UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA - UESB
DEPARTAMENTO DE CIÊNCIAS EXATAS - DCE
CURSO DE CIÊNCIA DA COMPUTAÇÃO
(Bacharelado)

O microcontrolador 8051 num sistema de segurança

André Juliano Santos Sampaio

Vitória da Conquista
Março/2004

UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA - UESB
DEPARTAMENTO DE CIÊNCIAS EXATAS - DCE
CURSO DE CIÊNCIA DA COMPUTAÇÃO
(Bacharelado)

O microcontrolador 8051 num sistema de segurança

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA PROJETO DE COMPUTAÇÃO SUPERVISIONADO NO CURSO DE CIÊNCIA DA COMPUTAÇÃO - BACHARELADO

André Juliano Santos Sampaio

Vitória da Conquista
Março/2004

004
5294m

SAMPAIO, André Juliano Santos

O microcontrolador 8051 num sistema de segurança./André Juliano Santos.-Vitória da Conquista: UESB, 2004. 62p.

Monografia (graduação). Orientador Prof. Marco Antônio Dantas Ramos. Universidade Estadual do Sudoeste da Bahia, 2004.

1 - microcontrolador 8051 - Computação. 2 - microcontrolador 8051 - Sistema de segurança. I-A.II-T.

O MICROCONTROLADOR 8051 NUM SISTEMA DE SEGURANÇA

ESTE TRABALHO DE CONCLUSÃO DE CURSO FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA PROJETO DE COMPUTAÇÃO SUPERVISIONADO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

_____

Prof. Marco Antônio Dantas Ramos - Orientador

_____

Prof. Alzira Ferreira da Silva - Coordenadora de PCS

BANCA EXAMINADORA:

_____

Prof. Marco Antônio Dantas Ramos

_____

Prof. Roque Mendes Prado Trindade

Sumário

Introdução

O crescente número de projetos nas diversas áreas de pesquisa do curso de Ciência de Computação da Universidade Estadual do Sudoeste da Bahia - UESB faz com que o número de usuários nos laboratórios aumente na mesma proporção, e em horários muito variados. Tal fator leva a instituição a investir em mão de obra especializada que muitas vezes é subtilizada pois estes profissionais ficam encarregados de organizar o uso dos laboratórios, quando poderiam estar trabalhando no gerenciamento da rede ou auxiliando em outras áreas, como manutenção das máquinas. Outro problema é o horário limitado de utilização que fica amarrado à disponibilidade do funcionário.

O sistema proposto por este trabalho provê uma interface com uma ferramenta em software para programação, sinais de entrada advindos de sensores infravermelhos e sinais de saída destinados ao alarme e aos LEDs. O sistema deverá emitir sinais em situações de alerta, como presença de pessoa em horário não permitido, horário programado incorreto, bateria fraca, etc.

Descrição do projeto

O sistema será composto por um microcontrolador da família 8051 com portal serial RS-232 e pinos de I/O. Dois pirossensores infravermelhos fornecerão a variação dos níveis de infravermelho do ambiente. Entre os sensores e o microcontrolador estarão presentes amplificadores e um conversor analógico-digital.

O microcontrolador, com base nos sinais provenientes dos sensores, será responsável pelo controle dos dispositivos de segurança, os quais serão LEDs e sirenes.

A programação do microcontrolador será feita através de ferramentas CAD. Se buscará desenvolver um hardware de baixo custo que atenda às necessidades dos laboratórios, e que permita um fácil e rápido upgrade.

Abaixo estão relacionados o cronograma bem como o orçamento previsto para uma confecção simples do dispositivo, isto é, uma montagem em placa de circuito impresso dos componentes sem levar em consideração a acomodação desta placa em um gabinete ou caixa adequados.

Cronograma

Atividade	**1º Mês**			**2º Mês**			**3º Mês**			**4º Mês**			**5º Mês**		
Revisão bibliográfica															
Teste de interfaciamento															
Programação															
Prototipagem e testes finais															

Orçamento

Material	**Valor**
Microcontrolador 8051	15,00
Conversor analógico-digital	5,00
Chave analógica	2,00
Pirossensores infravermelho	6,00
Pilhas	3,00
LEDs	1,00
Outros (resistores, capacitores, placa de circuito impresso, conectores, etc.)	38,00
TOTAL	**70,00**

De maneira geral, o projeto procurará atingir os objetivos acima especificados. O restante deste trabalho tratará dos assuntos a seguir: No capítulo 1 serão introduzidos os conceitos de sistemas embarcados e microcontroladores. Serão discutidas as diferenças entre eles bem como as duas principais arquiteturas disponíveis. No capítulo 2 trataremos dos DSP's (Digital Signal Processors ou processadores de sinal digital), suas funções e principais aplicações. No capitulo 3 será dada uma breve visão sobre a utilização atual dos microcontroladores nas mais diversas aplicações. O capítulo 4 mostra o descrição do projeto de aplicação do microcontrolador 8051 no estudo de caso. O capítulo 5 encerra o trabalho apresentando os materiais e os métodos empregados para gerar as simulações e testes.

Capítulo 1 - Sistemas Embarcados e Microcontroladores

Sistemas embarcados

Sistemas embarcados são sistemas eletrônicos computacionais dedicados a realizar tarefas específicas de uma determinada aplicação. Também pode ser definido como sistemas computacionais incluídos em outro sistema. Exemplos de sistemas embarcados são os terminais de caixa eletrônico, máquinas de refrigerante, controladores de temperatura, aparelhos eletrônicos em geral, como televisão, videocassete, microondas, etc. [7]

Não se deve confundir sistemas embarcados com microcontroladores apesar de às vezes ser difícil diferenciá-los. As principais características que devem ser observadas para defini-los corretamente são o tamanho e a complexidade. Microcontroladores geralmente são pequenos e empregam poucas CI's, enquanto que os sistemas embarcados são maiores e mais sofisticados, empregando um número maior de CI's. [7]

Diferente dos computadores de propósito geral, que oferecem funções básicas de processamento que são combinadas permitindo a execução de várias funções distintas utilizando o mesmo hardware, os sistemas embarcados possuem capacidade de processamento bem específicas.

Para o desenvolvimento de um sistema embarcado pode ser construído um hardware dedicado especialmente à aplicação e desse modo toda a computação pode ser feita por hardware ou podem ser utilizados microcontroladores programáveis.

A primeira abordagem possui um custo de desenvolvimento mais elevado e deve ser usada quando fatores como desempenho, volume e peso são muito relevantes para a aplicação.

No caso da segunda opção, um fator a ser observado durante a escolha do microcontrolador para o desenvolvimento de um sistema embarcado é a quantidade de recursos do componente. O ideal é que sejam utilizados microcontroladores com a capacidade mínima para a execução do sistema, minimizando o custo de produção e otimizando o uso dos recursos computacionais disponíveis.

Microcontroladores

Os microcontroladores, são componentes eletrônicos que integram, numa única pastilha, uma unidade de controle e vários outros dispositivos, como conversores AD (analógico/digital), memórias, temporizadores, interface de comunicação serial, etc. Desse modo, permitem redução de custos e tamanho físico, além de conferir versatilidade ao hardware.

Os microcontroladores geralmente são equipados com circuitos adicionais, a fim de torná-los um sistema completo, simplificando o projeto de circuitos eletrônicos ao oferecer num único chip várias funções que são comuns nestes sistemas. Um microcontrolador típico possui memória para dados e programas, portas de entrada e saída e temporizadores. Os fabricantes também disponibilizam microcontroladores com periféricos adicionais como conversor analógico/digital, adaptador multimídia, USB (*Universal Serial Bus*), relógio de tempo real, interface Ethernet e uma série de outros dispositivos. Tudo pronto para ser usado e oferecendo boa flexibilidade, uma vez que todos estes circuitos são controlados por uma CPU programável pelo projetista. [1] [3]

Os microcontroladores, de maneira geral, possuem as seguintes vantagens quando aplicados em sistemas embarcados [1]:

- Menos dispositivos são necessários
- Menor custo e tamanho
- Baixo consumo
- Hardware simplificado

Em contrapartida têm as seguintes limitações:

- Flexibilidade reduzida
- Expansão limitada
- Performance limitada

Arquitetura HARVARD x Von Neumann

Normalmente os microcontroladores utilizam memória separada para programa e dados. Este esquema permite transferências simultâneas de dados uma vez que é possível buscar instruções ao mesmo tempo em que se lê/escreve dados. Além disso os dados ficam protegidos contra uma possível execução indevida já que só é permitida execução na memória do programa. Esta arquitetura é chamada de Harvard ao contrário da arquitetura Von Neumann em que tanto dados como programa ficam na mesma memória.

Em arquiteturas Von Neumann, quando se deseja proteger dados contra execução, é necessário estabelecer um esquema complexo de estruturas de dados para uma unidade de gerenciamento de memória onde se aplica atributos (leitura, escrita, execução) em blocos de memória. Uma unidade de gerenciamento de memória naturalmente aumenta a complexidade do chip, algo que deve ser evitado num microcontrolador. Assim, microcontroladores baseados na arquitetura Harvard conseguem um esquema de proteção de memória eficiente e de implementação em circuito bastante simplificada. [1] [4]

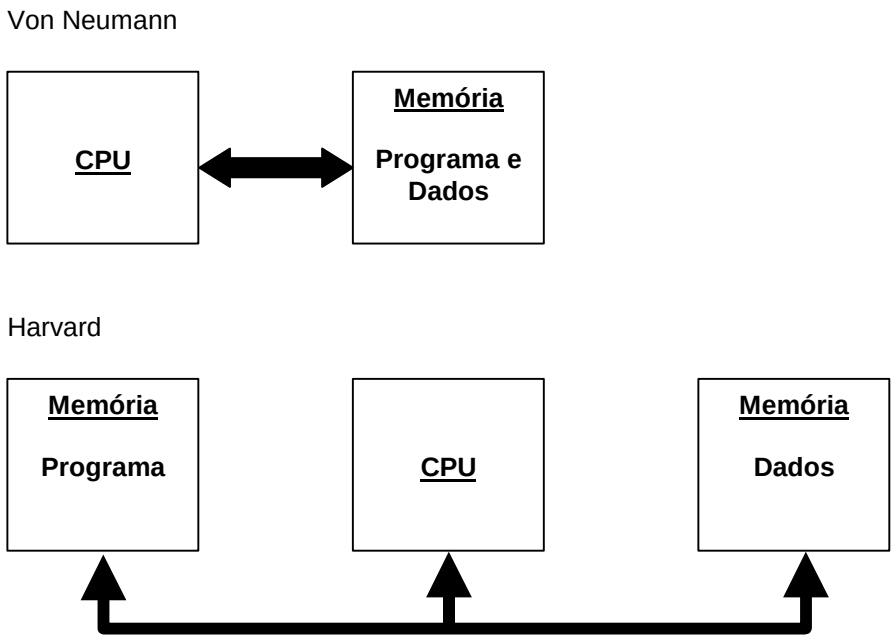

Figura 1 – Arquitetura Von Neumann x Harvard

Famílias de microcontroladores

Famílias de microcontroladores são grupos de dispositivos que compartilham as mesmas facilidades básicas, tais como CPU, características elétricas e dispositivos básicos, mas que diferem na funcionalidade dos periféricos. Uma família partilha o mesmo software, ferramentas de desenvolvimento, mas diferem nas funcionalidades individuais, como memória de I/O, número de temporizadores, formato do encapsulamento, tamanho e tipo (ROM, EPROM, Flash) da memória para programas. Os fabricantes oferecem mapas comparativos das facilidades de cada modelo. [1] [2]

Quando um novo modelo de uma família é fabricado, em alguns casos, é possível o *upgrade* do microcontrolador do sistema embarcado simplesmente substituindo-se o chip do mesmo pelo novo chip, sem necessidade de outras modificações no sistema. É comum microcontroladores de uma mesma família serem compatíveis em pinagem e em *opcodes* dos binários. Como exemplo, você pode começar seu projeto com um AT89LS51 que é um microcontrolador com as características básica da família 8051 e depois substituí-lo por um AT89S52 que, além de possuir o dobro da memória de programa, é duas vezes mais rápido. [3] [2]

Ferramentas de desenvolvimento

Num sistema embarcado onde serão utilizados microcontroladores, o desenvolvimento do software é uma etapa muito importante que, a depender do projeto, pode ser a que exija maior cuidado.

Para facilitar a depuração tanto do hardware quanto do software o processo de desenvolvimento implica na utilização da várias ferramentas. O conjunto mais básico de desenvolvimento consiste num placa que incorpora uma CPU em que exista uma interface com o mundo exterior (por exemplo, portas de entrada e saída), um assembler e um simulador. Estes permitem ao projetista escrever o código e verificar como funciona. [1]

Capítulo 2 – Dispositivos DSP

Os DSP's ou Digital Signal Processors (Processadores de Sinal Digital) são dispositivos eletrônicos capazes processar sinais digitais, tratando-os de algum modo, a semelhança de circuitos analógicos que processam sinais analógicos. A sigla também pode significar Digital Signal Processing (Processamento de Sinal de Digital) e, neste sentido, obviamente, se refere à técnica de tratamento do sinal e não ao dispositivo em si.

Para um DSP, um sinal é um fluxo de números binários que, na maioria das vezes, representa a intensidade equivalente de um sinal analógico. O sinal analógico passa por um conversor analógico-digital, que faz a conversão necessária para o formato digital, para então ser processado pelo DSP.

Sinais elétricos capturados em circuitos eletrônicos são quase sempre contaminados por interferências. Um sinal de um transdutor pode ser distorcido por ruído elétrico proveniente do exterior ou do próprio circuito onde ele está acoplado. Campos magnéticos ou flutuações de fornecimento de energia elétrica são exemplos de gerados de ruído. Processando-se o sinal através de filtros é possível remover ou reduzir bastante os efeitos dos ruídos. De maneira crescente, os DSP's vêm ganhando espaço no tratamento de sinais, seja na aplicação de filtros para melhorar a qualidade do sinal ou seja na extração de informações importantes. [14]

As aplicações para DSP's são das mais variadas. Hoje é possível encontrá-los em tocadores de CD, vídeo cassetes, controladores de disco rígido e modems. Em breve eles estarão substituindo circuitos analógicos de TV's e telefones. Outra aplicação importante dos DSP's está na compressão de dados. Num DVD, por exemplo, os dados estão compactados no disco e precisam ser descompactados antes de poderem ser transformados em sinais de vídeo. A descompressão precisa ser rápida o suficiente para que a reprodução não sofra atrasos. Para realizar uma tarefa tão complexa e em tempo real, um dispositivo especializado é fundamental.

À medida que os DSP's vão se tornando menores e mais velozes, novas aplicações vão os incorporando em seus sistemas. Gradativamente os filtros analógicos vão sendo substituídos por filtros digitais que apresentam várias vantagens sobre esses. Os DSP's são programáveis e, por isso, o filtro pode ser facilmente alterado mudando-se apenas o programa na memória. Um filtro analógico só pode ser alterado mudando-se o circuito eletrônico. Filtros digitais são extremamente estáveis por não sofrerem interferência de temperatura ou tempo e são capazes de processar sinais de baixa freqüência de maneira acurada, o mesmo não acontece com filtros analógicos [14]. DSP's são extremamente versáteis quanto a variedade de formas que podem empregar para processar um sinal. A descompressão de vídeo, por exemplo, é uma tarefa que seria extremamente difícil e provavelmente impraticável de ser realizada em um sistema analógico.

A título de ilustração, são apresentados a seguir alguns exemplos de filtros aplicados a sinais digitais. Apesar de serem exemplos de filtros bastante simples, um filtro digital pode empregar técnicas baseadas em teoria matemática complexa para transformar o sinal. As necessidades da aplicação é que irão ditar qual melhor filtro a ser empregado.

Para os exemplos a seguir, subtende-se que o fluxo do sinal já esteja na memória do DSP, assim x_0 se refere ao primeiro valor, x_1 ao segundo e assim sucessivamente. O sinal original de exemplo é o mostrado na **Figura 2,** já digitalizado.

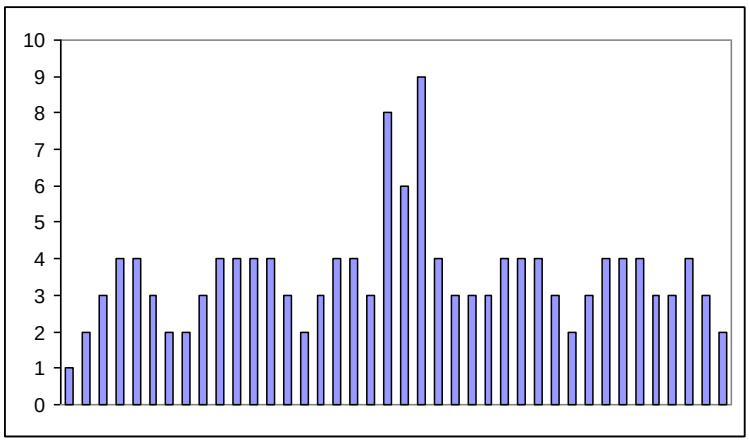

Figura 2 - Exemplo de sinal digitalizado

Um filtro do tipo “ganho simples” é obtido aplicando-se um ganho K a cada entrada:

$y_0 = Kx_0$

$y_1 = Kx_1$

$y_2 = Kx_2$

...

$y_n = Kx_n$

Para K > 0 o filtro se torna um amplificador, para K < 0 se torna um atenuador [14]. Na **Figura 3** pode-se observar a mudança provocada quando da aplicação do filtro de “ganho simples” com K = 0.5 nas entradas exibidas na **Figura 2**.

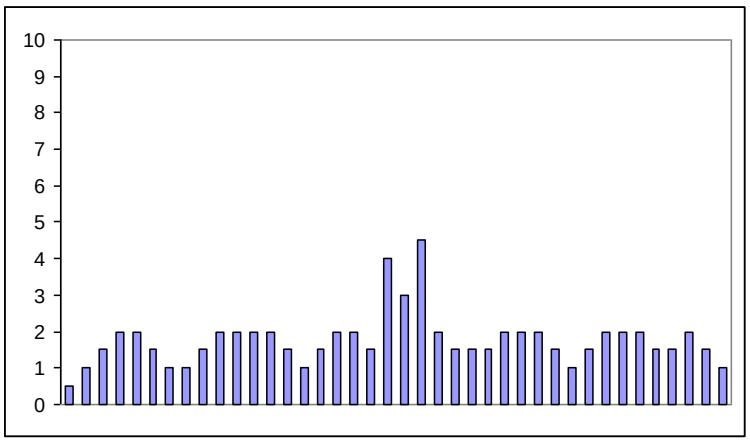

Figura 3 - Sinal após aplicação do filtro de ganho simples com K = 0.5

O filtro “média de dois termos”, dado por $y_n=(x_n+x_{n-1})/2$, é um exemplo simples de um filtro do tipo passa-baixa. Ele suaviza mudanças de alta freqüência no sinal. Ele pode ser estendido para fornecer a média para um número arbitrário de entradas, como num filtro de “média de três termos” que corresponderia à função $y_n=(x_n+x_{n-1}+x_{n-2})/3$. A **Figura 4** mostra a alteração provocada no sinal após ser processado por um filtro “média de três termos” e a **Figura 5** após ser aplicado o mesmo filtro apenas com a média sendo calculada sobre os 10 últimos termos. Para o caso x_{n-c} não existir é usual considerá-lo como valendo zero [14].

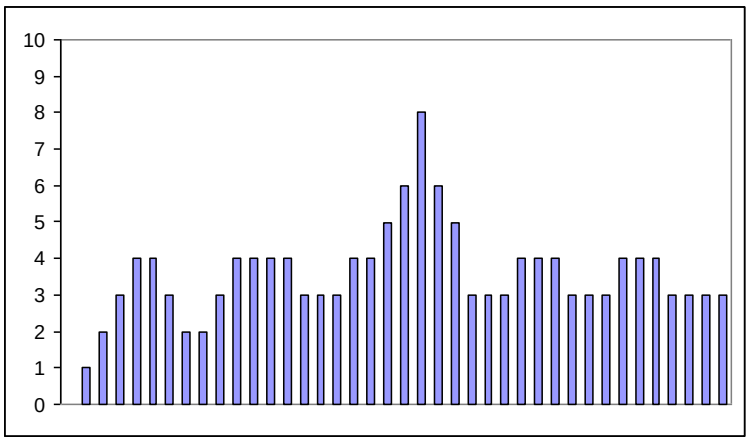

Figura 4 - Sinal após aplicação do filtro "média de três termos"

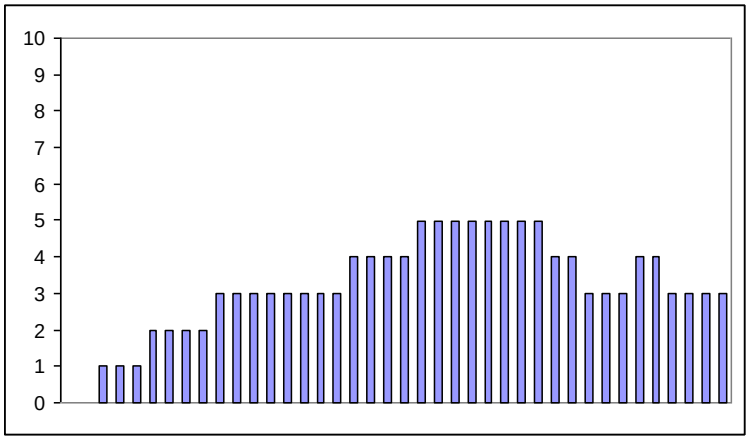

Figura 5 - Sinal após aplicação de um filtro do tipo "média de n termos", com n = 10

Se os sinais de nível 6 mostrados na **Figura 2** forem ruídos, pode-se ver claramente que o filtro “média de n termos” com n = 10, conseguiu suavizá-los. Como efeito colateral, no entanto, as outras entradas sofreram distorções por causa do filtro. Reduzindo-se o número de termos reduz-se estas distorções mas, por outro lado, a suavização dos ruídos é diminuída. Como foi dito anteriormente, este é um exemplo de filtro muito simples e seu emprego, assim como o de qualquer outro filtro, deve ser previamente avaliado para se assegurar que suas limitações não interfiram no resultado final da aplicação. De qualquer forma, filtros como este, são de fácil implementação e podem inclusive ser programados em controladores de propósito mais geral, como os microcontroladores, com boa performance final.

Capítulo 3 – Estado da Arte

Os sistemas embarcados e os microcontroladores estão cada vez mais presentes em nossas tarefas diárias. Basta dar uma olhada em casa para verificar. Eles estão presentes nos microondas, nos videocassetes, nos DVD's, nas TV's, nos rádios, nas impressoras e em uma larga gama de outros aparelhos eletrônicos. [8]

De acordo com um estudo da Semico Research, até 2005, uma pessoa comum deve utilizar ou pelo menos cruzar, diariamente, com 300 aparelhos que contém algum microcontrolador. [8]

Um dos maiores consumidores destes chips é a indústria automobilística que os utiliza na injeção eletrônica e em quase todo aparato eletrônico dos carros. Estima-se que em breve 70% de um carro será controlado por sistemas eletrônicos.

Os microcontroladores podem ser utilizados em vários projetos de eletrônica, substituindo vários componentes digitais, diminuindo custos, espaço físico e melhorando a produção. [9]

As câmaras fotográficas são um bom exemplo da utilização de circuitos eletrônicos na automatização e controle de tarefas. No início elas eram totalmente mecânicas. Pouco a pouco foram sendo introduzidos mecanismos para torná-las mais fáceis para fotógrafos leigos. Foi acrescentado um medidor de luz para controlar a abertura do diafragma, um motor para rodar o filme, um contador eletrônico para registrar o número de fotos batidas e um medidor de distância para regular o foco. Isso só foi possível graças à inclusão de um pequeno computador dentro da máquina, o microcontrolador. [7]

Os sistemas embarcados classificados como "simples" utilizam CPU's menos poderosas, geralmente de 8 bits. Eles são empregados em sinais de trânsito, alguns instrumentos médicos, controle de temperatura, etc. Os sistemas de "complexidade média" usam CPU's com mais recursos e são aplicados em tarefas mais sofisticadas, como pontos de venda, instrumentos médicos e computadores de bordo. Os sistemas

embarcados "sofisticados" são semelhantes a um computador moderno, apenas não possuem teclado, disco nem monitor. São empregados em sistemas multimídia, automação de centrais telefônicas, controle de plantas fabris, dentre outras. [7]

A Faculdade de Engenharia de Sorocaba apresentou um projeto de substituição de suas antigas trancas eletrônicas por novas trancas microcontroladas. A nova tranca possui interface com o usuário através de um LCD de duas linhas e de um teclado de 16 teclas. A senha para entrar nas salas pode ser trocada a qualquer momento usando-se o teclado, o que só era possível anteriormente abrindo-se a tranca e alterando o circuito. As chaves também podem ser controladas remotamente via sinais de rádio. [9]

Nesta mesma faculdade está em desenvolvimento uma cadeira de rodas elétrica comandada eletronicamente por meio de um teclado. O usuário pressiona teclas para ir para frente, para trás, girar à esquerda e à direita e para controlar a velocidade da cadeira. A velocidade correta é dada pelo microcontrolador através de um sinal PWM (Pulse Width Modulation).

Sérgio Akira Ito, da Universidade Federal do Rio Grande do Sul, publicou um artigo apresentando uma estratégia de desenvolvimento de aplicações embarcadas em Java, mantendo compatibilidade de software e tendo um microcontrolador como dispositivo alvo. O artigo propõe a síntese de um microcontrolador executor de bytecodes. No sistema, o projetista fornece uma aplicação Java que será rodada pelo microcontrolador batizado de *fentoJava*, capaz de executar instruções Java nativamente.

Glauco Fiorante aplica microcontroladores para processamento de sinais de imagem [12]. O trabalho consta de uma pesquisa e análise dos circuitos eletrônicos utilizados na aquisição de imagens em vídeo câmaras baseadas em sensores CCD (Charge-Coupled Device) e aplicações de visão estéreo, localização de objetos, reconhecimento de padrões e elaboração de plano de movimentos robóticos.

Outra aplicação para microcontroladores consiste na geração de sinais. Em [11] um PIC16C84 é utilizado para gerar sinais de vídeo. Através deste simples microcontrolador, o projetista construiu uma aplicação que consistia num pequeno jogo de vídeo game, o tetris, e num gerador de sinais de vídeo preto-e-branco que possibilitava conectar o microcontrolador numa TV e jogar através de um joystick. O limite está apenas na capacidade de processamento do dispositivo. O mesmo autor

adaptou o sistema para rodar num microcontrolador mais rápido de maneira que foi possível gerar imagens coloridas.

Os microcontroladores podem ainda ser empregados em interfaces ISDN, modems e outros dispositivos de rede e comunicação, dispositivos RKE (remote keyless entry) de automóveis, impedindo o uso do veículo sem a chave original; associado a sensores e DSP's, para controles de temperaturas, níveis de água e pressão; controles de braços mecânicos, de movimentos e de motores de passo e em muitas outras aplicações. [13]

Conforme diz Paulo Grimme, vice-presidente e gerente geral da divisão de microcontroladores da Motorola, "a cada dia aparece uma nova aplicação para os microcontroladores" e ainda "quanto mais baratos eles ficam, também descobrimos que é possível fabricar equipamentos com diversos chips desse tipo."

Capítulo 4 – O microcontrolador 8051 num sistema de segurança dos Laboratórios da UESB

Neste estudo de caso o microcontrolador 8051 foi utilizado para realizar as principais funções dentro de um sistema de segurança. Procurou-se concentrar o máximo de atividades do sistema dentro deste microcontrolador com o objetivo de se conseguir construir um hardware ao menor custo possível. O 8051 foi escolhido por ser um componente bastante barato e por ter ampla disponibilidade, além de oferecer bons recursos ao projetista, como um bom número de portas de entrada e saída, porta serial, timers e tratamento de interrupção.

Implementação

O sistema deverá emitir um alarme ou aviso quando detectar movimento na sala, fora dos horários permitidos. Para isso ele será equipado com dois sensores infravermelhos capazes de captar variações nos níveis da radiação infravermelha quando existe presença de pessoas no ambiente. Serão usados dois sensores porque o sistema informará também a posição do invasor dentro da sala.

Um microcontrolador da linha 8051 receberá o sinal de saída do sensor e acionará o alarme, se necessário. O alarme será composto de um alto-falante simples, interfaceado com o microcontrolador. A variação de sinal para produzir a freqüência desejada no alto-falante será produzida pelo próprio microcontrolador.

O software no microcontrolador será responsável por manter uma tabela de horários que informa em quais períodos a sala estará disponível ou não. É necessário que o software também mantenha um relógio para que se possa comparar a data e hora atual com a tabela de horários.

Em razão da memória RAM nativa do 8051 ser bastante reduzia (128 bytes), a tabela de horários deve ser o mais compacta possível. Procurou-se uma estrutura de dados que pudesse manter a tabela flexível e pequena. É possível optar pelo uso de RAM externa para aumentar a capacidade de armazenamento, mas, neste projeto, está-

se buscando a solução mais barata possível, por isso a decisão foi tentar aproveitar ao máximo a memória RAM interna do microcontrolador.

Um exemplo de tabela de horários poderia ser a seguinte:

Dia da semana	Hora inicial	Hora final
Segunda	08:00	12:00
Segunda	14:00	18:00
Terça	08:00	12:00
Terça	14:00	18:00

Pode-se observar que, nesta estrutura, a tabela permite configurar apenas horários disponíveis, neste caso, os horários indisponíveis estão implícitos, ou seja, a sala só estará aberta nos horários explicitamente presentes na tabela. Assume-se, também, que não é necessário que a hora seja expressa com precisão de segundos. Para nossa aplicação uma precisão de minutos é suficiente. Podemos criar um array para guardar a tabela, dando a cada coluna 1 byte de tamanho. Com 1 byte podemos representar o dia da semana codificados de 0 a 6. Nesta configuração, estaremos usando apenas 3 bits dos 8 possíveis.

Com apenas 1 byte não é possível guardar a hora completa. Podemos usar 1 byte para a hora e mais 1 byte para os minutos. Utilizando-se este array, nossa tabela passa a ocupar 5 bytes para cada registro incluído. Se, por exemplo, a sala fica aberta de segunda a sexta, das 8:00h às 12:00h e das 14:00h às 19:00h, tem-se uma tabela de horários que ocupará 50 bytes. O array assim estruturado ocupa muito espaço. No exemplo anterior, só a tabela de horários está consumindo 40% da memória RAM. É bom lembrar que dos 128 bytes disponíveis, 32 serão automaticamente ocupados pelos registradores e outra parte será usada pela pilha para guardar registradores e variáveis locais, além das variáveis globais que ocuparão a RAM durante todo tempo. A situação fica ainda pior se desejarmos incluir um outro horário livre na tabela, por exemplo, de segunda a sexta das 20:00h às 22:00h, caso em que o array ocuparia 75 bytes.

Verificando que a maioria dos horários vai valer para dias da semana em seqüência, a tabela poderia ser composta da seguinte maneira:

Dia inicial	Dia final	Hora inicial	Hora final
Segunda	Sexta	08:00	12:00
Segunda	Sexta	14:00	18:00
Segunda	Sexta	08:00	12:00
Segunda	Sexta	20:00	22:00

A tabela de horários que ocuparia 75 bytes, se estruturada da forma anterior, agora passa a ter apenas 24 bytes com as colunas "dia inicial" e "dia final" ocupando 1 byte cada uma. É possível reduzir ainda mais o array, concatenando as duas primeiras colunas num único byte. Como já foi visto, o dia da semana precisa de apenas 3 bits para ser representado. Reservamos os 3 primeiros bits de um byte para o dia inicial e os 3 seguintes para o dia final. Cada registro passa a ter 5 bytes de comprimento como no primeiro exemplo, mas com uma estrutura de armazenamento que pode ser melhor aproveitada.

Os sinais vindos dos sensores precisam ser filtrados para evitar que variações naturais nos níveis de infravermelho acionem o alarme. Um exemplo deste tipo de variação seriam as rajadas de calor trazidas por ondas de vento que passam periodicamente em ambientes não totalmente isolados.

O filtro utilizado não pode consumir muitos ciclos da máquina porque o 8051 não é um microcontrolador muito rápido (os modelos comuns chegam a um clock máximo de 20Mhz). Optou-se, pela utilização do filtro “média de n termos” que é rápido de ser calculado e consegue suavizar bem mesmo sinais fortemente distorcidos, bastando utilizar um número adequado de termos para se obter a média. O cálculo deste filtro consiste apenas em se somar os últimos níveis amostrados e dividir o total pelo número de amostras, ou seja, é a média das amostras. Estas duas operações são realizadas com boa eficiência pelo 8051.

A taxa de amostragem dos sinais dos sensores infravermelhos foi obtida experimentalmente, além de ter sido levado em consideração o filtro que seria aplicado a eles. Assumindo-se que os ruídos provocados pelas rajadas de calor durem menos de 1 segundo, a média dos sinais do último segundo foi avaliada como uma boa medida que caracterizasse o sinal. Experimentalmente foi percebido também que, com uma taxa de amostragem de 20 vezes por segundo é possível determinar a posição do intruso de maneira suficiente rápida.

Outra medida avaliada foi a resolução da conversão dos sinais analógicos para sinais digitais. Um conversor analógico-digital de 8 bits demonstrou ser suficiente. Se o sistema for instalado, por exemplo, numa sala de 10 x 10 metros, cada variação unitária representaria a mudança de 4 cm na posição se for adotado que 0 é a distância máxima

do invasor ao sensor e 255 a distância mínima (vale observar que o nível aumenta com a aproximação). Na prática, no entanto, o ambiente já estará com certo nível mínimo de radiação infravermelha, e a entrada de uma pessoa provocará o aumento do nível a partir deste mínimo. Mesmo numa situação extrema, como o nível mínimo valendo 128 e o máximo 200, por exemplo, ou seja, apenas 72 variações do nível de infravermelho, o sistema conseguiria detectar alterações de 14 cm na posição, o que é mais que suficiente para os propósitos da aplicação em questão.

A partir das informações acima, pode-se verificar que o filtro deverá utilizar 40 bytes de memória, 20 bytes para um sensor e 20 para o outro. Estes blocos de memória armazenarão as amostras anteriores, necessárias ao cálculo da média do sinal.

Como foi visto, o filtro tem o propósito de suavizar o sinal e evitar que ruídos gerem alarmes falsos. Mas o sistema também deve ser capaz de informar a posição do intruso na sala. Para realizar esta tarefa é que foram utilizados 2 sensores. A disposição dos sensores nos cantos esquerdo e direito da sala, conforme indicado na Figura 6, permite que a posição do intruso seja dada pela interseção dos arcos cujos raios sejam proporcionais ao nível de proximidade da pessoa ao sensor (neste caso, quanto mais próximo, menor o raio).

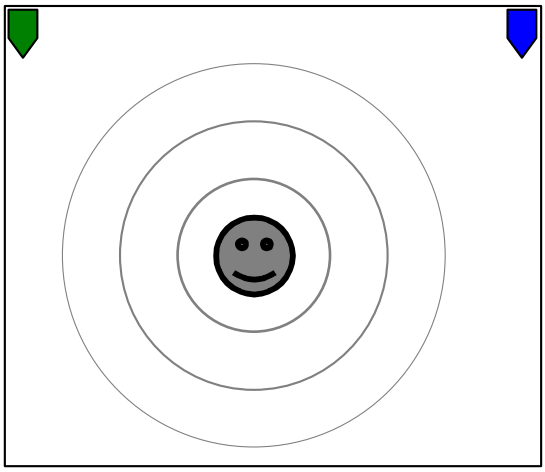

Figura 6 – Posição de 2 sensores para detecção de posição

A Figura 7 ilustra um exemplo desta técnica:

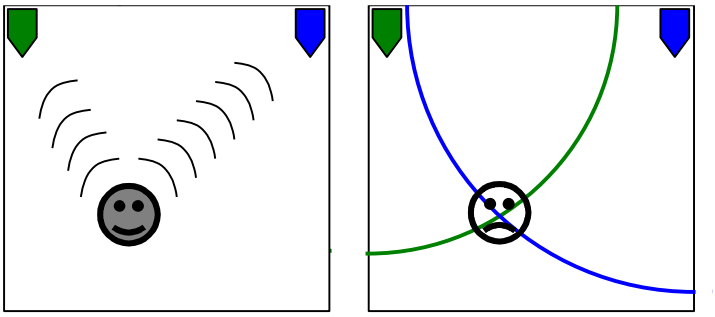

Figura 7 – a) O nível de infravermelho em cada sensor informa a distância do objeto até o sensor; b) Produzindo-se arcos onde o raio seja esta distância, é possível determinar a posição do objeto pela interseção destes arcos.

A técnica acima é válida desde que os sensores sejam estejam posicionados nos extremos da sala. É fácil ver que se ao invés de arcos, os sensores estivessem posicionados de maneira que produzissem círculos completos (se eles estivessem mais centralizados, por exemplo), haveria mais de uma interseção possível e, neste caso, não seria possível definir a posição do objeto com apenas 2 sensores.

Software

O software rodando no microcontrolador será o principal componente do sistema pois é através dele que o restante dos dispositivos serão gerenciados. Devido a pouco memória que estará disponível, tanto para os dados quanto para o programa, deve-se procurar desenvolver um software de código pequeno e com estruturas de dados as menores possíveis. Com este objetivo em mente, algumas funções de bibliotecas deverão ser evitadas por serem excessivamente grandes, a exemplo da função padrão *printf* que, em alguns compiladores, chega a aumentar em 1K o tamanho do código compilado. A linguagem em que será escrito o programa será C em razão das vantagens de ser uma linguagem de alto nível e também por resultar em código bem compacto após a compilação.

A rotina principal do sistema é a que trata da interrupção do timer 0. Esta rotina será chamada a cada interrupção do timer que será configurado como um temporizador. Ela ajustará o valor de um relógio que registrará a data e hora atual. Ela também verificará o status geral do sistema, como bateria fraca, hora errada, sinal de presença, dentre outros, tomando as ações necessárias em cada caso. Além disso, esta rotina será a

responsável por enviar sinais para a porta de entrada e saída onde estiver o alto-falante de maneira a gerar pulsos sonoros de alarme.

A verificação de hora errada usa um algoritmo simples. Ela é feita comparando-se o ano com um valor de referência, 2004 por exemplo. Se o ano for menor que este valor significa que a hora está errada pois o sistema nunca entraria no ar num ano menor que o de referência. Este caso acontecerá quando houver falta de energia elétrica, por exemplo, situação em que o valor do ano será 0 quando o sistema voltar.

A hora será representada por variáveis globais que armazenarão hora, minutos, dia e mês atuais. Para ajustar o relógio, a rotina principal contará o número de vezes que foi chamada. Quando este número for correspondente ao período de 1 minuto ela acerta a hora, incrementando e/ou ajustando as respectivas variáveis. O relógio terá precisão de 1 minuto que, para a aplicação em questão, é suficiente uma vez que a hora deste relógio será comparada com a tabela de horários que também é ajustada com esta precisão.

O sinais que o microcontrolador receberá virão das portas de entrada e saída. Não serão usadas interrupções porque o sistema não precisará dar respostas em tempo real, a emissão do alarme, por exemplo, poderia demorar até mais de 1 segundo que não comprometeria os requisitos de resposta uma vez que não seria possível alguém invadir a sala e furtá-la neste intervalo. As portas de entrada e saída serão amostradas periodicamente na rotina principal. O tempo desta amostragem será rápido o suficiente para que: seja dada uma resposta no tempo adequado; os sinais de entrada não sejam perdidos devido a uma amostragem muito lenta. Uma porta receberá o sinal dos sensores de movimento e outra o sinal da bateria quando esta estiver fraca.

Configuração do dispositivo

Como trata-se de um sistema de segurança programável é necessário algum mecanismo que permita ao operador ajustar o dispositivo. As tarefas principais de configuração seriam ajustar a hora atual e manter a tabela de horários. Poderia ser utilizado um teclado e um display LCD de maneira a permitir entrar com os comandos para programá-lo. Esta solução tem dois problemas principais: o custo da solução aumentaria substancialmente e toda vez que faltasse energia elétrica seria necessário a reprogramação manual. Para contornar estes problemas podemos aproveitar o fato de

que o 8051 vem com uma porta serial embutida que pode ser usada para comunicação com outros dispositivos. Assim, para fazer a programação do sistema, poderia se conectar o microcontrolador a um PC com software de comunicação que faria o papel de teclado e display. Esta é a solução adotada neste projeto.

O microcontrolador será então, programado de maneira a receber comandos de texto pela porta serial. O processamento do comando envolve: analisar o texto, extrair o comando e os parâmetros e executar a tarefa correspondente. Teremos que restringir bastante a sintaxe dos comandos em razão de que o código necessário para fazer o processamento do texto seria muito grande caso fosse permitida uma sintaxe flexível (comandos e parâmetros de tamanhos não-fixos, espaço arbitrário entre elementos do comando, etc.), veja **Tabela 1**. Observe que ao invés de ser configurado por comandos, o sistema poderia ser configurado por meio de menus de opção. Esta alternativa poderia ser implementado com um código menor mas que, por outro lado, resultaria num aumento do número de mensagens que o sistema teria de exibir (várias para cada menu). Estas mensagens, obviamente, serão gravadas na memória de programa que, como já sabemos, é escassa. A alternativa por meio de comandos tem também a vantagem de que o texto que o operador digitar para programar o sistema seja previamente salvo num arquivo no PC. Desta forma, caso ocorra falta de energia elétrica, ele poderá reprogramar o dispositivo rapidamente, bastando para isso, carregar o arquivo no software usado para comunicação, que este enviará a seqüência de comandos para o microcontrolador. Esta forma é usada atualmente em vários equipamentos como roteadores e switchs.

Exemplo de sintaxe flexível	**Exemplo de sintaxe rígida** [*1]	**Função**
Hora 8:30	Hr 08 30 31 01 2003 [*2]	Altera a hora
Senha 1234	Sn 1234	Muda a senha de proteção
Entrada Segunda Sexta 8:30 12:30	Ed 02 06 08 30 12 30 [*2]	Inclui entrada na tabela de horários

Tabela 1 - Exemplo de possíveis sintaxes

[*1] - Os nomes de comandos têm todos o mesmo tamanho
[*2] - Os parâmetros são numéricos e com o número de dígitos fixo.

Circuito

O circuito elétrico para implantação do projeto constará da interface entre o microcontrolador e os sensores infravermelhos bem como dele com os outros dispositivos que irão compor o sistema, um alto-falante para emissão do alarme e um LED para notificar outros eventos (bateria fraca, data errada, etc.). O diagrama de blocos da Figura 8 mostra o circuito:

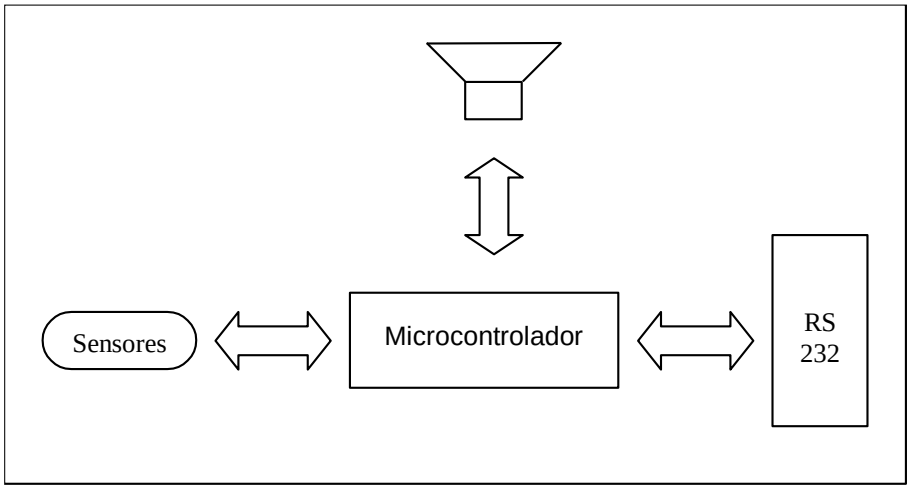

Figura 8 - Diagrama de blocos do circuito

A princípio o circuito exigiria 2 conversores analógico-digital, um para cada sensor, além de 2 portas de 8 bits no microcontrolador para receber os sinais digitalizados. Para simplificar e baratear este esquema pode-se utilizar uma chave analógica que comute os 2 sinais. Primeiro, o sinal de um dos sensores é digitalizado e recebido pelo microcontrolador, depois o do outro, seqüencialmente. Estas duas digitalizações seqüenciais é um processo rápido, num conversor AD típico cada uma leva em torno de 100 microssegundos. O microcontrolador é capaz de realizar as 20 amostras por segundo com bastante folga permitindo que um esquema deste tipo seja inserido sem que haja o perigo de uma amostragem demorar mais que 0,05 segundos (o tempo máximo que se tem para processar cada uma).

Capítulo 5 – Prototipagem e testes realizados

Para a realização dos testes foi montado um protótipo que pudesse se equiparar, o mais próximo possível, ao dispositivo real proposto. Em virtude da falta de recursos disponíveis, algumas blocos do circuito foram substituídos por soluções mais baratas e/ou mais disponíveis. A Figura 9 mostra o esquema eletrônico do circuito utilizado. Para melhorar a visualização não são exibidas as conexões dos CI's à fonte e ao terra.

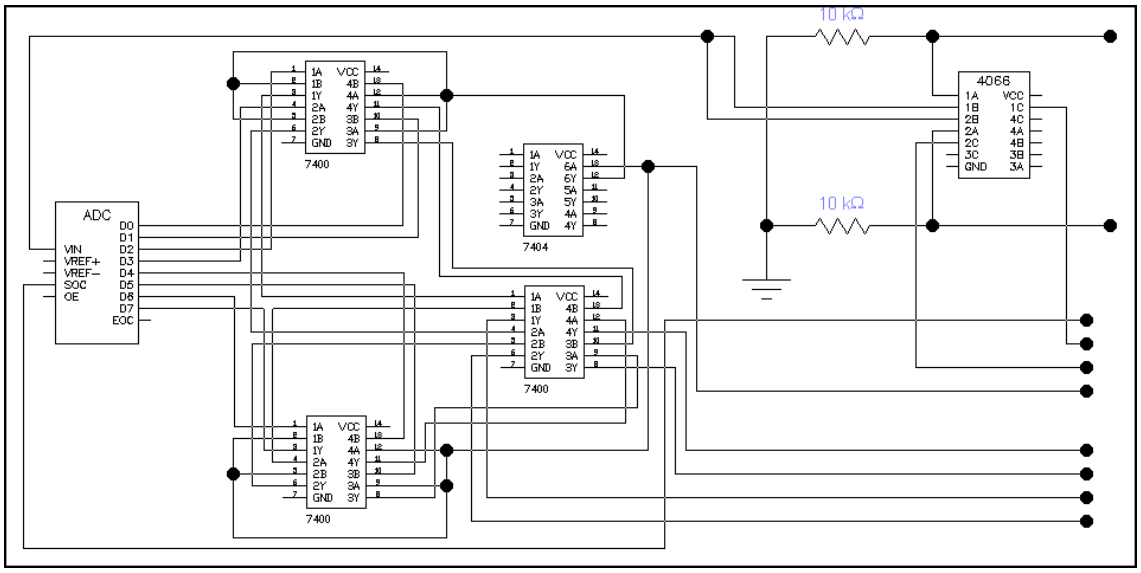

Figura 9 – Esquema eletrônico usado no protótipo. No extremo direito de figura temos conectores (círculos preenchidos conectados em só uma ponta) que representam entradas e saídas. De cima para baixo tem-se: os 2 primeiros são sensores LDR conectados a uma fonte de +5V; o próximo é uma entrada que ativa/desativa a conversão; os 2 seguintes selecionam o sensor; o próximo seleciona o nibble; os 4 últimos são as saídas do sinal digitalizado. Com exceção dos 2 primeiros, todos são conectados à porta paralela.

A primeira grande mudança do protótipo em relação ao dispositivo real é que o microcontrolador foi simulado num PC. Isso se deveu à falta de um programador para o microcontrolador, impossibilitando que um 8051 real fizesse parte do circuito.

Para que fosse possível ao PC receber e enviar dados para o restante do protótipo, foi utiliza a porta paralela como meio de comunicação. A porta paralela foi usada para lê os bits vindos do conversor AD bem como para acionar partes do circuito através de pinos de saída.

Normalmente a porta paralela é utilizada apenas para se escrever dados para uma impressora, 8 bits de cada vez, sendo também possível ler alguns bits de status. Numa porta paralela comum, a porta de envio de dados não pode receber dados [15]. Foi utilizada a porta de status para que fosse possível ler os bits vindos do conversor AD ao mesmo tempo em que a porta de dados foi utilizada para se enviar bits para o circuito eletrônico que fez parte do protótipo.

A porta de status possui apenas 5 bits que podem efetivamente ser lidos sem que seja necessário nenhum circuito adicional para realizar conversões e/ou ajustes [15]. Neste projeto foi utilizado este esquema por ser mais simples. No entanto, tem-se o problema de que devem ser lidos 8 bits do conversor AD. O mecanismo adotado foi ler o nibble menos significativo num primeiro passo e ler o nibble mais significativo num segundo passo. Esta seqüência precisa ser realizada 2 vezes, uma para ler os 8 bits de um dos sensores e outra para ler a do outro sensor.

Os bits da porta de dados são utilizados basicamente para selecionar o que se deseja ler, a única exceção é o bit 0 que ativa ou desativa a conversão (é necessário desativar e ativar a conversão a cada nova leitura) [16]. Os bits 1 e 2 são mutuamente exclusivos e selecionam o sinal de qual sensor entrará no conversor. O bit 3 seleciona qual nibble deverá ser lido.

Outra mudança importante em relação ao dispositivo proposto é que os sensores infravermelhos foram substituídos por sensores LDR, isto é, sensores de luz, ou ainda, resistores dependentes da luz. As razões fundamentais para esta substituição foram: os sensores infravermelhos são muito difíceis de se encontrar no mercado e, caso eles fossem utilizados, seria necessário um bloco de amplificação dentro do circuito para que fosse possível conectá-los ao conversor AD, aumentando a complexidade. Com os sensores LDR, tudo que é necessário é um simples divisor de tensão, ou seja, a conexão em série dos sensores com resistores.

Portas NAND foram utilizadas para se construir um multiplexador de 8 entradas e 4 saídas. Duas portas fazem a multiplexação propriamente dita e uma terceira a inversão final da saída. Sem esta terceira porta o sinal sairia invertido em relação à entrada. Este tipo de dispositivo existe comercialmente, no entanto, não foi encontrado no comércio local de componentes eletrônicos. O multiplexador é o responsável por

dividir a leitura do conversor em grupos de 4 bits [17]. Além do multiplexador, uma chave analógica também foi utilizada. Ela é a responsável por ora permitir que o sinal de um dos sensores chegue até o conversor, ora o do outro. Os bits 1 e 2 da porta de dados da paralela foram conectados às entradas desta chave de maneira que se tornou possível selecionar o sensor desejado pelo PC.

Naturalmente foi necessário desenvolver um software para rodar no PC de maneira a simular o microcontrolador. Além disso também foram incluídas algumas funções extras, como a exibição dos gráficos dos sinais originais e filtrados e também a posição da fonte de luz (que substitui a fonte de calor). Este software foi desenvolvido utilizando-se a ferramenta de desenvolvimento Delphi.

A Figura 10 dá um exemplo de telas geradas pelo programa. São exibidos os sinais dos dois sensores num ambiente iluminado por luz fluorescente no qual foram provocadas mudanças no nível da luz. Tanto os sinais originais quanto os após a aplicação do filtro são exibidos. O filtro aplicado foi o filtro original proposto, isto é, média dos últimos 20 termos.

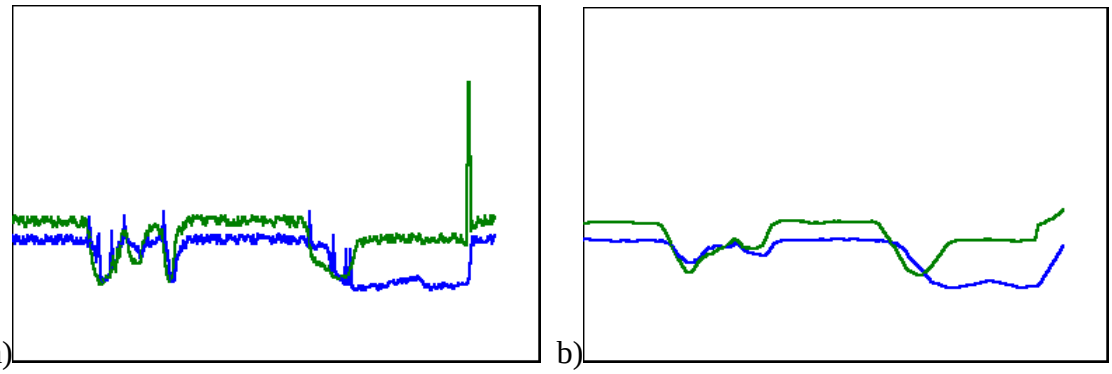

Figura 10 – Exemplo de telas geradas pelo programa. a) exibição do sinal original; b) exibição do sinal após o filtro.

Na Figura 11 têm-se ainda mais um exemplo de tela que o programa produz. Nela o programa dá a posição relativa da fonte de luz representada pelo círculo. As outras curvas são os arcos que foram gerados pelos níveis de luz em cada sensor.

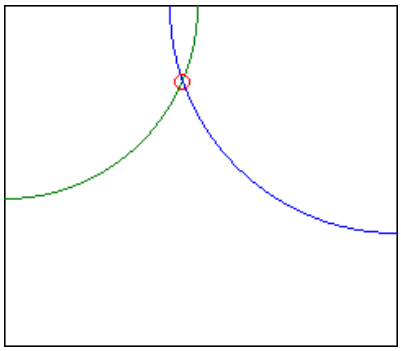

Figura 11 – Tela do programa mostrando a posição da fonte de luz

Apêndice 1 - O microcontrolador 8051 - Referência

O 8051 é o membro original de uma família de microcontroladores muito utilizada em sistemas embarcados. O 8051 padrão possui as seguintes características:

- CPU de 8 bits otimizada para aplicações de controle.
- Capacidade extensiva de processamento booleano (bit a bit).
- 64K de endereçamento de memória de programa.
- 64K de endereçamento de memória de dados.
- 4K de memória de programa *on-chip*.
- 128 bytes de memória RAM *on-chip*.
- 32 linhas de I/O bidirecionais e individualmente endereçáveis.
- 2 temporizadores/contadores de 16 bits.
- UART full duplex.
- 5 estruturas de vetores de interrupção com 2 níveis de prioridade.
- Oscilador de relógio *on-chip*.

A arquitetura básica do 8051 é mostrada na Figura 12: [5]

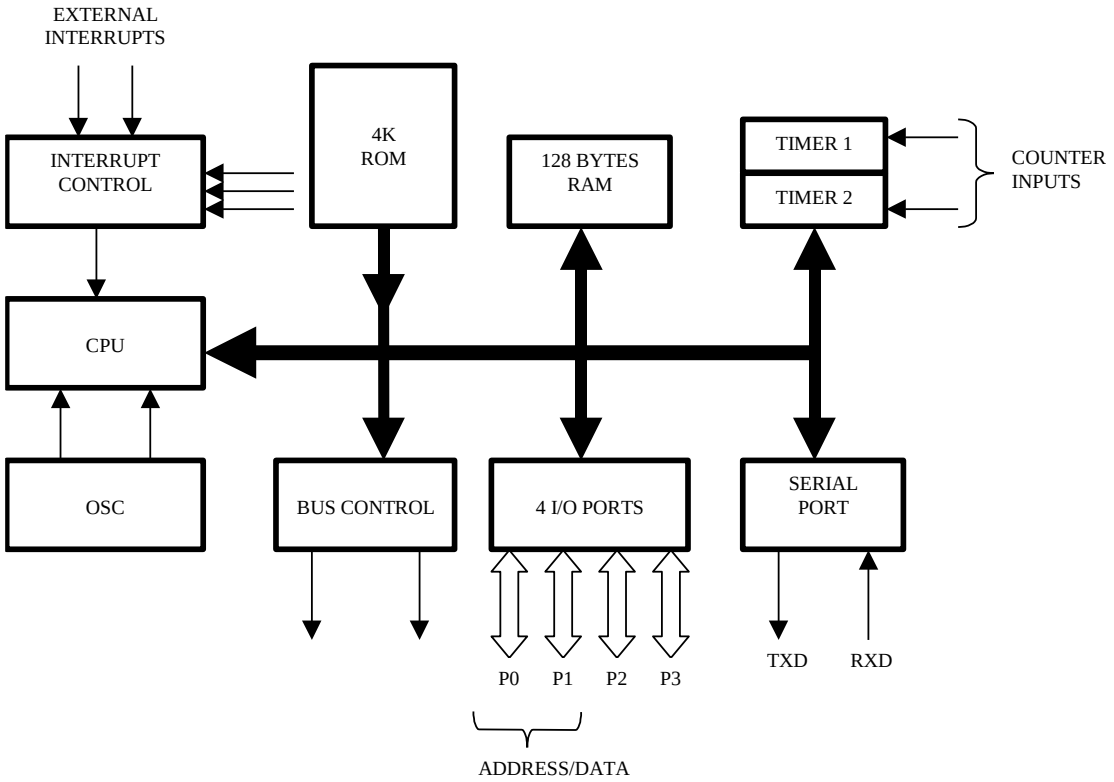

Figura 12 - Diagrama de bloco do núcleo do 8051

Organização de memória

Os dispositivos da família 8051 têm endereços separados para programas e dados (arquitetura Harvard). A memória de dados é acessada através de endereços de 8 bits que são tratados com mais rapidez pela CPU de 8 bits. Isso limita o espaço de memória a apenas 256 bytes. No entanto, o microcontrolador disponibiliza um registrador de 16 bits, o DPTR (Data Pointer), para acesso a memórias RAM de até 64K.

Pode-se ter até 64K de memória de programa. Esta é uma memória para leitura somente. Existem versões com 4K, 8K e até 16K de memória de programa integrada no chip do processador. A CPU suporta memória de programa externa ativando-se o sinal PSEN (Program Store Enable). [5]

Memória de Programa

Cada interrupção está atribuída a uma posição de memória de programa. O pedido de interrupção provoca o salto da CPU para esta posição de memória. Os

endereços das rotinas de tratamento de interrupção estão espaçados em intervalos de 8 bytes começando de 0003H. Isto dá a cada rotina um bloco de 8 bytes para código.

Se a rotina for bastante pequena, o que é comum em aplicações de controle, ela pode estar dentro destes 8 bytes. Caso contrário, a rotina de tratamento propriamente dita pode ser alocada em endereços superiores (acima da bloco destinado à rotinas de interrupção) da memória de programa, não ficando, assim, limitada aos 8 bytes. Para que esta rotina seja efetivamente chamada basta existir um instrução de salto (*jump*) que aponte para ela em algum lugar dentro do bloco de 8 bytes destinado à sua rotina de tratamento.

Caso uma interrupção não venha a ser utilizada, o espaço de memória dela poderá ser alocado livremente como memória de programa de propósito geral. [5]

A tabela abaixo mostra os endereços pré-determinados para algumas rotinas de interrupção: [1]

Endereço	**Rotina**
0000H	RESET
0003H	Interrupção externa 0
000BH	Interrupção do Timer 0
0013H	Interrupção externa 1
001BH	Interrupção do Timer 1
0023H	Interrupção da Porta Serial
...	...

Tabela 2 - Tabela de endereços de interrupção - exemplo

Memória de Dados

Como pode ser visto na Figura 13, a memória interna é uma faixa de endereços de 256 bytes (de 00h a FFh). Apesar disso ela possui efetivamente 384 bytes de comprimento uma vez que a área de memória alta, de 80h a FFh, acessa dois espaços de memória diferentes, a depender do tipo de instrução que é usada para acessá-la. Instruções de endereçamento direto acessam 128 bytes de memória de uso geral enquanto instruções de endereçamento direto acessam os *registradores de funções especiais* que estão mapeados nesta faixa de endereços. Somente dispositivos com 256 bytes de memória RAM disponibilizam acesso aos 128 bytes de memória alta de uso geral. [5]

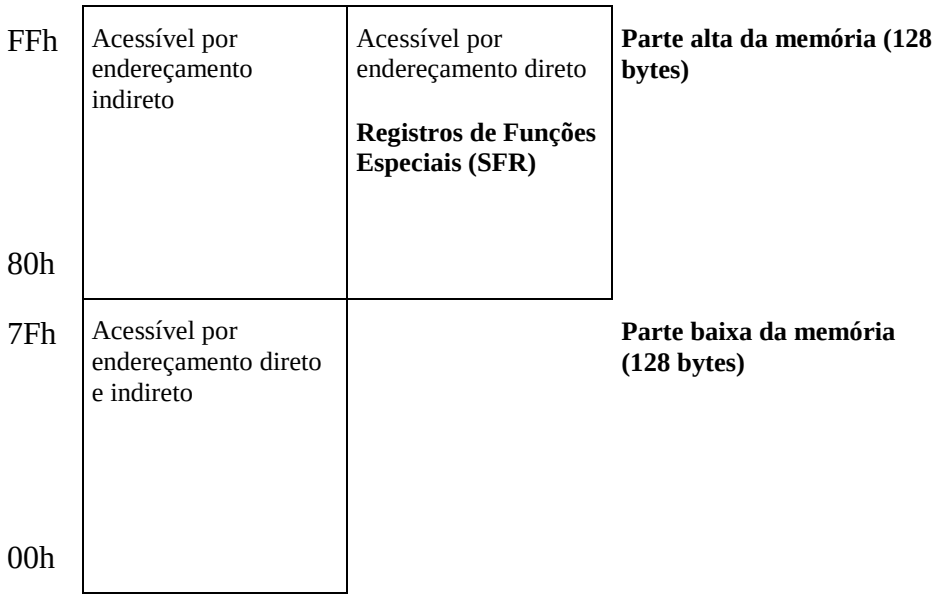

Figura 13 - Mapa de memória interna.

A área de memória baixa, os 128 primeiros bytes da memória interna, está organizada conforme é mostrado na Figura 14:

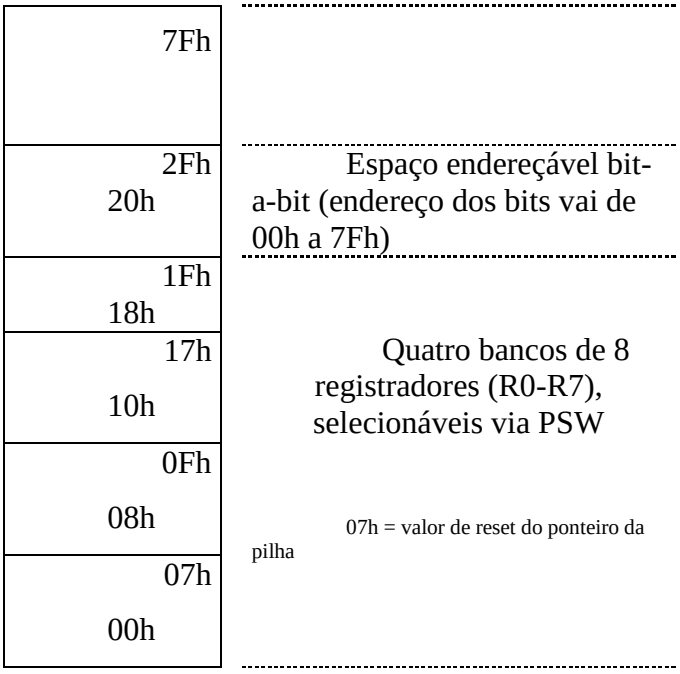

Figura 14 - Área baixa da memória interna

A área de memória baixa pode ser alocada quase que totalmente para uso geral. A exceção são os 32 primeiros bytes (00h-1Fh). Conforme mostra a Figura 14, de 00h a 1Fh temos 4 bancos de 8 registradores. Esta área pode ser acessada através dos registradores R0 a R7. Isto permite uso mais eficiente do espaço de programa uma vez que instruções que usam registradores são menores que instruções que usam

endereçamento direto. O bloco de registradores é selecionável através de 2 bits em PSW (Program Status Word).

Em 07h fica armazenado o endereço inicial do ponteiro da pilha, este valor pode ser alteado depois por programação.

De 20h a 2Fh temos um bloco de memória que pode ser acessado bit-a-bit. O 8051 disponibiliza várias instruções para acesso bit-a-bit. A faixa de endereços dos bits vai de 00h a 7Fh. [5]

Conjunto de Instruções

O conjunto de instruções do 8051 é otimizado para instruções de 8 bits. Todos os membros da família executam as mesmas instruções. O microcontrolador possui uma série de instruções de acesso rápido à memória RAM que facilitam operações de um byte em pequenas estruturas de dados. Também é oferecido suporte a uma vasta gama de instruções para operações de um bit.

O estado da CPU é obtido através do registrador PWS (Program Status Word). Este registrador possui uma série de bits que refletem várias situações em que o estado da CPU é alterado com a execução das instruções. O PSW é guardado no espaço de memória do SFR (Special Function Registers). [5]

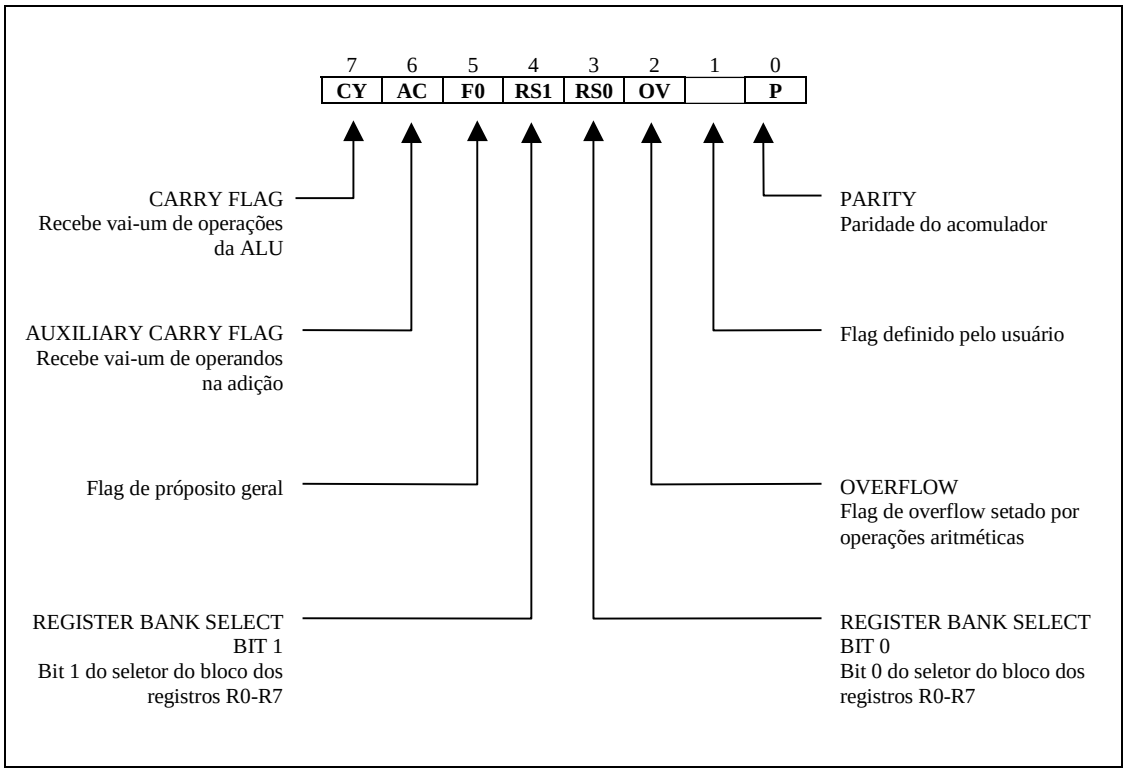

Figura 15 - O registrador PSW - Program Status Word

Modos de endereçamento

Os modos de endereçamento determinam como os valores dos operandos associados a endereços da memória são interpretados pelo microcontrolador. No 8051 os modos são os seguintes:

- Endereçamento direto;
- Endereçamento indireto;
- Instruções de registradores;
- Instruções de registradores específicos;
- Constantes imediatas;
- Endereçamento indexado;

Endereçamento direto

Neste tipo de endereçamento o endereço é dado através de um campo de 8 bits na instrução que aponta para o endereço desejado. Somente a RAM de dados interna e o bloco dos SFRs podem ser acessados diretamente.

Exemplos:

```
mov a, 0AFh       ; a = conteúdo em 0AFh
mov 0AFh, a       ; conteúdo em 0AFh = a
```

Endereçamento indireto

No endereçamento indireto o endereço é o conteúdo de um registrador. Tanto a RAM interna quanto externa podem ser acessadas indiretamente.

Os registradores R0, R1 ou o SP (Stack Pointer) podem ser usados quando um endereço de 8 bits é suficiente. Para endereçamentos de 16 bits existe somente o DPTR.

Exemplo:

```
mov a, @R0        ; a = conteúdo da memória apontada por R0
```

Instruções de registradores

Aqui, as operações guardam seus valores nos registradores de R0 a R7. Estas instruções têm a vantagem de reduzirem o código uma vez que um byte extra para endereço é eliminado. O banco de registradores a ser utilizado é selecionado através de 2 bits em PSW.

Exemplos:

```
mov a, R0         ; a = R0
mov R3, a         ; R3 = a
```

Instruções de registradores específicos

Algumas instruções são específicas para determinado registrador. Algumas instruções, por exemplo, só utilizam o acumulador, outras somente o DPTR. Para estas instruções não é necessário o byte de endereço. O op-code da instrução já determina qual registrador será utilizado.

Exemplos:

```
clr c            ; c = 0
swap a           ; troca os nibbles de a
inc dptr         ; dptr = dptr + 1
```

Constantes imediatas

O valor de uma constante segue o op-code da instrução e é usado como um valor constante.

Exemplo:

```
mov a, #100      ; a = 100
```

Endereçamento indexado

Somente a memória de programa pode ser acessada via endereçamento indexado e, portanto, só pode ser lida.

Este modo de endereçamento visa o acesso a tabelas look-up. Um registrador base de 16 bits (o DPTR ou o PC) apontam para a base da tabela e o acumulador é o deslocamento (offset) dentro da tabela. O endereço de entrada na tabela é dado pelo soma do registrador base com o acumulador.

Outro modo de endereçamento indexado é usado na instrução "case jump". O endereço de salto também é calculado a partir da soma do registrador base com o acumulador.

Exemplos:

```
movc a, @a+dtpr  ; a = conteúdo do endereço a+dptr
movc a, @a+pc    ; a = conteúdo do endereço a+pc
```

[1] [5]

Instruções Aritméticas

O conjunto de instruções do 8051 contempla as principais operações aritméticas: adição, subtração, multiplicação e divisão, além de BCD (decimal codificado em binário). Todas operando sobre números inteiros e retornando inteiros. A Tabela 3 sumaria as instruções aritméticas. O tempo de execução se baseia num microcontrolador com clock de 12 Mhz.

Algumas instruções só suportam operações sobre o acumulador, mas boa parte do conjunto pode utilizar todos ou quase todos os modos de endereçamento.

Uma das instruções INC opera no DPTR permitindo, deste forma, incremento de 1 em endereço de 16 bits. Além disso qualquer endereço dentro da memória de dados interna pode ser incrementado ou decrementado sem passar antes pelo acumulador.

A instrução MUL AB multiplica os registradores A (acumulador) e B e coloca o resultado de 16 bits na concatenação destes dois registradores, sendo que B recebe o byte mais significativo e A o menos. Já a instrução DIV AB divide A por B, atribuindo a A a resposta inteira da divisão e a B o resto. Esta instrução, no entanto, é mais usada em conversões de base e operações de deslocamento de bits (shift) do que realmente para divisão. O deslocamento de bits para a direita pode ser feito dividindo-se um número por 2. Neste caso B receberá os bits de A que forem deslocados para fora.

A instrução DA A ajusta o acumulador após este ter recebido a soma de dois números no formato BCD. Deve-se notar que DA não faz uma conversão de binário para BCD mas apenas um ajuste como uma instrução a ser usada na segunda etapa de uma operação em BCD. [5]

Mnemônico	**Operação**	**Modos de endereçamento**				**Tempo execução (μs)**
		Dir	**Ind**	**Reg**	**Imed**	
ADD A, <byte>	A = A + <byte>	X	X	X	X	1
ADDC A, <byte>	A = A + <byte> + C	X	X	X	X	1
SUBB A, <byte>	A = A - <byte> - C	X	X	X	X	1
INC A	A = A + 1	Somente acumulador				1
INC <byte>	<byte> = <byte> + 1	X	X	X		1
INC DPTR	DPTR = DPTR + 1	Somente DPTR				2
DEC A	A = A - 1	Somente acumulador				1
DEC <byte>	<byte> = <byte> - 1	X	X	X		1
MUL AB	B:A = BxA	Somente acumulador e B				4
DIV AB	A = Int [A/B] B = Mod [A/B]	Somente acumulador e B				4
DA A	Ajuste decimal	Somente acumulador				1

Tabela 3 - Conjunto de instruções aritméticas

Instruções Lógicas

As instruções que realizam operações booleanas (AND, OR, NOT) em bytes realizam a operação bit-a-bit. Isto é, se o acumulador contém 00110101B e <byte> contém 01010011B, então a instrução

```
ANL A, <byte>
```

deixará o acumulador com o valor 00010001B

Todas as instruções que utilizam que são específicas do acumulador executam em 1 cíclo máquina. As outras levam 2 cíclos.

As instruções lógicas podem ser executadas em qualquer dos 128 primeiros bytes da memória de dados interna ou no SFR usando endereçamento direto sem ter que passar pelo acumulador. Em rotinas de tratamento de interrupção isto pode eliminar uma instrução extra que seria necessária para salvar o acumulador na pilha antes de poder ser utilizado nas instruções booleanas da rotina.

A instrução SWAP troca a posição do nibble (4 bits) mais significativo com o menos significativo do acumulador. Esta é uma operação muito útil para operações de manipulação de BCD. Por exemplo, se o acumulador contém um número que se sabe ser menor que 100, este pode ser convertido rapidamente para BCD através do código [5]:

```
MOV B, #10
DIV AB
SWAP
ADD A,B
```

Transferência de dados

RAM Interna

As instrução MOV <destino>, <fonte> permite que os dados sejam transferidos entre duas localizações quaisquer da RAM interna ou do SFR sem Ter que passar pelo acumulador.

Em todos dispositivos 8051, a pilha reside na RAM on-chip e cresce para cima. A instrução PUSH, primeiro incrementa o SP (stack pointer), então copia o byte na pilha. PUSH e POP utilizam somente endereçamento direto para identificar o byte

sendo salvo o restaurado, mas a pilha mesmo é acessada por endereçamento indireto usando o registrador SP. Isto significa que a pilha pode ir até os 128 bytes superiores da memória interna, se implementados, mas não até o espaço do SFR.

Em dispositivos com apenas 128 de memória RAM interna, se o SP aponta para acima dos primeiros 128 bytes, bytes salvos com PUSH são perdidos e restaurados com POP têm seus valores indeterminados.

As instruções de transferência de dados incluem um MOV de 16 bits que pode ser utilizado para inicializar o DPTR para tabelas look-up na memória de programa ou para acessos de 16 bits à memória de dados externa.

A instrução XCH A, <byte> faz com que o acumulador e o byte endereçado troquem seus dados. A instrução XCHD A,@Ri é similar, com a diferença que apenas os nibbles menos significativos são trocados.

RAM externa

O 8051 dispõe da instrução MOVX para acesso à RAM externa. Somente endereçamento indireto pode ser usado. A escolha é se o endereçamento será de 8 bits ou 16 bits. No caso de endereçamento de 8 bits o registrador R0 ou o R1, do banco selecionado, funciona como ponteiro. Se o endereçamento for de 16 bits, o registrador DPTR é que se torna o ponteiro.

Existe uma desvantagem em se utilizar endereçamento de 16 bits, é que, neste caso, todos os 8 bits da porta P2 são usados como bus de endereço.

Todas as instruções de acesso à memória externa levam 2 ciclos da máquina para serem executadas e todas utilizam o acumulador como fonte ou destino do byte.

Tabelas lookup

Tabelas lookup são tabelas que residem na memória de programa e são lidas através da instrução MOVC (*move constant*). O endereço utiliza o DPTR ou o PC como base e o acumulador como deslocamento (*offset*).

A instrução

```
MOVC A, @A + DPTR
```

copia o byte no endereço A+DPTR para o acumulador. Como o deslocamento é de 1 byte, a tabela pode ter até 256 bytes de tamanho (de 0 a 255).

Quando se utiliza PC como endereço base, a tabela lookup deve ser acessada via uma subrotina.

Primeiro carrega-se o acumulador com o índice do item desejado dentro da tabela, em seguida chama-se a subrotina que efetivamente lerá o byte que o retornará também no acumulador:

```
MOV A, número_da_entrada
CALL tabela
```

A subrotina deverá parecer com algo como:

```
tabela:     MOVC A, @A + PC
            RET
```

A tabela em si deve vir logo após a instrução RET. Quando a instrução MOVC é executada o registrador PC contém o endereço da instrução RET, que é a próxima instrução a ser executada. O índice 0 (zero) da tabela é perdido pois aponta exatamente para a instrução RET, a tabela perde então 1 byte, passando a poder conter até 255 entradas (de 1 a 255). [5]

Instruções booleanas

O 8051 contém um conjunto completo de instruções booleanas (de 1 bit). A memória interna contém 128 bits endereçáveis individualmente, e o espaço do SFR pode suportar até mais 128 bits endereçáveis.

Todas as linhas das portas são endereçáveis bit-a-bit, e cada uma pode ser tratada como portas separadas de 1 bit.

Toda acesso bit-a-bit é endereçado diretamente. Os bits de 00h até 7Fh estão nos primeiros 128 bytes da RAM e os bits de 80h a FFh estão no espaço do SFR.

O exemplo abaixo demonstra como um bit pode ser movido facilmente para uma porta:

```
MOV C,FLAG
MOV P1.0,C
```

Neste exemplo FLAG é o endereço de um bit, ou seja, um número de 00h a FFh. O bit 0 da porta 1 é setado ou limpado dependendo do valor do bit em FLAG.

O bit c*arry* da PSW é usado como um acumulador de um bit pelo processador booleano. A maioria das instruções booleanas o utiliza como operador que vai receber o bit ou como o operador que contém o valor do bit. O bit *carry* também possui um endereço direto uma vez que ele reside na PSW que é endereçável bit-a-bit.

Existe uma série de instruções de salto condicional que testam a condição de um bit. Elas testam se o bit está setado (JC, JB, JBC) ou se está limpo (JNC, JNB).

A instrução JBC além de executar o salto, limpa o bit. Desta forma um flag pode ser testado e limpado em apenas uma instrução.

Como já foi visto, todos os bits da PSW podem ser endereçados diretamente, assim o bit de paridade ou as flags de propósito geral, por exemplo, estão disponíveis às instruções de teste bit-a-bit. [5]

Instruções de salto

O 8051 possui um bom repertório de instruções de salto condicionais e incondicionais, todas elas executando em 2 ciclos-máquina (a única exceção é a instrução NOP).

Para algumas instruções o endereço de destino é um endereço absoluto, para outras é um endereço relativo. Neste último caso, o endereço é um número em completo de 2 que define de quantos bytes será o salto a partir da instrução seguinte ao *jump*. Se o endereço relativo for negativo o salto será para trás.

A instrução SJMP codifica o endereço de destino como um deslocamento relativo de 1 byte. A distância do salto está limita à faixa de -128 a +127 bytes.

Já a instrução LJMP realiza um salto longo, ou seja, o destino é um endereço absoluto de 16 bits. O endereço pode estar em qualquer lugar na área de 64K da memória de programa.

A instrução AJMP codifica o endereço como uma constante de 11 bits. Esta instrução tem 2 bytes de comprimento ao contrário de LJMP que tem 3. Quando executada, os 11 primeiros bits do PC são alterados para o endereço passado a AJMP, o outros 5 bits permanecem inalterados. Desta maneira, o destino precisa estar dentro do mesmo bloco de 2K da instrução seguinte a AJMP.

O 8051 ainda suporta *case jumps* que são saltos em que o endereço é formado pela soma de dois registradores, neste caso A e DPTR. Num caso típico, DPTR é setado com o endereço base de uma tabela de saltos e A recebe o índice dentro desta tabela. Como exemplo veja o código seguinte:

```
MOV DPTR, #TABELA
MOV A, INDICE
RL A
JMP @A+DPTR
```

A instrução RL A multiplica o índice por 2 em função de que cada entrada na tabela de salto ocupa 2 bytes:

```
TABELA:
   AJMP CASE_00
   AJMP CASE_01
   AJMP CASE_02
   AJMP CASE_03
```

Semelhantemente às instruções JMP (SJMP, LJMP e AJMP) tem-se instruções CALL para chamadas de subrotinas. LCALL realiza um chamada longa com endereço absoluto de 16 bits e ACALL uma chamada com endereço de 11 bits que esteja dentro dos 2K do bloco da instrução seguinte.

As subrotinas devem terminar com a instrução RET que provoca o retorna de execução à instrução posterior à instrução CALL.

Quando o procedimento é uma rotina de tratamento de interrupção ela deve terminar com a instrução RETI. RETI informa ao sistema de controle de interrupção que a interrupção em curso terminou.

No 8051 todas as instruções de salto condicional passam um endereço relativo para o destino que, como foi dito acima, fica limitado a uma distância entre -128 a +127 bytes a partir da próxima instrução.

As instruções JZ e JNZ testam se o acumulador é zero, ou não, respectivamente e saltam para o endereço referenciado, caso positivo.

A instrução DJNZ (decrementa e pula se não for zero) é usada principalmente para controle de loops. Para executar um loop N vezes, carrega-se um contador (um registrador por exemplo) com N e chama-se a instrução DJNA no final do loop. O endereço para o salto deve ser o endereço inicial do loop. Exemplo:

```
       MOV Counter, #10
Loop: (início do loop)
       ...
       ...
       ...
       (fim do laço)
       DJNZ Counter, Loop
```

A instrução CJNE (compare e salte se não for igual) também pode ser usada para controle de loop. Dois bytes são especificados no campo de operandos. O salto só é executado se os 2 bytes não forem iguais.

Outro uso para a instrução CJNE é em comparações do tipo "maior que, menor que". Os dois bytes são interpretados pelo instrução como inteiros sem sinal. Se o primeiro é menor que o segundo, o bit Carry do PSW é setado (1), caso contrário ele é mantido em zero. [5]

Ciclo-máquina

Um ciclo-máquina consiste de uma seqüência de 6 estados, numerados de S1 a S6. Cada estado leva 2 períodos do oscilador. Se a freqüência do oscilador é de 12 MHz então um ciclo-máquina terá um tempo de 1 milissegundo.

A execução de uma instrução de 1 cliclo começa durante o estado 1 (S1) quando o opcode da instrução é carregado no registrador de instrução. Um segundo fetch ocorre durante S4 do mesmo cilco-máquina. A execução é terminada no estado S6. [5]

Estrutura de interrupção

O 8051 disponibilizada 5 fontes de interrupção: 2 externas, 2 interrupções de timers, e a interrupção da porta serial. Outros dispositivo da família podem ter um número maior de fontes de interrupção.

Cada fonte de interrupção pode ser individualmente habilitada ou desabilitada setando-se ou limpando-se o registrador IE (*Interrupt Enable*). Este registrador possui um bit de desabilitação global que quando em 0 (zero) desabilita todas as interrupções de uma única vez.

Prioridades

A prioridade de uma interrupção é definida através do registrador IP (*Interrupt Priority*) que controla individualmente o nível de prioridade de cada interrupção. A prioridade é definida em um de 2 níveis, alto e baixo.

Um interrupção de prioridade baixa pode ser interrompida por uma de prioridade alta, mas não por outra de baixa. Uma interrupção de alta prioridade não pode ser interrompida por nenhuma outra.

Se requisições de interrupção do mesmo nível de prioridade são recebidas simultaneamente, uma seqüência de um *poll* interno diz qual das interrupções será servida primeiro.

Ao chamar uma rotina de tratamento de interrupção, o registrador PC é guardado na pilha para que seja possível retornar à instrução interrompida pela requisição. Para melhorar o tempo de resposta das rotinas de tratamento de interrupções, somente este registrador é salvo. Caso seja necessário guardar outros registradores, o programador é quem deve fazer isso na próprio rotina de tratamento. [5]

Timers

O 8051 padrão vem equipado com 2 timers, ambos podem ser controlados e configurados individualmente. Existem três funções gerais que um timer pode desempenhar: 1) manter um relógio ou calcular o tempo entre eventos; 2) contar o número de vezes que um evento ocorre; 3) Gerar *baud rates* para a porta serial.

Um timer sempre faz uma contagem crescente. Não importa se ele é usado como um temporizador, um contador ou um gerador de baud rate. Ele é sempre incrementado pelo microcontrolador.

Quando um timer é configurado para medir o tempo, ele será incrementado de 1 a cada ciclo-máquina. Como já foi visto, um ciclo-máquina consiste em 12 pulsos do relógio. Desta forma, se o microcontrolador está operando a 12 Mhz, o timer será incrementado 1 milhão de vezes a cada segundo. [6]

A número de vezes que um timer incrementou pode ser obtido pelos registrados de 8 bits TH e TL ou mais precisamente TH0 e TL0 para o timer 0 e TH1 e TL1 para o timer1. Juntos eles formam uma palavra de 16 bits, sendo TH o byte mais significativo.

Como existem somente 2 bytes para guardar o valor do timer, o valor máximo que um timer pode chegar é 65535. Quando neste valor, o próximo incremento faz o valor do timer voltar para 0.

Existem 4 modos em que um timer pode ser configurado, os principais são os modos 1 e 2.

Quando no modo 1, o timer usa todos os bits de TH e TL para a contagem, sendo, portanto, um timer de 16 bits. A cada ciclo-máquina ele é incrementado em 1, voltando a zero após o valor 65535.

No modo 2 o timer se torna um timer de 8 bits em *auto-reload*. Neste modo de operação somente TL é incrementado, o valor do timer pode ir de 0 a 255. TH guarda o valor de recarga. Quando TL tem o valor 255 e é incrementado, ao invés de voltar para zero ele recebe o valor TH. O registrador TH é sempre o valor inicial do timer quando este atinge seu limite. Este modo é muito usado para se estabelecer o baud rate da porta serial.

Os timers também pode ser usados para contar eventos. Quando configurados para tal, os timers deixam de incrementar a cada ciclo-máquina e passam a fazer isso a cada evento recebido numa das linhas de interrupção externa. Um evento é a transição de 1 para 0 em P3.2 ou P3.3 que são as linhas de interrupção externa do 8051. O timer fica aguardando a ocorrência deste tipo de transição e incrementa a cada uma detectada. Um timer assim configurado é também chamado de contador. [6] [1]

Porta serial

Uma dos componentes de grande utilidade do 8051 é a porta serial integrada que o acompanha. A existência desta porta torna possível a integração do 8051 facilmente com outros dispositivos que também possuem esta porta.

Na falta de uma porta serial integrada, seria necessário o desenvolvimento de rotinas especiais só para este fim caso, a comunicação serial fosse necessária. Dentre outras funções, estas rotinas deveriam ligar e desligar uma dos linhas de I/O para sincronizar corretamente o envio de cada bit individual, incluindo os bits *start*, *stop* e de paridade. Ao invés de gastar tempo com todo esse trabalho, tudo o que é necessário é configurar a porta serial adequadamente e usar uma instrução MOV para ler ou escrever um byte nela. Além disso outra etapa que é necessária para transmissão via serial é apenas verificar se o byte já pode ser lido ou se já foi enviado através dos *flags* RI e TI, respectivamente.

O registrador SCON (Serial Control) é usado, dentre outras coisas, para configurar o modo de operação da serial e a transmissão/recepção do 9º bit quando se utiliza paridade. Nele também estão os bits RI e TI.

A porta serial pode operar num baud rate determinado pela freqüência do relógio dividida por 12 ou 32, ou ainda pelo overflow do Timer1. Neste último caso, é possível a obtenção de vários baud rates configurando-se adequadamente este timer. [6]

Apêndice 2 - Código Fonte da Aplicação de Teste

Abaixo vai o código fonte dos arquivos necessários à compilação da aplicação no Delphi 6.0. Além dos componentes e funções padrões do Delphi foi utilizada a biblioteca inpout32 que consiste de uma DLL e um driver de dispositivo para acesso à porta paralela. Foi utilizado também um componente OCX que faz parte da biblioteca de maneira que foi possível utilizar a biblioteca visualmente, adicionado o componente à janela através da IDE do Delphi..

Arquivo: unit1.pas

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  StdCtrls, OleCtrls, HWINTERFACELib_TLB, ExtCtrls,
funcoes;

type
  TForm1 = class(TForm)
    Timer1: TTimer;
    PaintBox1: TPaintBox;
    PaintBox2: TPaintBox;
    lblLevel: TLabel;
    GroupBox1: TGroupBox;
    edCalibMin1: TEdit;
    edCalibMax1: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    cbBit4: TCheckBox;
    cbBit3: TCheckBox;
    cbBit2: TCheckBox;
    cbBit1: TCheckBox;
    PaintBox3: TPaintBox;
    edCalibMin2: TEdit;
    edCalibMax2: TEdit;
    cbAutoCalib: TCheckBox;
    cbUseFilter: TCheckBox;
    cbPlotArcs: TCheckBox;
    Hwinterface1: THwinterface;
```

```
    procedure Button2Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action:
TCloseAction);
    procedure PaintBox1Paint(Sender: TObject);
    procedure PaintBox1MouseMove(Sender: TObject; Shift:
TShiftState; X,
      Y: Integer);
    procedure PaintBox2Paint(Sender: TObject);
    procedure PaintBox2MouseMove(Sender: TObject; Shift:
TShiftState; X,
      Y: Integer);
    procedure cbAutoCalibClick(Sender: TObject);
  private
    { Private declarations }
    nextX: double;
  public
    { Public declarations }
    function LeSensor(sensor: byte): byte;
    procedure DrawArc(cx, cy, x, y: integer; color:
TColor);
    procedure Flip(box: TPaintBox; graph: TGraphic);
    procedure Blank(graph: TBitmap);
    procedure DrawGraph(n: integer);
    procedure DrawArcs;
  end;

const MAX_TERMS = 20;

var
  Form1: TForm1;
  count: integer = 0;
  status: integer;
  back: TBitmap;
  x1G1, y1G1, x2G1, y2G1, x1G2, y1G2, x2G2, y2G2,
  S1, S2, fS1, fS2, pfS: integer;
  fArrayS1, fArrayS2: array[0..MAX_TERMS-1] of byte;

implementation

{$R *.DFM}

function Intersec(r1, r2, a: integer; var x,y: integer):
boolean;
var intX, intY: double;
begin
    result := false;
    if a = 0 then exit; // não intercepta
    intX := (sqr(r1)-sqr(r2)+sqr(a))/(2*a);
    intY := sqr(r1)-sqr(intX);
```

```
    if intY <=0 then exit; // não intercepta
    intY := sqrt(intY);
    x := round(intX);
    y := round(intY);
    result := true;
end;

procedure TForm1.Blank(graph: TBitmap);
begin
    graph.Canvas.Pen.Color := clWhite;
    graph.Canvas.Rectangle(0, 0, graph.Width,
graph.Height);
end;

procedure TForm1.Flip(box: TPaintBox; graph: TGraphic);
begin
    box.Canvas.Draw(0, 0, graph);
end;

procedure TForm1.DrawArc(cx, cy, x, y: integer; color:
TColor);
var r: integer;
begin
    r := round(sqrt(sqr(x-cx)+sqr(y-cy)));
    back.Canvas.Pen.Color := color;
    back.Canvas.Brush.Style := bsClear;
    back.Canvas.Ellipse(cx-r, cy-r, cx+r, cy+r);
end;

procedure TForm1.DrawGraph(n: integer);
var x1, y1, x2, y2: ^integer;
    PaintBox: TPaintBox;
    signal1, signal2: integer;
begin
    if n = 1 then begin
      x1 := @x1G1;
      y1 := @y1G1;
      x2 := @x2G1;
      y2 := @y2G1;
      PaintBox := PaintBox2;
      signal1 := S1;
      signal2 := S2;
    end
    else begin
      x1 := @x1G2;
      y1 := @y1G2;
      x2 := @x2G2;
      y2 := @y2G2;
      PaintBox := PaintBox3;
      signal1 := fS1;
      signal2 := fS2;
```

```
    end;

    PaintBox.Canvas.MoveTo(x1^, y1^);
    PaintBox.Canvas.Pen.Color := clBlue;
    PaintBox.Canvas.LineTo(round(nextX), PaintBox.Height-
signal1);
    x1^ := PaintBox.Canvas.PenPos.x;
    y1^ := PaintBox.Canvas.PenPos.y;

    PaintBox.Canvas.MoveTo(x2^, y2^);
    PaintBox.Canvas.Pen.Color := clGreen;
    PaintBox.Canvas.LineTo(round(nextX), PaintBox.Height-
signal2);
    x2^ := PaintBox.Canvas.PenPos.x;
    y2^ := PaintBox.Canvas.PenPos.y;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    back := TBitmap.Create;
    back.Width := PaintBox2.Width;
    back.Height := PaintBox2.Height;
    PaintBox2.Canvas.MoveTo(0, PaintBox2.Height);
    PaintBox2.Canvas.Pen.Color := clBlue;
    PaintBox2.Canvas.Pen.Width := 2;

    PaintBox3.Canvas.MoveTo(0, PaintBox3.Height);
    PaintBox3.Canvas.Pen.Color := clBlue;
    PaintBox3.Canvas.Pen.Width := 2;

    nextX := 0;
    pfS := 0;
end;

procedure TForm1.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
    back.Free
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
end;

// Lê paralela:

// Porta $378
// D0 = Ativa converssão analógico-digital *
// D1 = Seleciona sensor 1
// D2 = Seleciona sensor 2
// D3 = Seleciona nibble
```

```
// Porta $379
// S7 = Bit 4 *  (pino 11)
// S6 = Bit 3    (pino 10)
// S5 = Bit 2    (pino 12)
// S4 = Bit 1    (pino 13)

// * = invertido

// sensor = 1 ou 2
function TForm1.LeSensor(sensor: byte): byte;
var n1, n2, nr: byte;
begin
    sensor := sensor shl 1;
    Hwinterface1.OutPort($378, sensor or 1); // desativa
conversor
    sleep(10);
    Hwinterface1.OutPort($378, sensor); // ativa conversor
e seleciona primeiro nibble
    sleep(10);
    n1 := Hwinterface1.InPort($379); // lê nibble
    n1 := n1 and $f0;
    n1 := n1 xor $80;
    n1 := n1 shr 4;

    Hwinterface1.OutPort($378, 8 or sensor); // seleciona
segundo nibble
    sleep(10);
    n2 := Hwinterface1.InPort($379); // lê nibble
    n2 := n2 and $f0;
    n2 := n2 xor $80;
    //memo1.lines.add('N: '+IntToBin(n2 shr
4,4)+':'+IntToBin(n1, 4));

    result := n2 or n1;

    nr := (n2 shr 4) or n1;
    if (nr and 1)=1 then cbBit1.Checked := true else
cbBit1.Checked := false;
    if (nr and 2)=2 then cbBit2.Checked := true else
cbBit2.Checked := false;
    if (nr and 4)=4 then cbBit3.Checked := true else
cbBit3.Checked := false;
    if (nr and 8)=8 then cbBit4.Checked := true else
cbBit4.Checked := false;
    //timer1.enabled := false;
end;

procedure TForm1.DrawArcs;
var r1, r2, cMin1, cMax1, range1, cMin2, cMax2, range2:
integer;
```

```
    //boardRay: double;
    intX, intY, signal1, signal2: integer;
begin
    try
        cMin1 := StrToInt(edCalibMin1.Text);
        cMax1 := StrToInt(edCalibMax1.Text);
        cMin2 := StrToInt(edCalibMin2.Text);
        cMax2 := StrToInt(edCalibMax2.Text);
    except
        exit;
    end;
    if cbAutoCalib.Checked then begin
        if S1 < cMin1 then edCalibMin1.Text :=
IntToStr(S1);
        if S2 < cMin2 then edCalibMin2.Text :=
IntToStr(S2);
        if S1 > cMax1 then edCalibMax1.Text :=
IntToStr(S1);
        if S2 > cMax2 then edCalibMax2.Text :=
IntToStr(S2);
    end;
    if not cbUseFilter.Checked then begin
        signal1 := S1;
        signal2 := S2;
    end
    else begin
        signal1 := fS1;
        signal2 := fS2;
    end;
    range1 := cMax1-cMin1;
    range2 := cMax2-cMin2;
    if (range1 = 0) or (range2 = 0) then exit;
    //boardRay :=
Sqrt(Sqr(PaintBox1.Width)+Sqr(PaintBox1.Height));
    r1 := Round((1-(signal1-
cMin1)/range1)*PaintBox1.Height);
    r2 := Round((1-(signal2-
cMin2)/range2)*PaintBox1.Height);
    back.Canvas.Pen.Color := clBlack;
    back.Canvas.Brush.Style := bsSolid;
    back.Canvas.Brush.Color := clWhite;
    back.Canvas.Rectangle(0, 0, PaintBox1.Width,
PaintBox1.Height);
    back.Canvas.Pen.Color := clGreen;
    back.Canvas.Brush.Style := bsClear;
    if cbPlotArcs.Checked then back.Canvas.Ellipse(0-r2, 0-
r2, r2, r2);
    back.Canvas.Pen.Color := clBlue;
    if cbPlotArcs.Checked then
back.Canvas.Ellipse(PaintBox1.Width+r1, 0-r1,
PaintBox1.Width-r1, r1);
```

```
    back.Canvas.Pen.Color := clRed;
    if Intersec(r1, r2, PaintBox1.Width, intX, intY) then
        back.Canvas.Ellipse(PaintBox1.Width-intX-5, intY-5,
PaintBox1.Width-intX+5, intY+5);
    Flip(PaintBox1, back);
end;

procedure TForm1.Timer1Timer(Sender: TObject);
var i, mS1, mS2: integer;
begin
    S1 := LeSensor(1);
    S2 := LeSensor(2);
    fArrayS1[pfS] := S1;
    fArrayS2[pfS] := S2;
    // Cálcula Média de MAX_TERMS
    mS1 := 0;
    mS2 := 0;
    for i := 0 to MAX_TERMS-1 do begin
        mS1 := mS1 + fArrayS1[i];
        mS2 := mS2 + fArrayS2[i];
    end;
    mS1 := mS1 div MAX_TERMS;
    mS2 := mS2 div MAX_TERMS;
    fS1 := mS1;
    fS2 := mS2;
    inc(pfS);
    if pfS = MAX_TERMS then pfS := 0;
    DrawGraph(1);
    DrawGraph(2);
    DrawArcs;

    nextX := nextX + 1;
    if nextX > PaintBox2.Width then begin
        nextX := 0;
        PaintBox2.Canvas.Pen.Color := clBlack;
        PaintBox2.Canvas.Brush.Color := clWhite;
        PaintBox2.Canvas.Rectangle(0, 0, PaintBox2.Width,
PaintBox2.Height);
        PaintBox3.Canvas.Pen.Color := clBlack;
        PaintBox3.Canvas.Brush.Color := clWhite;
        PaintBox3.Canvas.Rectangle(0, 0, PaintBox3.Width,
PaintBox3.Height);
        x1G1 := 0;
        x2G1 := 0;
        x1G2 := 0;
        x2G2 := 0;
    end;

    inc(count);
end;
```

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
    Flip(PaintBox1, back);
end;

procedure TForm1.PaintBox1MouseMove(Sender: TObject; Shift:
TShiftState; X,
  Y: Integer);
begin
    if ssLeft in Shift then begin
      back.Canvas.Pen.Color := clWhite;
      back.Canvas.Brush.Style := bsSolid;
      back.Canvas.Brush.Color := clWhite;
      back.Canvas.Rectangle(0, 0, back.Width, back.Height);
      DrawArc(0, PaintBox1.Height, x, y, clGreen);
      DrawArc(PaintBox1.Width, PaintBox1.Height, x, y,
clBlue);
      back.Canvas.Pen.Color := clRed;
      back.Canvas.Brush.Style := bsClear;
      back.Canvas.Ellipse(x-5, y-5, x+5, y+5);
      Flip(PaintBox1, back);
    end;
end;

procedure TForm1.PaintBox2Paint(Sender: TObject);
begin
    Flip(Sender as TPaintBox, back);
end;

procedure TForm1.PaintBox2MouseMove(Sender: TObject; Shift:
TShiftState; X,
  Y: Integer);
begin
    lblLevel.Caption := IntToStr(255-y);
end;

procedure TForm1.cbAutoCalibClick(Sender: TObject);
begin
    if cbAutoCalib.Checked then begin
        edCalibMin1.Text := '255';
        edCalibMin2.Text := '255';
        edCalibMax1.Text := '0';
        edCalibMax2.Text := '0';
    end;
end;

end.
```

Arquivo: unit1.dfm

```
object Form1: TForm1
  Left = 226
  Top = 167
  Width = 800
  Height = 600
  Caption = 'Simula'#231#227'o'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  OnClose = FormClose
  OnCreate = FormCreate
  PixelsPerInch = 96
  TextHeight = 13
  object PaintBox1: TPaintBox
    Left = 264
    Top = 8
    Width = 241
    Height = 209
    OnMouseMove = PaintBox1MouseMove
    OnPaint = PaintBox1Paint
  end
  object PaintBox2: TPaintBox
    Left = 8
    Top = 224
    Width = 377
    Height = 255
    OnMouseMove = PaintBox2MouseMove
    OnPaint = PaintBox2Paint
  end
  object lblLevel: TLabel
    Left = 8
    Top = 480
    Width = 36
    Height = 13
    Caption = 'lblLevel'
  end
  object PaintBox3: TPaintBox
    Left = 392
    Top = 224
    Width = 377
    Height = 255
    OnMouseMove = PaintBox2MouseMove
    OnPaint = PaintBox2Paint
  end
```

```
object GroupBox1: TGroupBox
  Left = 8
  Top = 8
  Width = 153
  Height = 129
  Caption = 'Calibra'#231#227'o'
  TabOrder = 0
  object Label1: TLabel
    Left = 8
    Top = 24
    Width = 35
    Height = 13
    Caption = 'M'#237'nimo'
  end
  object Label2: TLabel
    Left = 8
    Top = 72
    Width = 36
    Height = 13
    Caption = 'M'#225'ximo'
  end
  object edCalibMin1: TEdit
    Left = 8
    Top = 40
    Width = 41
    Height = 21
    TabOrder = 0
    Text = '0'
  end
  object edCalibMax1: TEdit
    Left = 8
    Top = 88
    Width = 41
    Height = 21
    TabOrder = 1
    Text = '255'
  end
end
object cbBit4: TCheckBox
  Left = 8
  Top = 496
  Width = 17
  Height = 17
  TabOrder = 1
end
object cbBit3: TCheckBox
  Left = 24
  Top = 496
  Width = 17
  Height = 17
  Caption = 'cbBit3'
```

```
  TabOrder = 2
end
object cbBit2: TCheckBox
  Left = 40
  Top = 496
  Width = 17
  Height = 17
  Caption = 'cbBit2'
  TabOrder = 3
end
object cbBit1: TCheckBox
  Left = 56
  Top = 496
  Width = 17
  Height = 17
  Caption = 'cbBit1'
  TabOrder = 4
end
object edCalibMin2: TEdit
  Left = 64
  Top = 48
  Width = 41
  Height = 21
  TabOrder = 5
  Text = '0'
end
object edCalibMax2: TEdit
  Left = 64
  Top = 96
  Width = 41
  Height = 21
  TabOrder = 6
  Text = '255'
end
object cbAutoCalib: TCheckBox
  Left = 8
  Top = 144
  Width = 97
  Height = 17
  Caption = 'Auto calibrar'
  TabOrder = 7
  OnClick = cbAutoCalibClick
end
object cbUseFilter: TCheckBox
  Left = 512
  Top = 8
  Width = 97
  Height = 17
  Caption = 'Usar filtro'
  TabOrder = 8
end
```

```
  object cbPlotArcs: TCheckBox
    Left = 512
    Top = 32
    Width = 97
    Height = 17
    Caption = 'Arcos'
    Checked = True
    State = cbChecked
    TabOrder = 9
  end
  object Hwinterface1: THwinterface
    Left = 648
    Top = 16
    Width = 32
    Height = 32
    ControlData = {00000100560A00002B05000000000000}
  end
  object Timer1: TTimer
    Interval = 50
    OnTimer = Timer1Timer
    Left = 744
    Top = 16
  end
end
```

Arquivo: project1.dpr

```
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1},
  funcoes in 'funcoes.pas';

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Bibliografia

[1] Marcelino, Rui. *Microprocessadores II*. Universidade do Algarve, Escola Superior de Tecnologia, ADEE. Artigo disponível no endereço http://w3.ualg.pt/~rmarcel.

[2] Costa, Alcides Silveira; Martins, Alex Silveira; Cunha, Cristina; Abib, Elbio Renato Torres; Zan, Rafael; Drebes, Roberto. *Microcontrolador 8051*. Artigo disponível no endereço http://www.inf.ufrgs.br/pct/disciplinas/sem_8051.pdf.

[3] ATMEL. *C51 – Microcontrollers Selection Guide*. Artigo disponível no endereço http://www.atmel.com/atmel/acrobat/sel_guide.pdf.

[4] *Intel 80386 Reference Programmer's Manual*. WEBSter - The Place on the Net to Learn Assembly Language Programming. Artigo disponível no endereço http://webster.cs.ucr.edu/index.html.

[5] Intel. *MCS 51 Microcontroller Family User's Manual*. 1994. Artigo disponível no endereço http://www.intel.com/design/mcs51/manuals/27238302.pdf.

[6] jgerdes@ee.fit.edu. *Assembly Tutorial*. 1998. Artigo disponível no endereço http://www.ee.fit.edu/courses/ece1552/ATutor.htm.

[7] Zelenovsky, Ricardo; Mendonça, Alexandre. *Introdução aos Sistemas Embutidos*. Artigo disponível no endereço http://www.mzeditora.com.br/artigos/embut.htm.

[8] DPINFO. *Pequenos chips de controle devem invadir o dia-a-dia até 2005*. Ano1, edição 3, 06 de agosto de 2002. Fonte UOL. Artigo disponível no endereço http://www.fozdoiguacu.pr.gov.br/dpinfo/edicao03/.

[9] Floriano Pinheiro de Camargo, Alisson; José Montebeller, Sidney. *Aplicação de microcontroladores*. Faculdade de Engenharia de Sorocaba. Iniciação científica 2001. Artigo disponível no endereço http://www.facens.br/site/ensino/projetos/microcont_alisson/FACULDADE.htm.

[10] Akira Ito, Sérgio; Pezzuol Jacobi, Ricardo; Carro, Luigi. *Projeto de Aplicações Específicas com Microcontroladores Java Dedicados*. Artigo disponível no endereço http://www.inf.ufrgs.br/pos/SemanaAcademica/Semana99/ito/ito.html.

[11] Gunée, Rickard. *How to Generate Video Signals in Software Using PIC*. Rickard's eletronic projects page. Artigo disponível no endereço http://www.rickard.gunee.com/projects/video/pic/howto.php.

[12] Fiorante, Glauco Rogério Cugler. *Aplicações Em Visão Robótica Estéreo Com Sensores De Imagem Ccd's*. Escola Politécnica da Universidade de São Paulo. São Paulo. 23/08/2001. Artigo disponível no endereço http://sim.lme.usp.br/linhas/iinteligente/pimagem/sistaquisicao/aplicarobotica.pdf.

[13] Ferreira, Eduardo Alves et al. *Monografia Sobre Atualidades em Arquitetura de Computadores*. Tema: Microcontroladores.

[14] DSP Home Page. *DSP Tutorial - Introducing to Digital Signal Processing*. Artigo disponível no endereço http://www.dsptutor.freeuk.com/.

[15] Peacock, Craig. *Interfacing the Standard Parallel Port*. Artigo disponível no endereço http://www.beyondlogic.org/spp/parallel.htm.

[16] Messias, Antônio Rogério. *Porta Paralela*. Artigo disponível no endereço http://www.rogercom.com.

[17] Taub, Herbet. *Circuitos Digitais e Microprocessadores*. São Paulo. McGrawHill do Brasil, 1984.