UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO DE LABORATÓRIO DE EXPERIMENTAÇÃO REMOTA VIA WEB

NADER ZANOTTO

BLUMENAU
2006

2006/2-21

NADER ZANOTTO

PROTÓTIPO DE LABORATÓRIO DE EXPERIMENTAÇÃO REMOTA MULTIPLATAFORMA CLIENTE

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Miguel Alexandre Wisintainer - Orientador

BLUMENAU
2006

2006/2-21

PROTÓTIPO DE LABORATÓRIO DE EXPERIMENTAÇÃO REMOTA MULTIPLATAFORMA CLIENTE

Por

NADER ZANOTTO

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Miguel Alexandre Wisintainer – Orientador, FURB
Membro:	Prof. Nome do professor, Titulação – FURB
Membro:	Prof. Nome do professor, Titulação – FURB

Blumenau, 13 de dezembro de 2006

Dedico este trabalho a minha noiva Rafaella que muito esteve ao meu lado me dando força e sendo compreensiva nesta fase de minha vida, a minha família que nunca me falta e a todos os amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

À Deus, pelo seu imenso amor e graça.

À minha noiva Rafaella, te amo.

À minha família, que sempre esteve presente.

Aos meus amigos, pelos empurrões e cobranças.

Ao meu orientador, Miguel Alexandre Wisintainer, que me auxiliou na conclusão deste trabalho.

Os bons livros fazem “sacar” para fora o que a pessoa tem de melhor dentro dela.

Lina Sotis Francesco Moratti

RESUMO

Este trabalho apresenta o desenvolvimento de um protótipo que permite realizar experimentos reais com o microcontrolador 89C51. Este protótipo permite realizar consultas aos principais registradores do microntrolador, memória, e enviar estímulos. Também disponibiliza um montador para analisar e montar o código a ser processado pelo 89C51.

Palavras-chave: Laboratório. Microcontrolador 89C51. Acesso remoto.

ABSTRACT

This work presents the development of a prototype that allows to carry through real experiments with the microcontroller 89C51. This prototype allows to carry through consultations to the main registers of the microcontroller, memory, and to send stimulus. It also makes available an assembler to analyze ant set up the code to be processed by the 89C51.

Key-words: Laboratory. Microcontroller 89C51. Remote access.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

AGP – *Acelerated Graphics Port*

CI – Circuito Integrado

EEPROM – *Electrically Erasable Programmable Read-Only Memory*

EPROM – *Erasable Programmable Read-Only Memory*

GAL – *Gateway Array Logical*

HTML – *HyperText Markup Language*

ISA – *Industrial Standard Architecture*

PC – *Personal Computer*

PCI – *Peripheral Component Interconnect*

RAM – *Random Access Memory*

ULA – Unidade Lógica Aritmética

UML – *Unifield Modeling Language*

LISTA DE SÍMBOLOS

H - hexadecimal

SUMÁRIO

1 INTRODUÇÃO

Os serviços e aplicações remotas estão ganhando espaço cada vez maior em diversas áreas. Inúmeros recursos vêm sendo disponibilizados pela rede mundial de computadores, a Internet, para quebrar as barreiras geográficas e permitir que um número maior de pessoas possam ter acesso a esses recursos.

A educação é uma dessas áreas e um recurso que já vem sendo disponibilizado é o Laboratório de Experimentação Remota que, conforme Alves (1999), "[...] tem por objetivo ampliar a gama de utilização com a internet, propiciando interações com o mundo físico" e tentar suprir tanto a ausência de laboratórios como a insuficiência de equipamentos que acaba por restringir a interação prática. Segundo Wisintainer (1999, p. 1), "[...] o desafio é fornecer aos estudantes experiências práticas, não sendo limitadas pelos recursos finitos de laboratório", ou seja, suprir essa indisponibilidade de equipamentos de laboratório estendendo a possibilidade prática.

De acordo com Alves (1999), o Laboratório de Experimentação Remota é uma aplicação que permite obter informações reais através de um computador remoto acessando recursos sem precisar possui-los localmente, mas desde que se tenha acesso a Internet.

Na área da eletrônica, um dos recursos mais utilizados é o microcontrolador. São baratos, poderosos e com um mercado crescente, sendo facilmente encontrados e produzidos por diversos fabricantes.

Diante dos fatos apresentados acima, neste trabalho é proposto um protótipo de um laboratório para realizar experimentações remotas reais com o microcontrolador 89C51 através da linguagem de programação Assembly, para várias plataformas, fazendo uso de uma placa com a arquitetura do microcontrolador 89C51 descrita em Wisintainer (1999).

Wisintainer (1999) desenvolveu um Laboratório de Experimentação Remota com o 8051 com funcionalidade restrita à plataforma Windows e sugeriu, como continuação de trabalho, desenvolver um protótipo multiplataforma cliente na linguagem Java. Com o avanço tecnológico surgiu a plataforma .NET, tendo sido a tecnologia adotada para desenvolvimento deste trabalho.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um protótipo de um laboratório de experimentações práticas reais com o microcontrolador 89C51 via Web, através da plataforma .NET.

Os objetivos específicos do trabalho são:

a) possibilitar o desenvolvimento de software na linguagem Assembly em qualquer plataforma;
b) executar experimentos reais, como verificar o conteúdo dos registradores e memória com o microcontrolador 89C51, sem a necessidade de dispor do hardware localmente.

1.2 ESTRUTURA DO TRABALHO

No primeiro capítulo é apresentado uma contextualização do tema proposto, através de uma introdução ao trabalho desenvolvido. Também são apresentados os objetivos principais e específicos.

O segundo capítulo apresenta a fundamentação teórica na qual foi apoiado o desenvolvimento deste trabalho abordando tópicos como: microcontrolador 89C51, laboratório de experimentação remota, interação com o barramento do PC e aplicação cliente / servidor e aplicação em ambiente *web*.

No terceiro capítulo é mostrado como foi desenvolvido este trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

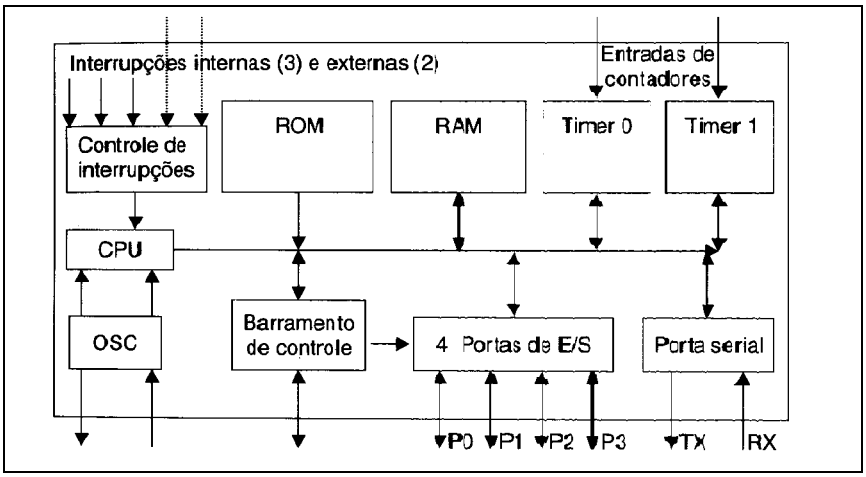
Este capítulo apresenta os aspectos teóricos dos temas abordados neste trabalho. Destacam-se o microcontrolador 89C51, laboratório de experimentação remota, interação com o barramento do PC, aplicação cliente / servidor e aplicação ambiente W*eb*. Na última seção são apresentados alguns trabalhos correlatos.

2.1 O MICROCONTROLADOR 89C51

Compatível com o microcontrolador 80C51, o 89C51 faz parte da família MCS-51 e suas principais características segundo Gimenez (2002) são:

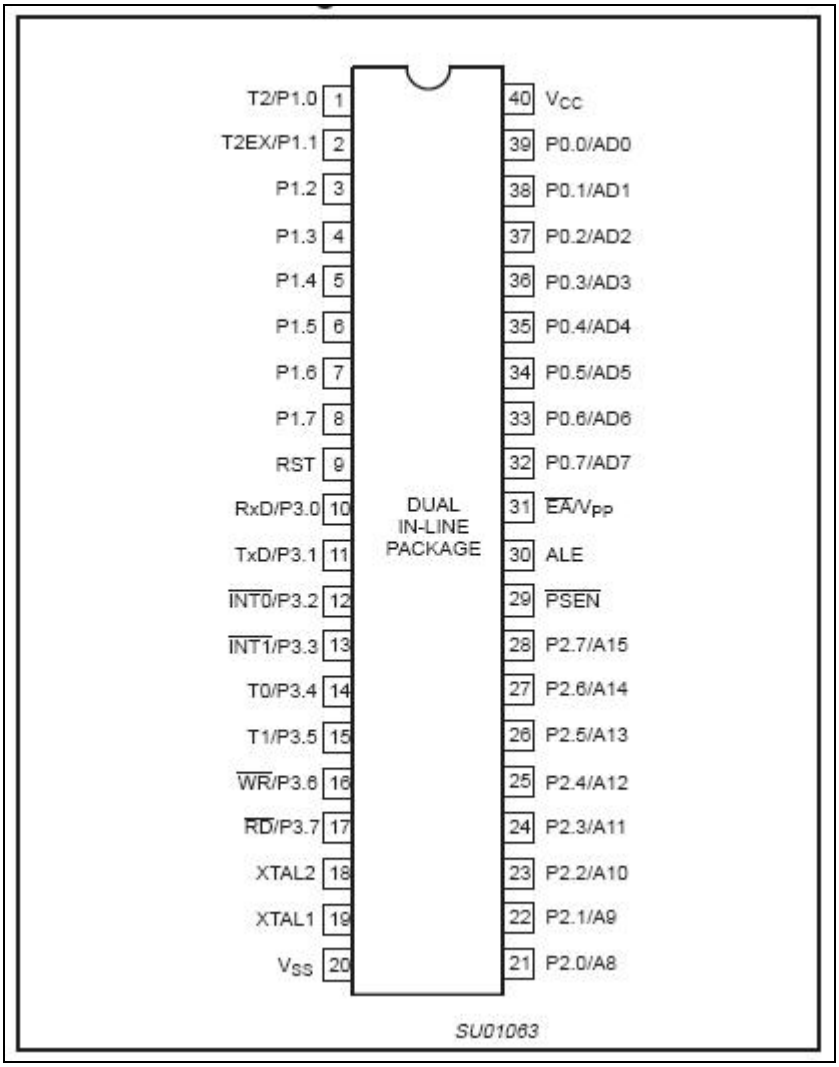
a) CPU de 8 bits otimizada para aplicações de controle;
b) Poderosa capacidade de processamento booleano, incluindo lógica individual de bits;
c) 64 Kbytes de endereçamento de memória de programa;
d) 64 Kbytes de endereçamento de memória de dados;
e) 4 Kbytes de memória de programa interna;
f) 128 bytes de memória RAM de dados interna;
g) 32 linhas de E/S bidirecionais, endereçáveis individualmente;
h) 2 *timers* /contadores de 16 bits;
i) 5 entradas de interrupções;
j) 1 oscilador interno de relógio.

A arquitetura básica do microcontrolador 89C51 pode ser vista na figura1. A pinagem é ilustrada na figura 2.



Fonte: Gimenez (2002, p. 18).

Figura 1 – Arquitetura do 89C51



Fonte: Alldatasheet (2006).

Figura 2 – Pinagem do 89C51

Os principais blocos e suas funções são descritos nos tópicos abaixo:

a) Memória: possui RAM e EPROM. A RAM é utilizada para ampliar o número de

variáveis guardadas pelo sistema e carregar programas dinamicamente, para posterior execução. A EPROM é o lugar onde o programa propriamente dito é inserido (WISINTAINER, 1999, p. 9);

b) Interface Serial: comunicação entre vários microcontroladores e outros periféricos que utilizam este dispositivo;
c) Temporizador: gera e controla ondas quadradas e permite criar dispositivos temporais;
d) Entradas / Saídas: interagem com o mundo real, controlando estados de diversos dispositivos;
e) Interrupções: monitora os eventos que ocorrem pedindo a atenção do microcontrolador. Tais eventos podem chamar essa atenção mudando de estado, de nível alto para baixo e vice e versa, na entrada do pino responsável quando a interrupção for externa e internamente através de um ato definido, como o estouro do *timer*;
f) Registradores: são posições de memória interna, reservadas e que tem funções especiais no 89C51, como os destinados à interrupção, à serial e os de uso geral (WISINTAINER, 1999, p. 9).

Há ainda para citar alguns principais registradores que são:

a) PSW: reflete a condição atual do processamento de um programa, possui um bit de utilização geral e dois selecionadores, e são influenciados diretamente pela ULA quando utilizada para funções aritméticas ou lógicas, ou seja, o registrador pode informar se houve uma condição de erro ou se a paridade do resultado obtido é par ou ímpar (GIMENES, 2002, p. 49);
b) PC: é o contador de programa, ele contabiliza as instruções executadas guardando o endereço da próxima instrução;
c) SP: guarda o contexto dos registradores, passagem dos parâmetros e retornos de função (WISINTAINER, 1999, p. 13).

No Quadro 1 pode-se ver todos os registradores de funções especiais disponíveis no 89C51.

Registradores	Endereço	Nome dos registradores
A ou ACC	E0h	Acumulador
B*	F0h	Registrador B
DPL	82h	Byte menos significativo do ponteiro de dados
DPH	83h	Byte mais significativo do ponteiro de dados
IE*	A8h	Habilitador de interrupções
IP	B8h	Priorizador de interrupções
SCON*	98h	Controlador da comunicação serial
SBUF	99h	Buffer de dados serial
PSW*	D0h	Palavra de status de programa
PCON	87h	Controle de potência
TCON*	88h	Controle do *timer*/contador
TMOD	89h	Modo de operação de *timer*/contador
TH0	8Ch	Byte mais significativo do *timer*/contador 0
TL0	8Ah	Byte menos significativo do *timer*/contador 0
TH1	8Dh	Byte mais significativo do *timer*/contador 1
TL1	8Bh	Byte menos significativo do *timer*/contador 1
P0*	80h	Porta 0
P1*	90h	Porta 1
P2*	A0h	Porta 2
P3*	B0h	Porta 3

Quadro 1 – Registradores de função especial

2.1.1 Estrutura para a execução de um programa

O microcotrolador executa um programa a partir da memória interna, a EEPROM, que ocupa o endereço 0000H à 1FFFH. Porém para se executar um programa que vá além deste bloco de endereços, faz-se necessário uma memória RAM externa que pode ser obtida utilizando os barramentos do microcontrolador que são:

a) barramento de dados: onde trafegam os dados que serão lidos ou escritos na RAM;
b) barramento de endereço: utilizado para endereçar um dado que será armazenado pela RAM ou EPROM e acessar outros dispositivos externos;
c) barramento de controle: informa o que acontece com o dado, se está sendo efetuado escrita ou leitura.

Este microcontrolador também é composto por um circuito de *reset* que serve para iniciar a execução de um programa e um oscilador obtido através de um cristal, imprescindível para que se tenha uma organização de todos os processos internos e que dá a base de tempo para o seu funcionamento (WISINTAINER, 1999, p. 16).

O 89C51 possui também um pino EA que serve para informar onde está o programa a

ser executado. Se este pino estiver em nível baixo ele busca o programa na memória externa e se estiver em alto na memória interna.

Para se acessar um dispositivo faz-se necessário um endereço, e para garantir que o microcontrolador acesse corretamente este dispositivo através do barramento de endereço, um componete é utilizado: o decodificador. Ele seleciona o dispositivo correto dado um endereço. Alguns microcontroladores possuem este decodificador embutido, mas o 89C51 necessita de um componente externo, e um CI muito utilizado é o 74138 (WISINTAINER, 1999, p.19).

2.1.2 Execução do programa

Dividindo em etapas a execução do programa do 89C51, o primeiro evento após a inicialização do microcontrolador pelo circuito de *reset* é a verificação do pino EA, para saber onde o programa está gravado e consequentemente o endereço inicial da primeira instrução.

A partir deste momento o registrador PC, que é o contador de programa já visto anteriormente, é incrementado e passa a apontar para o endereço da próxima instrução. Através do barramento de endereço a instrução chega ao microcontrolador e é executada.

Caso a instrução tenha a finalidade de escrever na RAM externa, o 89C51 devolve no barramento de endereço o endereço da RAM, no de dados o valor a ser escrito e no de controle o sinal de escrita. E se a instrução for ler um dado armazenado na RAM, o microcontrolador coloca no barramento de endereço o endereço da RAM e sinaliza ao barramento de controle que é uma leitura (WISINTAINER , 1999, p. 21).

2.1.3 Conjunto de instruções

Toda a família do MCS-51 possui o mesmo conjunto de instruções de 8 bits. Segundo Gimenez (2002, p.59) as principais operações que se pode realizar são:

a) transferência de dados;
b) aritméticas;
c) lógicas;
d) booleanas;
e) salto incondicional;
f) salto condicional;

g) sub-rotinas;

h) retorno de sub-rotinas.

No Quadro 2 pode-se ver as principais instruções disponíveis.

ACALL: Absolute call	JNZ: Jump if Accumulator Not Zero	RL: Rotate Accumulator Left
ADD, ADDC:Add	JZ: Jump if Accumulator Zero	RLC: Rotate Accumulator Left Through Carry
Accumulator (With Carry)	LCALL: Long Call	RR: Rotate Accumulator Right
AJMP Absolute Jump	LJMP: Long Jump	RRC: Rotate Accumulator Right Through Carry
ANL Bitwise AND	MOV: Move Memory	SETB: Set Bit
CJNE Compare and Jump if Not Equal	MOVC: Move Code Memory	SJMP: Short Jump
CLR. Clear Register	MOVX: Move Extend Memory	SUBB: Subtract From Accumulator With Borrow
CPL: Complement Register	MULAB Multiply Accumulator by B	SWAP: Swap Accumulator Nibbles
DA: Decimal Adjust	NOP: No Operation	XCH: Exchange Digits
DEC: Decrement Register	ORL: Bitwise OR	XRL: Bitwise
DIV: AB	POP: Pop Accumulator From Stack	
Divide Accumulator by	PUSH: Push Accumulator Onto Stack	
DJNZ: Decrement Register	RET: Return From Subroutine	
and Jump if Not Zero	RETI: Return From Internint	
INC: Increment register		
JB: Jump if Bit Set		
JBC: Jump if Bit Set and Clear Bit		
Jc: Jump if Carry Set		
JMP: Jump to Address		
JNB: Jump if Bit Not Set		
JNC Jump if Carry Not Set		

Fonte: Wisintainer (1999, p. 24).

Quadro 2 – Conjunto de instruções

2.1.4 Métodos de transferência de programas

A maneira mais tradicional de transferir um programa ao 89C51 é gravando na memória interna ou externa deste microcontrolador através de um gravador de EEPROM / EPROM, porém este é um método trabalhoso já que toda vez que se quiser alterar o programa, há a necessidade de retirar o CI e inseri-lo no gravador.

Segundo Wisintainer (1999, p. 26) um método alternativo é usar a interface serial ou paralela do 89C51 e através de um programa monitor receber o programa a ser executado.

A transferência de programa através da porta paralela do 89C51 pode ser feita

utilizando o barramento de dados e endereço do computador que podem ser acessados por meio dos *slots* do próprio computador, visto que alguns microcontroladores estão em placas de som, vídeo que usam estes *slots* para comunicação.

Segundo Wisintainer (1999, p.28) através dos *slots* é possível transferir remotamente um programa para o 89C51. Para isso, o computador necessita estar conectado a internet e utilizar uma interface para fazer essa ligação.

2.2 LABORATÓRIO DE EXPERIMENTAÇÃO REMOTA

O Laboratório de Experimentação Remota pode ser definido de um modo geral, como um conjunto de instrumentos ligados a computadores os quais estão conectados a Internet (WISINTAINER, 1999, p. 32).

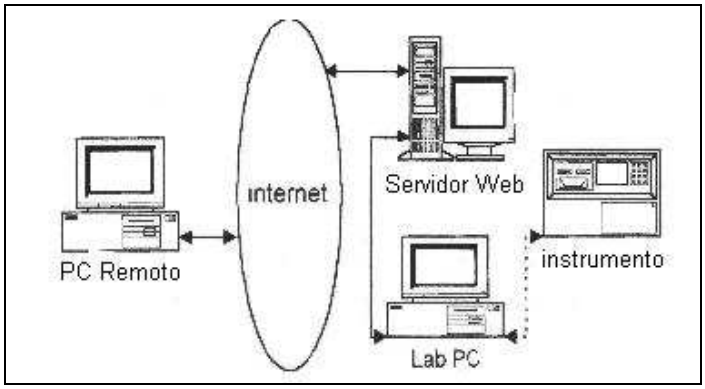
O primeiro passo para entender um laboratório de experimentação remota, é diferenciá-lo de simuladores, que não trazem resultados reais, visto que não interagem diretamente com o hardware e não podem garantir que os resultado obtidos estejam realmente corretos.

A experimentação remota permite a interação com o mundo físico através de controle eletrônico e sistemas de monitoramento gerenciados pelo computador e propiciam ao usuário controlar qualquer dispositivo ligado ao PC como se estivesse no próprio local, assegurando que os resultados obtidos sejam iguais quando manipulando-se diretamente o recurso (SEGUNDO, 2000, p. 34).

Segundo Wisintainer (1999) a experimentação remota apresenta algumas vantagens:

a) permite compartilhar o acesso a dispositivos, ampliando o número de usuários beneficiados pois não é preciso dispor do recurso, nem importa o local ou hora, basta para isso ter um computador ligado a internet;
b) permite ao usuário interagir com o recurso com mais segurança, pois não o manipula diretamente, o que é imprescindível no manuseio a equipamentos de risco;
c) assegura aos estudantes o sentimento prático, mesmo não estando nos laboratórios para realizar os experimentos;
d) reduz custos, pois com um mesmo equipamento pode-se favorecer muitos usuários.

A figura 3 mostra a estrutura básica de um laboratório de experimentação remota.



Fonte: Wisintainer (1999, p. 33).

Figura 3 – Laboratório de experimentação remota

2.3 INTERAÇÃO COM O BARRAMENTO DO PC

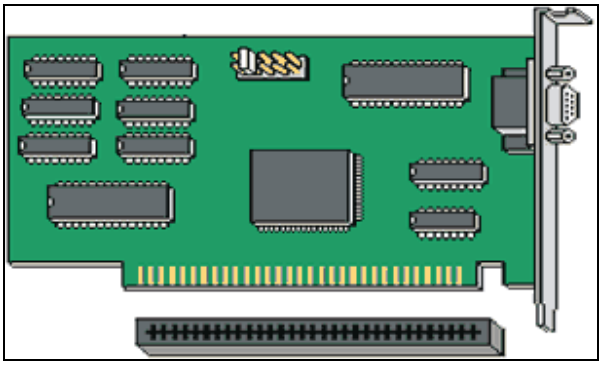
Segundo a Wikipédia (2006), pode-se definir o barramento de entrada e saída como "[...] um conjunto de circuitos e linhas de comunicação que se ligam ao resto do PC com a finalidade de possibilitar a expansão de periféricos e a instalação de novas placas no PC".

Os barramentos mais conhecidos atualmente são: ISA, PCI, AGP e PCI *Express*, cada um possui uma melhoria na comunicação com os componentes do computador.

Diante da relevancia do barramento ISA para este trabalho, deixa-se de lado o detalhamento dos outros barramentos citados.

O barramento ISA, surgiu inicialmente na versão 8 bits, o que permitia uma taxa de transferência de no máximo 8MB/s (FETTER, 2001). Era utilizado para a comunicação de periféricos nos processadores 8088 e foi o primeiro barramento de expansão.

A Figura 4 mostra uma placa adaptadora e o *slot* ISA de 8bits.

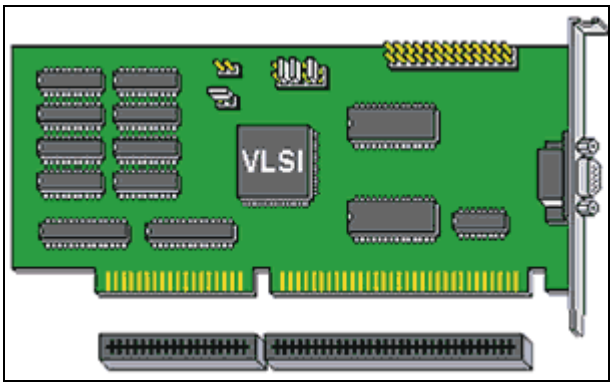


Fonte: ICEA(2005)

Figura 4 – Placa adaptadora e *slot* ISA 8 bits

A partir dos processadores 80286 da Intel o barramento ISA passou de 8 para 16 bits e fixando sua taxa máxima de transferência em 16MB/s, sendo compatível com o ISA 8bits através de técnicas de *wait-states* que nada mais são que retardos de tempo(WIKIPÉDIA, 2006).

A Figura 5 mostra uma placa adaptadora e o *slot* ISA de 16 bits.



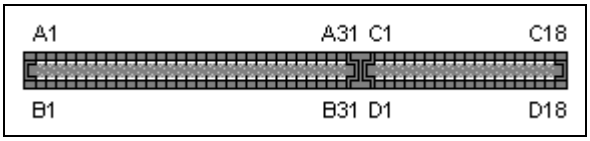
Fonte: ICEA(2005).

Figura 5 – Placa adaptadora e *slot* ISA 16 bits

Os principais pinos do *slot* ISA 16bits são:

a) A2 - A9: dados, onde são enviados e recebidos;
b) A12 – a31: endereços, onde se pode endereçar a placa inserida no *slot*;
c) B1: GROUND;
d) B3: +5V;
e) B9: +12V;

A Figura 6 ilustra a pinagem do barramento ISA 16 bits.



Fonte: ICEA(2005).

Figura 6 – pinagem do *slot* ISA 16 bits.

2.4 APLICAÇÃO CLIENTE SERVIDOR

Uma arquitetura cliente / servidor pode ser definida como uma abordagem da computação que separa processos em plataformas independentes e que permitem o compartilhamento de recursos obtendo o máximo de benefícios (BOCHENSKI, 1995, p. 8).

A funcionalidade básica desta arquitetura é de que o cliente solicita os serviços ao

servidor e este retorna os serviços solicitados. Ambos devem ter poder de processamento para realizar as suas ações.

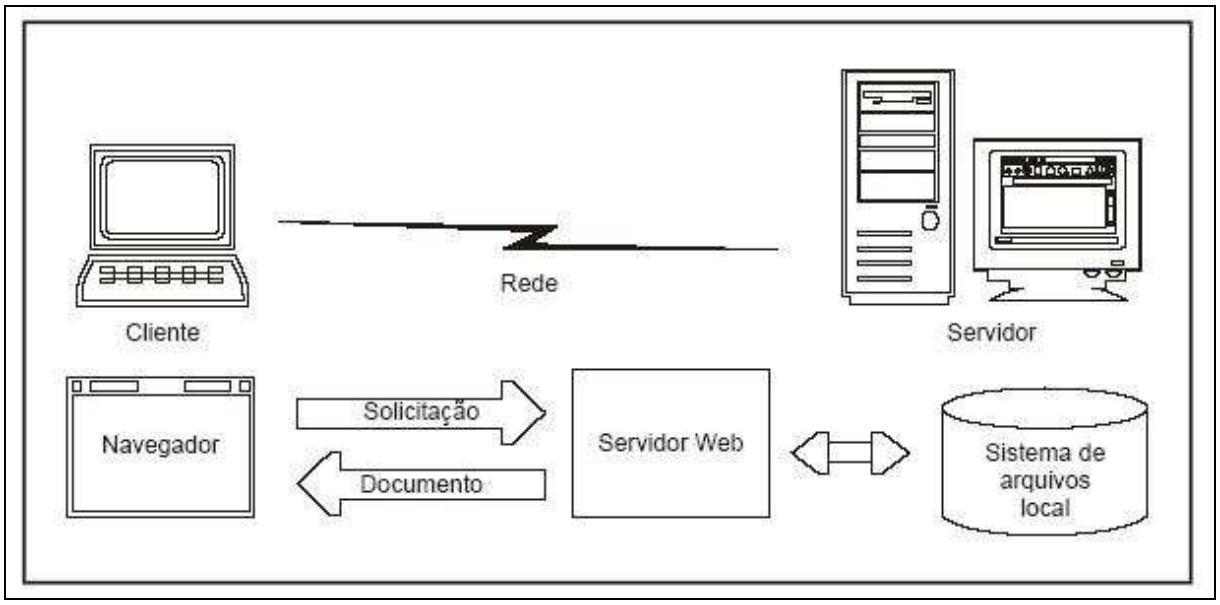
Bochenski (1995, p.9) ainda lista algumas características sobre a arquitetura cliente / servidor:

a) a parte cliente e a parte servidor pode operar em diferentes plataformas;
b) ambas podem ser atualizadas, sendo que se uma delas for, não necessariamente a outra tenha que atualizar-se também;
c) incluem algum tipo de operacionalidade em rede;
d) a ação em geral é iniciada no cliente.

2.5 APLICAÇÃO EM AMBIENTE WEB

Segundo Conallen (2003, p. 10), as aplicações *Web* evoluíram de *sites* ou sistemas *Web* que nos primórdios formavam um sistema que permitia a troca de documentos e informações publicadas através de um navegador que por sua vez, era uma aplicação executada no computador cliente, o qual solicitava um documento à outra aplicação onde esta, localizava-se no computador servidor.

A Figura 7 ilustra um sistema *Web* básico.



Fonte: Conallen (2003, p. 11).

Figura 7 – Sistema *Web* básico

Uma aplicação *Web* é uma extensão de um sistema *Web* adicionando-se as

funcionalidades de negócio, ou seja, permite ao usuário executar uma regra do negócio através de um navegador (CONALLEN, 2003, p.11).

Para que se tenha realmente uma aplicação *Web*, existe a necessidade de permitir ao usuário, além de fornecer mais do que as informações básicas na navegação, inserir uma variedade de dados, como textos simples, seleções de caixas de seleção e informações de arquivo.

2.5.1 Gerenciamento do estado do cliente

As aplicações *Web* utilizam tecnologias de ativação para tornar seu conteúdo dinâmico isto é, para que o usuário afete as regras de negócio diretamente no servidor. Para isso há a necessidade do gerenciamento do estado do cliente. Este gerenciamento pode ser feito através de *cookies*.

Um *cookie* é um fragmento de dado que um servidor *Web* pode pedir para o navegador manter e retornar cada vez que o usuário fizer uma nova solicitação a este servidor (CONALEN, 2003, p. 26).

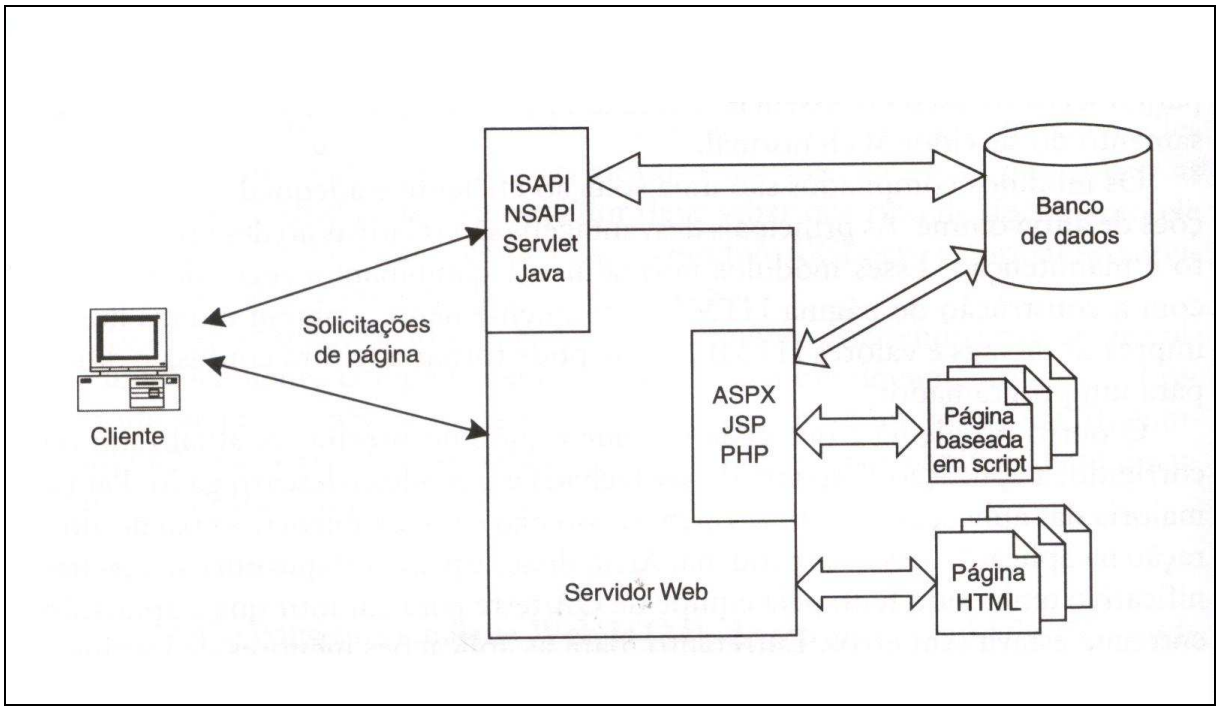
Os *cookies* podem manter também os estados das sessões que são criadas quando o usuário inicia a aplicação.

2.5.2 Tecnologias de ativação

Segundo Conallen (2003, p. 29), são tecnologias responsáveis, em parte, por tornar a aplicação *Web* dinâmica.

Elas são utilizadas através de *scripts*, que são códigos interpretados pelos servidores de aplicação. As páginas são baseadas nesses *scripts* e o servidor de aplicação identifica através da extensão do nome do arquivo.

A Figura 8 mostra o relacionamento entre algumas tecnologias e o servidor *Web*.



Fonte: Conallen (2003, p. 23).

Figura 8 – Tecnologias de ativação

Ainda na Figura 8 pode-se perceber que o cliente efetua a solicitação da página a ser visualizada, o servidor *Web* busca a página no diretório especificado, caso ela apresente algum *script*, ele interpreta interagindo com os recursos de servidor e monta a página, formatada adequadamente para enviar de volta ao cliente.

2.6 TRABALHOS CORRELATOS

Wisintainer (1999) apresenta um Laboratório de Experimentação Remota com o microcontrolador 89C51, desenvolvido na linguagem *Object Pascal*, utilizando comunicação por *sockets*, que é uma interface de comunicação bidirecional entre processos através de uma rede, entre cliente e servidor.

O usuário deve fazer o *download* do arquivo de instalação do software cliente, que está disponível somente na plataforma Windows, no *site* indicado no trabalho do autor.

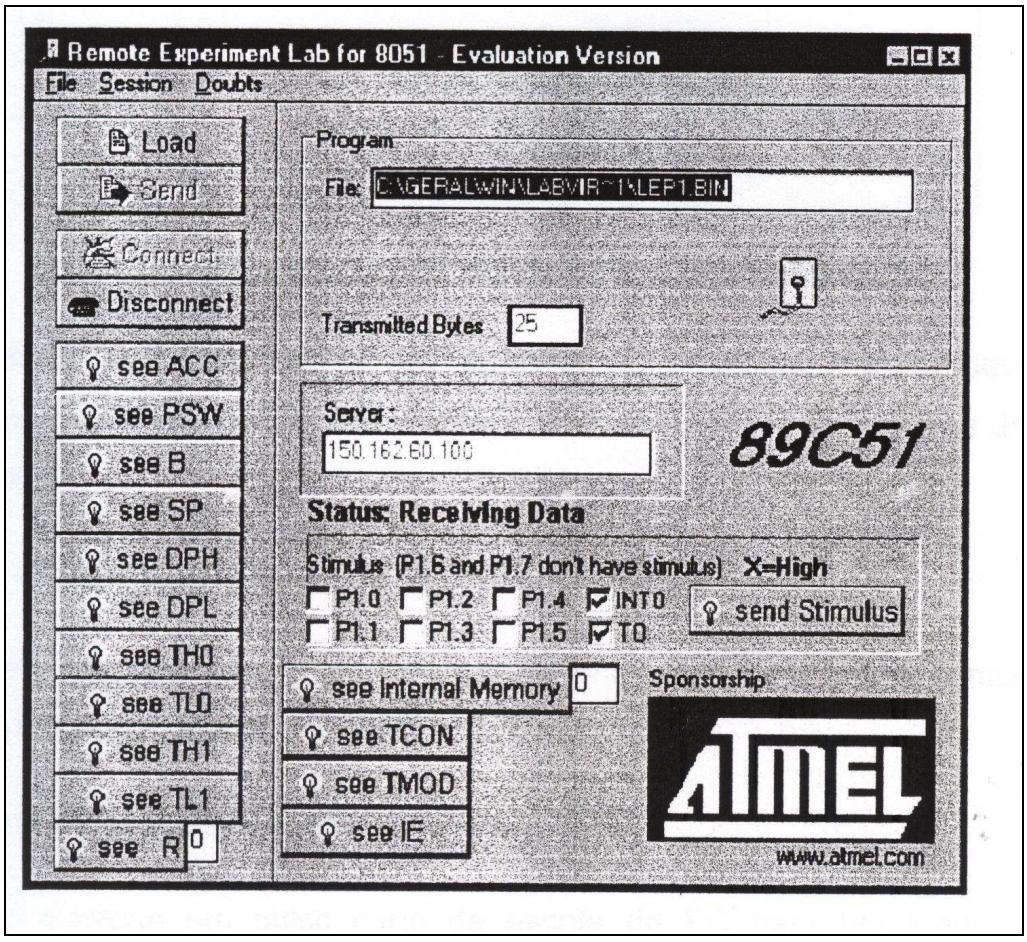
Fazendo o *download* do software cliente e instalado no PC é possível enviar um arquivo previamente montado para o servidor, para isso faz-se necessário também um programa montador para a linguagem *assembly*, pois tal função não está incorporada no trabalho.

O servidor recebe o arquivo e ele, através de funções de acesso direto a hardware, acessa uma placa construída pelo próprio autor deste laboratório.

A placa constitui-se de um circuito básico de um microcontrolador 89C51 e foi arquitetada e acoplada num *slot* ISA.

Após o envio do programa para o servidor, o usuário pode fazer verificações no conteúdo da memória, registradores do microcontrolador, além de enviar estímulos.

A Figura 9 ilustra a interface cliente deste laboratório.



Fonte: Wisintainer (1999, p. 33).

Figura 9 – Laboratório de experimentação remota

Gustavsson (1999) relata um Laboratório de Experimentação Remota para experimentos físicos em placas virtuais, simulando um laboratório de Eletrônica. Os usuários podem montar circuitos remotamente através de um software cliente, que pode ser encontrado no site indicado no trabalho do autor, e testá-los no servidor com a ajuda de um robô montador composto por um conjunto de placas que fazem a interface com diversos aparelhos de testes reais como osciloscópio e multímetro. O software do servidor foi desenvolvido na linguagem C++ e utiliza um banco de dados MySQL que guarda as informações do usuário. Para a interface gráfica do cliente foi utilizado o Visual Basic 6.

3 DESENVOLVIMENTO DO TRABALHO

Neste capítulo são apresentados os detalhes da especificação e implementação do protótipo. Na primeira parte é apresentada a descrição dos requisitos. A segunda mostra a especificação através de diagramas de casos de uso, atividades e classe. E na terceira parte, a implementação e as técnicas e ferramentas utilizadas, bem como a operacionalidade através de um caso de uso. Na quarta e última parte serão apresentados os resultados obtidos e discussões referentes aos trabalhos correlatos apresentados na fundamentação teórica.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos são classificados em: requisitos funcionais (RF) e requisitos não-funcionais (RNF). A camada cliente deverá:

a) fornecer uma área para digitação do código a ser traduzido (RF);
b) o código redigido deverá ser na linguagem Assembly (RNF);
c) enviar o código para o servidor (RF);
d) permitir acompanhar o experimento verificando o conteúdo de registradores e posições de memória (RF);
e) ser desenvolvido na linguagem Active Server Pages (ASP) .NET com C# (RNF).

A camada servidor deverá:

a) chamar um montador para verificar o código recebido (RF);
b) avisar através de mensagens ao usuário, se o código do programa Assembly está correto e se a placa do 89C51 está disponível para executar a experimentação (RF);
c) implementar as funções necessárias para prover a comunicação com a placa do microcontrolador 89C51 no formato de *Application Programming Interface* (API) (RNF);
d) ser desenvolvido na linguagem ASP .NET com C# (RNF);
e) utilizar o microcontrolador 89C51 da família do 8051 (RF).

3.2 ESPECIFICAÇÃO

Para definição da modelagem do sistema foi utilizada a UML que regulamenta os diagramas que definem o sistema.

Foram definidos os casos de uso, diagramas de atividades e diagramas de classe, detalhados a seguir.

3.2.1 Casos de uso

Os casos de uso representam a interação do usuário com o sistema, destacando as ações realizadas pelo usuário na utilização do sistema.

Neste protótipo foram definidos os seguintes casos de uso:

a) controle de acesso;
b) montar código;
c) executar programa;
d) consultar registradores e memória;
e) enviar estímulos;
f) comunicação com o hardware.

A Figura 10 mostra o caso de uso em que o usuário efetua *login* no sistema.

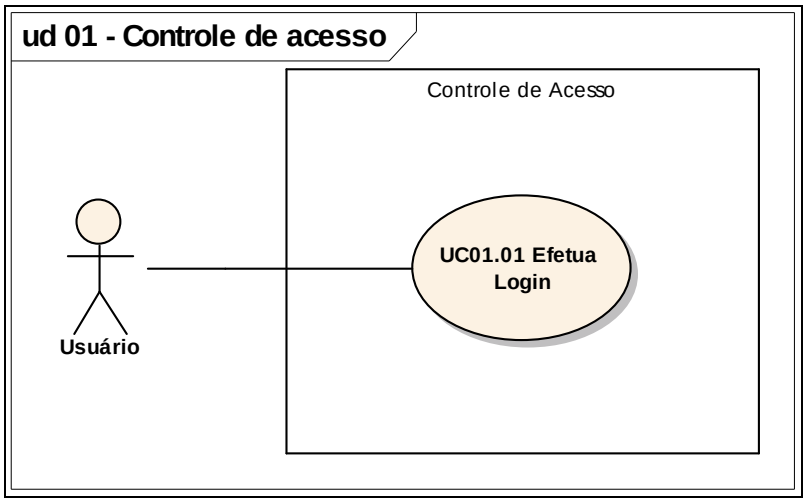


Figura 10 – Caso de uso Controle de acesso

O Quadro 3 descreve o caso de uso do controle de acesso.

CONTROLE DE ACESSO
Descrição: O usuário entra com login e senha **Ator principal:** Usuário do aplicativo **Cenário principal:** Controle de acesso 1. O sistema apresenta uma página solicitando o login e a senha do usuário. 2. O usuário preenche os dados (login/senha) e confirma. 3. O sistema valida a conta e senha fornecidas. 4. O sistema apresenta a página principal do sistema. **Cenário de exceção:** login ou senha inválidos, ou em branco. Caso o login ou a senha estejam em branco, ou incorretos o sistema apresenta mensagem "login ou senha inválidos!". **Pós-condição:** O usuário garantirá exclusividade no acesso ao sistema

Quadro 3 – Descrição do caso de uso controle de acesso

A Figura 11 mostra o caso de uso em que o usuário monta o código.

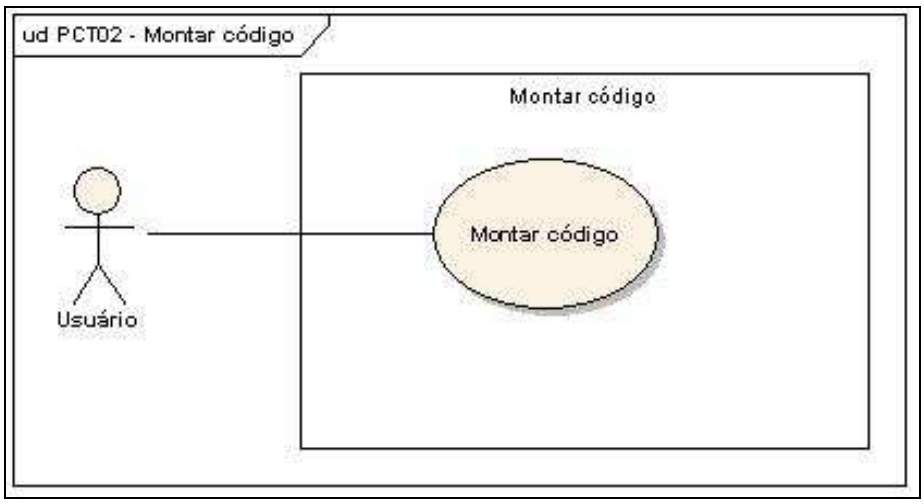


Figura 11 – Caso de uso montar código.

O Quadro 4 descreve o caso de uso montar código.

MONTAR CÓDIGO
Descrição: O usuário utiliza um montador para o código. **Ator principal:** Usuário do aplicativo **Cenário principal:** Montar Código 1. O sistema apresenta uma tela para selecionar arquivo com a extensão .asm redigido previamente a ser montado. 2. O usuário seleciona o arquivo e clica no botão montar. 3. O sistema valida o arquivo, faz um *upload* e efetua uma chamada a um montador que gera o código binário. 4. O sistema possibilita a opção para efetuar o *download* do arquivo binário gerado. 5. O usuário salva ou cancela o *download.* **Cenário de exceção:** arquivo com extensão inválida. Caso o arquivo não tenha a extensão .asm o sistema apresenta a mensagem "Extensão de arquivo inválida!" **Cenário de exceção:** Erro no código a ser montado. Caso o arquivo selecionado possua erro no seu código o sistema informará onde está o erro.

Quadro 4 – Descrição do caso de uso montar código

A Figura 12 mostra o caso de uso do usuário na execução do programa, consulta de registradores e memória e envio de estímulos.

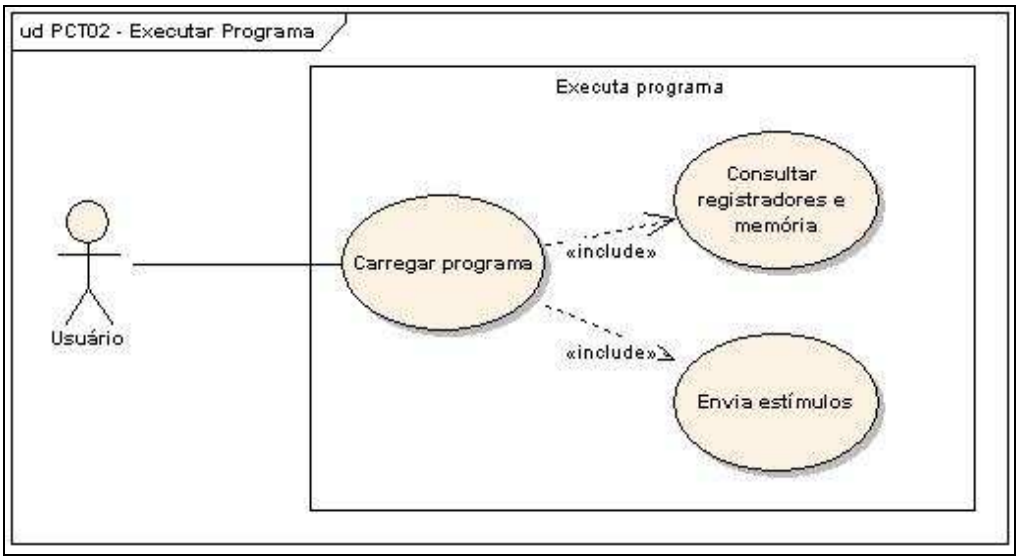


Figura 12 – Caso de uso executar programa

O Quadro 5 descreve o caso de uso carrega programa.

CARREGAR PROGRAMA
Descrição: O usuário carrega o programa na placa com o microcontrolador. **Ator principal:** Usuário do aplicativo. **Cenário principal:** Carregar programa. 1. O sistema apresenta um campo para selecionar um arquivo. 2. O usuário entra com o arquivo binário. 3. O sistema valida o arquivo e carrega na memória da placa com o microcontrolador. **Cenário de exceção:** arquivo inválido. Caso o arquivo não seja do tipo binário e maior que 32768 bytes o sistema apresenta mensagem "arquivo inválido!".

Quadro 5 – Descrição do caso de uso carregar programa

O Quadro 6 descreve o caso de uso consulta registradores e memória.

CONSULTA REGISTRADORES E MEMÓRIA

Descrição: O usuário consulta os valores dos registradores no microcontrolador.

Ator principal: Usuário do aplicativo.

Cenário principal: Consulta registradores.

1. O sistema apresenta opções para consulta dos registradores de funções especiais.
2. O sistema apresenta opções para consulta dos registradores R0-R7.
3. O sistema apresenta opção para consultar o conteúdo da memória interna.
4. O usuário seleciona o registrador ou o conteúdo da memória interna a ser consultado.
5. O sistema valida a escolha dos registradores.
7. O sistema valida a escolha da posição da memória interna.
8. O sistema consulta o registrador escolhido e retorna o valor do mesmo.
9. O sistema consulta a posição de memória interna escolhida e retorna o valor da mesma.

Cenário de exceção: Registrador inválido.

Caso o registrador escolhido não esteja entre 0 – 7 o sistema apresenta mensagem "registrador inválido!".

Cenário de exceção: Posição de memória inválida.

Caso a posição de memória escolhida não estiver entre 32 – 127 o sistema apresenta mensagem "Posição de memória inválida!".

Quadro 6 – Descrição do caso de uso consultar registradores

O Quadro 7 descreve o caso de uso envia estímulos. Os estímulos são bytes enviados as portas do microcontrolador com a finalidade de representar situações do mundo real.

ENVIA ESTÍMULOS
Descrição: O usuário envia estímulos para o microcontrolador, simulando ações do mundo real. **Ator principal:** Usuário do aplicativo. **Cenário principal:** Envia estímulos. 1. O sistema apresenta campos do tipo seleção para o envio de estímulos a placa do microcontrolador. 2. O usuário seleciona os estímulos. 3. O sistema envia os estímulos selecionados para a placa com o microcontrolador.

Quadro 7 – Descrição do caso de uso enviar estímulos

A Figura 13 representa o caso de uso do aplicativo servidor e hardware.

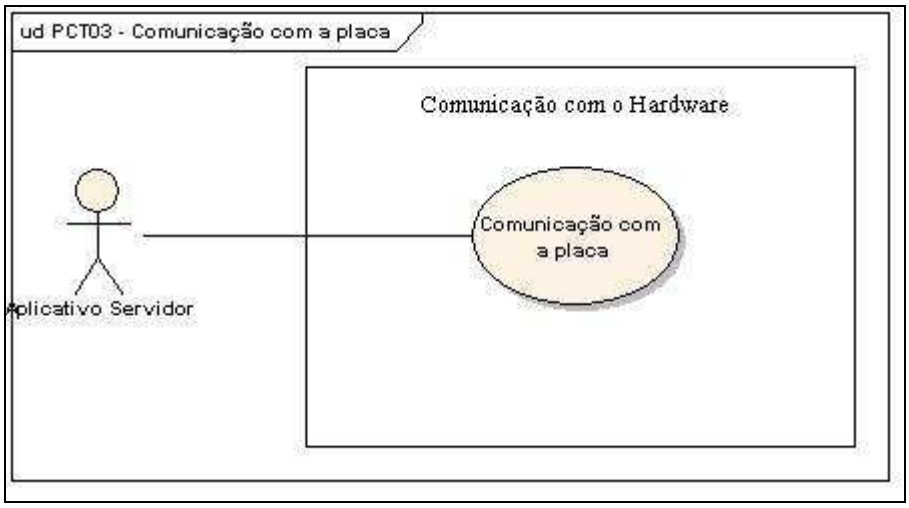


Figura 13 – Caso de uso Comunicação com o hardware

O Quadro 8 descreve o caso de uso Comunicação com o hardware.

COMUNICAÇÃO COM O HARDWARE
Descrição: O aplicativo servidor escreve e lê bytes na placa do microcontrolador através do barramento ISA. **Ator principal:** Aplicativo servidor. **Cenário principal:** Comunicação com o Hardware. 1. O aplicativo servidor escreve no barramento ISA um byte qualquer e o endereço do hardware. 2. O hardware recebe o byte e processa, escrevendo no barramento o valor processado. 3. O aplicativo servidor lê o dado no barramento informando o endereço do hardware.

Quadro 8 – Descrição do caso de uso comunicação com o hardware

3.2.2 Diagrama de atividades

Nesta seção, será apresentado os diagramas de atividades que ilustram o funcionamento do protótipo na interação com o usuário.

Para representar o funcionamento geral do protótipo são apresentados três diagramas, ilustrados nas figuras 14 à 16, que vão desde o *login*, passando pelo montador e finalizando com a execução do programa no microcontrolador.

A Figura 14 mostra o diagrama de atividades relacionado ao *login*.

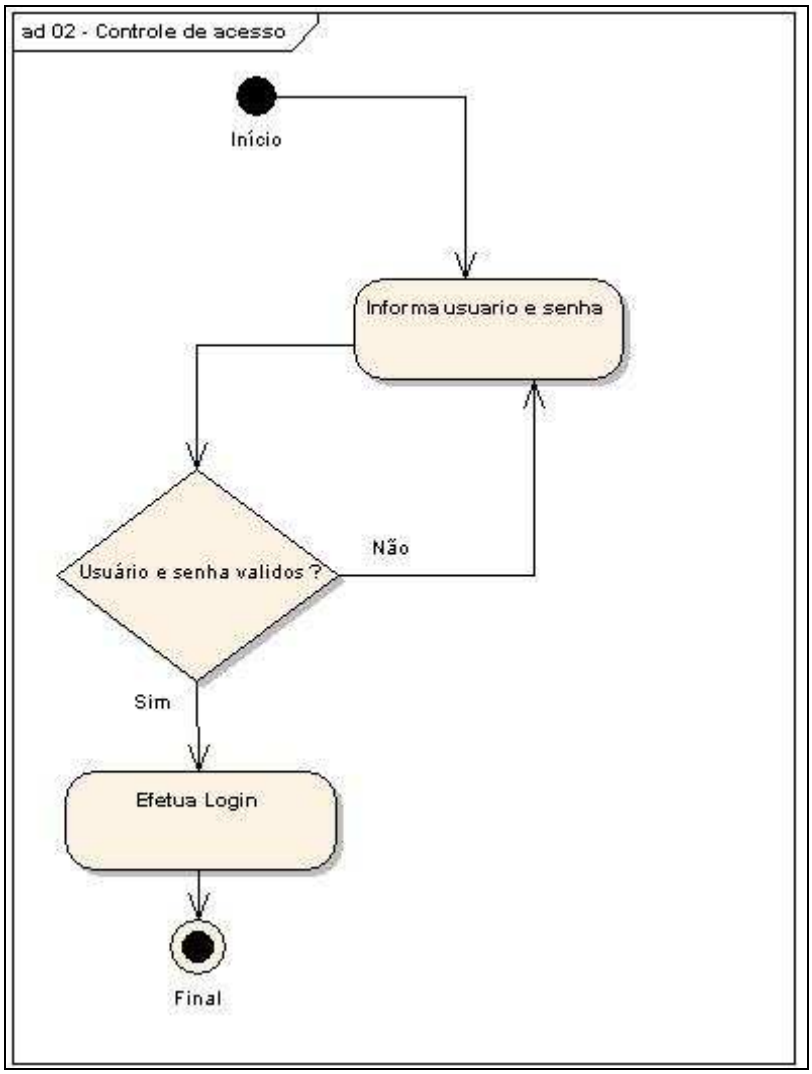


Figura 14 – Diagrama de atividades controle de acesso

A Figura 15 visualiza o diagrama de atividade relacionado ao montar o código do usuário.

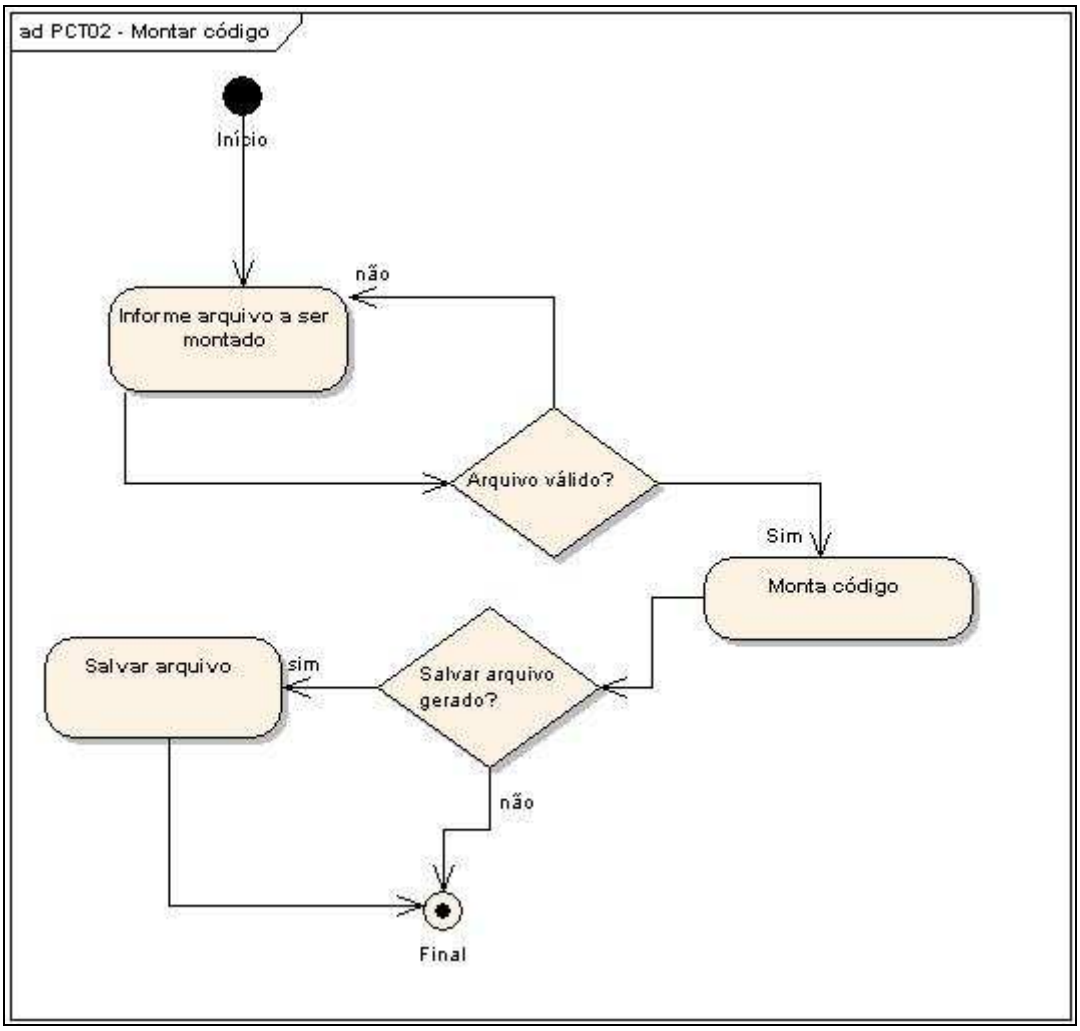


Figura 15 – Diagrama de atividades montar código

A Figura 16 mostra o diagrama de atividade relacionado à execução do código do usuário no microcontrolador.

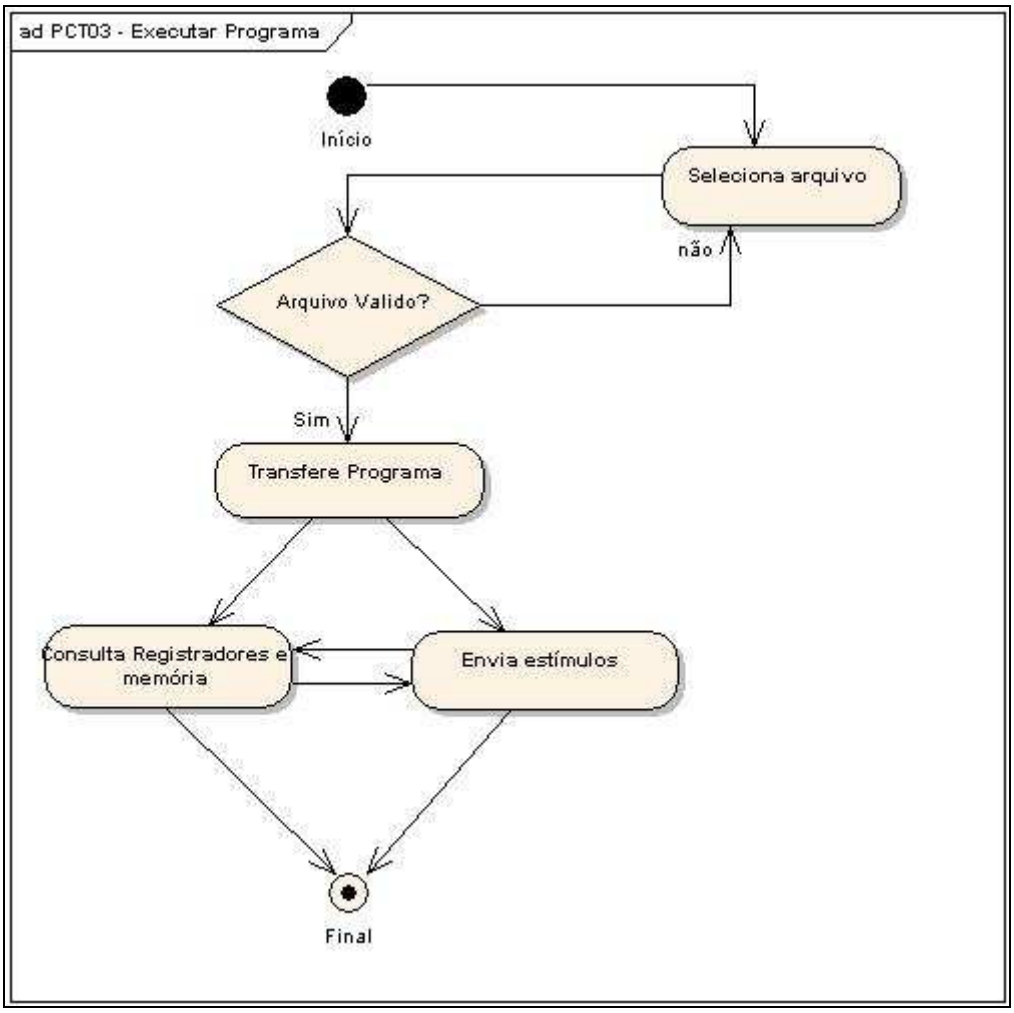


Figura 16 – Diagrama de atividades executar Programa

3.2.3 Diagrama de classes

A seguir são mostradas as classes que fazem parte do protótipo e em seguida é feito um breve relato das mesmas.

A Figura 17 ilustra o diagrama de classes do servidor.

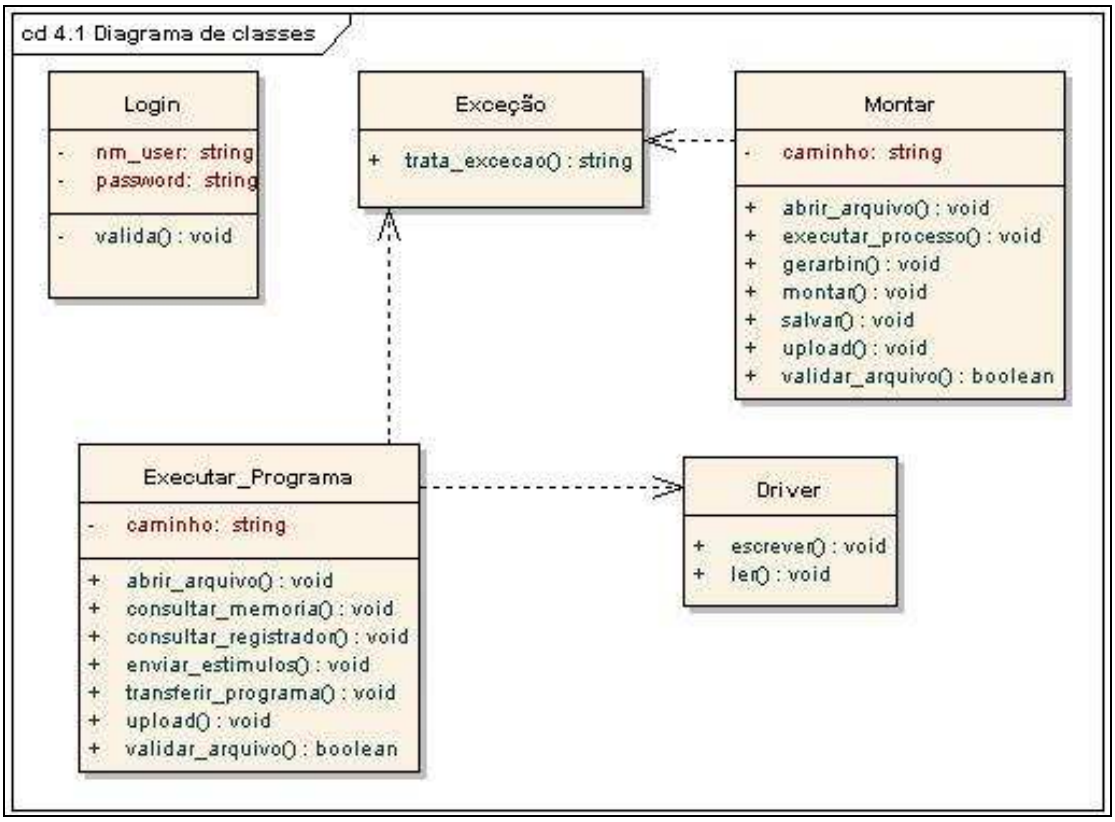


Figura 17 – Diagrama de classes

O protótipo é composto pelas seguintes classes:

a) Login: responsável por restringir o acesso e garantir a exclusividade do uso do protótipo;
b) Montar: executa a tarefa de analisar e montar o código redigido pelo usuário e indica os erros no código, caso ocorram;
c) Executar_Programa: responsável por transferir o programa do usuário para o microcontrolador, consultar os registradores e memória e enviar estímulos, faz uso dos métodos da classe Driver para tal;
d) Driver: efetua a comunicação com a placa com o microcontrolador;
e) Exceção: trata as exceções que possam vir a acontecer durante a execução do protótipo.

3.3 IMPLEMENTAÇÃO

Nesta seção são apresentados os detalhes da implementação do software, as ferramentas utilizadas, a operacionalidade através de um estudo de caso, bem como as

dificuldades encontradas no decorrer desta etapa.

3.3.1 Técnicas e ferramentas utilizadas

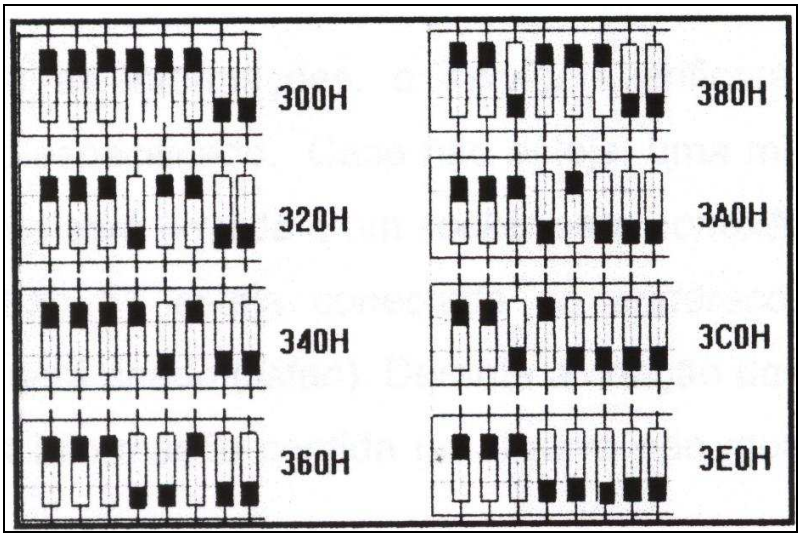
Para que fosse possível a realização deste projeto, foi utilizado o ambiente de desenvolvimento Microsoft Visual Studio 2003, a plataforma ASP.NET e a linguagem de programação C#. Também foi utilizada a placa com o microcontrolador projetada por Wisintainer (WISINTAINER, 1999), e para a comunicação do software com a placa foi empregado um *driver* proprietário da empresa Entech de Taiwan, o *RapidDriver*. Para montar o código *assembler* e gerar o código binário para o microcontrolador 89C51 foi utilizado o software ASM51 da empresa MetaLink Corporation. A especificação do protótipo foi realizada através da ferramenta *Enterprise Architect*.

3.3.1.1 A placa com o microcontrolador 89C51

O *hardware* projetado por Wisintainer (WISINTAINER, 1999), foi desenvolvido sobre um barramento padrão ISA do computador e é constituída basicamente de um microcontrolador 89C51, 4 *buffers*, 1 memória 6224, 3 monoestáveis 74121, 2 GAL e uma chave para endereçamento da placa.

Para controlar os componentes acima descritos, Wisintainer (WISINTAINER, 1999) também desenvolveu um software de monitoramento que fica armazenado na EEPROM do microcontrolador e como o próprio nome já diz, monitora o barramento ISA, aguardando por instruções destinadas ao seu endereço.

O endereço base é 3x0H onde “x” pode assumir valores que podem ser visualizados na Figura 18. Para acessar a placa, basta escrever nesses endereços.



Fonte: Wisintainer (1999, p. 85).

Figura 18 – Conjunto de endereços.

O software montador segue um padrão de execução descrito abaixo:

a) aguarda no endereço 3x1H um byte para resetar o microcontrolador;
b) aguarda no endereço 3x0H dois bytes, onde o primeiro informa o byte mais significativo do tamanho do programa a ser recebido e o segundo o byte menos significativo;
c) aguarda no endereço 3x0H os bytes correspondentes ao programa a ser transferido;
d) depois do ultimo byte do programa, aguarda no endereço 3x0H um byte que indica o inicio de execução de programa;
e) para efetuar consultas aos registradores aguarda no endereço 3x2H o número do registrador, estipulado no software monitor, a ser consultado e para efetuar consulta na memória aguarda no endereço 3x2H um número entre 32 e 127. Os valores são escritos no endereço 3x0H e;

Para saber qual registrador foi requisitado, o software montador define números para identificação de cada registrador.

O Quadro 9 mostra os registradores e seus respectivos números de identificação definidos pelo software montador.

Registrador	Número	Registrador	Número
ACC	0	R0	8
B	1	R1	9
DPH	2	R2	10
DPL	3	R3	11
TH0	4	R4	12
TL0	5	R5	13
TH1	6	R6	14
TL1	7	R7	15
TCON	16		
TMOD	17		
IE	18		
SP	19		
PSW	20		

Quadro 9 – Identificação dos registradores

A placa pode ser visualizada na Figura 19.

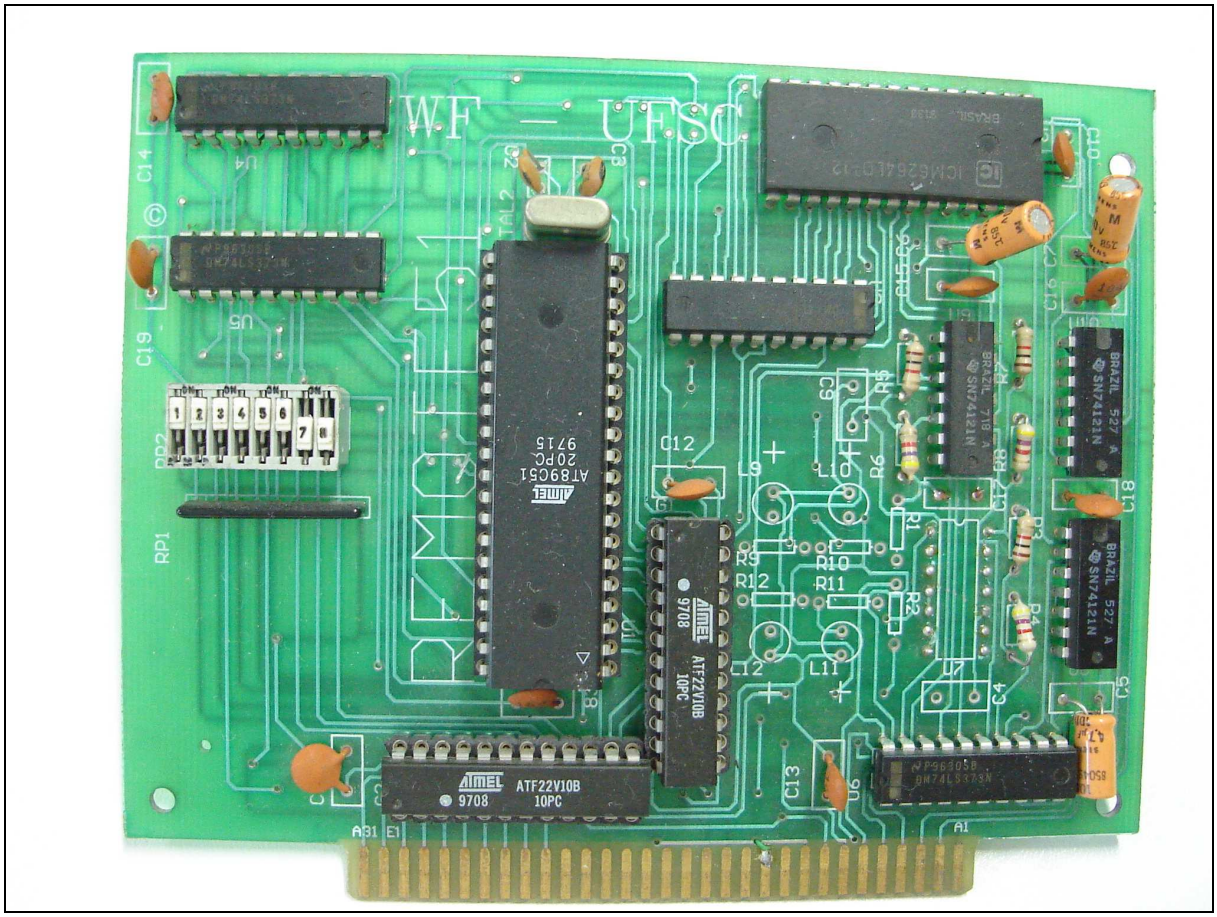


Figura 19 – Imagem da placa com o microcontrolador

3.3.1.2 *Driver* de comunicação

O Microsoft Windows XP, não permite acesso direto ao hardware (MESSIAS, 2006),

então, faz-se necessário um *driver* para efetuar a comunicação com a placa desenvolvida por Wisintainer (WISINTAINER, 1999).

O *driver* possui bibliotecas que dão suporte ao seu uso no desenvolvimento de aplicações.

A biblioteca que apresentou efetivo suporte ao barramento ISA e conseqüentemente o sucesso na comunicação, foi a *RapidDriver*.

O *driver* suporta acesso a vários barramentos, entre eles ISA e PCI. Possui versão de avaliação de 30 dias, apresenta também suporte nas principais linguagens e diversos exemplos da sua utilização.

Para poder usar as bibliotecas é necessário após a instalação do software *RapidDriver*, instalar o tipo de *driver* que irá fazer a comunicação com a placa. Para isso o software disponibiliza uma interface, onde se pode escolher o tipo do *driver*, baseado na aplicação que irá desenvolver. Neste protótipo utilizou-se o barramento ISA, sendo assim foi instalado um *driver* de comunicação com o barramento ISA.

Por se tratar de uma versão de avaliação, a interface tem que estar sempre ativa, para que o *driver* instalado possa ser aberto. A Figura 20 mostra a interface do software *RapidDriver*.

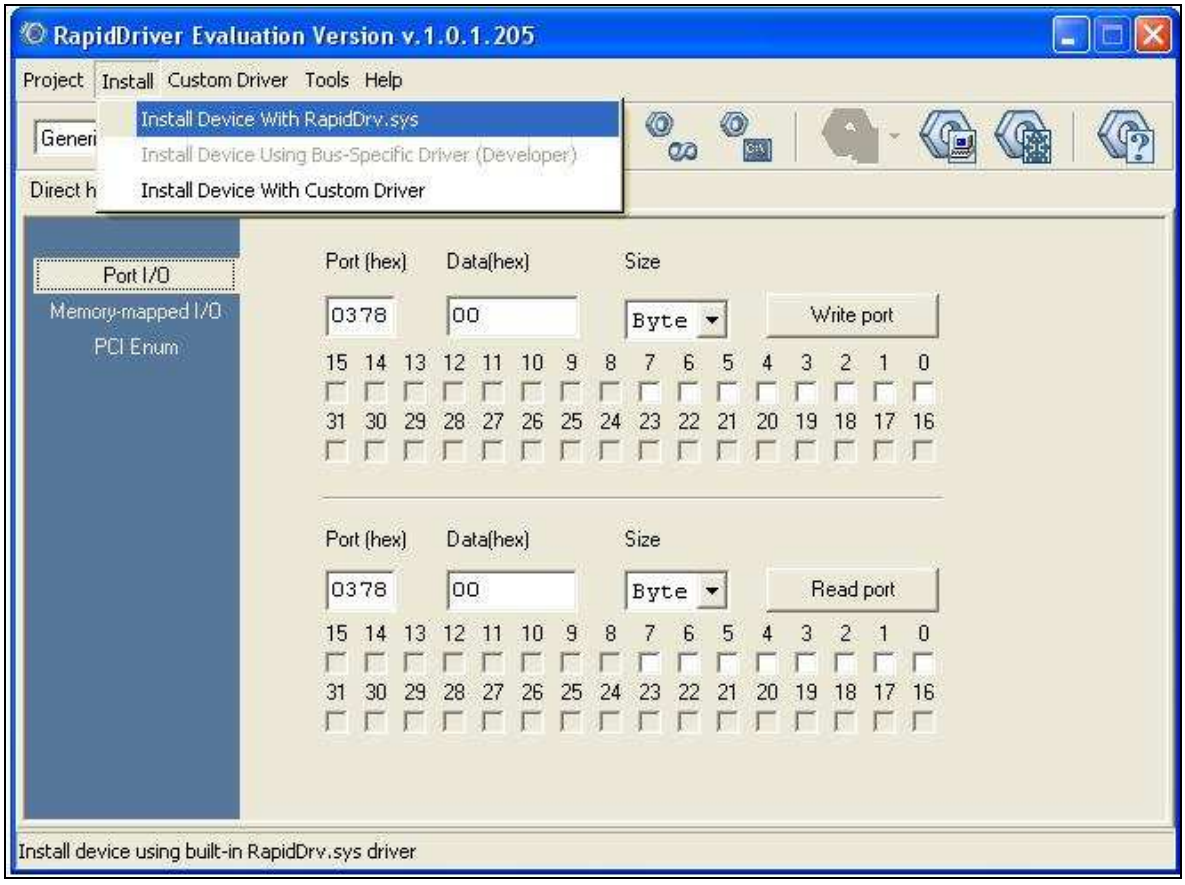


Figura 20 – Interface do *driver* de comunicação com o hardware

3.3.1.3 Controle de acesso

Ao ser iniciada a aplicação através do navegador do usuário, o qual digita o endereço do servidor onde a aplicação está sendo executada, a classe Login é instanciada, Figura 21.

Figura 21 – Página de *login*

Ao informar o nome de usuário e senha o método Valida é chamado para validar o usuário e a senha digitados e testar se não há nenhum usuário logado. O código da classe Login pode ser observado no Quadro 10.

```
private void Valida()
{
    string controle = Application["controle"].ToString();
    if (controle == "0")
    {
        //  login
        if (FormsAuthentication.Authenticate(txtUserName.Text.Trim(),
            txtPassword.Text.Trim()))
        {
            //trava a aplicação para que não haja acesso simultaneo
            Application.Lock();
            //cria controle para aplicação e libera somente um acesso
            Application["controle"] = "1";
            Application.UnLock();
            // Cria cookie de autenticação
            FormsAuthentication.SetAuthCookie(txtUserName.Text, false);
            //redireciona para a página
            Response.Redirect("Lab.aspx");

        }
        else // login falhou
        {
            // Envia msg de erro
            lblMessage.Text = "Login Invalido - Invalid Login.";

        }
    }
    else
    {
        lblMessage.Text = "software ocupado, por favor aguarde...";
    }


}
```

Quadro 10 – Validação dos usuários

Se não há usuário logado e o usuário e senha estão corretos o método mostrado no quadro acima cria uma sessão e redireciona o usuário a página principal do software.

3.3.1.4 Montador

Na página principal, Figura 22, o usuário tem a opção de selecionar um arquivo binário com o código a ser executado, ou ir para a página com o montador, para montar e gerar o código binário.

Se o usuário não tiver o arquivo binário, ele clica num *link button* que o redireciona a página com o montador que também pode ser visto na Figura 22.

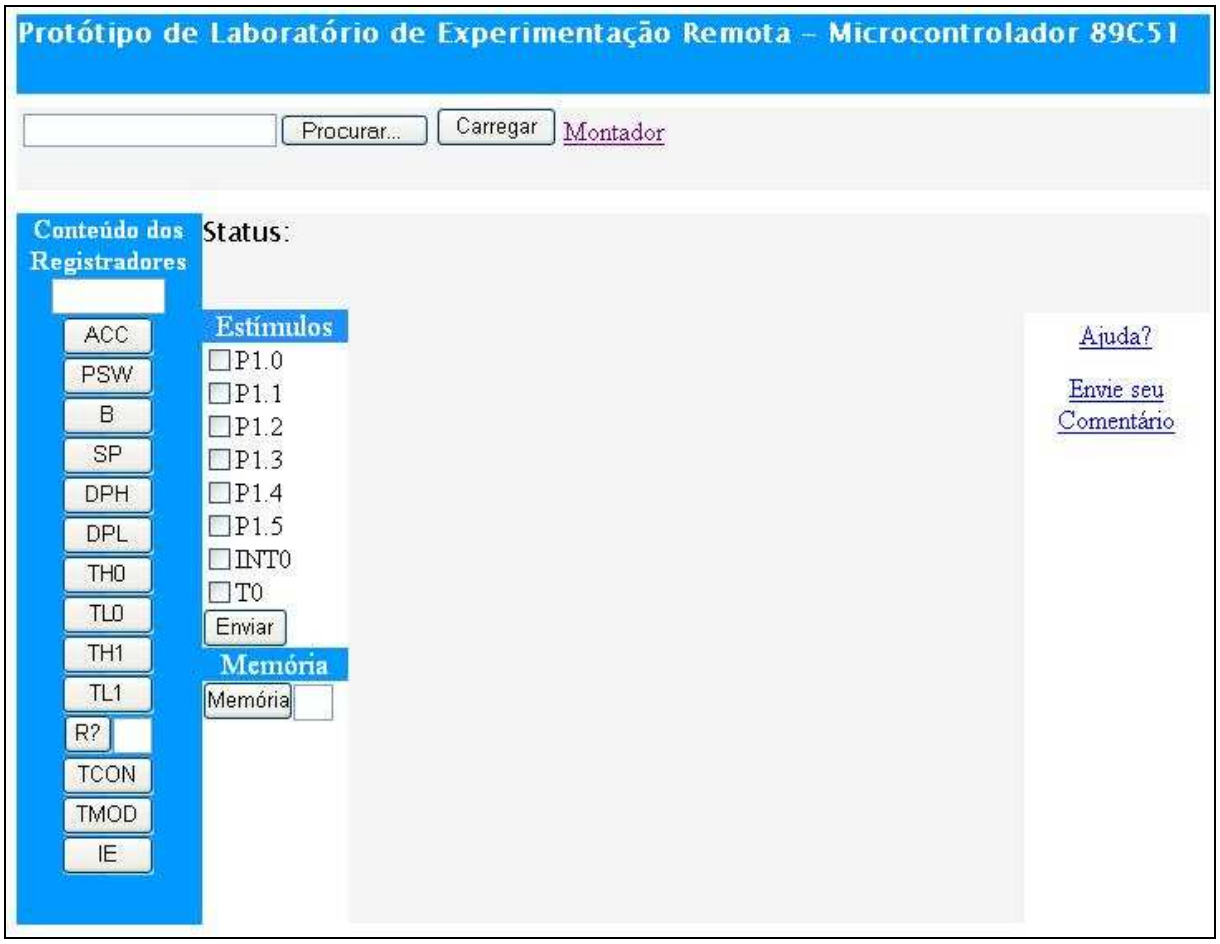


Figura 22 – Página Principal

Nesta página o usuário tem a opção de selecionar um arquivo e após a seleção clicar no botão **montar**, o método Valida_Arquivo é ativado para verificar se a extensão do arquivo é válida, como mostra o código no Quadro 11.

```
public bool Validar_Arquivo()
{
    if ( FindFile.Value != "")
    {
        string ext = Path.GetExtension(FindFile.PostedFile.FileName);
        //verifica a extensão
        if ( ext.ToUpper() != ".ASM")
        {
            lblmsgError.Text=" Somente são permitidos arquivos com extensão do tipo ASM";
            return false;
        }
        return true;
    }
    else
    {
        lblmsgError.Text=" Selecione um arquivo! ";
        return false;
    }
}
```

Quadro 11 – Validação dos nomes dos arquivos

Caso o arquivo seja válido o método Monta é chamado na seqüência que carrega através do método Upload o arquivo selecionado pelo usuário no diretório no servidor.

O código dos métodos podem ser vistos no Quadro 12.

```
private void Monta()
{
    try
    {
        if (Upload()){
            Executar_Processo(localexeproc);
            //carrega o arquivo .lst
            StreamReader reader = new StreamReader(localpath + onlyname + ".LST");
            TxtMsg.Value = reader.ReadToEnd();
            string str = Convert.ToString(TxtMsg.Value);
            int ind = str.IndexOf("0 ERRORS");
            //procurar se há erro no arquivo .lst
            reader.Close();
            if (ind > 0){
                Executar_Processo(localexeprocBin);
                TxtMsg.Value = "";
                TxtMsg.Value = "O código não apresentou erro" +"/n" +"Montado com sucesso";
                Salvar();
            }
        }
    }
    catch (Exception Ex){
        TxtMsg.Value = Excp.ExpIO(Ex);}
}
private bool Upload()
{
    try
    {
        //testa se o arquivo foi postado e se é maior que zero
        if ((FindFile.PostedFile != null) || (FindFile.PostedFile.ContentLength != 0))
        {
            filename = System.IO.Path.GetFileName(FindFile.PostedFile.FileName);
            //salva o arquivo
            FindFile.PostedFile.SaveAs(localpath + filename);
            //salva o nome do arquivo numa variavel de sessão
            Session["filename"] = filename;
            return true;
        }
        else
        {   //erro ao carregar arquivo
            TxtMsg.Value = "Erro ao carregar arquivo";
            return false;
        }

    }
    catch (Exception Ex)
    {
        TxtMsg.Value= Excp.ExpIO(Ex);
        return false;
    }
}
```

Quadro 12 – Salvar arquivo do usuário no servidor e montar código.

O método Monta também carrega outro método Executar_Processo, ele é responsável por criar uma chamada a um software independente: o ASM51, e a dois executáveis deste software. O primeiro é o asm51.exe que gera dois arquivos: um hexadecimal e outro com a extensão .lst. Este último informa ao usuário se o código redigido possui algum erro. O primeiro é utilizado por outro executável: o HexBin.exe que gera o arquivo binário.

Então o método Executar_Processo recebe primeiramente como parâmetro do método o caminho de onde se encontra o arquivo executável asm51.exe. Em seguida a classe interna do C# que manipula os processos, Process, é instanciada e são passados alguns parâmetros

fixos que a própria classe exige: caminho do executável e diretório de trabalho. Além do caminho e nome do arquivo a ser montado exigido pelo executável, antes de iniciar o processo.

O método Monta também analisa o arquivo com a extensão .lst e verifica se houve erro no código do usuário. Ocorrendo o erro, ele apresenta o arquivo ao usuário e informa a posição onde o erro foi detectado.

Caso contrário, ainda no método Monta o método Executar_Processo é chamado novamente e outro processo é disparado, desta vez para gerar o arquivo binário através do executável HexBin.exe do software ASM51. que exige parâmetros como: caminho e nome do arquivo hexadecimal, caminho e nome do arquivo binário a ser criado, e o tipo de microcontrolador utilizado, neste caso, compatível Intel.

Em seguida o método Salvar é chamado para que o usuário possa salvar o arquivo binário gerado.

O código dos métodos Executar_Processo e Salvar podem ser visualizados no Quadro 13.

```
private void Executar_Processo(string pathinf){
    //carrega a variavel filename com o nome do arquivo guardado em uma variavel de sessão
    filename = Session["filename"].ToString();
    //retira a extensão do arquivo
    onlyname = Path.GetFileNameWithoutExtension(localpath + filename);
    Process proc = new Process();
    //desabilita qualquer evento que ocorra ao iniciar o processo
    proc.EnableRaisingEvents = false;
    //deixa um saida padrão de dados que pode ser carregada em uma string
    proc.StartInfo.RedirectStandardOutput = true;
    //desabilita a execusão no shell
    proc.StartInfo.UseShellExecute =false;
    proc.StartInfo.FileName = pathinf;
    //informa o local onde está o .exe
    proc.StartInfo.WorkingDirectory = "C:\\ASM51\\";
    try
    {
        if (pathinf == localexeprocBin)
        {
            //carrega os argumentos para iniciar o processo
            proc.StartInfo.Arguments =" "+localpath + onlyname + ".hex" +
                                      " " +localpath + onlyname + ".bin" + " " + "i";
        }
        else
        {proc.StartInfo.Arguments =localpath + filename;}
    }
    catch (Exception Ex)
    {TxtMsg.Value = Excp.ExpIO(Ex);}
    //inicia processo
    proc.Start();
    //aguarda até que o processo seja encerrado
    proc.WaitForExit();
    //fecha processo
    proc.Close();
}
private void Salvar()
{
    //carrega arquivo para ser salvo
    FileStream SourceFile = new FileStream(localpath + onlyname +".bin",FileMode.Open);
    long FileSize;
    //informa o tamanho do arquivo
    FileSize = SourceFile.Length;
    //abre a opção para que o usário salve o arquivo
    Response.AddHeader("Content-Disposition", "attachment; filename=" +onlyname+".Bin");
    Response.AddHeader("Content-Length", FileSize.ToString());
    Response.ContentType = "application/octet-stream";
    SourceFile.Close();
    //determina o nome do arquivo a ser salvo
    Response.WriteFile(localpath + onlyname +".bin");
    Response.End();

}
```

Quadro 13 – Executar chamadas a programas externos e efetuar download

3.3.1.5 Executar código no microcontrolador

Na página principal, Figura 22, o usuário tendo posse do arquivo binário, o sistema possibilita selecionar o arquivo, e executa-lo no microcontrolador. Ao selecionar o arquivo, o usuário clica no botão **Carregar** e o método Validar_Arquivo é chamado para validar o

arquivo.

Se o arquivo for válido o método chamado na seqüência é o Upload, que carrega o arquivo para o servidor e chama o método Transferir_Programa, que inicialmente abre o *driver* de comunicação com a placa atravé de outro método chamando DriverOpen.

O método DriverOpen abre o *driver* de comunicação e transfere o código para o microcontrolador, através dos métodos da classe RapidIsa_cs. O código dos métodos Transferir_Programa e o DriverOpen podem ser visualizados nos Quadro 14 e 16 respectivamente.

```
private void Transferir_Programa()
{
    Atualiza();
    //carrega arquivo .BIN e efetua a leitura a partir do endereço 0x8000
    FileStream f = new FileStream(localpath + Session["filename"],FileMode.Open);
    f.Seek(0x8000,SeekOrigin.Begin);
    LblMSg.Text = "";
    //abrindo Driver (dll)
    driveropen();
    RapidIsa_Cs.RapidIsa.OpenRapidIsa(0);
    //detectando a placa do microcontrolador
    RapidIsa_Cs.RapidIsa.SetPortByte(hIsa,Address1,0);
    Thread.Sleep(200);
    LblMSg.Text = RapidIsa_Cs.RapidIsa.GetPortByte(hIsa,Address0).ToString();
    Thread.Sleep(time);
    RapidIsa_Cs.RapidIsa.SetPortByte(hIsa,Address1,0);
    Thread.Sleep(time);
    //informando o tamanho do programa
    int len = Convert.ToInt32(f.Length-0x8000);
    //dividindo o tamanho em 2 bytes
    byte ms = Convert.ToByte(len/256);
    byte ls = Convert.ToByte(len%256);
    //enviando o byte mais significativo
    RapidIsa_Cs.RapidIsa.SetPortByte(hIsa,Address0,ms);
    Thread.Sleep(time);
    //enviando o byte menos significativo
    RapidIsa_Cs.RapidIsa.SetPortByte(hIsa,Address0,ls);
    Thread.Sleep(time);
    //transferindo o programa
    while (f.Position < f.Length)
    {
        byte comando = Convert.ToByte(f.ReadByte());
        RapidIsa_Cs.RapidIsa.SetPortByte(hIsa,Address0,comando);
        Thread.Sleep(time);
    }
    //mandando um byte para informar final de programa
    RapidIsa_Cs.RapidIsa.SetPortByte(hIsa,Address0,0);
    Thread.Sleep(time);
    LblMSg.Text = "Programa Carregado com sucesso!";
    f.Close();
}
```

void RapidIsa.SetPortByte (uint hIsa, uint PortAddr, byte bValue)

Quadro 14 – Transferir o código binário para o microcontrolador

O código do método Transferir_Programa, foi baseado em um código redigido na

linguagem Pascal por Wisintainer. O Quadro 15 mostra este código.

```
uses crt;
begin
     clrscr;
     writeln('Detectando REXLAB...');
     port[$301]:=0;
     delay(200);
     if(port[$300]=128) then
       begin
          writeln('REXLAB detectado...');
          sound(900);
          delay(500);
          nosound;
       end
       else
       begin
          writeln('REXLAB nao detectado...');
          sound(300);
          delay(500);
          nosound;
          exit;
       end;
       port[$301]:=0;
       delay(200);
       {O PROGRAMA ABAIXO TEM 5 BYTES, VOU INFORMAR AO REXLAB}
       port[$300]:=0; {MSB}
       delay(300);
       port[$300]:=5; {LSB}
       delay(300);
       {TRANSFERINDO CODIGO DE MAQUINA}
       writeln('Transferindo Programa...');
       port[$300]:=$74; {74}
       delay(300);
       port[$300]:=222; {DECIMAL PARA ACUMULADOR}
       delay(300);
       port[$300]:=$02; {02}
       delay(300);
       port[$300]:=$80; {80}
       delay(300);
       port[$300]:=$02; {02}
       delay(300);
       {EXECUTE}
       writeln('Executando o Programa...');
       port[$300]:=0; {LIXO}
       delay(300);
       {CONSULTA ACUMULADOR}
       port[$302]:=0;
       delay(300);
       writeln('ACUMULADOR VALE ',PORT[$300]);
       DELAY(1000);
end.
```

Quadro 15 – Código exemplo em Pascal

```
//abre driver para comunicação com a placa
private void driveropen()
{
    RapidIsa_Cs.RapidIsa.HW_DEV_CONFIG Cfg;
    hIsa = RapidIsa_Cs.RapidIsa.OpenRapidIsa(DeviceInstance);
    DriverOpened = RapidIsa_Cs.RapidIsa.IsRapidIsaOpened(hIsa) != 0;
    if (DriverOpened)
    {
        Cfg = new RapidIsa_Cs.RapidIsa.HW_DEV_CONFIG();
        //inicializa o driver
        Cfg.Initialize();
        RapidIsa_Cs.RapidIsa.GetHardwareConfiguration(hIsa,ref Cfg);

    }
    else
    {
        LblMSg.Text = "Erro ao carregar driver de Comunicação";
    }
}
```

Quadro 16 – Abrir *driver* de comunicação com o hardware

Após a transferência do programa, o usuário poderá realizar consultas dos valores dos

registradores e memória através de *buttons* com o nome dos registradores. Esses buttons ativam o método Consulta_Registrador, que recebe como parâmetro um número que identifica cada registrador escolhido, através do evento *Click*.

O código para selecionar os registradores pode ser visto no Quadro 17.

```
//seleciona o registrador a ser consultado
private uint Consulta_Registrador(byte b)
{
    RapidIsa_Cs.RapidIsa.SetPortByte(hIsa,Address2,b);
    Thread.Sleep(300);
    return RapidIsa_Cs.RapidIsa.GetPortByte(hIsa,Address0);

}
```

Quadro 17 – Selecionar registradores

3.3.2 Operacionalidade da implementação

Esta seção descreve a funcionalidade do protótipo através de um estudo de caso, onde o cliente já efetuou o *login* e irá carregar o arquivo binário para ser executado.

O Quadro 18 exemplifica o código a ser executado no microcontrolador.

```
$MOD51
ORG 8000H
BEGIN:
        MOV A,#55
        MOV B,P1
SJMP BEGIN
END
```

Quadro 18 – Exemplo do código a ser executado

Pode-se ainda notar no código do Quadro 18, que o programa começa no endereço 8000H, pois a memória 6264 responsável por armazenar o programa, está mapeada para este endereço.

O código exemplificado move 55 para o registrador A e também efetua a leitura da porta P1 e coloca o valor lido no registrador B.

A Figura 23 mostra a página principal.

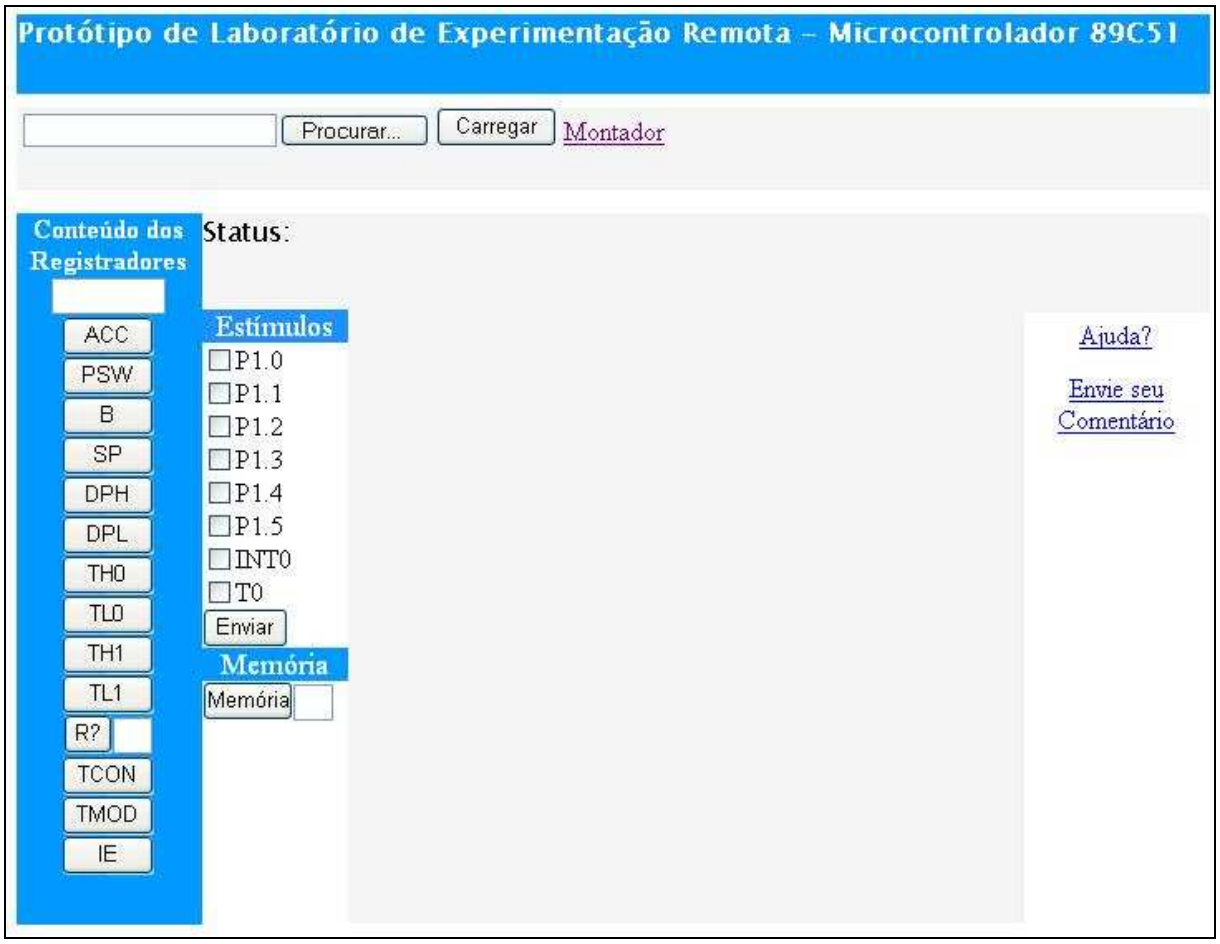


Figura 23 – Página principal

Informando o caminho ou através do botão Procurar o usuário localiza o arquivo binário a ser executado, conforme a Figura 24.

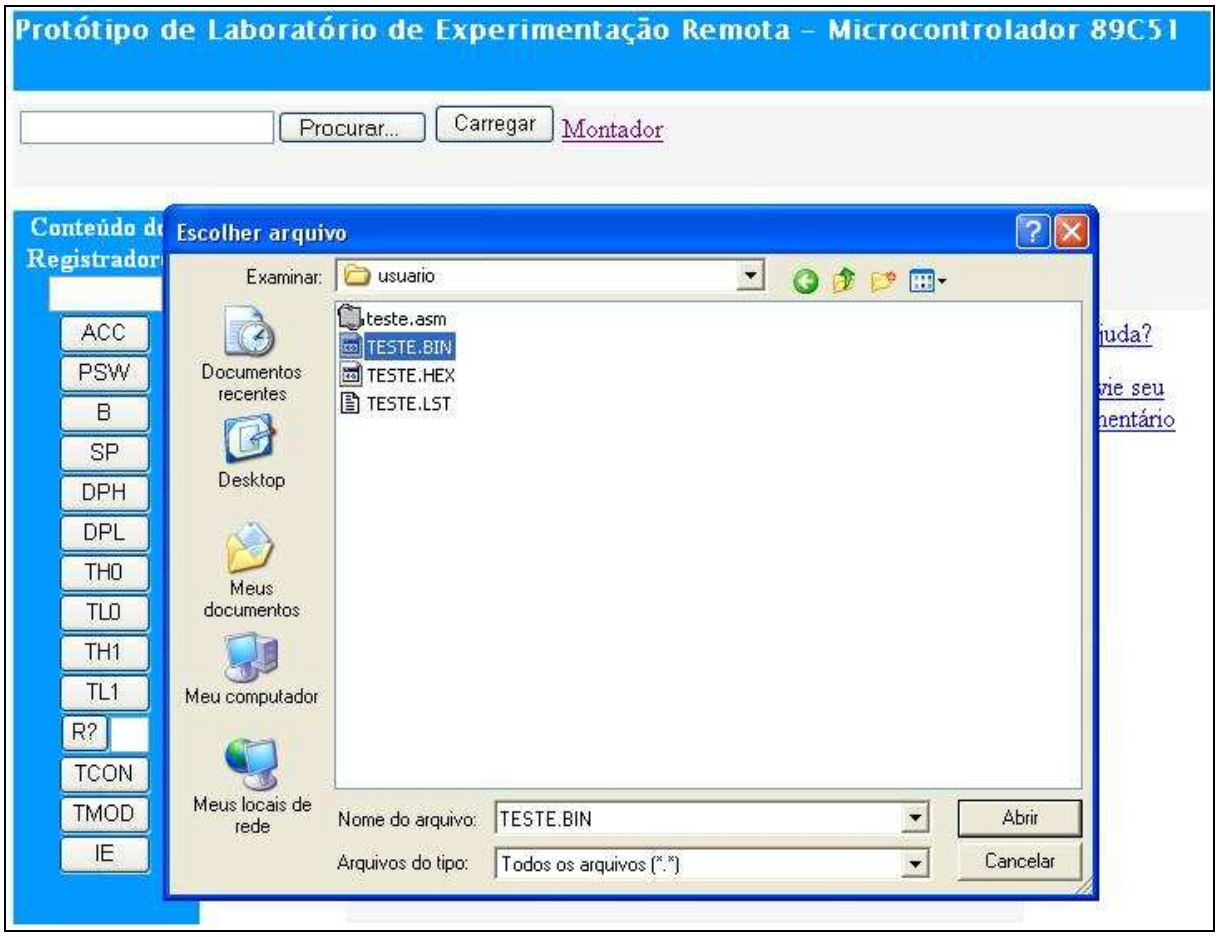


Figura 24 – Selecionando arquivo

Após selecionar o arquivo, o usuário clica no botão carregar, e uma mensagem avisando que o arquivo foi carregado com sucesso será exibida como na figura 25.

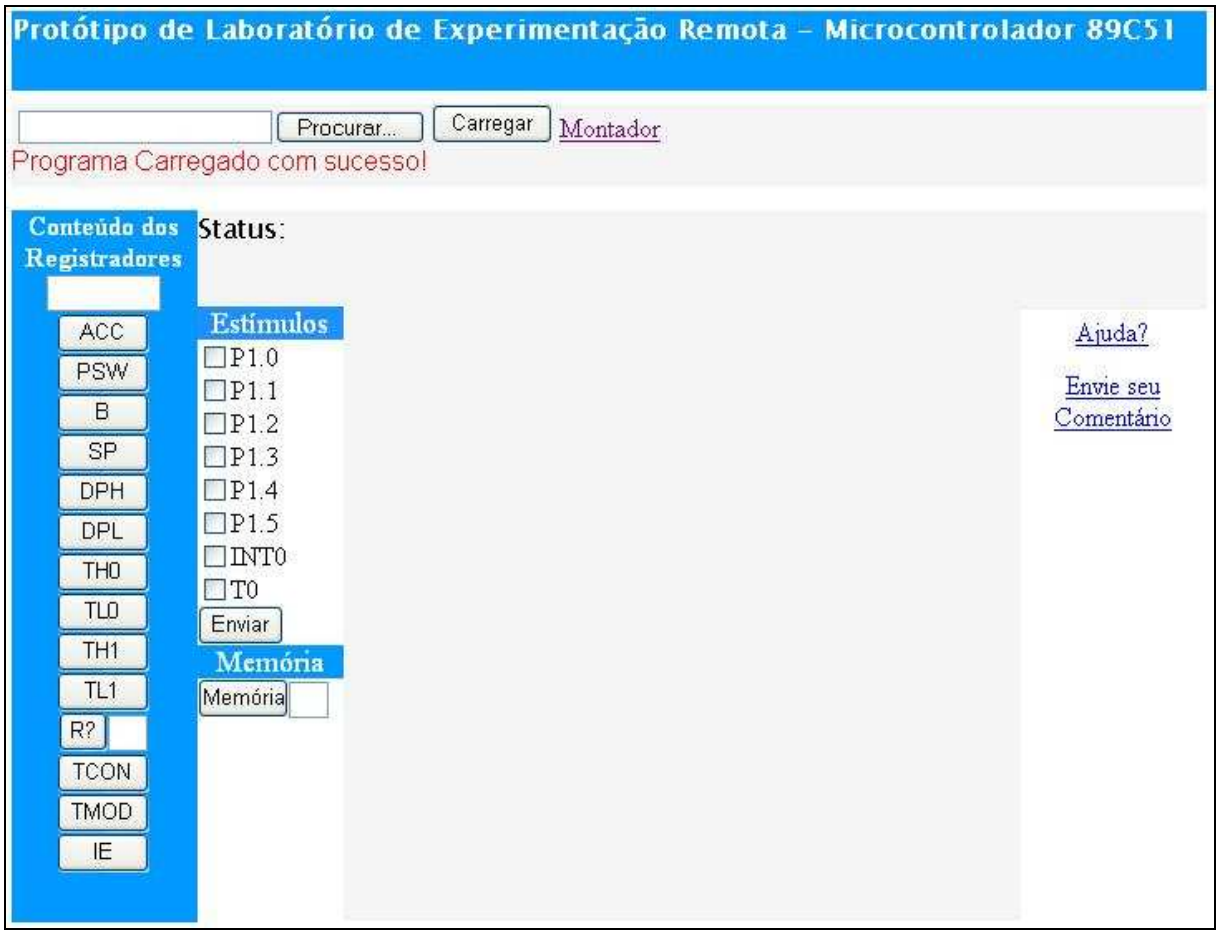


Figura 25 – Arquivo carregado

Na seqüência o usuário está apto para fazer a consulta a um registrador de sua escolha, através dos botões exibidos na página. Utilizando o exemplo do quadro 18 o valor do registrador A é mostrado no campo *textbox* logo acima da coluna com os botões dos registradores, como mostra a Figura 26.

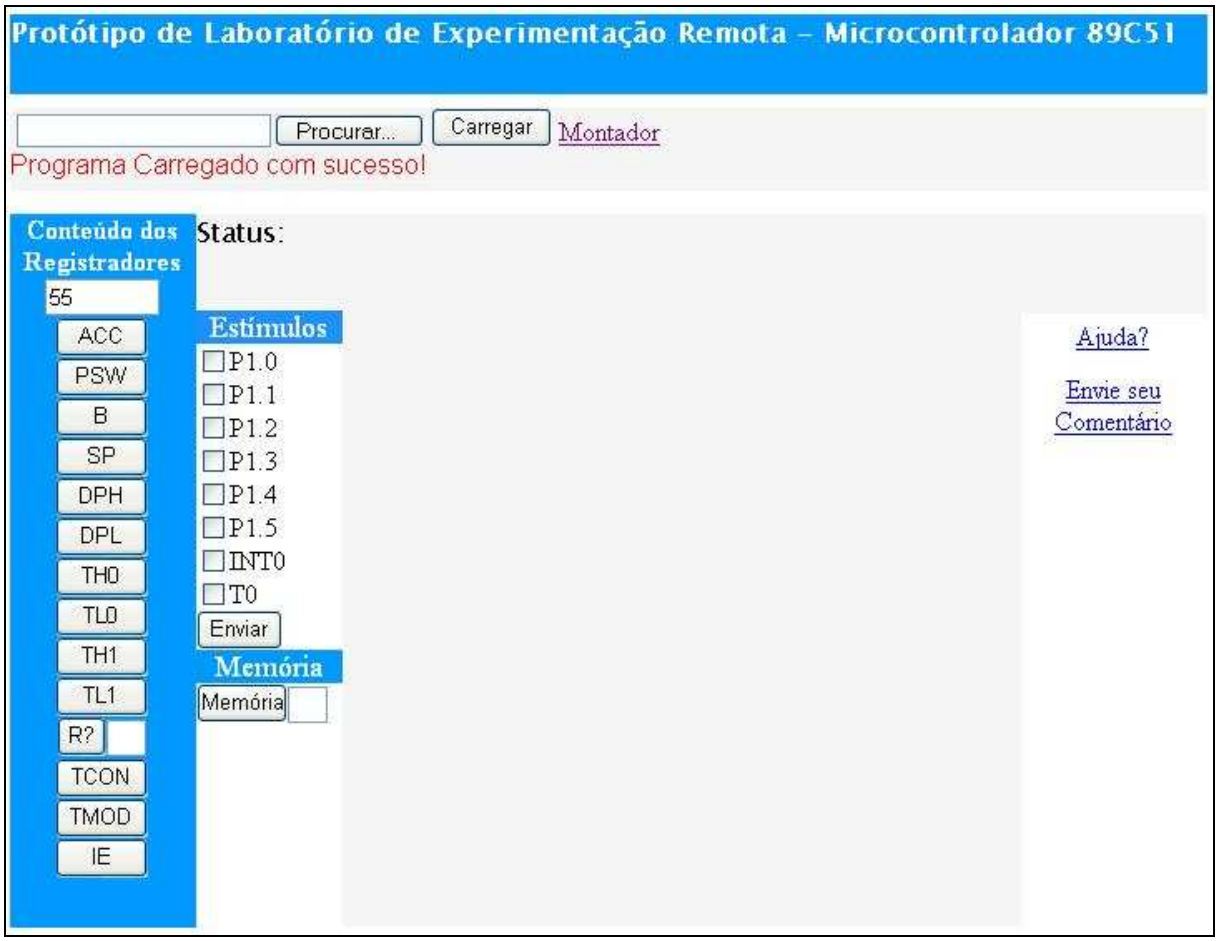


Figura 26 – Visualizando a consulta a um registrador

Ainda seguindo o exemplo do Quadro18 pode-se selecionar um estímulo a ser enviado e visualizar o resultado, selecionando o registrador B como mostra a Figura 27.

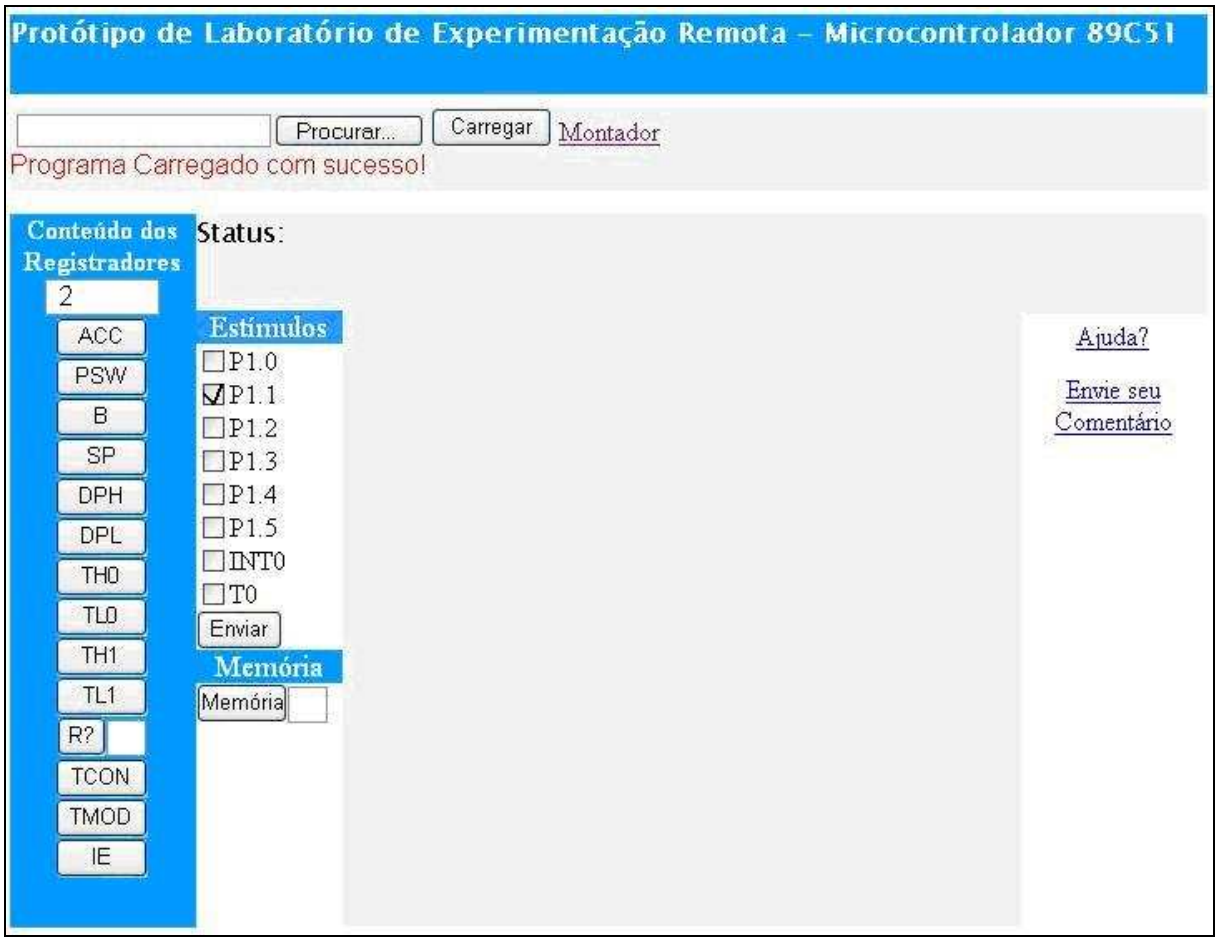


Figura 27 – Enviando um estímulo e visualizando o seu resultado em um registrador

3.4 RESULTADOS E DISCUSSÃO

Com a finalização deste protótipo pode-se verificar que versão inicial prevista para o desenvolvimento deste protótipo sofreu uma modificação.

A princípio a idéia inicial era disponibilizar um editor de texto onde o usuário poderia redigir seu código e salvar seu arquivo, sem precisar sair da aplicação e utilizar outro editor. Porém com o desenvolvimento deste projeto foram surgindo dificuldades que foram consumindo o tempo disponível para o desenvolvimento de tal função.

A maior dificuldade encontrada foi na utilização do *driver* e sua biblioteca de comunicação com o hardware pelo barramento ISA, por não ser um tipo de acesso comum e por este tipo de barramento ter sido descontinuado.

No início foram pesquisadas e encontradas três bibliotecas, duas gratuitas e uma

proprietária, que realizavam comunicação direta com *hardware*. O material encontrado a respeito dessas bibliotecas exemplificava o uso no acesso à porta paralela do PC. Embora não fora encontrado nenhum exemplo de utilização nem documentação que exemplificasse seu uso com o bararmento ISA, foram testadas uma a uma.

A primeira a ser testada foi a biblioteca IO.dll (HIDEOUT, 2006), uma biblioteca gratuita, mas que não apresentou comunicação com a placa, resultados aleatórios e dispersos eram obtidos.

A INPOUT32.dll (LOGIX4U, 2006), foi a segunda biblioteca gratuita testada, duas versões disponíveis foram testadas até a realização deste trabalho. Nos testes efetuados com esta biblioteca, obteve-se êxito parcial na comunicação com o *hardware*. O problema estava no funcionamento inconstante do envio e recebimento dos bytes pelo barramento ISA.

A terceira foi a TVicPort.dll (ENTECH, 2006), proprietária com trinta dias para avaliação. No *site* do fabricante não havia documentação sobre o barramento ISA. Nos testes realizados o funcionamento também apresentou-se muito esparso.

No intuito de vencer o desafio de efetuar a comunicação com o *hardware*, foram realizadas várias buscas na internet até que se encontrou um fabricante que tinha no seu *driver*, suporte ao barramento ISA.

O andamento do trabalho então pode ser avançado e os principais resultados esperados com a finalização deste trabalho foram confirmados. As consultas aos registradores e memória podem ser realizadas, os estímulos podem ser enviados, e o código pode ser analisado e montado, sem que o usuário precise ter em seu PC um montador instalado.

Com o término deste trabalho também pode se notar que houve uma continuidade no trabalho desenvolvido por Wisintainer (WISINTAINER, 1999).

Em relação aos trabalhos correlatos apresentados e o protótipo desenvolvido neste trabalho, destacam-se as seguintes características:

a) o trabalho de Wisintainer (1999) limita-se a uma plataforma, a Windows, e não disponibiliza a opção de motagem remota, ou seja, o programa na linguagem Assembly deve ser montado na própria máquina antes ser mandado ao servidor. O protótipo proposto disponibilizará o montador *assembly* no servidor e já que seu funcionamento é via Internet, independente da plataforma utilizada;
b) Gustavsson (1999) retrata a montagem de circuitos eletrônicos remotamente que, como no caso acima, restringe-se a plataforma Windows.

4 CONCLUSÕES

O protótipo de laboratório desenvolvido é uma alternativa aos desenvolvedores de projetos que utilizam o microcontrolador 89C51, pois podem realizar experimentos reais sem a necessidade de ter a estrutura que envolve tais experiências, basta ter um computador conectado a internet.

Esses experimentos podem ser feitos independentemente do sistema operacional, como Windows e Unix, e de qualquer lugar, já que o funcionamento do protótipo é baseado na Internet.

Os resultados obtidos são sempre reais, pois estão diretamente relacionados com componentes do mundo real, o que diferencia de aplicativos simuladores, que nem sempre informam resultados reais.

O protótipo também permite, testar e montar o código a ser transferido para o microcontrolador na própria aplicação, sem que o usuário tenha que ter tal função em seu PC.

Segundo (2000, p.13) relata que um laboratório de experimentação remota real com o microcontrolador 89C51 diminui os custos com equipamentos, já que o um único *hardware* pode estar ao alcance de muitos, desde que se tenha acesso a um computador ligado na internet.

A principal limitação é quanto ao acesso ao protótipo, que é restrito há um usuário, e enquanto este não desocupar o aplicativo outro não pode acessar, o que pode gerar um tempo de espera considerável na tentativa de usufruir o protótipo.

O protótipo não possui um editor para permitir a redação e edição do código, o usuário deve possuir um editor instalado na sua máquina para que o faça.

4.1 EXTENSÕES

Como extensões para este trabalho sugere-se:

a) mudar a arquitetura da placa para um barramento mais atual como o barramento PCI do computador;
b) aumentar o número de *hardware* gerenciado e consequentemente um número maior de usuários simultâneos.

REFERÊNCIAS BIBLIOGRÁFICAS

ALLDATASHEET. **DataSheet**. [S.l.], 2006. Disponível em: <http://www.alldatasheet.com/>. Acesso em: 09 set. 2006.

ALVES, J. M. **Laboratório de experimentação remota**. Florianópolis, 1999. Disponível em: <http://www.inf.furb.br/~jbosco/lexrem2p.htm>. Acesso em: 15 set. 2005.

BOCHENSKI B. **Implementando sistemas cliente / servidor de qualidade.** São Paulo: Makron Books, 1995.

CONALLEN J. **Desenvolvendo aplicações web com UML.** Rio de Janeiro: Campus, 2003.

ENTECH. **Toolkits**. [S.l.], 2006. Disponível em: <http://www.entechtaiwan.com/dev/index.shtm>. Acesso em: 12 set. 2006.

FETTER, W. **Barramentos ISA, EISA, VLB, PCI e AGP**. Porto Alegre, 2001**.** Disponível em: <http://www.ece.ufrgs.br/~fetter/proc/busses.pdf 2001>. Acesso em: 17 set. 2005.

GIMENEZ, S. P. **Microcontroladores 8051**. São Paulo: Pearson Education do Brasil, 2002.

GUSTAVSSON, I. et al. **A remote eletronics laboratory for phisical experiments using virtual breadboards**. Suécia, 1999. Disponível em: <http://zone.ni.com/devzone/conceptd.nsf/webmain/2C382C0EAC5FA5868625704B00746B16>. Acesso em: 19 set. 2005.

HIDEOUT, G. **IO.DLL.** [S.l.], 2006. Disponivel em: <http://www.geekhideout.com/iodll.shtml>. Acessado em: 12 set. 2006.

ICEA. **Barramento ISA.** [S.l.], 2005. Disponível em: <http://www.icea.gov.br/ead/anexo/24101.htm >. Acesso em: 10 set. 2006.

LOGIX4U. **INPOUT32.DLL.** [S.l.], 2006. Disponível em: <http://www.logix4u.net/inpout32.htm>. Acesso em: 12 set. 2006.

MESSIAS, A. R **Porta paralela**. Anádia, 2006. Disponível em: <http://www.rogercom.com/pparalela/ExemploImpOut32.htm>. Acesso em: 10 set. 2006.

SEGUNDO, F. R. **Laboratórios remotos no ensino**: estudo de caso do laboratório de experimentação remota para microcontroladores 8051. 2000. 91 f. Dissertação (Mestrado em Engenharia de Produção) – Programa Pós-graduação em Engenharia de Produção, Universidade Federal de Santa Catarina, Florianópolis.

WIKIPÉDIA. **Barramento ISA**. [2006]. Disponível em: < http://pt.wikipedia.org/wiki/ISA>. Acessado em 10 set. 2006.

WISINTAINER, M. A. **RExLab**: laboratório de experimentação remota com o microcontrolador 8051. 1999. 137 f. Dissertação (Mestrado em Ciências da Computação) – Programa de Pós-graduação em Ciências da Computação, Universidade Federal de Santa Catarina, Florianópolis.