_CONFIRMA
        RETLW   D'15'   ; TAMANHO DA MENSAGEM
        RETLW   'O'
        RETLW   'K'
        RETLW   ' '
        RETLW   'P'
        RETLW   '/'
        RETLW   ' '
        RETLW   'C'
        RETLW   'O'
        RETLW   'N'
        RETLW   'F'
        RETLW   'I'
        RETLW   'R'
        RETLW   'M'
        RETLW   'A'
```

```
        RETLW   'R'

MSG_ID
        RETLW   D'7'    ; TAMANHO DA MENSAGEM
        RETLW   'R'
        RETLW   'E'
        RETLW   'L'
        RETLW   'E'
        RETLW   ' '
        RETLW   'I'
        RETLW   'D'

MSG_DELTA
        RETLW   D'9'    ; TAMANHO DA MENSAGEM
        RETLW   'A'
        RETLW   'M'
        RETLW   'P'
        RETLW   'L'
        RETLW   'I'
        RETLW   'T'
        RETLW   'U'
        RETLW   'D'
        RETLW   'E'

MSG_MENOR
        RETLW   D'8'    ; TAMANHO DA MENSAGEM
        RETLW   'I'
        RETLW   ' '
        RETLW   'M'
        RETLW   'I'
        RETLW   'N'
        RETLW   'I'
        RETLW   'M'
        RETLW   'O'

MSG_NOMINAL
        RETLW   D'9'    ; TAMANHO DA MENSAGEM
        RETLW   'I'
        RETLW   ' '
        RETLW   'N'
        RETLW   'O'
        RETLW   'M'
        RETLW   'I'
        RETLW   'N'
        RETLW   'A'
        RETLW   'L'

MSG_PARTIDA
        RETLW   D'7'    ; TAMANHO DA MENSAGEM
        RETLW   'P'
        RETLW   'A'
        RETLW   'R'
        RETLW   'T'
        RETLW   'I'
        RETLW   'D'
        RETLW   'A'

MSG_UNIDADE
        RETLW   D'14'   ; TAMANHO DA MENSAGEM
        RETLW   'T'
        RETLW   'E'
        RETLW   'M'
        RETLW   'P'
        RETLW   'O'
        RETLW   ' '
        RETLW   '1'
        RETLW   '='
        RETLW   'c'
        RETLW   's'
        RETLW   '/'
        RETLW   '2'
        RETLW   '='
        RETLW   's'

MSG_I_UNIDADE
        RETLW   D'12'   ; TAMANHO DA MENSAGEM
        RETLW   'I'
        RETLW   ' '
        RETLW   'x'
        RETLW   ' '
        RETLW   '1'
        RETLW   '/'
        RETLW   '1'
```

```
        RETLW   '0'
        RETLW   '/'
        RETLW   '1'
        RETLW   '0'
        RETLW   '0'

;*****************************************************************************************************
; AS INTERRUPÇÕES VIERAM PARA CÁ PARA SOBRAR MAIS ESPAÇO PARA AS MENSAGENS (STACK 1)
R_INTERRUPCOES
        MOVWF           V_W_TEMP                ; Salva W e STATUS
        MOVF            STATUS, W
        MOVWF           V_S_TEMP
        MOVF            FSR, W
        MOVWF           V_FSR_TEMP
        CLRF            STATUS                  ; VAI PARA O BANK 0

        BTFSC           PIR1, TMR2IF
        GOTO            INT_TMR2        ; SE 1 É PORQUE OCORREU UMA INT DO TIMER2

INT_SER
        GOTO            R_SERIAL_RX             ; DENTRO DESTA ROTINA HÁ UM "GOTO INT_FIM"

INT_TMR2
        GOTO            R_TIMER2        ; DENTRO DESTA ROTINA HÁ UM "GOTO INT_FIM"

INT_FIM
        MOVF            V_FSR_TEMP, W
        MOVWF           FSR
        MOVF            V_S_TEMP, W
        MOVWF           STATUS
        SWAPF           V_W_TEMP, F
        SWAPF           V_W_TEMP, W             ; RECUPERA W SEM ALTERAR O STATUS (MOVF ALTERA O STATUS!)
        RETFIE

;*****************************************************************************************************
; ROTINA UTILIZADA PARA TESTES GERAIS DO PIC (DEBUG)
R_INICIO_TST
        CLRF            STATUS
        MOVLW           K_PORTA
        MOVWF           PORTA
        MOVLW           K_PORTB
        MOVWF           PORTB
        MOVLW           K_PORTC
        MOVWF           PORTC
;       MOVLW           K_INTCON
;       MOVWF           INTCON
;       MOVLW           K_ADCON0
;       MOVWF           ADCON0
;       MOVLW           K_T2CON         ; BANK 0
;       MOVWF           T2CON

        BSF                     STATUS, RP0             ; SELECIONA BANK 1
;       MOVLW           K_PR2
;       MOVWF           PR2
;       MOVLW           K_PIE1
;       MOVWF           PIE1
        MOVLW           K_TRISA                 ; K_TRISA
        MOVWF           TRISA
        MOVLW           K_TRISB                 ; K_TRISB
        MOVWF           TRISB
        MOVLW           K_TRISC                 ; K_TRISC
        MOVWF           TRISC
;       MOVLW           K_OPTION
;       MOVWF           OPTION_REG
;       MOVLW           K_ADCON1
;       MOVWF           ADCON1

        CLRF            STATUS

R_TST_LOOP
        MOVLW           0X00
        BTFSC           PORTB, 7
        GOTO            R_TST_SET
        CLRF            PORTB
        GOTO            R_TST_LOOP

R_TST_SET
        MOVLW           0XFF
        MOVWF           PORTB
        GOTO            R_TST_LOOP


;*****************************************************************************************************
```

```
; ROTINAS GERAIS PARA ACESSO AOS PERIFÉRICOS OU ROTINAS MATEMÁTICAS
;=================================================================================================

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE MULTIPLICA UM BYTE POR OUTRO
;       ENTRADA: V_NUM1 E V_NUM2 = BYTES A SEREM MULTIPLICADOS
;       SAÍDA: V_NUM4,V_NUM3 = WORD RESPOSTA (V_NUM4 É O MAIS SIGNIFICATIVO)
;       AFETA: W; STATUS; V_NUM1; V_NUM2; V_NUM3; V_NUM4; V_NUM5; V_NUM6
;       CHAMA: NADA (STACK: 1)
;-------------------------------------------------------------------------------------------------

R_MUL8
        CLRF            V_NUM3
        CLRF            V_NUM4
        CLRF            V_NUM5                  ; INICIA VARIÁVEIS LOCAIS
        MOVLW           0X08
        MOVWF           V_NUM6                  ; INICIA CONTADOR

R_MUL_LOOP
        RRF                     V_NUM1, F
        BTFSS           STATUS, C
        GOTO            R_MUL_PULA              ; COMO ERA UM BIT ZERO, NÃO PRECISA SOMAR!
        MOVF            V_NUM2, W
        ADDWF           V_NUM3, F               ; SOMA PARTE BAIXA
        BTFSS           STATUS, C
        GOTO            R_MUL_CONT              ; NÃO PRECISOU AJUSTAR PARTE ALTA
        INCF            V_NUM4, F

R_MUL_CONT
        MOVF            V_NUM5, W
        ADDWF           V_NUM4, F               ; SOMA PARTE ALTA

R_MUL_PULA
        RLF                     V_NUM2, F               ; SEMPRE CHEGA AQUI COM C = 0!
        RLF                     V_NUM5, F               ; MULTIPLICA POR 2 (PREPARA PARA TESTAR PRÓXIMO
BIT!)

        DECFSZ          V_NUM6
        GOTO            R_MUL_LOOP              ; ATÉ PASSAR POR TODOS OS BITS
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE DIVIDE UMA WORD POR UM BYTE
;       ENTRADA: V_NUM,V_NUM1 = DIVIDENDO; V_NUM2 = DIVISOR
;       SAÍDA: V_NUM5,V_NUM4 = QUOCIENTE; V_NUM3 = RESTO
;       AFETA: W; STATUS; V_NUM; V_NUM1; V_NUM3; V_NUM4; V_NUM5; V_NUM6
;       CHAMA: NADA (STACK: 1)
;-------------------------------------------------------------------------------------------------

R_DIV8
        CLRF            V_NUM3
        CLRF            V_NUM4
        CLRF            V_NUM5                  ; INICIA VARIÁVEIS LOCAIS
        MOVLW           0X10
        MOVWF           V_NUM6                  ; INICIA CONTADOR

R_DIV_LOOP
        RLF                     V_NUM1, F
        RLF                     V_NUM, F
        RLF                     V_NUM3, F               ; V_NUM3 <- BIT MAIS SIGNIFICATIVO
        BTFSC           STATUS, C
        GOTO            R_DIV_CY         ; SE DEU CARRY, SUBTRAI E GRAVA RESTO!

        MOVF            V_NUM2, W
        SUBWF           V_NUM3, W
        BTFSS           STATUS, C
        GOTO            R_DIV_NEG               ; RESULTADO DA SUBTRAÇÃO FOI NEGATIVO!
        GOTO            R_DIV_POS

R_DIV_CY
        MOVF            V_NUM2, W
        SUBWF           V_NUM3, W
        BSF                     STATUS, C               ; GARANTE INCREMENTO DO QUOCIENTE

R_DIV_POS
        MOVWF           V_NUM3                  ; GUARDA RESULTADO, POIS A SUBTRAÇÃO FOI POSITIVA

R_DIV_NEG
        RLF                     V_NUM4                  ; AJUSTA O QUOCIENTE
        RLF                     V_NUM5                  ; AJUSTA O QUOCIENTE
        DECFSZ          V_NUM6
        GOTO            R_DIV_LOOP              ; ATÉ PASSAR POR TODOS OS BITS
        RETURN
```

```
;-------------------------------------------------------------------------------------------------
;       ROTINA QUE CONVERTE UM NUMERO DE BINÁRIO PARA BCD
;       ENTRADA: W = NÚMERO EM BINÁRIO DE 0 ATÉ 99
;       SAÍDA: W = NÚMERO EM BCD
;       AFETA: W; STATUS; V_NUM; V_NUM1; V_NUM2; *V_NUM3; *V_NUM4; *V_NUM5; *V_NUM6
;       CHAMA: R_DIV8 (STACK: 2)
;-------------------------------------------------------------------------------------------------

R_BIN_TO_BCD
        MOVWF           V_NUM1
        CLRF            V_NUM                   ; PARTE ALTA DA WORD É NULA
        M_MOVLF         V_NUM2, D'10'
        CALL            R_DIV8
        SWAPF           V_NUM4, W
        IORWF           V_NUM3, W
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE CONVERTE UM NUMERO BCD PARA BINÁRIO
;       ENTRADA: W = NÚMERO EM BCD
;       SAÍDA: W = NÚMERO EM BINÁRIO
;       AFETA: W; STATUS; V_NUM1; V_NUM2; V_NUM3
;       CHAMA: NADA (STACK: 1)
;-------------------------------------------------------------------------------------------------

R_BCD_TO_BIN
        MOVWF           V_NUM1          ; SALVA BYTE A SER CONVERTIDO
        ANDLW           0X0F
        MOVWF           V_NUM3          ; SALVA PARTE BAIXA DO BCD
        SWAPF           V_NUM1, W
        ANDLW           0X0F
        MOVWF           V_NUM1          ; SALVA PARTE ALTA DO BCD
        BCF                     STATUS, C
        RLF                     V_NUM1, F       ; MULTIPLICA PARTE ALTA POR 2
        RLF                     V_NUM1, W       ; MULTIPLICA PARTE ALTA POR 4 E COLOCA EM W
        MOVWF           V_NUM2
        RLF                     V_NUM2, W       ; MULTIPLICA A PARTE ALTA POR 8 E COLOCA EM W
        ADDWF           V_NUM1, W       ; SOMA PARTE ALTA X 8 COM PARTE ALTA X 2 = PARTE ALTA X (8 + 2)
        ADDWF           V_NUM3, W       ; W = 10x(PARTE ALTA) + PARTE BAIXA = BINÁRIO DO BCD
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE LÊ VALOR DO ADC
;       ENTRADA: W = CANAL SELECIONADO (0..4)
;       SAÍDA: W = ADRESH = BYTE COM OS 8 BITS MAIS SIGNIFICATIVO DO VALOR CONVERTIDO
;       AFETA: ADRESH; ADRESL; V_W_AUX; W
;       CHAMA: NADA
;-------------------------------------------------------------------------------------------------

R_ADC_READ
        MOVWF           V_W_AUX
        MOVLW           B'11000111'
        ANDWF           ADCON0, F               ; ZERA O CANAL

        BCF                     STATUS, C               ; PARA FAZER A ROTAÇÃO
        BTFSC           V_W_AUX, 1              ; VERIFICA SE É CANAL 2 (FASE 3)
        RLF                     V_W_AUX, F              ; SE FOR, APONTA PARA O CANAL 4, POIS O 2 É
REFERÊNCIA!

        RLF                     V_W_AUX, F
        RLF                     V_W_AUX, F
        RLF                     V_W_AUX, F              ; POSICIONA O BIT ZERO NA TERCEIRA POSIÇÃO!

        MOVF            V_W_AUX, W
        IORWF           ADCON0, F               ; SELECIONA O CANAL A SER LIDO

        MOVLW           0X01
        CALL            R_DELAY         ; PEQUENO ATRASO PARA ESTABILIZAR O SINAL DO NOVO CANAL

;       BCF                     INTCON, GIE             ; DESABILITA INTS

        BSF                     ADCON0, GO      ; INICIA CONVERSÃO
        BTFSC           ADCON0, GO
        GOTO            $-1                     ; ESPERA PELO FINAL DA CONVERSÃO

        MOVF            ADRESH, W

;       BSF                     INTCON, GIE             ; HABILITA INTS
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE LÊ A EEPROM INTERNA
```

```
;       ENTRADA: EEADR = ENDEREÇO A SER LIDO
;       SAÍDA: W = BYTE LIDO
;       AFETA: W; STATUS; EECON1; EEDATA; EEADR
;       CHAMA: NADA (STACK: 1)
;-------------------------------------------------------------------------------------------------

R_EE_READ
        BSF                     STATUS, RP0
        BSF                     STATUS, RP1             ; APONTA PARA VARIÁVEIS NO BANK 3

        BCF                     EECON1, EEPGD   ; APONTA PARA MEMORIA DE DADOS (EEPROM)
        BSF                     EECON1, RD              ; INICIA LEITURA

        BCF                     STATUS, RP0             ; RETORNA PARA O BANCO 2
        MOVF            EEDATA, W               ; SALVA BYTE LIDO (8 BITS MENOS SIGNIFICATIVO)
        BCF                     STATUS, RP1             ; RETORNA PARA O BANCO 0

        BCF                     STATUS, RP0
        BCF                     STATUS, RP1             ; APONTA PARA VARIÁVEIS NO BANK 0
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE ESCREVE NA EEPROM INTERNA E VERIFICA SE A ESCRITA FOI CORRETA
;       ENTRADA: EEADR = ENDEREÇO A SER ESCRITO, EEDATA = BYTE A SER ESCRITO
;       SAÍDA: Z = 0 SE OK
;       AFETA: W; STATUS; EECON1; EECON2
;       CHAMA: NADA (STACK: 1)
;-------------------------------------------------------------------------------------------------

R_EE_WRITE
        BSF                     STATUS, RP0
        BSF                     STATUS, RP1                     ; APONTA PARA VARIÁVEIS NO BANK 3

        BCF                     EECON1, EEPGD   ; APONTA PARA MEMORIA DE DADOS (EEPROM)
        BSF                     EECON1, WREN    ; HABILITA A ESCRITA
        BCF                     INTCON, GIE             ; DESABILITA INTERRUPÇÕES

        M_MOVLF         EECON2, 0x55
        M_MOVLF         EECON2, 0xAA    ; PREPARA P/ A ESCRITA
        BSF                     EECON1, WR              ; INICIA ESCRITA

        BSF                     INTCON, GIE             ; HABILITA INTERRUPÇÕES
        BCF                     EECON1, WREN    ; DESABILITA A ESCRITA

        BTFSC           EECON1, WR
        GOTO            $-1                             ;   ESPERA   PELA   CONCLUSÃO   DA   ESCRITA   P/
VERIFICAÇÃO

        MOVF            EEDATA, W               ; GUARDA BYTE ESCRITO PARA VERIFICAÇÃO
        INCF            EEDATA, F               ; GARANTE QUE OS DOIS SEJAM DIFERENTES
        BSF                     EECON1, RD              ; INICIA LEITURA
        XORWF           EEDATA, W               ; TESTA SE BYTE LIDO = ESCRITO

        BCF                     STATUS, RP0
        BCF                     STATUS, RP1             ; APONTA PARA VARIÁVEIS NO BANK 0
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE LÊ A MEMÓRIA FLASH INTERNA
;       ENTRADA: EEADR, EEADRH = ENDEREÇO A SER LIDO (BANK = 2 SELECIONADO)
;       SAÍDA: W = BYTE LIDO = EEDATA (BANK = 2), EEDATH = 6 BITS RESTANTES (BANK = 2)
;       AFETA: W; STATUS; EECON1; EEDATA; EEDATH
;       CHAMA: NADA (STACK 1)
;-------------------------------------------------------------------------------------------------

R_FLASH_READ
        BSF                     STATUS, RP0             ; MUDA O BANCO 3

        BSF                     EECON1, EEPGD   ; APONTA PARA MEMORIA DE PROGRAMA (FLASH)
        BSF                     EECON1, RD              ; INICIA LEITURA
        NOP
        NOP                                                     ; É NECESSÁRIO O ATRASO DE 2 NOPs

        BCF                     STATUS, RP0             ; RETORNA PARA O BANCO 2
        MOVF            EEDATA, W               ; SALVA BYTE LIDO (8 BITS MENOS SIGNIFICATIVO)
        BCF                     STATUS, RP1             ; RETORNA PARA O BANCO 0

        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA LIGA O MOTOR E PREPARA VARIÁVEIS
;       ENTRADA: NENHUMA
;       SAÍDAS: V_FAIXA; V_PARTIDA; V_SER_FLAG SÃO ATUALIZADOS
```

```
;       AFETA: W; STATUS
;       CHAMA: R_EE_READ (STACK: 2)
;-------------------------------------------------------------------------------------------------

R_LIGA_MOTOR
        BTFSC           P_MOTOR
        GOTO            R_L_MOTOR_FIM           ; IGNORA COMANDO PORQUE O MOTOR JÁ ESTÁ LIGADO!
        CLRF            V_SER_FLAG                      ; DESLIGA O ALARME ATUAL
        M_EE_READ       E_PARTIDA
        MOVWF           V_PARTIDA                       ; INICIA CONTADOR DO TEMPO DE PARTIDA
        M_EE_READ       E_UNIDADE
        MOVWF           V_FAIXA                         ; PEGA A UNIDADE DA PARTIDA
        BSF                     P_MOTOR
R_L_MOTOR_FIM
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE TRATA A INTERRUPÇÃO DO TIMER2
;       ENTRADA: V_1MS; V_10MS; V_SEG; V_MIN; V_HOR; V_DIA; V_SEM
;       SAÍDAS: V_1MS; V_10MS; V_SEG; V_MIN; V_HOR; V_DIA; V_SEM SÃO ATUALIZADAS
;       AFETA: W; STATUS; V_1MS; V_10MS; V_SEG; V_MIN; V_HOR; V_DIA; V_SEM
;       CHAMA: NADA (STACK: 1)
;-------------------------------------------------------------------------------------------------

R_TIMER2
        BCF                     PIR1, TMR2IF    ; PREPARA PARA NOVA INTERRUPÇÃO

; QUANDO CHEGA AQUI É PORQUE PASSOU 1 ms

        INCF            V_MS, F
        MOVLW           D'10'
        XORWF           V_MS, W         ; VERIFICA SE JÁ CHEGOU
        M_JNZ           R_TMR2_OK       ; AINDA NÃO CHEGOU EM ZERO
        CLRF            V_MS

; QUANDO CHEGA AQUI É PORQUE PASSARAM 10 ms

        MOVF            V_PARTIDA, F    ; VERIFICA SE O TEMPO DE PARTIDA É ZERO
        M_JZ            R_TRM2_CS
        BTFSC           B_1CS_P                 ; TESTA SE DEVE DECREMENTAR A PARTIDA
        DECF            V_PARTIDA, F

R_TRM2_CS
        BTFSC           B_1CS                   ; TESTA SE DEVE INCREMENTAR CONTADOR DE SOBRECORRENTE
        BSF                     B_INC_TEMP              ;  PREPARA  FLAG  QUE  AVISA  PARA  INCREMENTAR  O
TEMPO E TESTAR

        INCF            V_CS, F
        MOVLW           D'100'
        XORWF           V_CS, W                 ; VERIFICA SE JÁ CHEGOU
        M_JNZ           R_TMR2_OK
        CLRF            V_CS

; QUANDO CHEGA AQUI É PORQUE PASSOU 1 s

        MOVF            V_TEC_TEMPO, F
        BTFSS           STATUS, Z                       ; CASO JÁ ESTEJA EM ZERO, IGNORE PRÓXIMO COMANDO
        DECF            V_TEC_TEMPO, F

R_TRM2_TEC_OK
        MOVF            V_PARTIDA, F            ; VERIFICA SE O TEMPO DE PARTIDA É ZERO
        M_JZ            R_TRM2_S
        BTFSC           B_1S_P                  ; TESTA SE DEVE DECREMENTAR A PARTIDA
        DECF            V_PARTIDA, F

R_TRM2_S
        BTFSC           B_1S                    ; TESTA SE DEVE INCREMENTAR CONTADOR DE SOBRECORRENTE
        BSF                     B_INC_TEMP              ;  PREPARA  FLAG  QUE  AVISA  PARA  INCREMENTAR  O
TEMPO E TESTA

        INCF            V_SEG, F
        MOVLW           D'60'
        XORWF           V_SEG, W        ; VERIFICA SE JÁ CHEGOU
        M_JNZ           R_TMR2_OK
        CLRF            V_SEG

; QUANDO CHEGA AQUI É PORQUE PASSOU 1 MINUTO
        INCF            V_MIN, F
        MOVLW           D'60'
        XORWF           V_MIN, W        ; VERIFICA SE JÁ CHEGOU
        M_JNZ           R_TMR2_OK
        CLRF            V_MIN
```

```
; QUANDO CHEGA AQUI É PORQUE PASSOU 1 h
        INCF            V_HOR, F
        MOVLW           D'24'
        XORWF           V_SEG, W        ; VERIFICA SE JÁ CHEGOU
        M_JNZ           R_TMR2_OK
        CLRF            V_HOR

R_TMR2_OK
        MOVF            V_PARTIDA, F            ; VERIFICA SE O TEMPO DE PARTIDA ACABOU
        M_JZ            R_TMR2_FIM                      ; SAI IGNORANDO A CONTAGEM DE TEMPO

R_TMR2_CONT
        BTFSS           P_MOTOR                         ; VERIFICA SE O MOTOR ESTÁ LIGADO
        GOTO            R_TMR2_FIM                      ; SAI SE ELE ESTIVER DESLIGADO

        BTFSS           B_INC_TEMP
        GOTO            R_TMR2_FIM
        BCF                     B_INC_TEMP                      ; LIMPA ESTE FLAG
        INCF            V_TEMPO
        MOVF            V_TEMPO_MAX, W
        SUBWF           V_TEMPO, W                      ; FAZ W = V_TEMPO - W
        BTFSS           STATUS, C
        GOTO            R_TMR2_FIM                      ; ENQUANTO NÃO ESTOURAR O TEMPO TUDO BEM!
        M_MOTOR         0                                       ; DESLIGA O MOTOR (DEVE ESTAR NO BANK 0
OU 2)
        BCF                     B_M_ALERTA                      ; FLAG PARA AVISAR AO PC DO PROBLEMA
        BSF                     B_M_SOBRE                       ; FLAG PARA AVISAR AO PC DO PROBLEMA

R_TMR2_FIM
        GOTO            INT_FIM                 ; FINALIZA INTERRUPÇÃO

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE ENVIA UM DADO PELA SERIAL
;       ENTRADA: W = BYTE COM O DADO A SER ENVIADO
;       SAÍDAS: NENHUMA
;       AFETA: TXREG; STATUS;
;       CHAMA: NADA (STACK: 1)
;-------------------------------------------------------------------------------------------------

R_SERIAL_TX
        BTFSS           PIR1, TXIF
        GOTO            $-1                             ; TESTA SE O TXREG ESTÁ VAZIO
        MOVWF           TXREG
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE RECEBE UM DADO PELA SERIAL
;       ENTRADA: NENHUMA
;       SAÍDAS: [V_SER_PTR] = W = BYTE RECEBIDO OU ERRO; V_SER_OK = SER_COUNT
;       AFETA: W; STATUS; FSR; V_SER_PTR; V_SER_OK
;       CHAMA: NADA (STACK: 1)
;-------------------------------------------------------------------------------------------------

R_SERIAL_RX
        BSF                     STATUS, IRP             ; PONTEIRO APONTANDO PARA BANK 2 OU BANK 3

        BTFSS           PIR1, RCIF
        GOTO            R_SER_RX_END            ; SAI SE NÃO HOUVER DADO NO BUFFER DE RECEPÇÃO

        M_MOVLF         V_SER_OK, SER_COUNT             ; EVITA TIMEOUT, POIS ACABOU DE CHEGAR UM BYTE

        BTFSC           RCSTA, FERR
        GOTO            R_SER_RX_ERROR          ; VERIFICA SE HOUVE ALGUM FRAMING ERROR
        BTFSC           RCSTA, OERR
        GOTO            R_SER_RX_ERROR          ; VERIFICA SE HOUVE ALGUM OVERRUN ERROR

        MOVF            V_SER_PTR, W            ; CARREGA O ENDEREÇO ATUAL DO PONTEIRO
        XORLW           SER_END                         ; TESTA SE ESTOUROU O PONTEIRO
        M_JZ            R_SER_RX_PTR

        M_MOVLF         FSR, SER_INFO
        CLRF            INDF                            ; GARANTE ZERO, POIS SÓ É UTILIZADO NA RESPOSTA
P/ O PC

        MOVF            V_SER_PTR, W            ; CARREGA O ENDEREÇO ATUAL DO PONTEIRO
        MOVWF           FSR                                     ; APONTA PARA ENDEREÇO DO BUFFER DE
RECEPÇÃO
        INCF            V_SER_PTR, F            ; APONTA PARA O PRÓXIMO END. DO BUFFER DE RECEPÇÃO

        MOVF            RCREG, W        ; LIBERA O BUFFER DA SERIAL (DESCARTA DADO!)
        MOVWF           INDF                    ; ESCREVE NO BUFFER
        GOTO            R_SER_RX_END
```

```
R_SER_RX_ERROR
        BCF                     RCSTA, CREN
        BSF                     RCSTA, CREN             ; RESETA COMUNICAÇÃO
        GOTO            R_SER_RX_END

R_SER_RX_PTR
        M_MOVLF         FSR, SER_INFO
        BSF                     B_ERRO_OVER             ;  GRAVA  O  ERRO  DE  ESTOURO!  (PRIMEIRO  ERRO
POSSÍVEL!)

        MOVF            RCREG, W        ; LIBERA O BUFFER DA SERIAL (DESCARTA DADO!)

R_SER_RX_END
        CLRF            STATUS
        GOTO            INT_FIM                 ; FINALIZA INTERRUPÇÃO

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE TRATA OS BYTES RECEBIDOS PELA SERIAL
;       ENTRADA: NENHUMA
;       SAÍDAS: NENHUMA
;       AFETA: W; STATUS; FSR; V_SER_PTR; V_SER_OK; *EEADR; *EEADRH; EEDATA
;       CHAMA: R_LIGA_MOTOR (2); R_EE_READ; R_EE_WRITE; R_ATUALIZA_EEPROM (STACK: 3)
;-------------------------------------------------------------------------------------------------

R_SERIAL
        CLRF            STATUS                  ; GARANTE BANK 0
        BSF                     STATUS, IRP             ; PONTEIRO PARA O BANK 2 E BANK 3
        BSF                     STATUS, RP1             ; PREPARA PARA LER DADOS DO BANK 2

        MOVF            V_SER_OK, F
        M_JZ            R_SER_END
        DECFSZ          V_SER_OK
        GOTO            R_SER_END

; CHEGA AQUI SE V_SER_OK ERA IGUAL A 1 E ACABOU DE OCORRER UM TIMEOUT!
        M_MOVLF         FSR, SER_INFO           ; APONTA PARA O BYTE DE INFORMAÇÃO

        MOVF            V_SER_PTR, W
        XORLW           SER_END
        M_JZ            R_SER_TIMEOUT_OK ; TESTA SE OCORREU UMA QUEBRA NA COMUNICAÇÃO (BREAK)
        BSF                     B_ERRO_BREAK            ;  ESCREVE  QUE  OCORREU  UM  ERRO  DE  QUEBRA  DE
COMUNICAÇÃO

R_SER_TIMEOUT_OK
; HÁ DADOS A SEREM TRATADOS PELA SERIAL
        MOVLW           SER_SIZE
        XORWF           SER_TAM, W                      ; TESTA SE O TAMANHO DO PACOTE ESTÁ OK
        M_JZ            R_SER_TST_OK
        BSF                     B_ERRO_TAM              ;  ESCREVE  QUE  OCORREU  UM  ERRO  DE  COMANDO
INVÁLIDO

R_SER_TST_OK

        MOVLW           SER_CMD_TST
        ANDWF           SER_CMD, W                      ; FILTRA BITS PARA TESTAR SE O COMANDO É VÁLIDO
        M_JZ            R_SER_COM_OK
        BSF                     B_ERRO_CMD              ;  ESCREVE  QUE  OCORREU  UM  ERRO  DE  COMANDO
INVÁLIDO

R_SER_COM_OK
; VERIFICA ID
        BSF                     B_ID_OK                 ; ACREDITA QUE O PC ESTEJA FALANDO COM ELE (ID
OK!)
        M_MOVLF         FSR, SER_ID             ; APONTA PARA O BYTE DA IDENTIFICAÇÃO
        M_EE_READ       E_ID
        XORWF           INDF, W
        BTFSS           STATUS, Z
        BCF                     B_ID_OK                 ; SE DIFERENTES ENTÃO ID NOK!

        M_MOVLF         FSR, SER_INFO    ; APONTA PARA O BYTE DA IDENTIFICAÇÃO
        MOVF            INDF, F                 ; ATUALIZA STATUS
        M_JNZ           R_SER_RESP              ; SE HOUVE QUALQUER ERRO, APENAS RESPONDE

; SE CHEGOU AQUI É PORQUE DEVE ATENDER AO COMANDO, PACOTE SERIAL ESTÁ OK!

        BTFSS           B_ID_OK
        GOTO            R_SER_RESP              ; SE DIFERENTES ENTÃO NÃO TRATA DADOS

        BCF                     STATUS, RP0
        BSF                     STATUS, RP1             ; GARANTE O BANK 2

; PREPARA PONTEIROS DOS ENDEREÇOS
        M_MOVLF         FSR, SER_EDATA0         ; APONTA P/ INÍCIO DOS DADOS DA EEPROM DO BUFFER SERIAL
```

```
        MOVLW           SER_EE_ADDR
        ANDWF           SER_EE, W               ; FILTRA BITS COM O ENDEREÇO DO BLOCO DA EEPROM
        MOVWF           EEADR                   ; APONTA P/ ENDEREÇO INICIAL DA EEPROM

; DEVE ESCREVER / LER OS BYTES NA EEPROM
R_SER_EEPROM
        BTFSC           B_EE_WR                 ; TESTA SE FOI UMA LEITURA OU ESCRITA
        GOTO            R_SER_EE_WRITE          ; SALTA SE É PARA FAZER UMA ESCRITA

; PROVIDENCIA UMA LEITURA NA EEPROM
        CALL            R_EE_READ
        MOVWF           INDF                    ; COPIA DADO P/ O BUFFER SERIAL
        GOTO            R_SER_EE_CONT

R_SER_EE_WRITE
        MOVF            INDF, W
        MOVWF           EEDATA
        CALL            R_EE_WRITE
        M_JNZ           R_SER_EE_NOK    ; SALTA SE HOUVE ERRO DE ESCRITA NA EEPROM!

R_SER_EE_CONT
        BCF                     STATUS, RP0
        BSF                     STATUS, RP1             ; APONTA PARA VARIÁVEIS DO BANK 2

        INCF            EEADR, F
        INCF            FSR, F                  ; INCREMENTA OS PONTEIROS E CONTINUA A CÓPIA

; TESTA SE JÁ COPIOU O ÚLTIMO DADO
        MOVLW           SER_EDATA0 + SER_EDATA
        XORWF           FSR, W
        BTFSS           STATUS, Z
        GOTO            R_SER_EEPROM    ; LOOP ATÉ COPIAR TODOS OS BYTES

R_SER_EE_OK
        BTFSS           B_EE_WR                 ; TESTA SE FOI UMA LEITURA OU ESCRITA
        GOTO            R_SER_EE_FIM    ; SALTA SE É PARA FAZER UMA ESCRITA

; COMO FOI UMA ESCRITA, ATUALIZA VARIÁVEIS AQUI!!!
        CALL            R_ATUALIZA_EEPROM
        GOTO            R_SER_EE_FIM

R_SER_EE_NOK
        M_MOVLF         FSR, SER_INFO           ; APONTA PARA O BYTE DE INFORMAÇÃO
        BSF                     B_ERRO_EEPROM           ; ESCREVE QUE OCORREU UM ERRO ESCRITA NA EEPROM

R_SER_EE_FIM
        BCF                     STATUS, RP0
        BCF                     STATUS, RP1                     ; GARANTE O BANK 0

; AS CORRENTES SÃO SOMENTE LEITURA
        M_MOVLF         FSR, SER_FASE           ; APONTA PARA O PRIMEIRO ENDEREÇO DA FASE NO BUFFER
SERIAL
        MOVF            V_I1, W
        MOVWF           INDF                    ; COPIA DADO P/ O BUFFER SERIAL
        INCF            FSR, F
        MOVF            V_I2, W
        MOVWF           INDF                    ; COPIA DADO P/ O BUFFER SERIAL
        INCF            FSR, F
        MOVF            V_I3, W
        MOVWF           INDF                    ; COPIA DADO P/ O BUFFER SERIAL
        INCF            FSR, F

; VERIFICA SE DEVE LIGAR / DESLIGAR O MOTOR
        BSF                     STATUS, RP1                     ; APONTA PARA VARIÁVEIS NO BANK 2

        BTFSS           B_MOTOR_SET
        GOTO            R_SER_MOTOR_READ ; SALTA POIS NÃO DESEJA LIGAR / DESLIGAR O MOTOR

        BTFSS           B_MOTOR_ON
        GOTO            R_SER_MOTOR_OFF
        M_MOTOR         1               ; LIGA O MOTOR (DEVE ESTAR NO BANK 0 OU 2)
        GOTO            R_SER_MOTOR_READ

R_SER_MOTOR_OFF
        M_MOTOR         0               ; DESLIGA O MOTOR (DEVE ESTAR NO BANK 0 OU 2)

R_SER_MOTOR_READ
        BCF                     B_MOTOR_ON                      ; CONSIDERA INICIALMENTE A CONDIÇÃO DE
MOTOR DESLIGADO
        BTFSS           P_MOTOR
        GOTO            R_SER_MOTOR_OK          ; RETORNA INDICANDO QUE O MOTOR ESTÁ DESLIGADO
        BSF                     B_MOTOR_ON                      ; RETORNA INDICANDO QUE O MOTOR ESTÁ
LIGADO
```

```
R_SER_MOTOR_OK
; INICIA VERIFICAÇÃO DO HORÁRIO
        M_MOVLF         FSR, SER_TIME           ; APONTA PARA O PRIMEIRO BYTE DO HORÁRIO NO BUFFER SERIAL
        BTFSS           B_TIME_WR                       ; SE FOR PARA ATUALIZAR SALTA O GOTO
        GOTO            R_SER_TIME_READ         ; SE FOR PARA APENAS LER O HORÁRIO SALTA
; O PIC DEVE ATUALIZAR SEU HORÁRIO INTERNO
        BCF                     STATUS, RP1                     ; APONTA PARA VARIÁVEIS NO BANK 0
        CLRF            V_MS
        CLRF            V_CS
        MOVF            INDF, W
        INCF            FSR, F
        MOVWF           V_SEG
        MOVF            INDF, W
        INCF            FSR, F
        MOVWF           V_MIN
        MOVF            INDF, W
        INCF            FSR, F
        MOVWF           V_HOR                   ; ATUALIZA HORÁRIO DO PIC
        GOTO            R_SER_RESP

R_SER_TIME_READ
; O PIC DEVE ENVIAR SEU HORÁRIO INTERNO
        BCF                     STATUS, RP1                     ; APONTA PARA VARIÁVEIS NO BANK 0
        MOVF            V_SEG, W
        MOVWF           INDF
        INCF            FSR, F
        MOVF            V_MIN, W
        MOVWF           INDF
        INCF            FSR, F
        MOVF            V_HOR, W
        MOVWF           INDF
        INCF            FSR, F                  ; CARREGA HORÁRIO NO BUFFER SERIAL

R_SER_RESP
        BCF                     STATUS, RP0
        BCF                     STATUS, RP1                     ; APONTA PARA VARIÁVEIS NO BANK 0

        MOVLW           SER_INFO
        MOVWF           FSR
        MOVF            V_SER_FLAG, W
        IORWF           INDF, F

        CLRF            V_SER_OK
        M_MOVLF         V_SER_PTR, SER_ID       ; INICIALIZA VARIÁVEIS DE CONTROLE DA SERIAL
        MOVWF           FSR                             ; APONTA PARA O END. INICIAL DO BUFFER
DA SERIAL
R_SER_LOOP
        MOVF            INDF, W
        CLRF            INDF

        BTFSS           B_ID_OK
        GOTO            R_SER_LOOP_PULA         ; SE DIFERENTES ENTÃO NÃO ENVIA DADOS

        BTFSS           PIR1, TXIF
        GOTO            $-1                             ; ESPERA TÉRMINO DO ÚLTIMO ENVIO
        MOVWF           TXREG                   ; ENVIA O DADO PELA SERIAL

R_SER_LOOP_PULA
        INCF            FSR, F
        MOVLW           SER_END                 ; TESTA SE JÁ PASSOU DO END. DO CHECK SUM
        XORWF           FSR, W
        M_JNZ           R_SER_LOOP              ; ENVIA TODOS OS BYTES ZERANDO O BUFFER

R_SER_END
        CLRF            STATUS                  ; RETORNA PARA CONDIÇÃO DE TUDO ZERO
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE CARREGA VARIÁVEIS PRINCIPAIS DA EEPROM PARA A MEMÓRIA RAM (ESPELHO)
;       ENTRADA: NENHUMA
;       SAÍDAS: NENHUMA
;       AFETA: W; STATUS; FSR; V_SER_PTR; V_SER_OK; *EEADR; *EEADRH; EEDATA
;       CHAMA: R_EE_READ (STACK: 2)
;-------------------------------------------------------------------------------------------------

R_ATUALIZA_EEPROM
        M_EE_READ       E_DELTA
        MOVWF           V_DELTA
        M_EE_READ       E_MENOR
        MOVWF           V_MENOR
        RETURN
```

```
;***********************************************************************************************
; Inicio do Programa Principal
;***********************************************************************************************

R_INICIO
        CLRF            STATUS                          ; GARANTE BANK = 0

; LIMPA VARIÁVEIS GLOBAIS
        M_MOVLF         FSR, V_BANKS            ; APONTA P/ O INICIO DOS BYTES COMUNS À TODOS OS BANCOS

R_INI_LOOP1
        CLRF            INDF
        INCFSZ          FSR, F
        GOTO            R_INI_LOOP1                     ; LOOP ATÉ ZERAR TODAS AS VARIAVEIS GLOBAIS

        M_MOVLF         FSR, V_NUM1
        M_MOVLF         V_NUM, 0X20
R_INI_ZERAR
        CLRF            INDF
        INCF            FSR, F
        DECFSZ          V_NUM, F
        GOTO            R_INI_ZERAR

; LIMPA BUFFER SERIAL
        M_MOVLF         FSR, SER_ID             ; APONTA P/ O INICIO DO BUFFER SERIAL
        BSF                     STATUS, IRP
R_INI_LOOP2
        CLRF            INDF
        INCF            FSR
        MOVLW           SER_END                 ; TESTA SE JÁ PASSOU DO END. DO CHECK SUM
        XORWF           FSR, W
        M_JNZ           R_INI_LOOP2             ; LOOP P/ ZERAR O BUFFER SERIAL

; INICIA VARIÁVEIS DE CONTROLE
        M_MOVLF         RCSTA, K_RCSTA          ; CONFIGURA RECEPÇÃO DA SERIAL

        M_MOVLF         ADCON0, K_ADCON0 ; INICIA CONFIGURAÇÕES DO AD

        MOVLW           K_T2CON         ; BANK 0
        MOVWF           T2CON

; INICIALIZA VALORES DAS PORTAS
        M_MOVLF         PORTA, K_PORTA
        M_MOVLF         PORTB, K_PORTB
        M_MOVLF         PORTC, K_PORTC

; INICIA VARIÁVEIS DO BANK 1
        BSF                     STATUS, RP0

        M_MOVLF         ADCON1, K_ADCON1 ; TERMINA CONFIGURAÇÕES DO AD

        M_MOVLF         PR2, K_PR2                      ; INICIA CONTADOR DO TIMER2

        M_MOVLF         SPBRG, K_SPBRG          ; CONFIGURA O BAUD RATE
        M_MOVLF         TXSTA, K_TXSTA          ; CONFIGURA TRANSMISSÃO

        M_MOVLF         PIE1, K_PIE1            ; HABILITA INTERRUPÇÕES

        M_MOVLF         OPTION_REG, K_OPTION
        M_MOVLF         TRISA, K_TRISA
        M_MOVLF         TRISB, K_TRISB
        M_MOVLF         TRISC, K_TRISC          ; INICIALIZA DIREÇÃO DAS PORTAS

        M_CHBK          10              ; SELECIONA BANK 0

; INICIALIZA VALORES DAS PORTAS
        M_MOVLF         PORTA, K_PORTA
        M_MOVLF         PORTB, K_PORTB
        M_MOVLF         PORTC, K_PORTC

        BSF                     P_NTEC                  ; DESABILITA O TECLADO
        M_MOTOR         0                               ; DESLIGA O MOTOR

; INICIALIZAÇÕES DAS VARIÁVEIS DE CONTROLE
        M_MOVLF         V_I_PTR, V_I1           ; INICIA PONTEIRO COM A CORRENTE DA PRIMEIRA FASE
        M_MOVLF         V_SER_PTR, SER_ID       ; INICIALIZA VARIÁVEIS DE CONTROLE DA SERIAL

        CALL            R_LCD_START             ; INICIALIZA O LCD

        M_MOVLF         INTCON, K_INTCON ; CONFIGURA CONTROLE DE INTERRUPÇÕES

; ATUALIZA VARIÁVEIS ESPELHO AQUI
        CALL            R_ATUALIZA_EEPROM
```

```
R_MAIN
        CALL            R_MSG_INICIO     ; ESCREVE MSG INICIAL NO LCD
;       CALL            R_LCD_I                  ; MOSTRA CORRENTE ATUAL NO LCD
;       CALL            R_TESTA_I                ; VERIFICA SE HÁ SOBRECORRENTE

        CALL            R_TEC_UP
        M_JZ            R_MAIN                   ; ESPERA POR UMA TECLA

;DEBUG!!!
        GOTO            R_MAIN
;FIM DO DEBUG!!!

        BTFSC           V_TECLA, TEC_MENU
        GOTO            R_MAIN_MENU                      ; FOI TECLA MENU
        BTFSC           V_TECLA, TEC_OK
        GOTO            R_MAIN_OK                        ; FOI TECLA OK

; SÓ PODE TER SIDO TECLAS UP OU DOWN
        BSF                     STATUS, C
        BTFSC           V_TECLA, TEC_UP
        GOTO            R_MAIN_UP
R_MAIN_DOWN
        BCF                     STATUS, C
R_MAIN_UP
        CALL            R_I_PTR
        GOTO            R_MAIN

R_MAIN_MENU
        CALL            R_MSG_MOTOR
        CALL            R_TECLA
        BTFSC           V_TECLA, TEC_OK
        GOTO            R_MAIN_MENU_OK           ; FOI TECLA OK
        BTFSS           V_TECLA, TEC_MENU
        GOTO            R_MAIN_FIM                       ; CASO CONTRÁRIO VOLTA AO LOOP
; TRATA BOTÃO MENU DENTRO DO MENU LIGA / DESLIGA MOTOR
        MOVLW           E_ID
        CALL            R_AJUSTA_EEPROM
        MOVLW           E_DELTA
        CALL            R_AJUSTA_EEPROM
        MOVLW           E_MENOR
        CALL            R_AJUSTA_EEPROM
        MOVLW           E_NOMINAL
        CALL            R_AJUSTA_EEPROM
        MOVLW           E_PARTIDA
        CALL            R_AJUSTA_EEPROM
        MOVLW           E_UNIDADE
        CALL            R_AJUSTA_EEPROM
        MOVLW           E_I_UNIDADE
        CALL            R_AJUSTA_EEPROM

        CALL            R_ATUALIZA_EEPROM
        GOTO            R_MAIN_FIM                       ; POR HORA APENAS SAI!

R_MAIN_MENU_OK
; TRATA BOTÃO OK PARA LIGA / DESLIGA MOTOR
        BTFSS           P_MOTOR                          ; VERIFICA SE O MOTOR ESTÁ LIGADO
        GOTO            R_MAIN_MOT_ON            ; SALTA PARA LIGAR O MOTOR, POIS ESTÁ DESLIGADO
        M_MOTOR         0
        GOTO            R_MAIN_FIM

R_MAIN_MOT_ON
; O MOTOR ESTÁ DESLIGADO, DEVE LIGÁ-LO
        M_MOTOR         1
        GOTO            R_MAIN_FIM

R_MAIN_OK
; NADA IMPLEMENTADO PARA ESTA OPÇÃO ATÉ AGORA

R_MAIN_FIM
        MOVLW           LCD_CLR
        CALL            R_LCD_CMD
        M_WAIT          D'100'                           ; ESPERA UM POUCO ANTES DE CONTINUAR
        GOTO            R_MAIN


;-------------------------------------------------------------------------------------------------
;       ROTINA QUE CAUSA UM ATRASO DE APROXIMADAMENTE N x 51,2 us
;       ENTRADA: W = N ESPERAS
;       SAÍDA: NENHUMA
;       AFETA: W; STATUS; TMR0
;       CHAMA: NADA (STACK: 1)
;-------------------------------------------------------------------------------------------------
```

```
R_DELAY
        SUBLW           0XFF
        MOVWF           TMR0                    ; APROXIMADAMENTE W x 256 x 4/Fosc ~= W x 50 us
        INCF            TMR0, F                 ; AJUSTE DA SUBTRAÇÃO

        BCF                     INTCON, T0IF    ; PREPARA PARA VERIFICAR SE HOUVE OVERFLOW
        BTFSS           INTCON, T0IF
        GOTO            $-1                             ;   ESPERA   POR   OVERFLOW,   ISTO   É,   PASSOU
APROXIMADAMENTE 50xW us

        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE CAUSA UM ATRASO DE APROXIMADAMENTE N ms
;       ENTRADA: W = N ESPERAS
;       SAÍDA: NENHUMA
;       AFETA: W; STATUS; V_WAIT; TMR0
;       CHAMA: NADA (STACK: 1)
;-------------------------------------------------------------------------------------------------

R_WAIT
        MOVWF           V_WAIT                  ; CARREGA CONTADOR DE TEMPO
R_WIT_LOOP
        MOVLW           D'20'
        CALL            R_DELAY                 ; APROXIMADAMENTE 20 x 256 x 4/Fosc ~= 1000 us

        DECFSZ          V_WAIT, F
        GOTO            R_WIT_LOOP              ; REPETE ATÉ PASSAR O TEMPO PEDIDO

        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE ESPERA POR UMA TECLA POR UM TEMPO MÁXIMO
;       ENTRADA: V_TEC_TEMPO = NÚMERO DE SEGUNDOS QUE DEVE ESPERAR PELA TECLA = 7 SEGUNDOS SE ZERO!
;       SAÍDA: V_TECLA = TECLA PRECIONADA
;       AFETA: W; STATUS; V_TECLA; V_TEC_TEMPO
;       CHAMA: R_TEC_UP (4) (STACK: 5)
;-------------------------------------------------------------------------------------------------

R_TECLA
        MOVF            V_TEC_TEMPO, W
        BTFSC           STATUS, Z               ; VERIFICA SE O TEMPO É ZERO
        MOVLW           D'7'                    ; SE FOR ZERO ENTÃO COLOCA O DEFAULT VALOR
        MOVWF           V_TEC_TEMPO

        CLRF            V_TECLA                 ; PRIMEIRAMENTE NENHUMA TECLA PRECIONADA

R_TEC_LOOP
        MOVF            V_TEC_TEMPO, F
        M_JZ            R_TEC_FIM               ; TESTA SE JÁ ESTOUROU TEMPO DE ESPERA DA TECLA

        CALL            R_TEC_UP        ; RETORNA Z = 0 SE NADA PRECIONADO
        M_JZ            R_TEC_LOOP              ; SE NADA TECLADO, SAI IMEDIATAMENTE
R_TEC_FIM
        CLRF            V_TEC_TEMPO
        MOVF            V_TECLA,F               ; APENAS ATUALIZA STATUS DE ACORDO COM A TECLA
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE LÊ ÚLTIMA TECLA PRECIONADA (ESPERA SOLTAR A TECLA SE FOR O CASO)
;       ENTRADA: NENHUMA
;       SAÍDA: V_TECLA = 0 SE NADA; = TECLA PRECIONADA; STATUS, Z = 0
;       AFETA: W; STATUS; V_TECLA
;       CHAMA: R_SERIAL (3); R_LE_TEC (1) (STACK: 4)
;-------------------------------------------------------------------------------------------------

R_TEC_UP
        CLRF            V_TECLA                 ; PRIMEIRAMENTE NENHUMA TECLA PRECIONADA

R_TEC_UP_LOOP
        CALL            R_SERIAL        ; VERIFICA A COMUNICAÇÃO SERIAL

        CALL            R_LE_TEC        ; RETORNA Z = 0 SE NADA PRECIONADO
        M_JZ            R_TEC_UP_FIM    ; SE NADA TECLADO, SAI IMEDIATAMENTE

        MOVWF           V_TECLA                 ; SALVA TECLA PRECIONADA
        GOTO            R_TEC_UP_LOOP

R_TEC_UP_FIM
        MOVF            V_TECLA,F               ; APENAS ATUALIZA STATUS DE ACORDO COM A TECLA
        RETURN
```

```
;-------------------------------------------------------------------------------------------------
;       ROTINA QUE ESPERA POR UMA TECLA POR UM TEMPO MÁXIMO, MAS NÃO ESPERA SOLTAR A TECLA
;       ENTRADA: V_TEC_TEMPO = NÚMERO DE SEGUNDOS QUE DEVE ESPERAR PELA TECLA = 7 SEGUNDOS SE ZERO!
;       SAÍDA: V_TECLA = TECLA PRECIONADA
;       AFETA: W; STATUS; V_TECLA; V_TEC_TEMPO
;       CHAMA: R_LE_TEC; R_WAIT (STACK: 2)
;-------------------------------------------------------------------------------------------------

R_TECLA_WAIT
        MOVF            V_TEC_TEMPO, W
        BTFSC           STATUS, Z               ; VERIFICA SE O TEMPO É ZERO
        MOVLW           D'7'                    ; SE FOR ZERO ENTÃO COLOCA O DEFAULT VALOR
        MOVWF           V_TEC_TEMPO

        CLRF            V_TECLA                 ; PRIMEIRAMENTE NENHUMA TECLA PRECIONADA

R_TEC_WIT_LOOP
        MOVF            V_TEC_TEMPO, F
        M_JZ            R_TEC_WIT_FIM           ; TESTA SE JÁ ESTOUROU TEMPO DE ESPERA DA TECLA

        M_WAIT          D'100'
        CALL            R_LE_TEC                ; RETORNA Z = 0 SE NADA PRECIONADO
        M_JZ            R_TEC_WIT_LOOP          ; SE NADA TECLADO, SAI IMEDIATAMENTE
        MOVWF           V_TECLA                         ; SALVA TECLA PRECIONADA
        M_WAIT          D'50'
R_TEC_WIT_FIM
        CLRF            V_TEC_TEMPO
        MOVF            V_TECLA,F               ; APENAS ATUALIZA STATUS DE ACORDO COM A TECLA
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE VERIFICA QUAIS AS TECLAS QUE ESTÃO PRESSIONADAS (NÃO ESPERA SOLTAR TECLA)
;       ENTRADA: NENHUMA
;       SAÍDAS: W = TECLA PRECIONADA ; Z = 0 SE NÃO E Z = 1 SE SIM
;       AFETA: W; TRIS_TEC; STATUS; *TMR0; V_WAIT
;       CHAMA: R_WAIT (STACK: 1)
;-------------------------------------------------------------------------------------------------

R_LE_TEC
        BSF                     STATUS, RP0             ; BANK 1
        BCF                     OPTION_REG, 7   ; HABILITA PULL UP
        BCF                     STATUS, RP0             ; BANK 0

        BCF                     P_NTEC                  ; HABILITA TECLADO
        M_WAIT          D'10'                   ; ESPERA 10 ms PARA FILTRAR RUÍDOS
        MOVF            PORT_TEC, W             ; LÊ TECLAS
        BSF                     P_NTEC                  ; DESABILITA TECLADO

        BSF                     STATUS, RP0             ; BANK 1
        BSF                     OPTION_REG, 7   ; DESABILITA PULL UP
        BCF                     STATUS, RP0             ; BANK 0

        ANDLW           K_TECS                  ; FILTRA OS PINOS VÁLIDOS DO TECLADO
        XORLW           K_TECS                  ; COMPLEMENTA OS BITS, SE Z = 0 ENTAO NÃO HÁ TECLAS
PRECIONADAS!
        RETURN


;-------------------------------------------------------------------------------------------------
;       ROTINA DE INICIALIZAÇÃO DO LCD
;       ENTRADA: NENHUMA
;       SAÍDAS: NENHUMA
;       AFETA: W; STATUS; V_LCD; *V_WAIT
;       CHAMA: R_WAIT; LCD_CMD (STACK: 4)
;-------------------------------------------------------------------------------------------------

R_LCD_START
        M_CHBK          01
        MOVLW           TRIS_LCD_OUT
        ANDWF           TRIS_LCD, F             ; CONFIGURA BARRAMENTO DE DADOS COMO OUTPUT
        M_CHBK          10

; ZERA O BARRAMENTO DE DADOS E O BARRAMENTO DE CONTROLE DO LCD
        BCF                     P_EN            ; GARANTE QUE O LCD ESTEJA DESABILITADO

        M_WAIT          D'200'          ; ESPERA AS TENSÕES DO CIRCUITO ESTABILIZAR

        BCF                     P_RS
        BCF                     P_READ          ; GARANTE COMANDO DE ESCRITA DE COMANDOS NO MÓDULO DO LCD

; ESCREVE O PRIMEIRO COMANDO NO LCD (INICIALIZA O LCD)
        MOVLW           K_NDATA
        ANDWF           PORT_LCD, F             ; ESCREVE ZERO NO BARRAMENTO DE DADOS DO LCD
```

```
        MOVLW           0X30
        IORWF           PORT_LCD, F             ; ESCREVE O DADO NO BARRAMENTO DE DADOS
        MOVLW           0X03
        MOVWF           V_LCD
R_LCD_STT_LOOP
        BSF                     P_EN            ; HABILITA LCD
        NOP
        NOP
        NOP                                             ;  ESPERA  MAIS  DE  450ns  ENTRE  O  SINAL  QUE
HABILITA E O QUE DESABILITA
        BCF                     P_EN            ; DESABILITA LCD
        M_WAIT          D'20'           ; ESPERA MAIS DE 15ms
        DECFSZ          V_LCD, F
        GOTO            R_LCD_STT_LOOP

; AGORA PROGRAMA O LCD PARA 4 BITS DE DADOS
        BSF                     P_EN            ; HABILITA LCD
        BCF                     P_D0            ; ESCREVE 2 NO BARRAMENTO DE DADOS
        NOP
        NOP                                             ;  ESPERA  MAIS  DE  450ns  ENTRE  O  SINAL  QUE
HABILITA E O QUE DESABILITA
        BCF                     P_EN            ; DESABILITA LCD
        M_WAIT          D'1'            ; ESPERA UM POUCO ANTES DE CONTINUAR

; AGORA O LCD ESTÁ PRONTO PARA COMUNICAR-SE VIA 4 BITS DE DADOS (INICIA PROGRAÇÃO DO LCD)
        MOVLW           LCD_CFG
        CALL            R_LCD_CMD

        MOVLW           LCD_OFF
        CALL            R_LCD_CMD

        MOVLW           LCD_ON
        CALL            R_LCD_CMD

        MOVLW           LCD_CLR
        CALL            R_LCD_CMD

        MOVLW           LCD_CUR_R
        CALL            R_LCD_CMD

        MOVLW           LCD_CUR_INPUT
        CALL            R_LCD_CMD

        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE LE UM BYTE DO LCD
;       ENTRADA: NENHUMA
;       SAÍDAS: V_LCD = BYTE LIDO DO LCD
;       AFETA: W; STATUS; TRIS_LCD = INPUT; CTRL_LCD; V_LCD
;       CHAMA: NADA (STACK: 1)
;-------------------------------------------------------------------------------------------------

R_LCD_READ
        BSF                     P_READ          ; CONFIGURA COMO LEITURA

        M_CHBK          01
        MOVLW           TRIS_LCD_IN
        IORWF           TRIS_LCD, F             ; CONFIGURA PORT_LCD COMO INPUT
        M_CHBK          10

        BSF                     P_EN            ; HABILITA LCD
        NOP
        NOP                                             ; ESPERA MAIS DE 350ns PELA RESPOSTA DO LCD
        MOVF            PORT_LCD, W
        ANDLW           K_DATA          ; LÊ E FILTRA DADOS LIDOS
        MOVWF           V_LCD
        BCF                     P_EN            ; DESABILITA LCD

        NOP
        NOP
        NOP                                             ;  ESPERA  MAIS  DE  500ns  PARA  MONTAR  A  ONDA
QUADRADA DO ENABLE

        BSF                     P_EN            ; HABILITA LCD
        NOP
        NOP                                             ; ESPERA MAIS DE 350ns PELA RESPOSTA DO LCD
        SWAPF           PORT_LCD, W
        ANDLW           K_NDATA
        IORWF           V_LCD, F ; MONTA O BYTE COMPLETO RECEBIDO DO LCD
        BCF                     P_EN            ; DESABILITA LCD

        RETURN
```

```
;-------------------------------------------------------------------------------------------------
;       ROTINA QUE ESCREVE UM BYTE NO LCD
;       ENTRADAS: W = BYTE A SER ESCRITO NO LCD
;       SAÍDAS: NENHUMA
;       AFETA: STATUS; V_LCD; V_W_AUX
;       CHAMA: NADA (STACK: 1)
;-------------------------------------------------------------------------------------------------

R_LCD_WRITE
        MOVWF           V_LCD           ; SALVA BYTE A SER ESCRITO
        BCF                     P_READ          ; CONFIGURA COMO ESCRITA

        M_CHBK          01
        MOVLW           TRIS_LCD_OUT
        ANDWF           TRIS_LCD, F             ; CONFIGURA PORT_LCD COMO OUTPUT
        M_CHBK          10

; ENVIA NIBBLE SUPERIOR
        MOVLW           K_NDATA
        ANDWF           PORT_LCD, F             ; ESCREVE ZERO NO BARRAMENTO DE DADOS DO LCD

        BSF                     P_EN            ; HABILITA LCD
        BTFSC           V_LCD, 7
        BSF                     P_D3
        BTFSC           V_LCD, 6
        BSF                     P_D2
        BTFSC           V_LCD, 5
        BSF                     P_D1
        BTFSC           V_LCD, 4
        BSF                     P_D0
        NOP                                             ; ESPERA PELO MENOS 195ns COM O BARRAMENTO DE
DADOS PRONTO
        BCF                     P_EN            ; DESABILITA LCD

        MOVLW           K_NDATA
        ANDWF           PORT_LCD, F             ; ESCREVE ZERO NO BARRAMENTO DE DADOS DO LCD

; ENVIA NIBBLE INFERIOR
        BSF                     P_EN            ; HABILITA LCD
        BTFSC           V_LCD, 3
        BSF                     P_D3
        BTFSC           V_LCD, 2
        BSF                     P_D2
        BTFSC           V_LCD, 1
        BSF                     P_D1
        BTFSC           V_LCD, 0
        BSF                     P_D0
        NOP                                             ; ESPERA PELO MENOS 195ns COM O BARRAMENTO DE
DADOS PRONTO
        BCF                     P_EN            ; DESABILITA LCD

        M_CHBK          01
        MOVLW           TRIS_LCD_IN
        IORWF           TRIS_LCD, F             ; CONFIGURA PORT_LCD COMO INPUT
        M_CHBK          10

        MOVF            V_LCD, W ; RECUPERA DADO ENVIADO ENVIADA
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE ESPERA O LCD FICAR DESOCUPADO
;       ENTRADA: NENHUMA
;       SAÍDAS: NENHUMA
;       AFETA: CTRL_LCD; *W; *STATUS; *TRIS_LCD; *V_LCD
;       CHAMA: R_LCD_READ (STACK: 2)
;-------------------------------------------------------------------------------------------------

R_LCD_BF
        BCF                     P_RS            ; CONFIGURA COMO COMANDO
        CALL            R_LCD_READ      ; LÊ BYTE DE COMANDO DO LCD
        BTFSC           V_LCD, 7
        GOTO            R_LCD_BF ; CONTINUA ATÉ O LCD DESOCUPAR!
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE ENVIA UM COMANDO PARA O LCD
;       ENTRADA: W = BYTE COM O COMANDO A SER ENVIADO
;       SAÍDAS: NENHUMA
;       AFETA: STATUS; V_W_AUX; *V_LCD; *TRIS_LCD; *CTRL_LCD
;       CHAMA: R_LCD_BF; R_LCD_WRITE (STACK: 3)
;-------------------------------------------------------------------------------------------------
```

```
R_LCD_CMD
        MOVWF           V_W_AUX         ; SALVA COMANDO
        CALL            R_LCD_BF ; ESPERA O LCD DESOCUPAR

        MOVF            V_W_AUX, W
        CALL            R_LCD_WRITE
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE ENVIA UM DADO PARA O LCD
;       ENTRADA: W = BYTE COM O DADOS A SER ENVIADO
;       SAÍDAS: NENHUMA
;       AFETA: STATUS; V_W_AUX; *V_LCD; *TRIS_LCD; *CTRL_LCD
;       CHAMA: R_LCD_BF; R_LCD_WRITE (STACK: 3)
;-------------------------------------------------------------------------------------------------

R_LCD_CHAR
        MOVWF           V_W_AUX         ; SALVA DADO
        CALL            R_LCD_BF ; ESPERA O LCD DESOCUPAR

        BSF                     P_RS            ; CONFIGURA COMO DADO
        MOVF            V_W_AUX, W
        CALL            R_LCD_WRITE
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE MOSTRA UM BCD NO LCD
;       ENTRADA: W = BCD A SER MOSTRADO
;       SAÍDAS: NENHUMA
;       AFETA: W; STATUS; V_W_AUX; V_NUM1; *V_LCD; *TRIS_LCD; *CTRL_LCD
;       CHAMA: R_LCD_BF; R_LCD_WRITE (STACK: 3)
;-------------------------------------------------------------------------------------------------

R_LCD_BCD
        MOVWF           V_W_AUX         ; SALVA NUMERO A SER MOSTRADO
        ANDLW           0X0F
        ADDLW           '0'                     ; CONVERTE NO ASCII EQUIVALENTE
        MOVWF           V_NUM1          ; SALVA NIBBLE INFERIOR DO BCD
        SWAPF           V_W_AUX, W      ; PEGA NIBBLE SUPERIOR DO BCD
        ANDLW           0X0F
        ADDLW           '0'                     ; CONVERTE NO ASCII EQUIVALENTE

        MOVWF           V_W_AUX         ; SALVA DADO
        CALL            R_LCD_BF ; ESPERA O LCD DESOCUPAR

        BSF                     P_RS            ; CONFIGURA COMO DADO
        MOVF            V_W_AUX, W
        CALL            R_LCD_WRITE     ; MOSTRA NIBBLE SUPERIOR

        MOVF            V_NUM1, W       ; RECUPERA NIBBLE INFERIOR
        GOTO            R_LCD_CHAR      ; MOSTRA NIBBLE INFERIOR
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE MOSTRA UM BYTE EM DECIMAL NO LCD
;       ENTRADA: W = BYTE COM O DADOS A SER ENVIADO
;       SAÍDAS: NENHUMA
;       AFETA: W; STATUS; V_NUM; V_NUM1; V_NUM2; V_NUM3; V_NUM4; *V_NUM5; *V_NUM6; V_NUM7
;                       *V_W_AUX; *V_LCD; *TRIS_LCD; *CTRL_LCD
;       CHAMA: R_DIV8; R_LCD_BF; R_LCD_WRITE (STACK: 3)
;-------------------------------------------------------------------------------------------------

R_LCD_DECIMAL
        CLRF            STATUS          ; GARANTE BANK 0
        MOVWF           V_NUM3          ; SALVA DADO

        M_MOVLF         V_NUM7, 0X03
        MOVLW           D'100'
R_LCD_DEC_LOOP
        MOVWF           V_NUM2
        MOVF            V_NUM3, W
        MOVWF           V_NUM1
        CLRF            V_NUM                   ; PARTE ALTA DA WORD É NULA
        CALL            R_DIV8                  ; PEGA O RESTO PARA DIVIDIR POR 10 (RESTO EM V_NUM3)
        MOVF            V_NUM4, W
        BTFSS           STATUS, Z
        GOTO            R_LCD_DEC_SHOW
        BTFSS           V_NUM7, 4               ; VERIFICA SE DEVE MOSTRAR
        GOTO            R_LCD_DEC_OK

R_LCD_DEC_SHOW
        ADDLW           '0'
```

```
        MOVWF           V_NUM1          ; SALVA DADO EM ASCII
        CALL            R_LCD_BF ; ESPERA O LCD DESOCUPAR
        MOVF            V_NUM1, W
        BSF                     P_RS            ; CONFIGURA COMO DADO
        CALL            R_LCD_WRITE
        BSF                     V_NUM7, 4       ; FLAG TEMPORÁRIO PARA INFORMAR QUE DEVE MOSTRAR ZEROS!

R_LCD_DEC_OK
        DECF            V_NUM7, F
        MOVLW           0X03
        ANDWF           V_NUM7, W
        M_JZ            R_LCD_DEC_FIM

        MOVLW           D'10'
        BTFSS           V_NUM7, 0
        GOTO            R_LCD_DEC_LOOP
        BSF                     V_NUM7, 4       ; FLAG TEMPORÁRIO PARA INFORMAR QUE DEVE MOSTRAR ZEROS!
        MOVLW           D'1'
        GOTO            R_LCD_DEC_LOOP

R_LCD_DEC_FIM
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE ENVIA UMA MENSAGEM PARA O DISPLAY A PARTIR DA POSIÇÃO ATUAL DO CURSOR
;       ENTRADA: W = ENDEREÇO DA MENSAGEM (PRIMEIRO BYTE INFORMA O TAMANHO DA MENSAGEM)
;       SAÍDAS: NENHUMA
;       AFETA: W; STATUS; V_NUM; V_NUM1; *V_W_AUX; *V_LCD; *TRIS_LCD; *CTRL_LCD
;       CHAMA: R_LCD_CHAR (STACK: 4)
;-------------------------------------------------------------------------------------------------

R_MENSAGEM
        MOVWF           V_NUM           ; GUARDA ENDEREÇO DA MENSAGEM
        CALL            R_CALL_MSG
        MOVWF           V_NUM1
R_MSG_LOOP
        INCF            V_NUM, F
        MOVF            V_NUM, W
        CALL            R_CALL_MSG
        CALL            R_LCD_CHAR
        DECFSZ          V_NUM1, F
        GOTO            R_MSG_LOOP
        RETURN

; ROTINA AUXILIAR PARA PEGAR CADA CARACTER DA MENSAGEM
R_CALL_MSG
        MOVWF           PCL

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE ENVIA A MENSAGEM DO LOOP PRINCIPAL NA PRIMEIRA LINHA
;       ENTRADA: NENHUMA
;       SAÍDAS: NENHUMA
;       AFETA: W; STATUS; *V_W_AUX; *V_LCD; *TRIS_LCD; *CTRL_LCD
;                       *V_NUM; *V_NUM1; *V_NUM2; *V_NUM3; *V_NUM4; *V_NUM5; *V_NUM6
;       CHAMA: R_MENSAGEM (5), R_LCD_CHAR; R_LCD_CMD; R_BIN_TO_BCD; R_LCD_BCD (STACK: 5)
;-------------------------------------------------------------------------------------------------

R_MSG_INICIO
        MOVLW           LCD_L0
        CALL            R_LCD_CMD       ; POSICIONA NA PRIMEIRA LINHA

        M_SHOW_MSG      MSG_INICIO      ; MOSTRA A MENSAGEM DO LOOP PRINCIPAL

; MOSTRA O HORÁRIO NO FORMATO HH:MM:SS
        MOVLW           LCD_L0 + D'8'
        CALL            R_LCD_CMD

        MOVF            V_HOR, W
        CALL            R_BIN_TO_BCD
        CALL            R_LCD_BCD
        MOVLW           ':'
        CALL            R_LCD_CHAR
        MOVF            V_MIN, W
        CALL            R_BIN_TO_BCD
        CALL            R_LCD_BCD
        MOVLW           ':'
        CALL            R_LCD_CHAR
        MOVF            V_SEG, W
        CALL            R_BIN_TO_BCD
        CALL            R_LCD_BCD
        RETURN

;-------------------------------------------------------------------------------------------------
```

```
;       ROTINA QUE ENVIA A MENSAGEM QUE PERGUNTA SE LIGA OU DESLIGA O MOTOR
;       ENTRADA: NENHUMA
;       SAÍDAS: NENHUMA
;       AFETA: W; STATUS; *V_W_AUX; *V_LCD; *TRIS_LCD; *CTRL_LCD; *V_NUM; *V_NUM1
;       CHAMA: R_MENSAGEM (5), R_LCD_CHAR; R_LCD_CMD (STACK: 5)
;-------------------------------------------------------------------------------------------------

R_MSG_MOTOR
        MOVLW           LCD_CLR
        CALL            R_LCD_CMD       ; LIMPA O DISPLAY

        M_SHOW_MSG      MSG_CONFIRMA    ; MOSTRA A MENSAGEM PARA CONFIRMAR OPERAÇÃO

        MOVLW           LCD_L1
        CALL            R_LCD_CMD       ; POSICIONA NA SEGUNDA LINHA

        MOVLW           LCD_L1 + 3
        CALL            R_LCD_CMD       ; POSICIONA NA SEGUNDA LINHA PARA ESCREVER "LIGAR?"
        M_SHOW_MSG      MSG_MOTOR       ; MOSTRA A MENSAGEM PARA LIGAR OU DESLIGAR O MOTOR

        BTFSS           P_MOTOR                         ; VERIFICA SE O MOTOR ESTÁ LIGADO
        GOTO            R_MSG_MOT_FIM           ; SALTA, POIS ESTÁ DESLIGADO E DEVE LIGÁ-LO MESMO
        MOVLW           LCD_L1
        CALL            R_LCD_CMD       ; POSICIONA NA SEGUNDA LINHA PARA ESCREVER "DES"
        MOVLW           'D'
        CALL            R_LCD_CHAR
        MOVLW           'E'
        CALL            R_LCD_CHAR
        MOVLW           'S'
        CALL            R_LCD_CHAR

R_MSG_MOT_FIM
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA DE INTERFACE PARA AJUSTAR OS VALORES DE CONFIGURAÇÃO NA EEPROM
;       ENTRADA: W = ENDEREÇO DA EEPROM A SER AJUSTADO
;       SAÍDAS: NENHUMA
;       AFETA: W; STATUS; V_NUM2; *V_W_AUX; *V_LCD; *TRIS_LCD; *CTRL_LCD; *V_NUM; *V_NUM1;
;                       *V_NUM3; *V_NUM4; *V_NUM6; *V_NUM7
;       CHAMA: R_MENSAGEM (5), R_TECLA_WAIT (2); R_EE_READ; R_EE_WRITE;
;                       R_LCD_CHAR (3); R_LCD_CMD; R_LCD_DECIMAL (STACK: 6)
;-------------------------------------------------------------------------------------------------

R_AJUSTA_EEPROM
        MOVWF           V_EE_OP                         ; SALVA ENDEREÇO DA EEPROM

        BSF                     STATUS, RP1                     ; MUDA PARA O BANCO 2
        MOVWF           EEADR                           ; GRAVA ENDEREÇO DA EEPROM
        CLRF            EEADRH                          ; GARANTE ENDEREÇO VÁLIDO
        BCF                     STATUS, RP1                     ; VOLTA PARA O BANCO 0

        MOVWF           V_EE_OP                 ; SALVA ENDEREÇO DA EEPROM
        MOVLW           LCD_CLR
        CALL            R_LCD_CMD       ; LIMPA O DISPLAY

        MOVLW           E_ID
        XORWF           V_EE_OP, W
        M_JNZ           R_AJT_EE_DELTA
        MOVLW           MSG_ID
        GOTO            R_AJT_EE_VALOR

R_AJT_EE_DELTA
        MOVLW           E_DELTA
        XORWF           V_EE_OP, W
        M_JNZ           R_AJT_EE_MENOR
        MOVLW           MSG_DELTA
        GOTO            R_AJT_EE_VALOR

R_AJT_EE_MENOR
        MOVLW           E_MENOR
        XORWF           V_EE_OP, W
        M_JNZ           R_AJT_EE_NOMINAL
        MOVLW           MSG_MENOR
        GOTO            R_AJT_EE_VALOR

R_AJT_EE_NOMINAL
        MOVLW           E_NOMINAL
        XORWF           V_EE_OP, W
        M_JNZ           R_AJT_EE_PARTIDA
        MOVLW           MSG_NOMINAL
        GOTO            R_AJT_EE_VALOR
```

```
R_AJT_EE_PARTIDA
        MOVLW           E_PARTIDA
        XORWF           V_EE_OP, W
        M_JNZ           R_AJT_EE_UNIDADE
        MOVLW           MSG_PARTIDA
        GOTO            R_AJT_EE_VALOR

R_AJT_EE_UNIDADE
        MOVLW           E_UNIDADE
        XORWF           V_EE_OP, W
        M_JNZ           R_AJT_EE_I_UNIDADE
        MOVLW           MSG_UNIDADE
        GOTO            R_AJT_EE_VALOR

R_AJT_EE_I_UNIDADE
        MOVLW           E_I_UNIDADE
        XORWF           V_EE_OP, W
        M_JNZ           R_AJT_EE_FIM
        MOVLW           MSG_I_UNIDADE
        GOTO            R_AJT_EE_VALOR

R_AJT_EE_VALOR
        CALL            R_MENSAGEM              ; MOSTRA A MENSAGEM EQUIVALENTE
        MOVLW           ':'
        CALL            R_LCD_CHAR

        CALL            R_EE_READ               ; CHAMA ROTINA DE LEITURA DA EEPROM
        MOVWF           V_EE_VALOR              ; SALVA VALOR DA EEPROM

; INICIA O AJUSTE DO VALOR PROPRIAMENTE DITO
R_AJT_EE_LOOP
; LIMPA 3 CARACTERES DO DISPLAY
        MOVLW           LCD_L1
        CALL            R_LCD_CMD
        MOVLW           ' '
        CALL            R_LCD_CHAR
        CALL            R_LCD_CHAR
        CALL            R_LCD_CHAR
        MOVLW           LCD_L1
        CALL            R_LCD_CMD
; MOSTRA O VALOR ATUAL DA VARIÁVEL
        MOVF            V_EE_VALOR, W
        CALL            R_LCD_DECIMAL

        CALL            R_TECLA_WAIT
        BTFSC           V_TECLA, TEC_OK
        GOTO            R_AJT_EE_GRAVA          ; FOI TECLA OK
        BTFSS           V_TECLA, TEC_MENU       ; SE FOI MENU OU NADA, ENTÃO SAI
        BTFSC           STATUS, Z
        GOTO            R_AJT_EE_FIM            ; NADA TECLADO POR MUITO TEMPO, ENTÃO SAI DO LOOP

; DEVE INCREMENTAR / DECREMENTAR O VALOR
        BTFSC           V_TECLA, TEC_UP
        GOTO            R_AJT_EE_INC            ; DEVE INCREMENTAR
        DECF            V_EE_VALOR, F
        GOTO            R_AJT_EE_LOOP
R_AJT_EE_INC
        INCF            V_EE_VALOR, F
        GOTO            R_AJT_EE_LOOP

R_AJT_EE_GRAVA
        MOVF            V_EE_VALOR, W
        BSF                     STATUS, RP1                             ; MUDA PARA O BANCO 2
        MOVWF           EEDATA
        CALL            R_EE_WRITE                      ; CHAMA ROTINA DE ESCRITA DA EEPROM

R_AJT_EE_FIM
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE LÊ AS 3 CORRENTES DE FASES DO ADC E MOSTRA A CORRENTE SELECIONADA NO LCD
;       ENTRADA: V_I_PTR = END. DA VARIÁVEL COM O VALOR DA CORRENTE A SER MOSTRADA
;       SAÍDAS: NENHUMA
;       AFETA: W; STATUS; FSR; V_NUM; V_NUM7; *V_W_AUX; V_LCD; *TRIS_LCD; *CTRL_LCD;
;                       V_NUM1; V_NUM2; *V_NUM; *V_NUM3; *V_NUM4; *V_NUM5; *V_NUM6
;       CHAMA: R_ADC_READ; R_LCD_DECIMAL; R_LCD_CHAR; R_LCD_CMD; R_DIV8; R_MUL8 (STACK: 4)
;-------------------------------------------------------------------------------------------------

R_LCD_I
        MOVLW           LCD_L1
        CALL            R_LCD_CMD

        MOVLW           'I'
```

```
        CALL            R_LCD_CHAR
        MOVLW           V_I1
        SUBWF           V_I_PTR, W
        ADDLW           '1'                             ; DETERMINA A CORRENTE FASE A SER LIDA
        CALL            R_LCD_CHAR
        MOVLW           '='
        CALL            R_LCD_CHAR

        M_MOVLF         V_NUM1, 3
R_LCD_I_LOOP
        BSF                     STATUS, C
        CALL            R_I_PTR                 ; INCREMENTA O PONTEIRO DA CORRENTE ATUAL
        MOVF            V_I_PTR, W
        MOVWF           FSR                             ; GUARDA ENDEREÇO DA FASE ATUAL EM FSR
        MOVLW           V_I1
        SUBWF           V_I_PTR, W              ; DETERMINA O CORRENTE CANAL A SER LIDO
        CALL            R_ADC_READ              ; LÊ O ADC
        MOVWF           INDF                    ; SALVA VALOR ATUAL
        DECFSZ          V_NUM1
        GOTO            R_LCD_I_LOOP

; MOSTRA A CORRENTE CALCULADA NO LCD
        MOVWF           V_NUM1
        MOVF            V_DELTA, W
        MOVWF           V_NUM2                  ; MULTIPLICADOR DELTA
        CALL            R_MUL8
        MOVF            V_NUM3, W
        MOVWF           V_NUM1                  ; SALVA RESULTADO MENOS SIGNIFICATIVO
        MOVF            V_NUM4, W
        ADDWF           V_MENOR, W              ;  SOMA  MAIS  SIGNIFICATIVO  COM  O  MENOR  VALOR  DA  ESCALA
(OFFSET)
        MOVWF           V_NUM                   ; SALVA RESULTADO DA SOMA NO MAIS SIGNIFICATIVO

; FAZ O AJUSTE DA UNIDADE DA CORRENTE
        M_EE_READ       E_I_UNIDADE             ; PEGA O COEFICIENTE DA UNIDADE DA CORRENTE
        MOVWF           V_NUM2
        CALL            R_DIV8                  ; DIVIDE O VALOR I * DELTA + MENOR PELO COEFICIENTE
        MOVF            V_NUM4, W
        MOVWF           V_I_AUX                 ; SALVA BYTE MENOS SIGNIFICATIVO
        MOVF            V_NUM5, W               ; PEGA O BYTE MAIS SIGNIFICATIVO
        CALL            R_LCD_DECIMAL   ; MOSTRA PARTE INTEIRA DO RESULTADO

; MOSTRA AS CASAS DECIMAIS
        MOVLW           ','
        CALL            R_LCD_CHAR
        MOVLW           D'10'
        MOVWF           V_NUM2                  ; PREPARA PARA MOSTRAR A PARTE DECIMAL
        MOVF            V_I_AUX, W              ; PEGA PARTE BAIXA
        MOVWF           V_NUM1
        CALL            R_MUL8
        MOVLW           '0'
        ADDWF           V_NUM4, W               ; CONVERTE PARA ASCII
        CALL            R_LCD_CHAR

        MOVLW           D'10'
        MOVWF           V_NUM2                  ; PREPARA PARA MOSTRAR A PARTE DECIMAL
        MOVF            V_NUM3, W               ; PEGA PARTE BAIXA
        MOVWF           V_NUM1
        CALL            R_MUL8
        MOVLW           '0'
        ADDWF           V_NUM4, W               ; CONVERTE PARA ASCII
        CALL            R_LCD_CHAR

        MOVLW           'A'
        CALL            R_LCD_CHAR
        MOVLW           ' '
        CALL            R_LCD_CHAR
        MOVLW           '('
        CALL            R_LCD_CHAR
        MOVF            INDF, W
        CALL            R_LCD_DECIMAL
        MOVLW           ')'
        CALL            R_LCD_CHAR
        MOVLW           ' '
        CALL            R_LCD_CHAR
        CALL            R_LCD_CHAR
        CALL            R_LCD_CHAR
        CALL            R_LCD_CHAR
        CALL            R_LCD_CHAR

        RETURN

;-------------------------------------------------------------------------------------------------
```

```
;       ROTINA QUE INC/DEC PONTEIRO DA CORRENTE
;       ENTRADA: V_I_PTR = END. DA VARIÁVEL COM O VALOR DA CORRENTE; C = 1 P/ INCREMENTAR
;       SAÍDAS: NENHUMA
;       AFETA: W; STATUS; V_I_PTR
;       CHAMA: NADA (STACK: 1)
;-------------------------------------------------------------------------------------------------

R_I_PTR
        M_JC            R_I_PTR_INC
        DECF            V_I_PTR, F
        MOVLW           V_I1 - 1
        XORWF           V_I_PTR, W
        M_JNZ           R_I_PTR_END
        MOVLW           V_I3
        MOVWF           V_I_PTR
        GOTO            R_I_PTR_END

R_I_PTR_INC
        INCF            V_I_PTR, F
        MOVLW           V_I3 + 1
        XORWF           V_I_PTR, W
        M_JNZ           R_I_PTR_END
        MOVLW           V_I1
        MOVWF           V_I_PTR

R_I_PTR_END
        RETURN

;-------------------------------------------------------------------------------------------------
;       ROTINA QUE VERIFICA SE HÁ SOBRECORRENTE
;       ENTRADA: V_I_PTR = END. DA VARIÁVEL COM O VALOR DA CORRENTE
;       SAÍDAS: V_TEMPO_MAX; V_FAIXA; V_TEMPO SÃO ATUALIZADOS
;       AFETA: W; STATUS; V_TEMPO_MAX; V_FAIXA; V_TEMPO
;       CHAMA: NADA (STACK: 1)
;-------------------------------------------------------------------------------------------------

R_TESTA_I
        MOVF            V_I_PTR, W              ; W = ENDEREÇO DA CORRENTE DA FASE ATUAL
        MOVWF           FSR                             ; GUARDA ENDEREÇO

        BTFSS           P_MOTOR                 ; VERIFICA SE O MOTOR ESTÁ LIGADO
        GOTO            R_TST_I_CLR             ; LIMPA TODAS AS VARIÁVEIS INTERNAS E SAI
        MOVF            V_PARTIDA, F
        BTFSS           STATUS, Z               ; SE ZERO NÃO ESTÁ EM PARTIDA
        GOTO            R_TST_I_CLR             ; LIMPA TODAS AS VARIÁVEIS INTERNAS E SAI

        BSF                     STATUS, RP1             ; MUDA PARA O BANK 2
        MOVLW           E_MAX                   ; PEGA O ENDEREÇO BASE DAS CORRENTES
        MOVWF           EEADR                   ; APONTA PARA ESTE ENDEREÇO NA EEPROM
        BCF                     STATUS, RP1             ; MUDA PARA O BANK 0

        MOVLW           K_FAIXA
        MOVWF           V_NUM                   ; PREPARA CONTADOR
R_TST_I_LOOP
        CALL            R_EE_READ
        SUBWF           INDF, W
        BTFSC           STATUS, C
        GOTO            R_TST_I_OK              ; HÁ SOBRECORRENTE E LOCALIZOU O SEU TEMPO!

        BSF                     STATUS, RP1             ; MUDA PARA O BANK 2
        INCF            EEADR                   ; APONTA PARA O TEMPO MÁXIMO
        INCF            EEADR                   ; APONTA PARA A PRÓXIMA CORRENTE CADASTRADA
        BCF                     STATUS, RP1             ; MUDA PARA O BANK 0

        DECFSZ          V_NUM                   ; CONTA OS 8 CADASTROS EXISTENTES
        GOTO            R_TST_I_LOOP    ; VARRE TODAS AS FAIXAS GRAVADAS

R_TST_I_CLR
; LIMPA TODAS AS VARIÁVEIS INTERNAS (MENOS O FLAG DE SOBRECORRENTE)
        CLRF            V_TEMPO                 ; NÃO HÁ SOBRECORRENTE!
        CLRF            V_TEMPO_MAX
        BCF                     B_1CS
        BCF                     B_1S
        BCF                     B_M_ALERTA              ; LIMPA O ALERTA DA VARIÁVEL
        GOTO            R_TST_I_FIM             ; NÃO HÁ SOBRECORRENTE!

R_TST_I_OK
        BSF                     B_M_ALERTA              ; FLAG PARA AVISAR AO PC QUE HÁ UM ALERTA NO
MOTOR

; CARREGA DADOS O TEMPO MÁXIMO DA SOBRECORRENTE LOCALIZADA!
        BSF                     STATUS, RP1             ; MUDA PARA O BANK 2
        INCF            EEADR                   ; APONTA PARA O TEMPO MÁXIMO
```

```
        CALL            R_EE_READ
        MOVWF           V_TEMPO_MAX             ; GUARDA O TEMPO MÁXIMO DESTA SOBRECORRENTE

; CALCULA A FAIXA E VERIFICA SE DEVE ZERAR O CONTADOR
        MOVLW           K_FAIXA/2 + 1
        SUBWF           V_NUM, W        ; CALCULA A FAIXA
        BTFSC           STATUS, C
        GOTO            R_TST_I_1CS             ; ESTÁ NA FAIXA DE CENTÉSIMOS DE SEGUNDOS

; OBSERVOU QUE A FAIXA É DE SEGUNDOS
        BTFSC           B_1S                    ; VERIFICA SE A FAIXA ANTERIOR ERA A MESMA
        GOTO            R_TST_I_FIM
        CLRF            V_TEMPO                 ; LIMPA CONTADOR SE A FAIXA MUDOU!
        BCF                     B_1CS
        BSF                     B_1S                    ; ACIONA O FLAG DA FAIXA
        GOTO            R_TST_I_FIM

R_TST_I_1CS
; OBSERVOU QUE A FAIXA É DE CENTÉSIMOS DE SEGUNDOS
        BTFSC           B_1CS                   ; VERIFICA SE A FAIXA ANTERIOR ERA A MESMA
        GOTO            R_TST_I_FIM
        CLRF            V_TEMPO                 ; LIMPA CONTADOR SE A FAIXA MUDOU!
        BCF                     B_1S
        BSF                     B_1CS                   ; ACIONA O FLAG DA FAIXA

R_TST_I_FIM
        RETURN

        END



;***********************************************************************************************
;       DEFINIÇÕES DAS CONSTANTES
;       Arquivo: Main.inc
;***********************************************************************************************

;===============================================================================================
;       TIMER2

#DEFINE  K_T2CON          B'00100101'             ; TIMER2 LIGADO COM PRE=4 E POST=5
#DEFINE  K_PR2            D'249'                  ; O TIMER2 IRÁ CONTAR DE 0 ATÉ 249 (250 VEZES)

;===============================================================================================
;       CONSTANTES DE INICIALIZAÇÃO

#DEFINE  K_TRISA          B'11101111'             ; SELECIONA COMO I/O (=1 -> INPUT / =0 -> OUTPUT)
#DEFINE  K_TRISB          B'11110000'             ; SELECIONA COMO I/O (=1 -> INPUT / =0 -> OUTPUT)
#DEFINE  K_TRISC          B'11111110'             ; SELECIONA COMO I/O (=1 -> INPUT / =0 -> OUTPUT)

#DEFINE  K_PORTA          0XFF                    ; UTILIZADO PARA CONTROLE DOS PERIFÉRICOS
#DEFINE  K_PORTB          0X00                    ; UTILIZADO PARA I/O DE DADOS E COMUNICAÇÃO SERIAL
#DEFINE  K_PORTC          0X00                    ; UTILIZADO PARA I/O DE DADOS E COMUNICAÇÃO SERIAL

#DEFINE  K_OPTION B'10000111'             ; DESABILITA PULL UP DO PORTB E PREESCALA PARA TIMER 0
#DEFINE  K_INTCON B'11000000'             ; HABILITA INTERRUPÇÕES
#DEFINE  K_PIE1           B'00100010'             ; HABILITA A INT SERIAL DE RECEPÇÃO E TIMER2 (BANK 1)

#DEFINE  K_ADCON0 B'10000001'             ; FREQ. DE CONVERSÃO E CANAL ESCOLHIDO
#DEFINE  K_ADCON1 B'00001011'             ; 4/2 CANAIS

;===============================================================================================
;       TECLADO

#DEFINE TRIS_TEC TRISB
#DEFINE PORT_TEC PORTB
#DEFINE P_NTEC           PORTA, 4

#DEFINE  TEC_MENU 7
#DEFINE  TEC_OK           6
#DEFINE  TEC_UP           5
#DEFINE  TEC_DOWN 4

#DEFINE  P_MENU           PORT_TEC, TEC_MENU
#DEFINE  P_OK             PORT_TEC, TEC_OK
#DEFINE  P_UP             PORT_TEC, TEC_UP
#DEFINE  P_DOWN           PORT_TEC, TEC_DOWN

#DEFINE  K_TECS           B'11110000'             ; TODOS OS BITS SETADOS = NENHUMA TECLA

;===============================================================================================
;       LCD MODULE
```

```
#DEFINE TRIS_LCD          TRISB
#DEFINE  TRIS_LCD_OUT     0X0F
#DEFINE TRIS_LCD_IN               0XF0             ; CONSTANTES UTILIZADAS PARA CONFIGURAR DIREÇÃO DO I/O
#DEFINE PORT_LCD          PORTB

#DEFINE  P_RS     PORTB, 1
#DEFINE  P_READ   PORTB, 2
#DEFINE  P_EN     PORTB, 3

#DEFINE K_DATA_BASE       4                                                ; OFFSET PARA O PRIMEIRO BIT DO
BARRAMENTO DE DADOS
#DEFINE P_D0     PORT_LCD, K_DATA_BASE + 0; BIT MENOS SIGNIFICATIVO DO BARRAMENTO DE DADOS
#DEFINE P_D1     PORT_LCD, K_DATA_BASE + 1
#DEFINE P_D2     PORT_LCD, K_DATA_BASE + 2
#DEFINE P_D3     PORT_LCD, K_DATA_BASE + 3; BIT MAIS SIGNIFICATIVO DO BARRAMENTO DE DADOS

#DEFINE  K_DATA           B'11110000'              ; TODOS OS BITS UTILIZADOS NO BARRAMENTO DE DADOS DO LCD
#DEFINE K_NDATA           B'00001111'              ; TODOS OS DEMAIS BITS DA PORTA QUE NÃO É DADOS DO LCD

#DEFINE  LCD_CFG                  0X28
#DEFINE  LCD_OFF                  0X08             ; DESLIGA LCD
#DEFINE  LCD_ON                   0X0C             ; LIGA LCD E DESLIGA O CURSOR!
#DEFINE  LCD_CUR_ON               0X0E             ; LIGA CURSOR
#DEFINE  LCD_CURSOR               0X0D             ; ATIVA CURSOR "_"
#DEFINE  LCD_CUR_INPUT    0X0F            ; ATIVA CURSOR ALTERNANTE "X" E "_"
#DEFINE  LCD_CUR_HOME     0X02
#DEFINE  LCD_CUR_R                0X06             ; DESLOCA O CURSOR PARA DIREITA APÓS ENTRAR DADOS
#DEFINE  LCD_CUR_L                0X04             ; DESLOCA O CURSOR PARA ESQUERDA APÓS ENTRAR DADOS
#DEFINE  LCD_CLR                  0X01
#DEFINE LCD_L0                    0X80
#DEFINE LCD_L1                    0XC0

;=============================================================================================
;        SERIAL

#DEFINE  K_TXSTA          B'01100100'              ; STATUS DA TRANSMISSÃO SERIAL
#DEFINE  K_RCSTA          B'11010000'              ; STATUS DA RECEPÇÃO SERIAL
#DEFINE  K_SPBRG          D'129'                   ; 129 = 9600 OU 64 = 19200 (0,16%)

#DEFINE  SER_COUNT        0X10                     ; VEZES QUE PASSA PELA LE_TEC E ESPERA PRÓXIMO BYTE

;        BUFFER SERIAL
#DEFINE  SER_EDATA        D'32'                    ; 32 BYTES DE DADOS DA EEPROM DENTRO DO PACOTE
#DEFINE  SER_SIZE D'48'                    ; TOTAL DE BYTES DE UM PACOTE DE COMUNICAÇÃO

#DEFINE  SER_ID           0X120 + 0X00                     ; ENDEREÇO DO ID NO PACOTE
#DEFINE  SER_CMD          SER_ID + D'1'                    ; END. DO COMANDO NO PACOTE
#DEFINE  SER_INFO SER_ID + D'2'                    ; END. DO ERRO (PIC -> PC) NÃO UTILIZADO (PC -> PIC)
#DEFINE  SER_EE           SER_ID + D'3'                    ; END. BASE DA EEPROM
#DEFINE  SER_FASE SER_ID + D'4'                    ; END. DA PRIMERIA FASE
#DEFINE  SER_TIME SER_ID + D'7'                    ; END. DO PRIMEIRO BYTE DO HORÁRIO (SEG;MIN;HOR)
#DEFINE  SER_RES          SER_ID + D'10'                   ;  END.  RESERVADOS  PARA  FUTURAS  IMPLEMENTAÇÕES
(10..14)
#DEFINE  SER_EDATA0       SER_ID + D'15'                   ; END. DO PRIMEITO BYTE DE DADOS DO PACOTE
#DEFINE  SER_TAM          SER_ID + SER_SIZE - 1    ; END. DO BYTE DE TESTE (= TAMANHO DO PACOTE)

#DEFINE  SER_END          SER_TAM + 1                              ; PRIMEIRO ENDEREÇO INVÁLIDO (P/ TESTAR
ENDEREÇO!)

; ERROS DA SERIAL (PARA O BYTE INFO)
#DEFINE  B_ERRO_OVER              INDF, 0  ; ERRO DE ESTROURO DE BYTES RECEBIDOS
#DEFINE  B_ERRO_BREAK     INDF, 1  ; ERRO DE QUEBRA DE RECEPÇÃO
#DEFINE  B_ERRO_CMD               INDF, 2  ; ERRO DE COMANDO INVÁLIDO
#DEFINE  B_ERRO_EEPROM    INDF, 3  ; ERRO DE ESCRITA NA EEPROM
#DEFINE  B_ERRO_TAM               INDF, 4  ; ERRO DO TAMANHO DO PACOTE
#DEFINE B_ERRO_ALERTA     INDF, 5  ; ESTÁ EM ALERTA DE SOBRECORRENTE
#DEFINE B_ERRO_SOBRE      INDF, 6  ; HOUVE UMA SOBRECORRENTE

; COMANDOS DA SERIAL
#DEFINE  SER_CMD_TST      B'00110110'              ; TODO COMANDO ENVIADO DEVE TER ESTES BITS ZERADOS
#DEFINE  B_TIME_WR        SER_CMD, 7               ;  BIT  RESPONSÁVEL  PELA  ATUALIZAÇÃO  DO  HORÁRIO  (=1-
>ATUALIZA)
#DEFINE B_EE_WR           SER_CMD, 6               ; BIT QUE INFORMA SE DEVE ESCREVER NA EEPROM
#DEFINE B_MOTOR_SET       SER_CMD, 3               ; BIT QUE INFORMA SE DEVE ESCREVER O NOVO STATUS DO MOTOR
#DEFINE  B_MOTOR_ON       SER_CMD, 0               ; BIT QUE INFORMA O STATUS DO MOTOR (LIGADO / DESLIGADO)

; ENDEREÇOS DA EEPROM PARA EVITAR QUE SOBREESCREVE DADOS ERRADOS
#DEFINE  SER_EE_ADDR      B'11110000'              ; ENDEREÇO INICIAL DOS BLOCOS DE 16 BYTES DA EEPROM

;=============================================================================================
;        ACIONAMENTO DO MOTOR

#DEFINE  P_MOTOR  PORTB, 0
```

```
;*************************************************************************************************
;       VARIÁVEIS DA RAM
;*************************************************************************************************

V_BANK0          EQU     0X20                    ; INÍCIO DA RAM DO BANK 0 (80 BYTES)
V_BANK1          EQU     0XA0                    ; INÍCIO DA RAM DO BANK 1 (80 BYTES)
V_BANK2          EQU     0X120                   ;  INÍCIO  DA  RAM  DO  BANK  2  (80  +  16  BYTES)  APENAS
16F876/77!!!
V_BANK3          EQU     0X1A0                   ;  INÍCIO  DA  RAM  DO  BANK  3  (80  +  16  BYTES)  APENAS
16F876/77!!!
V_BANKS          EQU     0XF0                    ;  INÍCIO  DA  RAM  PARA  QUALQUER  BANK  (16  BYTES)  APENAS
16F876/77!!!

;-------------------------------------------------------------------------------------------------
;       VARIÁVEIS COM ACESSO ATRAVÉS DO BANK 0
; VARIÁVEIS DE CONTROLE
V_TECLA          EQU     V_BANK0 + 0X00   ; VARIÁVEL DE LEITURA DA TECLA
V_WAIT           EQU     V_BANK0 + 0X01   ; CONTADOR PARA ATRASO EM ms DA ROTINA R_WAIT
V_LCD            EQU     V_BANK0 + 0X02   ; BYTE CONTENDO O DADO/COMANDO DO/PARA LCD
V_I_PTR          EQU     V_BANK0 + 0X03   ; BYTE COM O ENDEREÇO DA CORRENTE MOSTRADA NO LCD
; BYTES PARA AS FUNÇÕES MATEMÁTICAS
V_NUM            EQU     V_BANK0 + 0X04   ; BYTE UTILIZADO NAS ROTINAS MATEMÁTICAS
V_NUM1           EQU     V_BANK0 + 0X05   ; BYTE UTILIZADO NAS ROTINAS MATEMÁTICAS
V_NUM2           EQU     V_BANK0 + 0X06   ; BYTE UTILIZADO NAS ROTINAS MATEMÁTICAS
V_NUM3           EQU     V_BANK0 + 0X07   ; BYTE UTILIZADO NAS ROTINAS MATEMÁTICAS
V_NUM4           EQU     V_BANK0 + 0X08   ; BYTE UTILIZADO NAS ROTINAS MATEMÁTICAS
V_NUM5           EQU     V_BANK0 + 0X09   ; BYTE UTILIZADO NAS ROTINAS MATEMÁTICAS
V_NUM6           EQU     V_BANK0 + 0X0A   ; BYTE UTILIZADO NAS ROTINAS MATEMÁTICAS
V_NUM7           EQU     V_BANK0 + 0X0B   ; BYTE UTILIZADO NAS ROTINAS MATEMÁTICAS
; BYTES PARA GUARDAR AS ÚLTIMAS LEITURAS DO ADC
V_I1             EQU     V_BANK0 + 0X0C   ; CORRENTE DA PRIMEIRA FASE DO MOTOR (END.= BASE + 8)
V_I2             EQU     V_BANK0 + 0X0D   ; CORRENTE DA SEGUNDA FASE DO MOTOR
V_I3             EQU     V_BANK0 + 0X0E   ; CORRENTE DA TERCEIRA FASE DO MOTOR
; BYTES PARA A CONTAGEM DO TEMPO
V_MS             EQU     V_BANK0 + 0X0F   ; BYTE QUE CONTA MILÉZIMOS DE SEGUNDOS (0..9)
V_CS             EQU     V_BANK0 + 0X10   ; BYTE QUE CONTA OS CENTÉZIMOS DE SEGUNDOS (0..99)
V_SEG            EQU     V_BANK0 + 0X11   ; BYTE QUE CONTA OS SEGUNDOS (0..59)
V_MIN            EQU     V_BANK0 + 0X12   ; BYTE QUE CONTA OS MINUTOS (0..59)
V_HOR            EQU     V_BANK0 + 0X13   ; BYTE QUE CONTA AS HORAS (0..23)
; BYTES PARA A CONVERSÃO DO ADC
V_DELTA          EQU     V_BANK0 + 0X14   ; ESPELHO DO E_DELTA
V_MENOR          EQU     V_BANK0 + 0X15   ; ESPELHO DO E_MENOR
; BYTES PARA A CONTAGEM DE TEMPO
V_PARTIDA        EQU     V_BANK0 + 0X16   ; CONTADOR DO TEMPO DE PARTIDA DO MOTOR
V_UNIDADE        EQU     V_BANK0 + 0X17   ; INDICADOR DA UNIDADE DO TEMPO DE PARTIDA DO MOTOR
V_TEMPO_MAX      EQU     V_BANK0 + 0X18   ; CONTÉM O TEMPO MÁXIMO QUE O RELE DEVE MANTER O MOTOR LIGADO
V_TEMPO          EQU     V_BANK0 + 0X19   ; CONTÉM A CONTAGEM DE TEMPO (TEMPO QUE JÁ PASSOU)
V_TEC_TEMPO      EQU     V_BANK0 + 0X1A   ; CONTÉM O CONTADOR DE TEMPO PARA A ROTINA DE LEITURA DA TECLA
V_FAIXA          EQU     V_BANK0 + 0X1B   ; CONTÉM O PONTEIRO PARA A FAIXA DA SOBRECORRENTE
; DEFINIÇÕES SOBRE A FAIXA E OS TEMPOS DAS CORRENTES (NIBBLE INF. SÓ DE FLAGS P/ OS CONTADORES)
#DEFINE  K_FAIXA         D'16'                   ; NÚMERO DE SOBRECORRENTES CADASTRADAS NA EEPROM
#DEFINE  B_1CS_P         V_FAIXA, 0              ;  INDICA  QUE  A  FAIXA  ESTÁ  EM  CENTÉSIMOS  DE  SEGUNDOS
(PARTIDA)
#DEFINE  B_1S_P          V_FAIXA, 1              ; INDICA QUE A FAIXA ESTÁ EM SEGUNDOS (PARTIDA)
#DEFINE  B_1CS           V_FAIXA, 2              ; INDICA QUE A FAIXA ESTÁ EM CENTÉSIMOS DE SEGUNDOS
#DEFINE  B_1S            V_FAIXA, 3              ; INDICA QUE A FAIXA ESTÁ EM SEGUNDOS
#DEFINE  B_INC_TEMP      V_FAIXA, 4              ; INDICA QUE DEVE INCREMENTAR O CONTADOR DE TEMPO
; VARIÁVEIS AUXILIARES DE USO GERAL
V_I_AUX          EQU     V_BANK0  + 0X1C  ; CONTÉM A PARTE BAIXA DO CÁLCULO DA CORRENTE EM R_LCD_I
V_EE_VALOR       EQU     V_BANK0 + 0X1D   ; CONTÉM UM VALOR ESPELHO DA EEPROM UTILIZADO EM R_AJUSTA_EEPROM
V_EE_OP          EQU     V_BANK0 + 0X1E   ; CONTÉM O ENDEREÇO (OPÇÃO DA EEPROM) EM R_AJUSTA_EERPOM

;-------------------------------------------------------------------------------------------------
; VARIÁVEIS COM ACESSO INDEPENDENTE DO BANCO (16 BYTES)
V_W_TEMP EQU     V_BANKS  + 0x00   ; WREG TEMPORÁRIO PARA INTERRUPÇÕES
V_S_TEMP EQU     V_BANKS + 0x01   ; STATUS TEMPORÁRIO PARA INTERRUPÇÕES
V_FSR_TEMP       EQU     V_BANKS + 0x02   ; FSR TEMPORÁRIO PARA INTERRUPÇÕES
V_W_AUX          EQU     V_BANKS + 0x03   ; VARIÁVEL AUXILIAR PARA O W (DESTRUÍDO NAS SUBROTINAS)
; VARIÁVEIS GLOBAIS PARA COMUNICAÇÃO SERIAL
V_SER_PTR        EQU     V_BANKS + 0x04   ; PONTEIRO PARA BUFFER DA SERIAL
V_SER_OK EQU     V_BANKS + 0x05   ; CONTADOR PARA TIME OUT DA SERIAL
V_SER_FLAG       EQU V_BANKS + 0X06       ; CONTÉM BITS DE STATUS PARA SER ENVIADO AO PC
#DEFINE B_M_ALERTA       V_SER_FLAG, 5    ; ESTÁ EM ALERTA DE SOBRECORRENTE
#DEFINE B_M_SOBRE        V_SER_FLAG, 6    ; HOUVE UMA SOBRECORRENTE
; BYTES COM FLAGS PARA USO GERAL NAS ROTINAS
V_FLAGS          EQU     V_BANKS + 0X07   ; FLAGS PARA USO GERAL EM ROTINAS INTERNAS
#DEFINE  B_ID_OK         V_FLAGS, 0       ; FLAG QUE INDICA SE A COMUNICAÇÃO É COM O RELE (1=SIM)


;*************************************************************************************************
;       VARIÁVEIS DA EEPROM
```

```
;*************************************************************************************************

; 32 BYTES: CONFIGURAÇÃO GERAIS (EX.: DA CORRENTE E DO ADC)
E_ID            EQU     0X00            ; BYTE COM IDENTIFICAÇÃO DO RELE
E_DELTA         EQU     0X01            ; VARIAÇÃO MÁXIMA DA LEITURA DO ADC (MÁXIMO VALOR - MÍNIMO VALOR)
E_MENOR         EQU     0X02            ; MENOR VALOR LIDO NO ADC (QUANTO O ADC LER BYTE 0)
E_NOMINAL       EQU     0X03            ; BYTE COM A CORRENTE NOMINAL DO MOTOR (VALOR DE 0 ATÉ 255!)
E_PARTIDA       EQU     0X04            ; TEMPO DE PARTIDA DO MOTOR
E_UNIDADE       EQU     0X05            ; UNIDADE DO TEMPO DE PARTIDA (1=T x 1cs;2=T x 1s)
E_I_UNIDADE     EQU     0X06            ; MULTIPLICADOR DA UNIDADE DA CORRENTE (1, 10 OU 100)

; 32 BYTES: BUFFER COM O CADASTRO DAS MÁXIMAS CORRENTES (16 x 2 = 32 BYTES USADOS)
; PARA OS PRIMEIROS 16 BYTES (8 CORRENTES) A UNIDADE É 1cs, NAS DEMAIS SERÁ 1s
; AS CORRENTES ESTÃO EM ORDEM DECRESCENTE DE CORRENTE E CRESCENTE DE TEMPO
E_MAX           EQU     0X20            ; END. BASE PARA O PRIMEIRO DOS CINCO CADASTROS
E_I_MAX         EQU     D'0'            ; OFFSET PARA A CORRENTE
E_T_MAX         EQU     D'1'            ; OFFSET PARA O TEMPO QUE DEVE ESPERAR ANTES DE DESLIGAR O MOTOR



;*************************************************************************************************
;       MACROS
;*************************************************************************************************

;-------------------------------------------------------------------------------------------------
;       MACRO PARA ATRASOS

M_DELAY MACRO   N
        MOVLW           N
        CALL            R_DELAY         ; TEMPO EM N x 4 us
        ENDM

M_WAIT  MACRO   N
        MOVLW           N
        CALL            R_WAIT          ; TEMPO EM N x 1 ms
        ENDM

;-------------------------------------------------------------------------------------------------
;       MACRO PARA CONTROLE DO MOTOR

M_MOTOR MACRO   LIGA    ; SÓ FUNCIONA NOS BANK 0 E BANK 2
        IF LIGA == 1
;       BSF                     P_MOTOR
        CALL            R_LIGA_MOTOR
        ENDIF
        IF LIGA == 0
        BCF                     P_MOTOR
        ENDIF
        ENDM

;-------------------------------------------------------------------------------------------------
;       MACROS PARA O ADC DESTE PROJETO

M_RD_AD MACRO   CANAL
        MOVLW           CANAL
        CALL            R_RD_AD
        ENDM

;-------------------------------------------------------------------------------------------------
;       MACROS PARA A EEPROM

M_EE_READ       MACRO   ENDERECO
        BSF             STATUS, RP1                             ; MUDA PARA O BANCO 2
        M_MOVLF EEADR, LOW(ENDERECO)
        CLRF    EEADRH                                  ; GARANTE ENDEREÇO VÁLIDO
;       M_MOVLF EEADRH, HIGH(ENDERECO)  ; SELECIONA ENDEREÇO A SER LIDO (NÃO P/ PIC16F87X)
        CALL            R_EE_READ                       ; CHAMA ROTINA DE LEITURA DA EEPROM
        ENDM

M_EE_WRITE      MACRO   ENDERECO, DADO
        BSF             STATUS, RP1                             ; MUDA PARA O BANCO 2
        M_MOVLF EEADR, LOW(ENDERECO)
        CLRF    EEADRH                                  ; GARANTE ENDEREÇO VÁLIDO P/ PIC16F873
;       M_MOVLF EEADRH, HIGH(ENDERECO)  ; SELECIONA ENDEREÇO A SER LIDO (NÃO P/ PIC16F87X)
        M_MOVLF EEDATA, DADO
        CALL            R_EE_WRITE                      ; CHAMA ROTINA DE ESCRITA DA EEPROM
        ENDM


;-------------------------------------------------------------------------------------------------
;       MACROS PARA A FLASH

M_RD_FLASH      MACRO   ENDERECO
```

```
        BSF             STATUS, RP1                     ; MUDA PARA O BANCO 2
        M_MOVLF  EEADR, LOW(ENDERECO)
        M_MOVLF  EEADRH, HIGH(ENDERECO)
        CALL            R_FLASH_READ     ; CHAMA ROTINA DE LEITURA DA FLASH
        ENDM

;-------------------------------------------------------------------------------------------------
;       MACROS PARA O DISPLAY

M_DPY_CHR       MACRO   DADO
        MOVLW           DADO
        CALL            R_LCD_CHAR
        ENDM

M_DPY_CMD       MACRO   DADO
        MOVLW           DADO
        CALL            R_LCD_CMD
        ENDM

M_SHOW_MSG      MACRO   ENDERECO
        MOVLW           ENDERECO
        CALL            R_MENSAGEM
        ENDM
```

Anexo B Atigo apresentado no UNINDU 2005

Veja o arquivo "intelligent relay.pdf".