UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO DE INTERFACE PARA TROCA DE INFORMAÇÕES ENTRE DOIS MICROCOMPUTADORES PADRÃO PC

VALTER PARASKI

BLUMENAU
2003

2003/2-40

VALTER PARASKI

PROTÓTIPO DE INTERFACE PARA TROCA DE INFORMAÇÕES ENTRE DOIS MICROCOMPUTADORES PADRÃO PC

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 Ciência da Computação — Bacharelado.

Prof. Antônio Carlos Tavares - Orientador

BLUMENAU
2003

2003/2-40

PROTÓTIPO DE INTERFACE PARA TROCA DE INFORMAÇÕES ENTRE DOIS MICROCOMPUTADORES PADRÃO PC

Por

VALTER PARASKI

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

Presidente:	Prof. Antônio Carlos Tavares – Orientador, FURB
Membro:	Prof. Nome do professor, FURB
Membro:	Prof. Nome do professor, FURB

Blumenau, 19 de novembro de 2003

Dedico este trabalho a meus pais por tudo que têm feito por mim, à minha esposa pelo seu amor e por limitar meu caos, aos meus filhos a quem devo meu renascimento.

Ninguém é tão grande que não possa aprender,
nem tão pequeno que não possa ensinar.

Autor: Desconhecido

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

À minha família, que mesmo longe, sempre esteve presente.

Ao meu orientador, Antônio Carlos Tavares, por ter acreditado na conclusão deste trabalho.

RESUMO

Este trabalho apresenta o desenvolvimento de um protótipo de interface para trocar informações entre dois microcomputadores padrão PC utilizando o barramento ISA e o circuito integrado 82C55A. Mostra o estudo de todos os sinais de entrada e saída do barramento e do circuito integrado e distinguindo os que foram utilizados na confecção do protótipo. Informa também a forma de configurar o circuito integrado 82C55A adequadamente e, no final acrescenta um software para transmissão de arquivos entre dois microcomputadores demonstrando o funcionamento do conjunto.

Palavras chaves: Barramento ISA; Interface de Hardware; 82C55A.

ABSTRACT

This work presents the development of an interface prototype to change information between two microcomputers PC compatible using the ISA bus and the integrated circuit 82C55A. It shows the study of all the input and output bus signs and the integrated circuit, also distinguish those that was used in the prototype. It also informs the way to configuring the integrated circuit 82C55A appropriately and, in the end increases a software for transmission of files between two microcomputers demonstrating the functionality of the group.

Key-Words: ISA Bus; Hardware Interfacing; 82C55A.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

ABNT – Associação Brasileira de Normas Técnicas

SBC – Sociedade Brasileira de Computação

DSC – Departamento de Sistemas e Computação

BCC – Curso de Ciências da Computação – Bacharelado

ISA – Industry Standard Architecture

MCA – Microchannel Architecture

EISA – Extended Industry Standard Architecture

VLB – VESA Local Bus

PCI – Peripheral Component Interconnect

AGP – Accelerated Graphics Port

USB – Universal Serial Bus

IrDa – Infrared Developers Association

LISTA DE SÍMBOLOS

@ - arroba

% - por cento

- sustenido

p - pi

$ - cifrão

SUMÁRIO

1 INTRODUÇÃO

A computação moderna evoluiu junto com a humanidade a patamares jamais alcançados por outra tecnologia. O primeiro dispositivo de cálculo que o homem inventou foi o ábaco, usado desde 2.000 a.C., e ainda é encontrado no Japão e em outros países. O computador nada mais é do que uma máquina capaz de efetuar cálculos com um grupo de números, recuperar números já computados e, ainda, realizar novas operações com estes valores. Devido a essa flexibilidade, existe um imenso campo para estudos e desenvolvimentos de novos dispositivos de hardware e software.

Nos computadores atuais, uma necessidade crescente de elevar seu poder de processamento leva o homem a pesquisar novas tecnologias que utilizam todo o potencial das máquinas novas e também das que se encontram atualmente instaladas. Neste quesito, podem ser implementados dispositivos para processamento em paralelo utilizando as portas de entrada e saída com a finalidade de compartilhar informações com outras máquinas. Uma delas é através do barramento, que permite uma performance bem superior que as mais tradicionais, como as portas paralela e serial.

O termo barramento (*bus*) refere-se aos percursos entre os componentes de um computador. Há dois barramentos principais em um computador: o barramento de dados e o barramento de endereços. O mais conhecido é o barramento de dados, portanto, quando se referencia apenas "barramento", em geral é uma abreviação ao barramento de dados.

Para que uma simples placa de vídeo ou disco possam ser utilizados em qualquer microcomputador, independente do processador instalado, utiliza-se diversos modelos de barramentos de expansão. Dentre eles pode-se destacar: *Industry Standard Architecture* (ISA); o *Microchannel Architecture* (MCA); o *Extended Industry Standard Architecture* (EISA); o VESA *Local Bus* (VL); o *Peripheral Component Interconnect* (PCI); o *Accelerated Graphics Port* (AGP); o *Universal Serial Bus* (USB); *Firewire* (também chamado IEEE 1394) e o *Infrared Developers Association* (IrDa) (TORRES, 1998).

Quando a IBM apresentou o PC-AT (*Advanced Technology*), a melhoria mais significativa foi um barramento aperfeiçoado, que atendia as potencialidades de um novo processador, o 80286. Esse barramento foi denominado ISA, sendo que possui uma taxa de transferência de 8 MHz em 16 bits (TORRES, 1998).

Mais tarde, a IBM lançou o barramento MCA, uma nova arquitetura de barramento para tirar proveito das CPUs de 32 bits. Essa nova arquitetura de barramento era significativamente mais rápida que o barramento ISA. Entretanto, as placas de expansão ISA não eram compatíveis com o barramento MCA, tendo a IBM quebrado a tendência de compatibilidade ascendente (PELISSON, 2000).

Em resposta ao MCA, um grupo de fabricantes de hardware uniram-se para desenvolver um barramento de 32 bits alternativo, mas que ainda aceitasse placas de expansão ISA. Esse barramento tornou-se conhecido como EISA, e sua performance ficou entre o barramento ISA e o MCA (PELISSON, 2000).

No final de 1992, criou-se a VESA "A" *Local Bus* (*Video Electronics Standards Association*) para desviar o tráfego mais intenso, como o vídeo, com um barramento local conectado diretamente ao barramento da CPU. Este barramento foi criado tendo em vista aumentar a velocidade de transferência de dados entre a placa de CPU e a placa SVGA, mas outras placas de expansão também poderiam utilizá-lo. Desta forma, o barramento de dados não teria problemas com um tráfego tolerável entre os dispositivos periféricos. Não necessitava de chips especiais como era o caso do EISA, era uma arquitetura aberta ao contrário do MCA, e tratava-se de um padrão industrial, uma grande vantagem sobre os barramentos proprietários. É fisicamente representada por um conector especial de expansão (PELISSON, 2000).

Com a evolução da velocidade dos processadores, o padrão VESA "A" foi tornando-se cada vez mais lento. Para resolver o problema de latência do barramento inerente a essa situação, foi criado um *virtual local bus* e conectado ao barramento da CPU via *buffer*. Essa solução ficou conhecida como VESA "B". Esses *buffers* permitem que os sinais sejam armazenados por breves períodos, enquanto o barramento estiver ocupado. Mesmo assim, o que acontece quando mais de um dispositivo necessita utilizar o barramento da CPU ao mesmo tempo? Acontece uma arbitragem e apenas um dispositivo utiliza o barramento enquanto os demais aguardam a sua liberação. Com isso, pode ocorrer uma latência e o sistema possivelmente opera de forma ineficiente. Isto fez surgir, então, o barramento PCI (PELISSON, 2000).

A arquitetura PCI é semelhante à VESA quanto à conexão ao barramento local da CPU, mas é muito mais completa ao utilizar uma ponte PCI-HOST, um dispositivo de *cache*

que provê uma única interface entre a CPU, memória e o PCI local bus. A arquitetura PCI possibilita que a CPU continue buscando informações do *cache* enquanto o controlador deste *cache* permite que um dispositivo de expansão acesse a memória do sistema, ou seja, operações concorrentes no mesmo barramento (PELISSON, 2000).

Outra grande vantagem do barramento PCI é que até 256 dispositivos podem ser conectados a um único PCI local bus, e 256 barramentos PCI podem existir em um único sistema (PELISSON, 2000).

O barramento AGP é uma nova porta ou *slot* introduzida no *chipset* LX do Pentium II (na placa mãe), para prover uma conexão de grande largura de banda entre a memória do sistema e o subsistema gráfico. Com isso, removeu-se o acelerador gráfico do gargalo do barramento PCI, liberando este e conseqüentemente aumentando a largura de banda para operações de entrada e saída e tráfego da rede (PELISSON, 2000).

O barramento USB utiliza basicamente duas taxas de transferência: 12Mbps é usada por periféricos que exigem mais velocidade (como câmeras digital, impressoras, *scanner*, etc.) e 1,5Mbps, para periféricos mais lentos (como teclados, *joysticks*, mouse, etc). No barramento USB é possível conectar até 127 dispositivos diferentes no mesmo conector da placa mãe (TORRES, 1998).

O barramento *Firewire* é uma idéia muito parecida com o USB, com a diferença de estar focado em substituir dispositivos de alto desempenho como o SCSI. Atualmente sua taxa de transferência é de 200 Mbps, podendo atingir 400 Mbps em sua segunda versão. É possível conectar até 63 periféricos e seu cabo pode alcançar uma distância de até 4,5 metros (OGREN, 2001).

O barramento IrDA é um barramento sem fios: a comunicação é feita através de luz infravermelha. Existem dois padrões: o IrDA 1.0, para comunicação de até 115.200 bps e IrDA 1.1, para conexão até 4.194.304 bps ou 4 Mbps (TORRES, 1998).

Considerando estas informações, esta proposta tem por objetivo aperfeiçoar o trabalho elaborado por Santos (2002), que viabiliza a comunicação entre dois computadores através do barramento de dados de um microcomputador padrão PC. Com a implementação de novos recursos (interrupções, largura de banda, etc.) na placa de expansão, será possível alcançar performance superior às conquistadas por Santos (2002). Para avaliação de seu desempenho,

será implementado um software para monitorar a entrada e a saída dos dados entre os dois microcomputadores. Algumas características seriam a transferência de dados entre duas máquinas, à curta distância.

1.1 OBJETIVOS DO TRABALHO

O objetivo desta proposta é o desenvolvimento de duas placas controladoras para troca de informações entre dois computadores, baseado no TCC elaborado por Santos (2002).

Os objetivos específicos são:

a) utilizar o barramento padrão ISA;

b) transferência de dados em 16 bits, utilizando circuitos integrados 82C55A;

c) funções de interrupções;

d) desenvolvimento de um software para tornar possível a comunicação entre dois computadores.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está organizado da seguinte maneira:

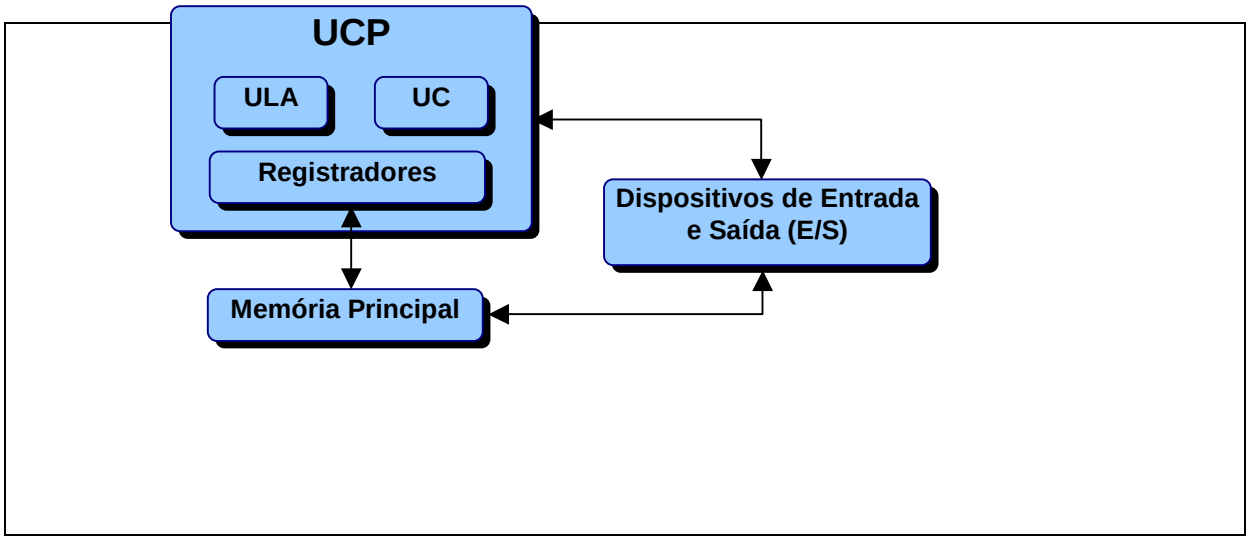
- No primeiro capítulo, é apresentada uma visão geral deste trabalho, seus objetivos, sua relevância e a sua estrutura.
- O segundo capítulo apresenta uma fundamentação e apresentação sobre Arquitetura de Computadores bem como um histórico de sua evolução e suas famílias com as principais características que as diferem umas das outras. Apresenta também uma visão sobre o barramento ISA, sua arquitetura e formas de interfaceamento e sobre as interrupções do microcomputador.
- O terceiro capítulo trata do desenvolvimento, especificação e implementação da parte de hardware e software que compõem este protótipo, e uma análise dos resultados obtidos no protótipo.
- E finalmente, no quarto capítulo encontram-se as conclusões e sugestões de continuidade do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

A arquitetura básica de um computador moderno segue ainda de forma geral os conceitos estabelecidos pelo Professor da Universidade de Princeton, John Von Neumann (1903-1957), que desde a sua criação até hoje muita coisa evoluiu no projeto inicial, com algumas adaptações chegou-se ao microcomputador de hoje.

As UCPs tornaram-se mais rápidas, a capacidade de memória aumentou, as redes transmitem uma grande quantidade de dados em curto espaço de tempo, etc. Todo esse progresso é acompanhado por uma diminuição gradual de custo, o que torna a tecnologia computacional mais acessível e aumenta o leque de aplicações que podem ser computacionalmente resolvidas de maneira eficiente (Turcotte, 1993).

Todos os componentes de um computador são agrupados basicamente em três subsistemas básicos: unidade central de processamento (UCP), memória principal e dispositivos de entrada e saída. Esses componentes podem ser observados na Figura 1.



Fonte: Turcotte (1993, p.)

Figura 1: Unidades funcionais de um computador

2.1 A UNIDADE CENTRAL DE PROCESSAMENTO (UCP)

A UCP tem como função principal unificar todos os componentes e controlar as funções realizadas por cada unidade funcional, ou seja, é o “cérebro” do computador.

Ela também é responsável pela execução de todos os programas do sistema, que obrigatoriamente deverão estar armazenados na memória (Machado, 1992).

A UCP é composta de várias partes distintas: Unidade de Controle (UC), que coordena a execução das instruções através da emissão de pulsos elétricos e a Unidade Lógica e Aritmética (ULA), que realiza operações aritméticas (adição e subtração, por exemplo) e operações booleanas (AND, OR) necessárias para realizar as instruções. A UCP contém pequenas memórias de alta velocidade usadas para armazenamento temporário de informações. Estas memórias consistem de registradores, cada um com uma determinada função. O registrador mais importante é o *program counter* (PC) ou Contador de Instruções (CI), o qual aponta a próxima instrução a ser executada. Outro registrador importante é o *instruction register* (IR) ou Registrador de Instrução (RI), o qual armazena a instrução que está sendo executada. A maioria dos microcomputadores tem esses e muitos outros registradores (Tanenbaum, 1984).

Dentro de uma UCP existe uma memória interna de alta velocidade, que funciona como um *cache* interno e acelera a movimentação dos dados. Essa *cache* é uma memória extremamente rápida, pois trabalha na velocidade do UCP.

A especificação da velocidade de processamento de uma UCP é determinada pelo número de instruções que o UCP executa por unidade de tempo, normalmente o segundo. Como exemplo tem-se: MIPS (milhões de instruções por segundo) e MFLOPS/GFLOPS (milhões/bilhões de instruções de ponto flutuante por segundo).

Um número bastante elevado de UCPs vem se tornando disponível ao longo dos últimos anos. Dentre os processadores que se destacam nos computadores pessoais (PC - *Personal Computer*), os processadores Intel da linha 80x86 são os que se tornaram mais populares. A Tabela 1 ilustra o inicio da evolução desses processadores, com sua data de lançamento, quantidade de instruções por segundo, número de transistores e o tamanho do barramento interno.

Tabela 1: Evolução dos processadores da linha Intel

Nome	Data de Lançamento	Freqüência do Clock	Largura Barramento	Nº Transistores	Memória Endereçável	Memória Virtual	Descrição Sumária
4004	15/11/71	108 KHz	4 bits	2.300 (10 microns)	640 bytes	-	1º chip microprocessador; manipulação aritmética
8008	01/04/72	108 MHz	8 bits	3.500	16 kbytes	-	Manipulação de dados e caracteres
8080	01/04/74	2 MHz	8 bits	6.000 (6 microns)	64 kbytes	-	10 vezes o desempenho do 8008
8086	08/06/78	5 a 10 MHz	16 bits	29.000 (3 microns)	1 Mbyte	-	10 vezes o desempenho do 8080
8088	01/06/79	5 MHz e 8 MHz	16 bits int / 8 bits ext.	29.000 (3 microns)	1 Mbyte	-	Idêntico ao 8086, mas com barramento externo de 8 bits
80286	01/02/82	8 a 12 MHz	16 bits	134.000 (1,5 microns)	1 Gbyte	-	De 3 a 6 vezes o desempenho do 8086

386 DX	17/10/85	16 a 33 MHz	32 bits	275.000 (1 micron)	4 Gbytes	64 Tbytes	1º chip x86 a manipular dados em 32 bits
386 SX	16/06/88	16 e 20 MHz	32 bits int./ 16 bits ext.	275.000 (1 micron)	4 Gbytes	64 Tbytes	Idêntico ao DX, exceto barramento externo de 16 bits
486 DX	10/04/89	25 a 50 MHz	32 bits	1.200.000 (1 micron)	4 Gbytes	64 Tbytes	*Cache* nível 1 e co-processador Aritmético embutidos no chip
486 SX	22/04/91	16 a 33 MHz	32 bits	1.185.000 (0,8 micron)	4 Gbytes	64 Tbytes	Idêntico ao DX porém sem co-processador aritmético
Pentium	22/03/93	60 a 266 MHz	32 bits	3.100.000 (0,8 micron)	4 Gbytes	64 Tbytes	Arquitetura super-escalar, 5 vezes o desempenho do 486 DX
Pentium Pro	27/03/95	150 a 266 MHz	32 bits	5.500.000 (0,32micron)	4 Gbytes	64 Tbytes	Arquitetura de Execução dinâmica

Fonte: PUC/Rio (2002)

2.2 MEMÓRIA

A memória é a parte do computador onde os programas e/ou dados são armazenados. É na memória que o processador lê ou escreve informações. O armazenamento pode ser feito de duas maneiras:

- armazenamento primário ou memória principal;
- armazenamento secundário.

A Memória tem por finalidade armazenar toda a informação que é manipulada pelo computador - programas e dados. Para que um programa possa ser manipulado pela máquina, ele primeiro precisa estar armazenado na memória principal.

2.2.1 Tecnologias

As primeiras tecnologias utilizadas em memórias foram as memórias de núcleos magnéticos, hoje apenas uma curiosidade. As memórias modernas são compostas por circuitos semicondutores, com novas tecnologias sendo criadas a cada ano permitindo que grandes quantidades de células de memória sejam encapsuladas em pequenas pastilhas.

2.2.2 Hierarquia de Memória

A memória principal não é o único dispositivo de armazenamento de um computador. Em função de características como tempo de acesso, capacidade de armazenamento, custo, etc., pode ser visto na Tabela 2 uma hierarquia de dispositivos de armazenamento em computadores.

Tabela 2: Hierarquia dos Dispositivos de Armazenamento

Tipo	**Capacidade**	**Velocidade**	**Custo**	**Localização**	**Volatilidade**
Registrador	Bytes	muito alta	muito alto	UCP	Volátil
Memória *Cache*	Kbytes	alta	alto	UCP/placa	Volátil
Memória Principal	Mbytes	Média	médio	Placa	Volátil
Memória Auxiliar	Gbytes	Baixa	baixo	Externa	Não Volátil

Fonte: PUC/Rio (2002)

A UCP vê nesta ordem e acessa primeiro a que está mais próxima. Subindo na hierarquia, quanto mais próximo da UCP, maior velocidade, maior custo, porém menor capacidade de armazenamento.

2.2.3 Registradores

Registradores são dispositivos de armazenamento temporário, localizados na UCP, extremamente rápidos, com capacidade para apenas um dado (uma palavra). Devido a sua tecnologia de construção e por estar localizado como parte da própria pastilha ("chip") da UCP é muito caro. O conceito de registrador surgiu da necessidade da UCP de armazenar temporariamente dados intermediários durante um processamento. Por exemplo, quando um dado resultado de operação precisa ser armazenado até que o resultado de uma busca da memória esteja disponível para com ele realizar uma nova operação.

Máquinas RISC são geralmente construídas com um grande conjunto de registradores, de forma a trazer os dados para o mais próximo possível da UCP, de forma a que o programa opere sempre sobre dados que estão em registradores. Registradores são voláteis, isto é, dependem de estar energizados para manter armazenado seu conteúdo.

2.2.4 Memória *Cache*

Com o desenvolvimento da tecnologia de construção da UCP, as velocidades foram ficando muito mais altas que as das memórias, que não tiveram a mesma evolução. Desta forma, os tempos de acesso às memórias foram ficando insatisfatórios e a UCP ao buscar um dado na memória precisa ficar esperando muitos ciclos até que a memória retorne o dado buscado ("*wait states*"), configurando um gargalo ("*bottleneck*") ao desempenho do sistema. Por esse motivo, desenvolveram-se outras arquiteturas de memória privilegiando a velocidade de acesso. A arquitetura da memória *cache* é muito diferente da arquitetura da memória principal e o acesso a ela é muitas vezes mais rápido (por exemplo: 5 ns contra 70 ns).

No entanto, o custo de fabricação da memória *cache* é muito maior que o da memória principal. Desta forma, não é econômico construir um computador somente com tecnologia de memória *cache*. Criou-se então um artifício, incorporando-se ao computador uma pequena porção de memória *cache*, localizada entre a UCP e a memória principal, e que funciona como um espelho de parte da memória principal.

Desenvolveram-se ainda algoritmos que fazem com que, a cada momento, a memória *cache* armazene a porção de código ou dados (por exemplo, uma sub-rotina) que estão sendo usados pelas UCP. Esta transferência (MP <--> *Cache*) é feita pelo hardware: ela independe do software, que ignora se existe ou não memória *cache*, portanto ignora essa transferência; nem o programador nem o sistema operacional têm que se preocupar com ela.

A memória *cache* opera em função de um princípio estatístico comprovado: em geral, os programas tendem a referenciar várias vezes pequenos trechos de programas, como *loops*, sub-rotinas e funções. Ela somente tem sentido porque programas executados linearmente, seqüencialmente, são raros. Desta forma, algoritmos (chamados algoritmos de *cache*) podem controlar qual parte do código ficará copiado na *cache*, a cada momento.

Quando a memória principal busca um determinado trecho de código e o encontra na *cache*, dá-se um "*cache hit*", enquanto se o dado não estiver presente na *cache* será necessário requisitar o mesmo à memória principal, acarretando atraso no processamento e dá-se um "*cache fault*".

O índice de *cache hit* ou taxa de acerto da *cache* é geralmente acima de 90%. Memórias *cache* também são voláteis, isto é, dependem de estar energizadas para manter gravado seu conteúdo.

2.2.5 Memória Principal

A Memória Principal é a parte do computador onde programas e dados são armazenados para processamento. A informação permanece na memória principal apenas enquanto for necessário para seu emprego pela UCP, após isto, esta área pode ser liberada para ser posteriormente sobre-gravada por outra informação. Quem controla a utilização da memória principal é o Sistema Operacional. A memória precisa ter uma organização que permita ao computador guardar e recuperar informações quando necessário.

Não teria nenhum sentido armazenar informações que não fosse possível recuperar depois. Portanto, não basta transferir informações para a memória. É preciso ter como encontrar essa informação mais tarde, quando ela for necessária, e para isso é preciso haver um mecanismo que registre exatamente onde a informação foi armazenada.

A memória principal é organizada em células. Célula é a menor unidade da memória que pode ser endereçada (não é possível buscar uma "parte" da célula) e tem um tamanho fixo (para cada máquina). As memórias são compostas de um determinado número de células ou posições. Cada célula é composta de um determinado número de bits. Todas as células de um dado computador têm o mesmo tamanho, isto é, todas as células daquele computador terão o mesmo número de bits.

Cada célula é identificada por um endereço único, pela qual é referenciada pelo sistema e pelos programas. As células são numeradas seqüencialmente, uma a uma, de 0 a (N-1), chamado o endereço da célula. Endereço é o localizador da célula, que permite identificar univocamente uma célula. Assim, cada célula pode ser identificada pelo seu endereço.

Unidade de transferência é a quantidade de bits que é transferida da memória em uma única operação de leitura ou transferida para a memória em uma única operação de escrita.

O tamanho da célula poderia ser igual ao da palavra, e também à unidade de transferência, porém por razões técnicas e de custo, são freqüentemente diferentes.

Computadores comerciais (tais como, por exemplo, os baseados nos processadores Intel 486) podem ter o tamanho da palavra definido como de 32 bits, porém sua estrutura de memória tem células de 16 bits.

2.2.6 Memórias Auxiliares

As memórias auxiliares resolvem problemas de armazenamento de grandes quantidades de informações. A capacidade da MP é limitada pelo seu alto custo, enquanto as memórias auxiliares têm maior capacidade e menor custo; portanto, o custo por bit armazenado é muito menor.

Outra vantagem importante é que as memórias auxiliares não são voláteis, isto é, não dependem de estar energizadas para manter gravado seu conteúdo.

Os principais dispositivos de memória auxiliar são: discos rígidos (ou *Hard Disk*), drives de disquete, unidades de fita, CD-ROM, DVD, unidades ótico-magnéticas, etc.

Obs.: *Cache* de disco não é a mesma tecnologia da memória *cache*. Trata-se do emprego do mesmo conceito da memória *cache*, para acelerar a transferência de dados entre disco, MP e UCP, usando um programa (um software, por ex.: SmartDrive) para manter um espelho do conteúdo de parte do disco (a mais provável de ser requisitada a seguir pela UCP) gravado em uma parte da Memória Principal. Recentemente, as unidades de disco passaram a incorporar em sua interface chips de memória - tipicamente 32 a 64 Kbytes - para acelerar a transferência de dados, utilizando um algoritmo de *cache*.

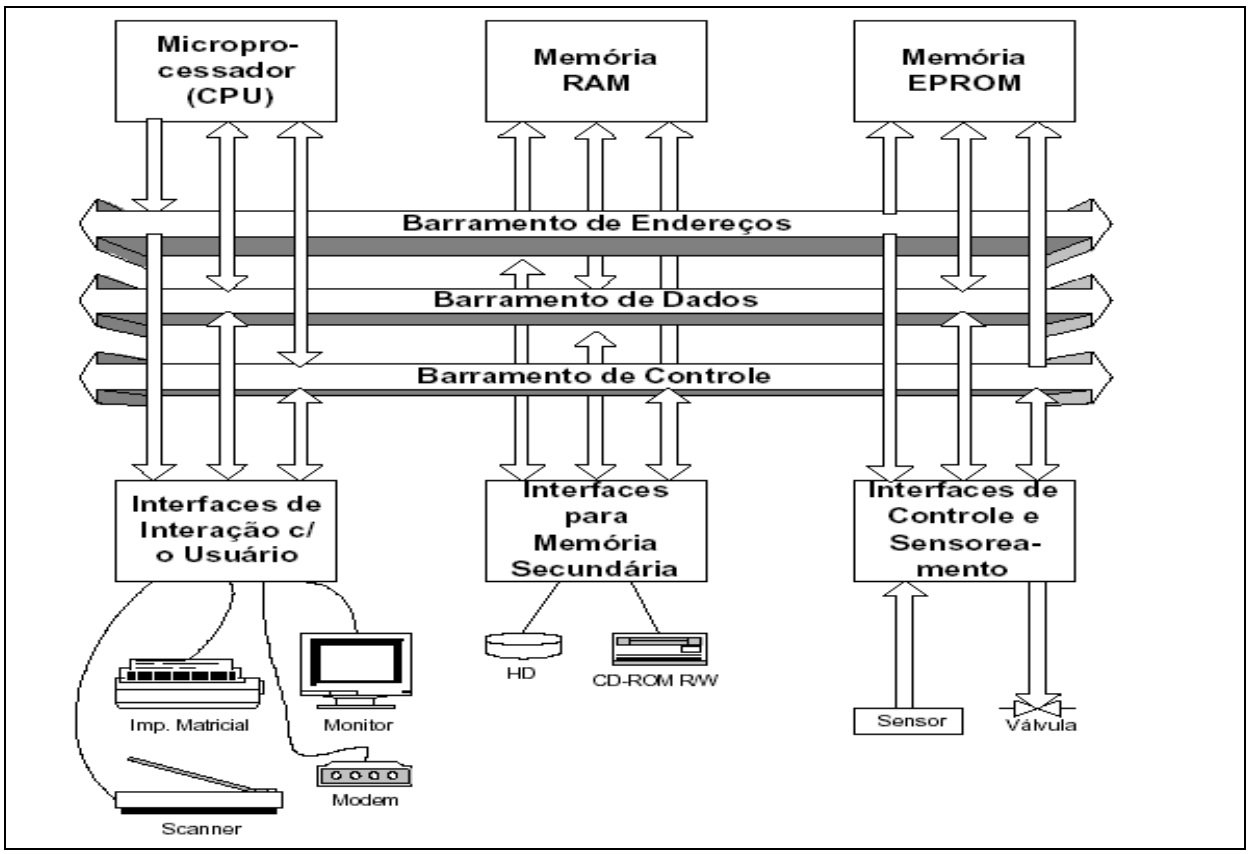
2.3 DISPOSITIVOS DE ENTRADA E SAÍDA

A utilização de computadores para a solução de problemas inclui três etapas básicas:

- entrada do programa e dados;
- processamento;
- saída dos resultados.

A entrada e saída de dados são efetuadas nos computadores através de dispositivos de entrada e saída. É necessário ressaltar que nem toda entrada é fornecida por pessoas assim como nem toda saída é destinada a pessoas. Os dispositivos de entrada e saída são os componentes que viabilizam a interface com o usuário, tais como: portas seriais, portas paralelas, conversores análogo-digitais, etc (Barretto, 2000).

Alguns dispositivos de entrada e saída podem transmitir uma grande quantidade de dados em um curto intervalo de tempo. Se a UCP tem que processar cada caractere separadamente, muito tempo de UCP é desperdiçado. Para impedir que a UCP fique presa por muito tempo em operações de entrada e saída, muitos computadores de médio e grande porte têm um ou mais processadores especializados e de baixo custo dedicados para tal tarefa. Devido a entrada e saída ser realizada por esses processadores, a UCP fica disponível para outros processamentos. Os processadores de entrada e saída trabalham em paralelo com a UCP, ou seja, enquanto a UCP está processando, esses processadores se dedicam a operações de entrada e saída. Na Figura 2 pode-se observar o tráfego de entrada e saída em um microcomputador típico.



Fonte: Marcelino (2002)

Figura 2: Microcomputador típico e suas interfaces

2.3.1 *Chipset*

Em uma placa-mãe existem diversos circuitos de apoio, que genericamente chamamos de *chipset*. O *chipset* define diversos fatores, como desempenho, quantidade máxima de memória que a placa-mãe aceitará e muito mais (Weber, 2000).

Tão importantes quanto as UCPs, e talvez mais que as memórias, os *chipsets* são circuitos os quais estão ligados diretamente ao chip da UCP e são responsáveis pela maioria das trocas de informações entre a UCP, memórias e barramentos como:

- interface IDE;
- controle das memórias RAM e de *cache* externa;
- controle de barramentos ISA e PCI;
- controle de DMA e interrupções.

A maioria dos *chipsets* foi projetada para operar com os processadores a partir do 486. Os *chipsets* pertencem à mesma classe de tecnologia dos chips de UCP, a tecnologia VLSI (*Very Large Scale Integration*). Um exemplo de *chipset* é a linha Intel 430FX.

2.3.2 Barramento

Também chamado em inglês de *bus*, é o meio físico responsável pela troca de dados entre circuitos, placas e equipamentos. Em um computador, são as trilhas e circuitos (quando existentes) responsáveis pela troca de dados entre o UCP, a memória, os dispositivos anexados e as placas do microcomputador (Torres, 1999).

Os barramentos podem ser classificados como unidirecionais (transmissão em um só sentido) ou bidirecionais (transmissão em ambos os sentidos). O sistema de barramentos de um microcomputador é composto de três barramentos independentes em suas funções elétricas (conforme Figura 2): o barramento de endereços, o barramento de dados e o barramento de controle (Barretto, 2000).

O Barramento de Endereços é apenas de saída (em relação à UCP) e define o caminho de comunicação dentro do sistema.

O Barramento de Dados é bidirecional, ou seja, da UCP para os periféricos e dos periféricos para UCP. Na saída de dados da UCP, estes são enviados à uma unidade que é selecionada pelo barramento de endereços. Na entrada de dados, estes são gerados por um periférico ou outra unidade independente e enviados ao microprocessador.

O Barramento de Controle, como o próprio nome indica, envia e recebe os sinais de controle necessários à transferência de dados no sistema. Este barramento é composto, basicamente de 4 tipos de sinais: leitura de memória ativa, escrita de memória ativa, entrada através de dispositivo externo ativo e saída através de dispositivo externo ativo. Podemos observar o funcionamento de uma placa mãe na Figura 3.

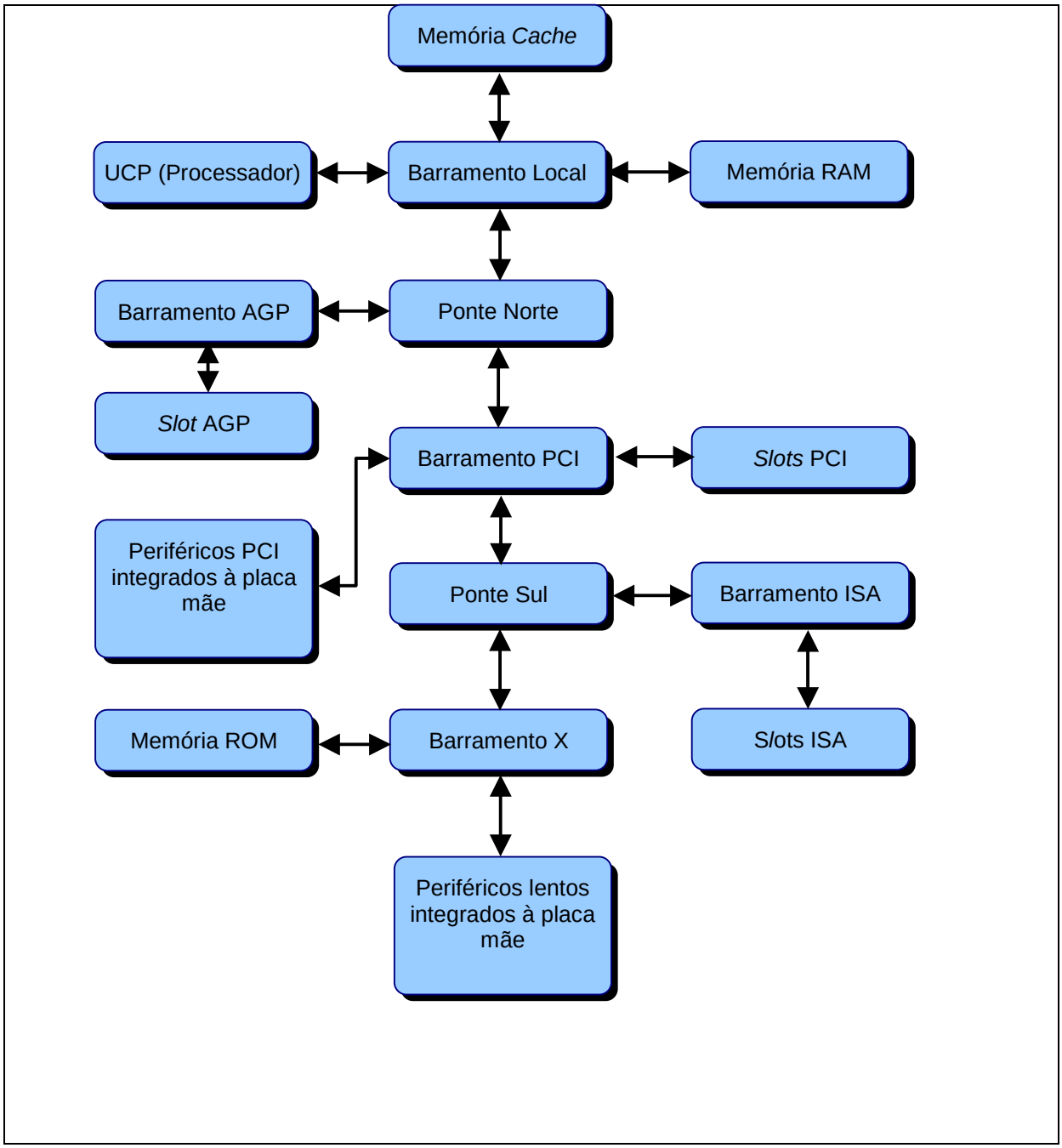


Figura 3: Esquema de funcionamento de uma placa-mãe

2.3.2.1 Barramento Local

O barramento local é utilizado na comunicação do UCP com os circuitos básicos que demandam velocidade (especialmente a memória RAM e a memória *cache*). O barramento local pode ser subdividido em Barramento de dados, Barramento de endereços e Barramento de controle (Barreto, 2000).

Algumas características deste barramento para os microcomputadores atuais:

- barramento de dados de 64 bits;
- barramento de endereços de 32 bits;
- freqüência de operação de 66, 100, 133, 200, 266, 400 MHz (as freqüências 200 e 266 referem-se ao barramento de 100 e 133 operando em DDR – *Double Data Rate*, ou seja, ocorre duas operações em um mesmo ciclo de barramento, enquanto a freqüência de 400 refere-se ao barramento de 100 operando em QDR – *Quadruple Data Rate*).

2.3.2.2 Barramento PCI

Após o advento do barramento PCI, a construção de *chipsets* e de placas-mãe tornou-se ainda mais fácil, devido a características próprias desse barramento. *Chipsets* para o barramento PCI possuem basicamente dois circuitos: Ponte Norte e Ponte Sul.

2.3.2.2.1 Funcionamento da Ponte Norte

Também chamada controladora de sistema, possui as seguintes funções: ponte barramento local-PCI, controlador de memória e controlador de *cache* L2 (exceto em placas-mãe para Pentium Pro, Pentium II e superiores, onde o controlador de *cache* L2 está no próprio processador).

Como podemos observar na Figura 3, a Ponte Norte é o circuito mais importante do *chipset*. Ela faz a conversão dos sinais do barramento local (que, em micros onde o barramento local trabalha a 66 MHz atinge 528 MB/s) para o barramento PCI (que atinge, no máximo, 132 MB/s).

Como possui embutido o controlador de memória RAM, a Ponte Norte também influi no desempenho do micro. Uma Ponte Norte pode buscar dados na memória RAM utilizando menos pulsos de clock que outro modelo de Ponte Norte, por exemplo. Isso significa que um *chipset* pode ser mais rápido do que o outro. De uma maneira mais ampla, chegamos à conclusão que uma placa-mãe pode ser mais rápida do que outra dependendo dos *chipsets* que elas utilizem.

Da mesma forma, a utilização de uma determinada tecnologia de memória RAM é definida pelo controlador de memória RAM que está embutido na Ponte Norte. Por exemplo, não podemos instalar memória EDO na maioria das placas-mãe para 486 porque a maioria dos *chipsets* para placas-mãe soquete 3 (mais especificamente a Ponte Norte) não sabe trabalhar com esse tipo de memória. Da mesma forma, a Ponte Norte do *chipset* Intel 430FX não sabe como trabalhar com memórias SDRAM, o que não acontece com os *chipsets* Intel 430VX e 430TX, por exemplo.

E, a freqüência de operação da placa-mãe é basicamente definida pela Ponte Norte. Por exemplo, A Ponte Norte dos *chipsets* Intel para placa-mãe com soquete 7 trabalham somente a até 66 MHz, enquanto a Ponte Norte dos *chipset* Via para placa-mãe com soquete 7 podem trabalhar a até 75 MHz (o *chipset* Via Apollo MVP3 trabalha a até 100 MHz). Isso permite a correta utilização dos processadores Cyrix de 75 MHz, dos processadores soquete 7 de 100 MHz (AMD K6-2 e Cyrix MII), bem como a configuração de *overclock*.

2.3.2.2.2 Funcionamento da Ponte Sul

Também chamada controlador de periféricos, possui as seguintes funções: ponte barramento PCI-ISA, controlador de interrupção, controlador de DMA, controle dos periféricos "*on board*" (controladora da unidade de disquete, porta serial, porta paralela e portas IDE).

A Ponte Sul possui como característica básica fazer a comunicação (ponte) entre o barramento ISA e o barramento PCI. Além disso, possui integrado o controlador de interrupções, controlador de DMA e também faz o controle dos periféricos "*on board*" básicos.

No caso de placas-mãe com outros periféricos integrados - como vídeo e áudio - o controle será feito por um processador de vídeo e/ou um processador de áudio. Esses processadores são conectados diretamente ao barramento PCI.

A grande preocupação em relação à Ponte Sul é saber se ela é capaz de trabalhar com o padrão Ultra-ATA (UDMA, Ultra-DMA), que permite discos rígidos IDE atingirem uma taxa de transferência de até 133 MB/s.

Outro detalhe sobre a Ponte Sul é que o barramento USB é conectado à ele.

2.3.2.3 Largura do Barramento

Um barramento é um canal no qual a informação circula, quanto maior for o número de linhas do barramento mais informação pode circular no barramento. O Barramento ISA original tem 8 bits, o barramento ISA usado atualmente tem 16 bits. Os outros barramentos (incluído VLB e PCI) são de 32 bits. O barramento local dos processadores Pentium são de 64 bits de largura.

2.3.2.4 Velocidade do Barramento

A velocidade do barramento reflete a quantidade de bits de informação que podem ser transferidos por segundo. A maioria dos barramentos transmite um bit por linha, por ciclo de relógio, no entanto os barramento de alta-performance como no AGP podem movimentar 2 bits por ciclo de *clock*, duplicando sua performance. Já o barramento ISA gasta em média 4 ciclos para transferir um dado (Marcelino, 2002).

2.3.2.5 Largura da Banda do Barramento

A Largura de banda refere-se à quantidade de dados que podem ser transferidos no barramento numa dada unidade de tempo.

A Tabela 3, mostra a largura de banda teórica que os barramentos podem suportar hoje em dia. Note-se que os barramentos podem trabalhar a diferentes velocidades (Marcelino, 2002).

Tabela 3: Taxa máxima de transferência dos barramentos mais conhecidos

Barramento	**Largura (bits)**	**Velocidade (MHz)**	**Largura da Banda (Taxa de Transferência Máxima) (MB/s)**
8-bit ISA	8	8,3	7,9
16-bit ISA	16	8,3	15,9
EISA	32	8,3	31,8
VLB	32	33	127,2
PCI	32	33	127,2
64-bit PCI 2.1	64	66	508,6
AGP	32	66	254,3
AGP (x2 *mode*)	32	66x2	508,6
AGP (x4 *mode*)	32	66x4	1017,3

Fonte: (Marcelino, 2002)

2.3.2.6 Interface do Barramento

Num sistema onde existem muitos barramentos distintos, devem ser previstos circuitos no *chipset* para interligar os barramentos e permitir aos diferentes dispositivos a comunicações entre eles. Estes dispositivos são chamados *bridges* ou “Pontes”, assim existe a PCI-ISA *bridge*, que faz parte do sistema do *chipset*, e faz a conversão do barramento PCI pra o ISA e vice-versa, o barramento PCI também tem uma *bridge* para o barramento do UCP ou *local bus* (Torres, 1999).

2.3.2.7 Ciclos do Barramento

Existem duas classificações gerais de tipos de ciclos de barramento, o ciclo do UCP, ou seja, ciclos gerados a partir do UCP e o ciclo de barramento de DMA gerado pelo controlador de DMA. O UCP pode gerar um dos cinco tipos de ciclos: leitura na memória, escrita na memória, leitura na porta de entrada e saída, escrita na porta de entrada e saída, atender às interrupções. Já os gerados pelo controlador de DMA são DMA *read* I/O e DMA *write* I/O. A Tabela 4 demonstra os tipos de ciclos de barramentos e seus propósitos.

Tabela 4: Tipos de Ciclos de Barramento

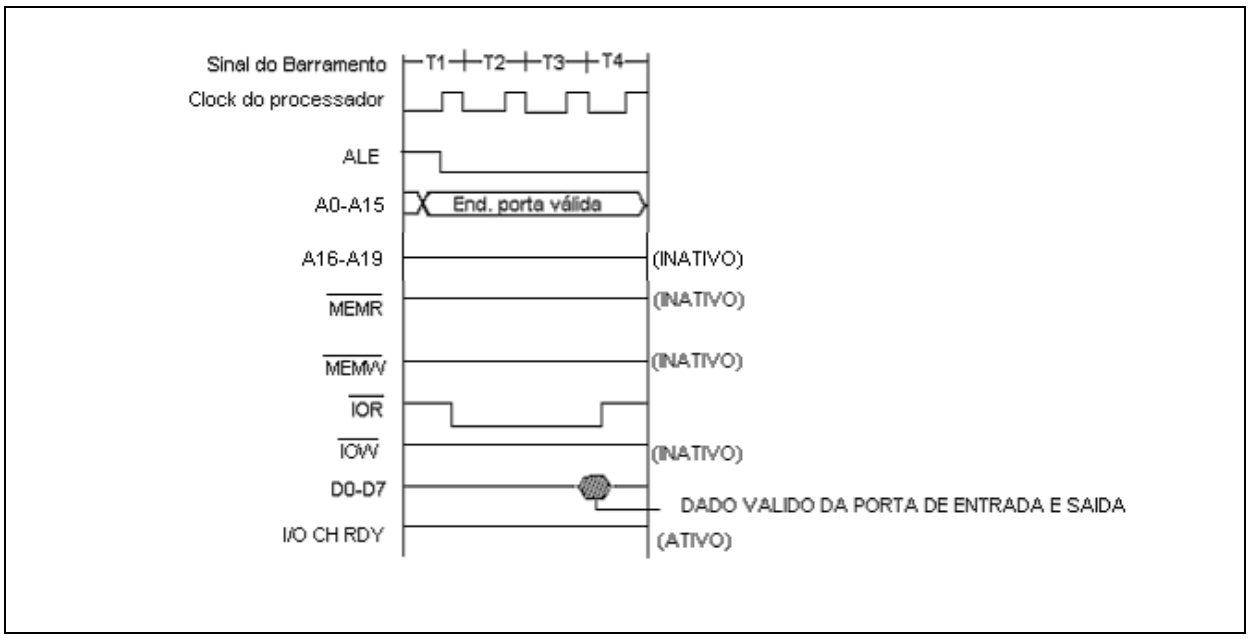
Tipo de Ciclo	Propósito	Fluxo dos dados
Leitura da Memória.	UCP busca dados ou instruções.	Memória para UCP.
Escrita da Memória.	UCP escreve dado.	UCP para memória.
Leitura da porta de entrada e saída.	UCP busca dados do dispositivo de entrada e saída.	Porta de entrada e saída para o UCP
Escrita da porta de entrada e saída.	UCP envia dado a dispositivo de entrada e saída.	UCP para porta de entrada e saída.
Requisição de interrupções	Envio de interrupção ao UCP.	Controlador de interrupção para UCP
Escrita da porta de entrada e saída por DMA.	Envio de dados da memória para a interface de entrada e saída.	Memória para dispositivo de Entrada e Saída
Leitura da porta de entrada e saída por DMA.	Envido de dados da interface de entrada e saída para a memória.	Dispositivo de Entrada e Saída para memória

Fonte: Adaptado de Eggebrecht (1995).

2.3.2.8 Ciclo de Leitura da Porta de Entrada e Saída

O ciclo de leitura na porta de entrada e saída é executado quando uma instrução IN e executada pelo UCP, este ciclo é similar ao ciclo de leitura da memória descrita acima, seu propósito é de buscar o dado contido no dispositivo de entrada e saída endereçado no barramento de endereço. Este ciclo possui aproximadamente 5 ciclos de *clock*, e este dispositivo pode estender o tamanho do ciclo desativando o sinal de barramento *ready* (Eggebrecht, 1995).

O ciclo de leitura da porta de entrada e saída inicia no tempo T1 do *clock* com o sinal de ALE sendo ativado ou posto em alta, isto indica que o barramento de endereço conterá um endereço de uma porta válida, e seguindo aproximadamente no T2, o sinal de IOR é ativado no barramento de controle, que indica que o ciclo é um ciclo de leitura da porta de entradas e saída e que o dispositivo de I/O cujo endereço for igual ao endereço contido no barramento de dados deverá disponibilizar seu dado no barramento de dados. A Figura 4 demonstra graficamente este ciclo.



Fonte: Adaptado de Eggebrecht (1995).

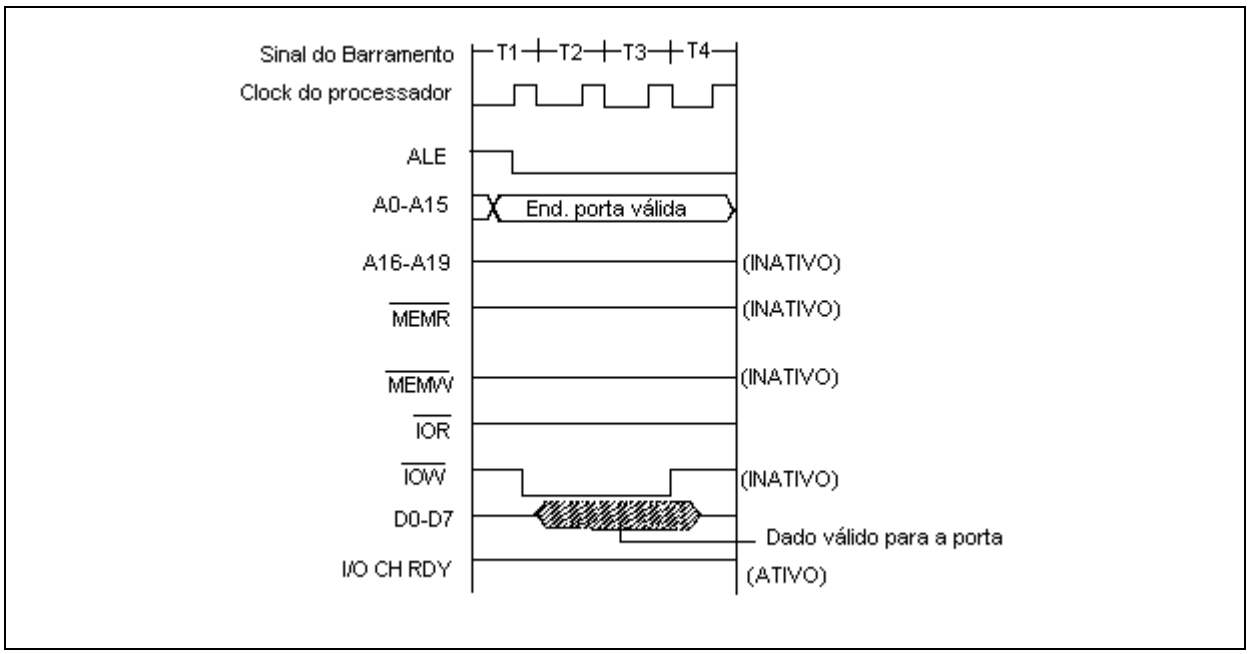
Figura 4: Ciclo de leitura da porta de entrada e saída

2.3.2.9 Ciclo de Escrita na Porta de Entrada e Saída

O ciclo de escrita na porta de entrada e saída é executado quando uma instrução OUT e executada pelo UCP, este ciclo escreve o dado contido num registrador do UCP em uma porta especifica. Este ciclo possui aproximadamente cinco ciclos de *clock*, e este dispositivo pode estender o tamanho do ciclo desativando o sinal de barramento READY (Eggebrecht, 1995).

O ciclo de escrita na porta de entrada e saída inicia no tempo T1 do *clock* com o sinal de ALE sendo ativado ou posto em alta, isto indica que o barramento de endereço conterá um endereço válido na porta. No T2, o sinal de IOW é ativado no barramento de controle (“0”), que indica que o ciclo é para escrita na porta e que o dispositivo cujo endereço for igual ao endereço contido no barramento de dados está sendo selecionado.

Após T2 o UCP envia ao barramento de dados o dado que devera ser recebido pelo dispositivo selecionado, e no início de T4 o sinal de IOW é desativado (“1”) e no final de T4 o ciclo estará completo. A Figura 5 demonstra graficamente este ciclo.



Fonte: Adaptado de Eggebrecht (1995).

Figura 5: Ciclo de escrita na porta de entrada e saída

2.3.3 Barramento ISA

O barramento ISA (*Industry Standard Architecture*) foi o primeiro barramento de expansão a ser criado. Em seus *slots* pode-se conectar qualquer placa ISA.

Nos primeiros PC's, os 8088 eram utilizados barramentos de dados de 8 bits, sendo assim os *slots* de expansão também eram de 8 bits. É interessante notar, que na época do PC e do PC XT o barramento ISA era conectado diretamente ao barramento local da máquina, já que essas máquinas utilizavam baixas freqüências de operação (Barreto, 2000) .

Com a introdução do micro AT, o barramento e o *slot* ISA aumentaram de tamanho de forma a acompanhar as características do UCP 80286, conforme relacionado abaixo:

- barramento de dados de 16 bits;
- barramento de endereços de 24 bits;
- freqüência de operação de 8 MHz;
- acesso a 16 MB de memória RAM;
- taxa de transferência máxima de 8 MB/s para o barramento ISA 8 bits e 16 MB/s para o ISA 16 bits.

O barramento ISA opera a uma freqüência máxima de 8MHz independendo se é um PC AT 286 ou um Pentium II isto ocorre por motivos de que uma placa antiga não funcionaria

em micros novos, e também dispositivo que utilizam baixa taxa de transferência como placas de som e placas de fax/modem que não necessitam de altas taxas de transferência.

Para a comunicação do barramento ISA com o barramento local do micro (que são bastante diferentes hoje em dia basta pensar que o barramento ISA trabalha no máximo a 16 *bits* e o barramento local de um Pentium e de no mínimo 64 bits), há um circuito próprio para o interfaceamento, chamando de controlador de barramento ISA. Este circuito está integrado ao *chipset* da placa-mãe e converte todas as informações de um barramento para o outro, este procedimento é chamando de ponte PCI-ISA (ou ponte sul) conforme esquema ilustrado na Figura 2 (Torres, 1999).

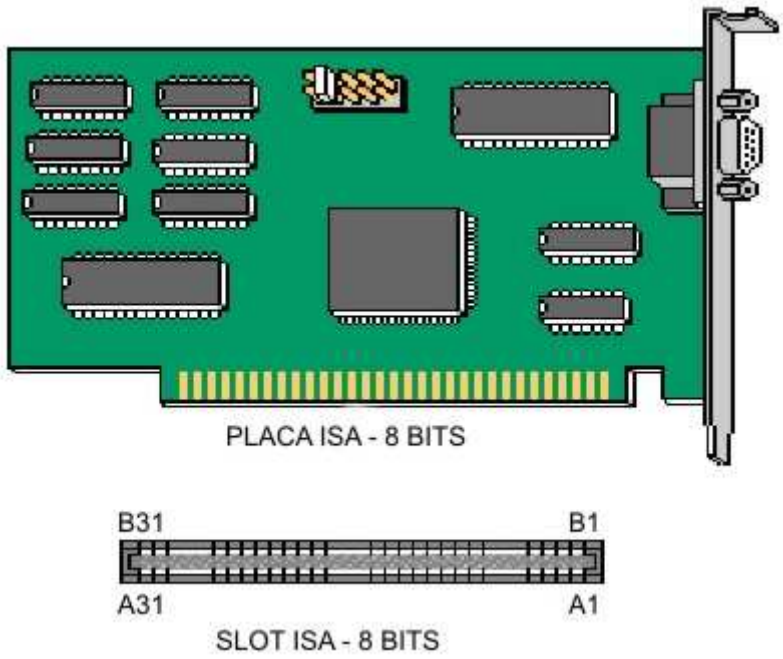
2.3.3.1 Recursos do Barramento ISA

Desde o primeiro PC, o controle dos recursos de hardware se dá através dos endereços de entrada e saída, linhas de interrupção (IRQ) e canais de DMA. Praticamente todas as placas ISA utilizam pelo menos um destes recursos. Por exemplo, uma placa de som típica utiliza o endereço 220h, interrupção IRQ5 e canais de DMA 1 e 5. O barramento ISA traz estes recursos da seguinte forma:

- endereços de entrada e saída: 1 KB (de 000h a 3FFh);
- interrupções: 15 linhas;
- canais de DMA: 8 canais.

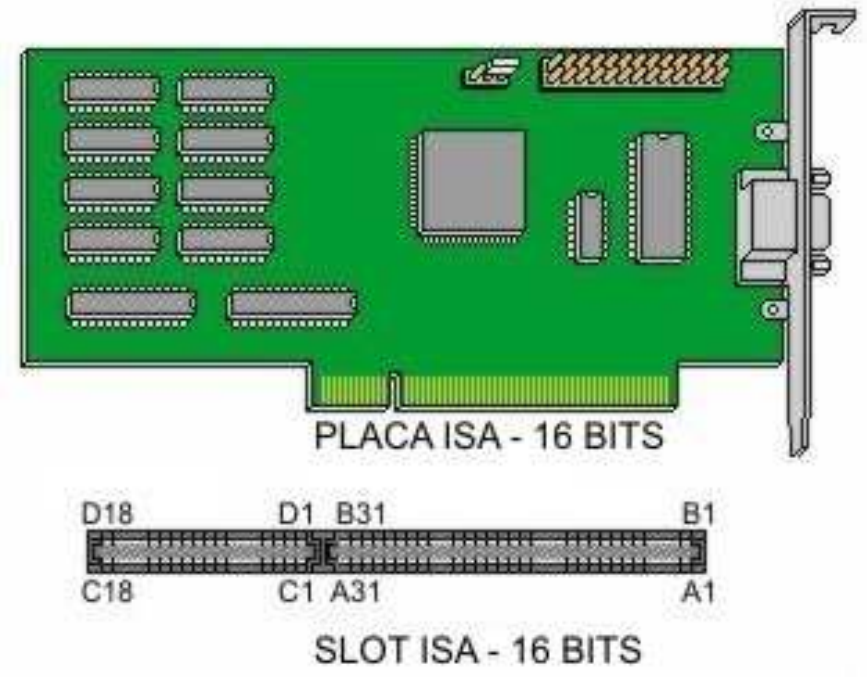
2.3.3.2 Descrição dos Sinais do Barramento ISA

A interligação do micro-computador com uma placa de expansão é feita através dos barramentos de dados, controle e endereço. As Figuras 6 e 7 e a Figura 8 mostram os tipos de conectores e placas de expansão ISA que podem ser de 8 ou 16 bits. A Tabela 5 apresenta todos os sinais do barramento ISA de 8 e a Tabela 6 os sinais do barramento de expansão de 16 bis.



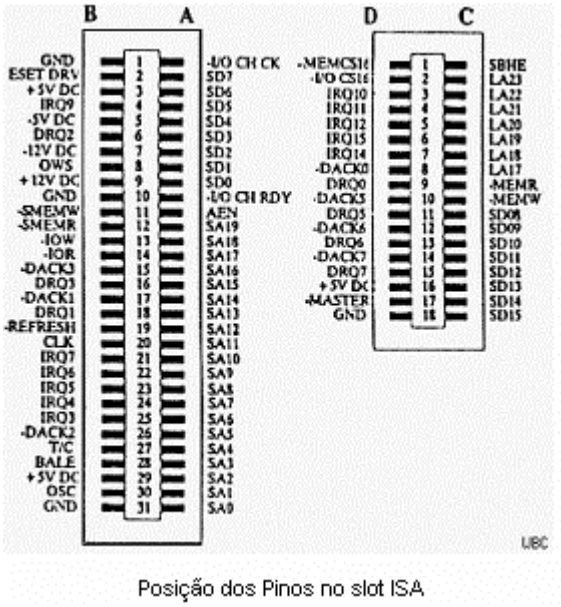
Fonte: Adaptado de Eggebrecht (1995).

Figura 6: Arquitetura do Barramento ISA - 8 bits



Fonte: Adaptado de Eggebrecht (1995).

Figura 7: Arquitetura do Barramento ISA - 16 bits



Fonte: Adaptado de Eggebrecht (1995).

Figura 8: Arquitetura do Barramento ISA – Pinagem

Tabela 5: Parte A/B dos pinos do barramento ISA

Sinal	**Nome**	**E/S**	**Descrição**
A0-A19	A0-A19	S	Barramento de endereços (sinal A0 pino A31, sinal A19 pino A12).
AEN	*Address Enable*	S	Quando ativo, este sinal indica que a operação executada no barramento é de DMA. Devemos utilizar este sinal em nossos protótipos de modo que o decodificador não capture um dado erroneamente durante uma operação de DMA do micro.
ALE	*Address Latch Enable*	S	Indica que há um endereço válido no barramento de endereços. Não é ativado durante operações de DMA.
CLK	*Clock*	S	*Clock* do barramento ISA, de 8 MHz. Ele não é simétrico, apresentando um ciclo de carga de 33%, ou seja, 1/3 em nível alto e 2/3 em nível baixo.
D0-D7	D0-D7	E/S	Barramento de dados (D0 pino A9, D7 pino A2).
/DACK1	DMA *Acknowledge* 1	S	Indica que o pedido de DMA 1 foi aceito e que o controlador de interrupção irá iniciá-lo.
/DACK2	DMA *Acknowledge* 2	S	Indica que o pedido de DMA 2 foi aceito e que o controlador de interrupção irá iniciá-lo.
/DACK3	DMA *Acknowledge* 3	S	Indica que o pedido de DMA 3 foi aceito e que o controlador de interrupção irá iniciá-lo.
DRQ1	DMA *Request* 1	E	Faz um pedido de DMA nível 1 ao controlador de DMA.
DRQ2	DMA *Request* 2	E	Faz um pedido de DMA nível 2 ao controlador de DMA.
DRQ3	DMA *Request* 3	E	Faz um pedido de DMA nível 3 ao controlador de DMA.
/I/O CH CK	I/O *Channel Check*	E	Ativando este sinal, gera-se uma interrupção não-mascarável (NMI). É ativado em situações de erro, como erro de paridade (Obs: este sinal deve ser utilizado com um circuito de coletor aberto).

/I/O CH RDY	I/O *Channel Ready*	E	Gera *wait states* para operações de I/O ou de memória. Se o circuito necessitar de wait states, basta ativar esta linha após a decodificação do endereço e dos sinais /MEMR, /MEMW, /IOR ou /IOW. Devemos temporizar esta linha com muito cuidado, para não inserirmos *wait states* desnecessários (Obs: este sinal deve ser utilizado com um circuito de coletor aberto).
/IOR	I/O *Read*	S	Indica que uma leitura em I/O está sendo executada. É ativado também durante ciclos de DMA.
/IOW	I/O *Write*	S	Indica que uma escrita em I/O está sendo executada. É também ativado durante operações de DMA.
IRQ3	*Interrupt Request* 3	E	Faz um pedido de interrupção nível 3 ao controlador de interrupções.
IRQ4	*Interrupt Request* 4	E	Faz um pedido de interrupção nível 4 ao controlador de interrupções.
IRQ5	*Interrupt Request* 5	E	Faz um pedido de interrupção nível 5 ao controlador de interrupções.
IRQ6	*Interrupt Request* 6	E	Faz um pedido de interrupção nível 6 ao controlador de interrupções.
IRQ7	*Interrupt Request* 7	E	Faz um pedido de interrupção nível 7 ao controlador de interrupções.
IRQ9	*Interrupt Request* 9	E	Faz um pedido de interrupção nível 9 ao controlador de interrupções.
/MEMR	*Memory Read*	S	Indica que está sendo executada uma operação de leitura em memória. É também ativado durante operações de DMA.
/MEMW	*Memory Write*	S	Indica que está sendo executada uma operação de escrita em memória. É também ativado durante operações de DMA.
OSC	Oscilador	S	Sinal com freqüência de 14,31818 MHz, utilizado para a sincronização do vídeo CGA. Não é sincronizado com o *clock* do micro, portanto, cuidado.
RESET DRV	*Reset Driver*	S	Sinal de *reset* do sistema.
TC	*Terminal Count*	S	Indica que um dos canais de DMA acabou de realizar a transferência de DMA programada.
+5V	+5V	S	+5V
-12V	-12V	S	-12V
-5V	-5V	S	-5V
GND	Terra	S	Terra

Fonte: Adaptado de Eggebrecht (1995).

Tabela 6: Parte D/C dos pinos do barramento ISA

Sinal	Nome	E/S	Descrição
A17-A23	A17-A23	S	Barramento de endereços, parte alta. Estas linhas não são válidas durante todo o ciclo do barramento, por isto devemos utilizar um *latch* para armazenarmos seus valores (sinal A17 pino C8, sinal A23 pino C2).
D8-D15	D8-D15	E/S	Barramento de dados, parte alta (sinal D8 pino C11, sinal D15 pino C18).
/DACK0	DMA *Acknowledge* 0	S	Indica que o pedido de DMA 0 foi aceito e que o controlador de interrupção irá iniciá-lo.
/DACK5	DMA *Acknowledge* 5	S	Indica que o pedido de DMA 5 foi aceito e que o controlador de interrupção irá iniciá-lo.
/DACK6	DMA *Acknowledge* 6	S	Indica que o pedido de DMA 6 foi aceito e que o controlador de interrupção irá iniciá-lo.
/DACK7	DMA *Acknowledge* 7	S	Indica que o pedido de DMA 7 foi aceito e que o controlador de interrupção irá iniciá-lo.
DRQ0	DMA *Request* 0	E	Faz um pedido de DMA nível 0 ao controlador de DMA.
DRQ5	DMA *Request* 5	E	Faz um pedido de DMA nível 5 ao controlador de DMA.
DRQ6	DMA *Request* 6	E	Faz um pedido de DMA nível 6 ao controlador de DMA.
DRQ7	DMA *Request* 7	E	Faz um pedido de DMA nível 7 ao controlador de DMA.
/IOCS16	I/O *Chip Select* 16 bits	E	Indica que haverá uma transferência de 16 bits no barramento de dados, utilizando endereçamento em I/O (Obs: este sinal deve ser utilizado com um circuito de coletor aberto).
IRQ10	*Interrupt Request* 10	E	Faz um pedido de interrupção nível 10 ao controlador de interrupções.
IRQ11	*Interrupt Request* 11	E	Faz um pedido de interrupção nível 11 ao controlador de interrupções.
IRQ12	*Interrupt Request* 12	E	Faz um pedido de interrupção nível 12 ao controlador de interrupções.
IRQ14	*Interrupt Request 14*	E	Faz um pedido de interrupção nível 14 ao controlador de interrupções.
IRQ15	*Interrupt Request* 15	E	Faz um pedido de interrupção nível 15 ao controlador de interrupções.
/MASTER	*Máster*	E	Este sinal é ativado quando queremos que algum outro dispositivo assuma o controle do barramento (*bus master*). Deve ser acionado em conjunto com o sinal DRQ. Quando o sinal /DACK correspondente for devolvido, todos os circuitos conectados ao barramento são colocados em *tri-state*, permitindo que o dispositivo manipule o barramento como bem entender. O novo mestre de barramento deverá

			obedecer aos sinais de temporização e devolver o controle no máximo em 15 μs.
/MEM CS16	*Memory Chip Select 16 bits*	E	Indica que haverá uma transferência de 16 bits no barramento de dados, utilizando endereçamento em memória (Obs: este sinal deve ser utilizado com um circuito de coletor aberto).
/MEM R	*Memory Read*	S	É ativado quando é feita uma leitura em uma posição de memória acima de 1 MB.
/MEM W	*Memory Write*	S	É ativado quando é feita uma escrita em uma posição de memória acima de 1 MB.
SBHE	*System Bus High Enable*	S	Indica que a transferência de dados utilizará a parte alta do barramento de dados (D8-D15). Este sinal e A0 são decodificados para informar que tipo de transferência será efetuada (vide tabela).
+5V	+5V	S	+5 V

Fonte: Adaptado de Eggebrecht (1995).

2.4 ENDEREÇOS DE ENTRADA E SAÍDA

Os endereços de entrada e saída são utilizados na comunicação do UCP com um dispositivo através das instruções "*In*" e "*Out*". Como exemplo, a porta paralela que se comunica com seus periféricos no endereço de entrada e saída 378h. Caso for utilizar uma impressora para imprimir um texto, isso será feito enviando dados através desse endereço. A Tabela 7 mostra os endereços tipicamente utilizados do PC AT (Eggebrecht, 1995).

Tabela 7: Mapa de endereços de entrada e saída do PC AT

Faixa de Endereço (em Hexadecimal)	**Dispositivo**
000-01F	Controlador de DMA 1
020-03F	Controlador de interrupção 1
040-05F	*Timer*
060-06F	Teclado
070-07F	*Clock* de tempo real, NMI mask
080-09F	Registrador de DMA 74LS612
0A0-0BF	Controlador de interrupção 2
0C0-0DF	Controlador de DMA 2
0F0	Limpa o co-processador matemático ocupado
0F1	Reseta o co-processador matemático
0F8-0FF	Co-processador matemático
1F0-1F8	Disco Rígido
200-207	E/S para jogos
278-27F	Porta paralela 2 (LPT2)
2F8-2FF	Porta serial 2 (com2)
300-31F	Porta para prototipar placas
360-36F	Reservado
378-37F	Porta paralela 1 (LPT1)
380-38F	Bi-sincronizador SDLC 2

3A0-3AF	Bi-sincronizador SDLC 1
3B0-3BF	Adaptador de vídeo monocromático
3C0-3CF	Reservado
3D0-3DF	Adaptador para monitor colorido
3F0-3F7	Controlador de disquete
3F8-3FF	Porta serial 1 (com1)

Fonte: Adaptado de Eggebrecht (1995).

2.5 INTERRUPÇÕES

Pode-se dizer que interrupções e *traps* são as forças que controlam os sistemas operacionais. Um sistema operacional somente recebe o controle da execução quando ocorre alguma interrupção ou *trap*.

Uma interrupção é um sinal de hardware que faz com que o UCP interrompa a execução de um programa e passe a executar uma rotina específica que trata tal interrupção de acordo com seu código. Ao terminar a execução da sub-rotina, restaura os registradores com o programa que foi interrompido e a partir daí, continua executando tal programa como se nada houvesse ocorrido. Este tipo de interrupção é chamado de interrupção por hardware.

Interrupções podem ser originadas pelos vários dispositivos periféricos (terminais, discos, impressoras, etc.), pelo operador (através do teclado ou mouse) ou pelo relógio do sistema. O relógio (*timer*) é um dispositivo de *hardware* que decrementa automaticamente o conteúdo de um registrador ou posição de memória, com uma freqüência constante, e interrompe a UCP quando o valor decrementado atinge zero. O sistema operacional garante que ocorrerá pelo menos uma interrupção e voltará dentro de um intervalo de tempo **t**, colocando no relógio um valor que demore **t** unidades de tempo para ser decrementado até zero. Esta atribuição de valor ao relógio é feita imediatamente antes do sistema operacional entregar a UCP para um programa de usuário.

Uma interrupção não afeta a instrução que está sendo executada pela UCP no momento em que ela ocorre: a UCP detecta interrupções apenas após o término da execução de uma instrução (antes do início da execução da instrução seguinte).

Os computadores possuem instruções para mascarar (desabilitar, inibir) o sistema de interrupções. Enquanto as interrupções estão mascaradas elas podem ocorrer, mas não detectadas pelo UCP. Neste caso, as interrupções ficam pendentes (enfileiradas) e somente serão tratadas quando uma instrução que desmascara as mesmas é executada.

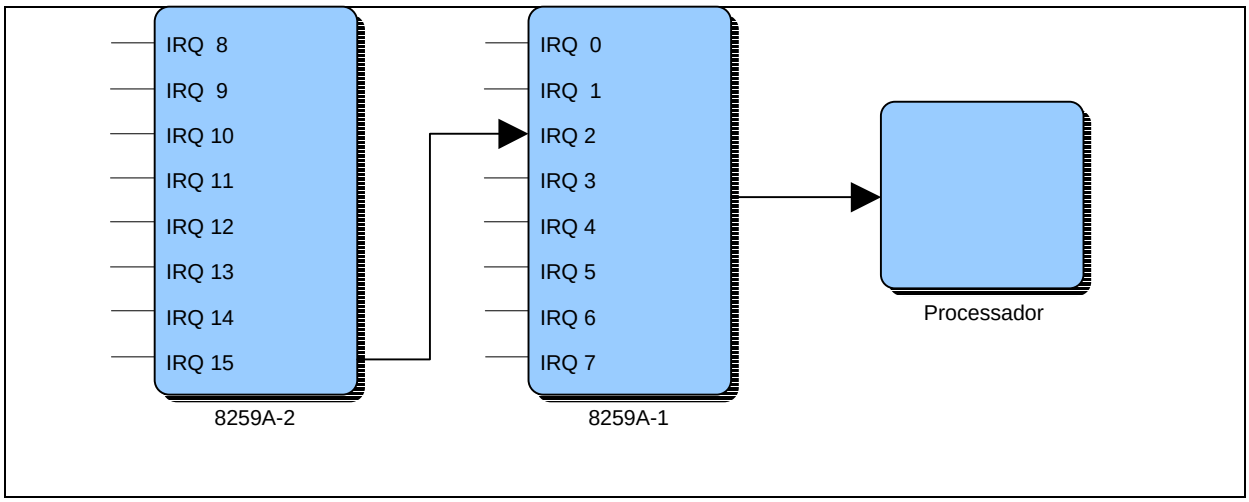
Uma *trap* é uma instrução especial que, quando executada pelo UCP interrompe a execução de um programa e aciona uma sub-rotina, conforme uma interrupção de hardware. A diferença é que uma *trap* é dada por software, por isso é denominada interrupção de software.

Pode-se dizer que *trap* é uma interrupção prevista, programada no sistema pelo próprio programador. Uma interrupção de hardware por outro lado, é completamente imprevisível, ocorrendo em pontos que não podem ser pré-determinados.

As *traps* têm a finalidade de permitir aos programas dos usuários a passagem do controle da execução para o sistema operacional. Por esse motivo também são denominados como “chamadas do sistema” (*supervisor call* ou *system call*). As *traps* são necessárias, principalmente, nos computadores que possuem **instruções protegidas** (privilegiadas). Nesses computadores, o registrador (palavra) de estado do UCP possui um bit para indicar se a UCP está em **estado privilegiado** (estado de sistema, estado de supervisor, estado mestre) ou não privilegiado (estado de usuário, estado de programa, estado escravo). Sempre que ocorre uma interrupção de hardware ou um *trap*, o novo valor carregado no registrador de estado do UCP, indica estado privilegiado de execução.

No estado de supervisor qualquer instrução pode ser executada e no estado de usuário apenas as instruções não protegidas podem ser executadas. Exemplos de instruções protegidas são instruções para desabilitar ou habilitar interrupções e instruções para realizar operações de entrada e saída. Operações que envolvam o uso de instruções protegidas somente podem ser executadas pelo sistema operacional. Quando um programa de usuário necessita executar alguma dessas operações, o mesmo deve executar uma *trap*, passando como argumento o número que identifica a operação que está sendo requerida.

A primeira versão do barramento ISA e dos microcomputadores, o PC XT possuía somente 8 níveis de interrupções, nas com a chegada dos PCs ATs foram introduzidos mais um circuito com mais 8 níveis de interrupções em cascata ao primeiro criando assim 15 níveis de interrupções. Estes controladores de interrupções estão interligados no circuito do *chipset* da placa-mãe chamado de Ponte sul, que é o circuito responsável por controlar o barramento ISA, a Figura 9 demonstra como esta ligação ocorre (Torres, 1999).



Fonte: Gabriel Torres (1999).

Figura 9: Como Funciona o esquema de interrupções

Na realidade, não há todas estas linhas disponíveis. Observando no mapa de interrupções (mapa da tabela 8), é possível verificar que há diversas interrupções automaticamente alocadas pela placa-mãe. Um PC típico, possui apenas 4 interrupções disponíveis caso não utilize nenhum periférico extra.

Tabela 8: Mapa de Interrupções por ordem de Prioridade

Interrupção	**Descrição**
IRQ0	Temporizador da placa-mãe (conectado ao *chipset*)
IRQ1	Teclado (Conectado ao *chipset*)
IRQ8	Relógio de tempo real (conectado ao *chipset*)
IRQ9	Interface de vídeo
IRQ10	Normalmente disponível
IRQ11	Normalmente disponível
IRQ12	Mouse de barramento (Bus Mouse, mouse OS/2)
IRQ13	Co-processador matemático (conectado ao *chipset*)
IRQ14	Porta IDE primária
IRQ15	Porta IDE secundaria
IRQ2	Conexão em cascata (conectado ao *chipset*)
IRQ3	COM2 e COM4 (comunicação serial)
IRQ4	COM1 e COM3 (comunicação serial)
IRQ5	Placa de som
IRQ6	Unidade de disquete
IRQ7	Porta Paralela

Fonte: Adaptado de Torres (1999)

3 DESENVOLVIMENTO DO TRABALHO

Este capítulo descreve todos os passos da implementação do trabalho desenvolvido. Inicialmente, foi utilizado como base o projeto elaborado por Santos (2002), onde foi estudado todo seu mecanismo de funcionamento. Devido à utilização de componentes com baixo índice de integração, necessitando uma área muito grande para montagem dos mesmos, esse projeto base foi abandonado. Com base numa nova pesquisa, chegou-se a componentes projetados especificamente para essa finalidade com um excelente índice de integração. Com um ponto de partida definido, o restante do trabalho ocorreu em três etapas:

- estudo das pinagens que seriam utilizados no protótipo e em seguida a elaboração do diagrama esquemático das interligações dos componentes;
- confecção das placas controladoras;
- desenvolvimento de um software para testes do hardware.

Para a elaboração do diagrama esquemático foram estudados todos os contatos elétricos que seriam necessários utilizar o barramento ISA, e sua interconexão com os demais componentes utilizados. Para isto, foi utilizado o software Liatro por sua praticidade. Através do diagrama foi possível definir um melhor *lay-out* para os componentes na placa. Para confecção de duas placas controladoras foi criado um desenho elaborado com o *software* AutoCad.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos deste protótipo são efetuar a comunicação entre dois computadores utilizando duas placas de comunicação conectadas no barramento ISA, com controle de fluxo por interrupções e banda de transmissão de dados em 16 bits. Para elaboração deste trabalho foram utilizados dois computadores um Pentium 233MMX e outro Pentium 166MMX que dispunham de *slots* disponíveis no padrão ISA.

3.2 ESPECIFICAÇÃO

As montagens do primeiro protótipo foram utilizadas duas placas de prototipação tipo "*Protoboards*" para testes com os componentes.

O multímetro que é um dispositivo capaz de medir intensidade da corrente, foi um instrumento muito utilizado para realizar os testes das conexões, correntes e tensões no circuito; ele é um item muito importante no manuseio de equipamentos de eletrônica e informática, pois sua praticidade e capacidade múltipla de medição fez com que eliminamos vários outros instrumentos, como o amperímetro e o voltímetro (Ribeiro, 2002).

Para a confecção dos esquemas das placas foi utilizado o editor de esquemas Liatro que é um pacote de software utilizado para montagens de esquemas e projetos de desenvolvimento de circuitos eletrônicos.

Para criar dois protótipos de placa de interface de comunicação primeiramente foi definido utilizar o barramento ISA, devido às complexidades técnicas em elaborar uma placa de circuito impresso para o barramento PCI ou *firewire* no espaço de tempo disponível para elaborar este trabalho, e também por se tratar de um barramento mais simples e com vasta bibliografia. O barramento ISA é um conector que pode ser encontrado em quase todos os microcomputadores mais antigos desde o PC-XT até ao Pentium III.

Foi estudado o barramento ISA, seus sinais bem como o funcionamento de suas funções de escrita e leitura nas portas de entrada e saída que foram detalhadas no tópico *2.3.2.6 Ciclos de Barramento* do capítulo anterior. Também foi definida uma faixa de endereço de entrada e saída para as placas que vai do 200H até ao 2E0H, para definir os componentes que podem ser usados para a decodificação dos endereços, pois como já foi visto o processador consegue acessar os adaptadores através de suas portas ou endereços. O endereço 2C0H foi definido como o padrão para enviar e receber os pacotes de dados, o 2C2H foi utilizado para status da placa, ou seja, se ela está ou não disponível para escrita. O 2C3H foi utilizado para escrever a palavra de controle do circuito integrado 82C55A.

Os sinais utilizados para confecção do protótipo foram:

- A0, A1, A5, A6, A7, A8 a A9: sinais do barramento de endereço utilizados para endereçar a porta do protótipo;
- D0 a D7: sinais do barramento de dados da parte baixa (padrão ISA 8 bits) utilizados para tratar o dado lido ou escrito no dispositivo;
- D8 a D15: sinais do barramento de dados da parte alta (padrão ISA 16 bits) utilizados para tratar o dado lido ou escrito no dispositivo;

- AEN: sinal do controlador de DMA que quando vai a HIGH “Alta”, ou 5 volts indica que o barramento de endereço possui um endereço válido para um ciclo de leitura ou escrita;
- IOR: sinal utilizado para indicar um ciclo ou operação de leitura na porta endereçada pelo barramento de endereço;
- IOW: sinal utilizado para indicar um ciclo ou operação de escrita na porta endereçada pelo barramento de endereço;
- IRQ 5, IRQ 7, IRQ 9 e IRQ 10: são os sinais correspondentes ao controle de interrupções;
- Vcc: Sinal do barramento ISA que representa a corrente elétrica de 5 volts no circuito;
- GND: Sinal do barramento ISA para apresentar o neutro ou terra do dispositivo.

Com estes sinais, foi desenvolvido um diagrama esquemático do circuito que pode ser visto no diagrama da Figura 10.

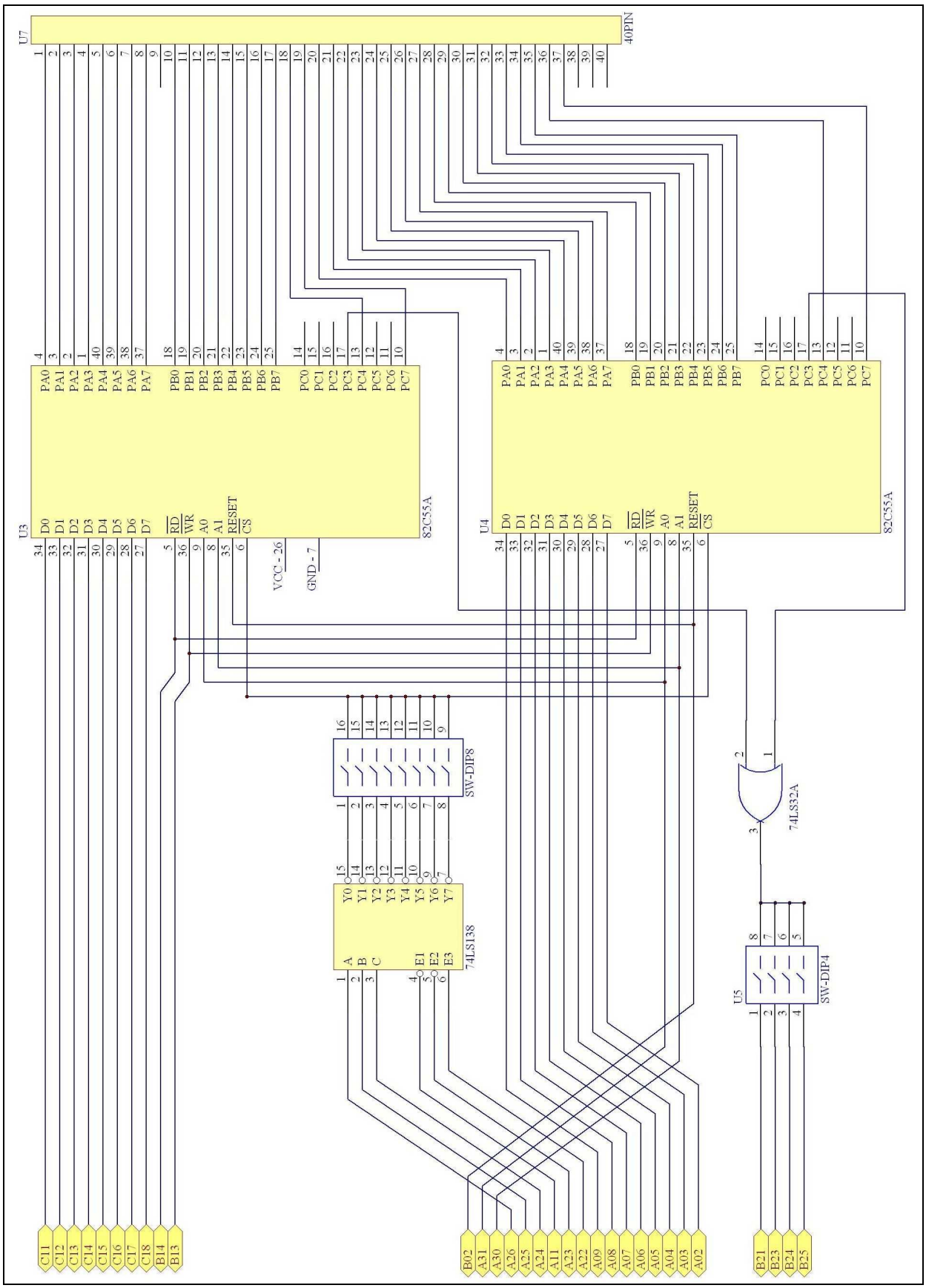


Figura 10: Diagrama esquemático do protótipo.

Para criação deste circuito foram utilizados os circuitos integrados 82C55A, SN74LS138 e o SN74LS32.

3.2.1 O SN74LS138

O circuito integrado SN74LS138 é um decodificador do tipo multiplexador de 3 entradas e 8 saídas utilizadas para habilitar o endereço da placa para leitura e escrita. Nele são conectados os sinais A5 no pino 1, A6 no pino 2, A7 no pino 3, AEN no pino 4, o A8 no pino 5 e o A9 no pino 6. No pino 8 é conectado o sinal GND (terra) e no pino 16 é conectado o sinal Vcc (5 volts). Para estabilização, foi feita uma "ponte" do Vcc (no pino 16) ao sinal GND (terra) com um capacitor cerâmico nº 104. O pino 7 é um sinal de saída que vai definir o endereço base como sendo o 2E0H, o pino 9 vai definir o endereço base 2C0H, o pino 10 define o endereço base como sendo o 2A0H, o pino 11 define o endereço base como sendo o 280H, o pino 12 define o endereço base como sendo 240H, o pino 13 define o endereço base como o 220H e o pino 14 define como o endereço base o 200H. A Tabela 9 mostra os endereços decodificados por este circuito integrado e qual pino de saída representa tal endereço.

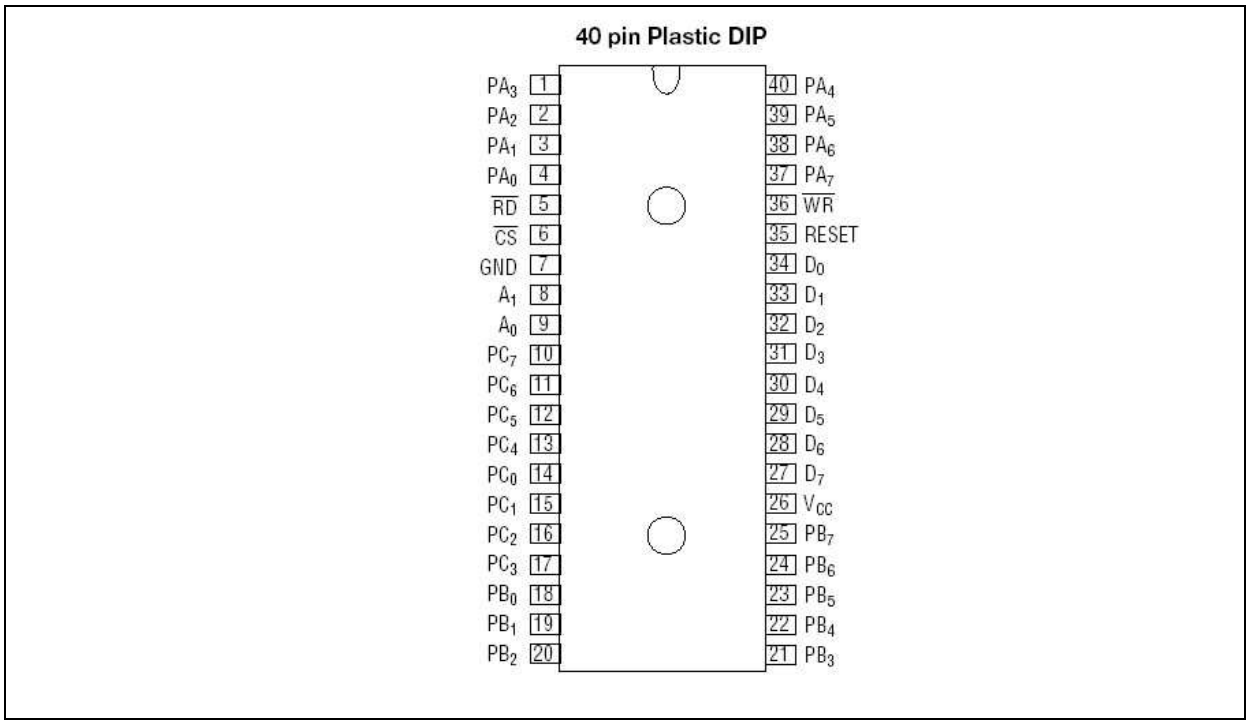
Tabela 9: Valores de endereçamento possível com o SN74LS138.

Endereço Hexadecimal	**Endereço Binário**	**Pino Afetado**
200H	1000000000	15
220H	1000100000	14
240H	1001000000	13
260H	1001100000	12
280H	1010000000	11
2A0H	1010100000	10
2C0H	1011000000	9
2E0H	1011100000	7

3.2.2 O 82C55A

O circuito integrado 82C55A é um dispositivo de entrada e saída, programável, de uso geral para uso em periféricos com até 8Mhz de velocidade. Este circuito possui três portas de 8 bits para escrita e leitura, funcionando nos modos 0, 1 e 2 de operação. No modo 0, todas as portas funcionam basicamente com leitura e escrita. Já no modo 1 e 2, as três portas se transformam em grupo A e grupo B. O grupo A é formado pela porta A e pelos bits altos da porta C. O grupo B é formado pela porta B e pelos 3 primeiros bits da porta C. Estes bits da porta C, que cada grupo utiliza, faz o "*handshake*", ou seja, faz o controle do sinal de escrita e

leitura e interrupções. Podemos observar o lay-out da pinagem deste componente na Figura 11.



Fonte: Oki (1998).

Figura 11: Pinagem do 82C55A.

Pode-se utilizar, por exemplo, a porta B para o protótipo. Logo, teremos que utilizar os sinais 0, 1 e 2 da porta C para controle da porta B. Usaremos, então, no modo de leitura o sinal 2 da porta C para o STB (*strobe input*), em baixa este sinal de entrada faz com que o 82C55A carregue o conteúdo da porta B para o *latch* de entrada e após concluído manda um Vcc para o sinal 1 da porta C que chamamos de IBF (buffer de entrada foi carregado) e em seguida o sinal 0 da porta C interrompe a CPU para executar uma rotina de tratamento de interrupção. Será utilizado dois 82C55A para elaborar nosso protótipo. Um para a parte baixa dos dados e outro para a parte alta. Para estabilização, foi feita uma "ponte" do Vcc (no pino 26) ao sinal GND com um capacitor cerâmico nº 104. Na Tabela 10 pode ser visto a associação da pinagem do 82C55A com a pinagem do barramento ISA na parte baixa dos dados.

Tabela 10: Conexão do 82C55A com a parte baixa dos dados do barramento ISA.

Pino do 82C55A	**Pino do barramento ISA**	**Função**
01	não utilizado	bit 3 de transmissão de dados para a outra placa.
02	não utilizado	bit 2 de transmissão de dados para a outra placa.
03	não utilizado	bit 1 de transmissão de dados para a outra placa.
04	não utilizado	bit 0 de transmissão de dados para a outra placa.
05	B14	RD (Sinal de leitura no barramento).
06	não utilizado	sinal CS vindo do *switch* após o SN74LS138.
07	B1	sinal terra.
08	A1	sinal de endereço do barramento ISA.
09	A0	sinal de endereço do barramento ISA.
10	não utilizado	
11	não utilizado	
12	não utilizado	
13	não utilizado	
14	não utilizado	saída do sinal que ativará a interrupção na outra placa.
15	não utilizado	
16	não utilizado	
17	não utilizado	
18	não utilizado	bit 0 de transmissão de status para a outra placa.
19	não utilizado	bit 1 de transmissão de status para a outra placa.
20	não utilizado	bit 2 de transmissão de status para a outra placa.
21	não utilizado	bit 3 de transmissão de status para a outra placa.
22	não utilizado	bit 4 de transmissão de status para a outra placa.
23	não utilizado	bit 5 de transmissão de status para a outra placa.
24	não utilizado	bit 6 de transmissão de status para a outra placa.
25	não utilizado	bit 7 de transmissão de status para a outra placa.
26	B3	sinal de 5 volts do barramento.
27	A2	bit 7 da parte baixa do barramento de dados.
28	A3	bit 6 da parte baixa do barramento de dados.
29	A4	bit 5 da parte baixa do barramento de dados.
30	A5	bit 4 da parte baixa do barramento de dados.
31	A6	bit 3 da parte baixa do barramento de dados.
32	A7	bit 2 da parte baixa do barramento de dados.
33	A8	bit 1 da parte baixa do barramento de dados.
34	A9	bit 0 da parte baixa do barramento de dados.
35	B2	*Reset*
36	B13	WR (sinal de escrita no barramento)
37	não utilizado	bit 7 de transmissão de dados para a outra placa.
38	não utilizado	bit 6 de transmissão de dados para a outra placa.
39	não utilizado	bit 5 de transmissão de dados para a outra placa.
40	não utilizado	bit 4 de transmissão de dados para a outra placa.

Na Tabela 11 demonstra a associação da pinagem do 82C55A com a pinagem do barramento ISA na parte alta dos dados.

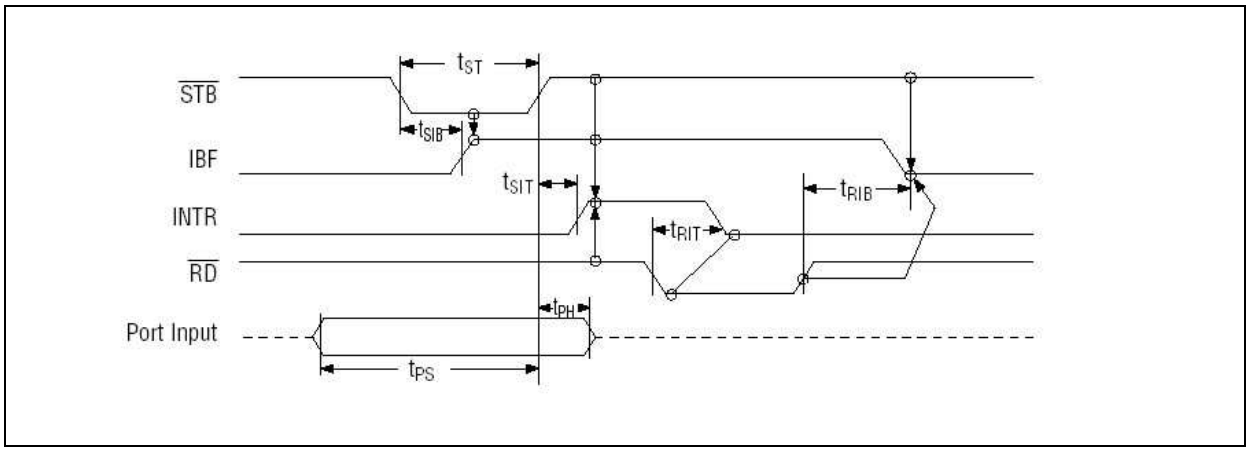
Tabela 11: Conexão do 82C55A com a parte alta dos dados do barramento ISA.

Pino do 82C55A	**Pino do barramento ISA**	**Função**
01	não utilizado	bit 11 de transmissão de dados para a outra placa.
02	não utilizado	bit 10 de transmissão de dados para a outra placa.
03	não utilizado	bit 9 de transmissão de dados para a outra placa.
04	não utilizado	bit 8 de transmissão de dados para a outra placa.
05	B14	RD (Sinal de leitura no barramento).
06	não utilizado	sinal CS vindo do *switch* após o SN74LS138.
07	D18	sinal terra.
08	A1	sinal de endereço do barramento ISA.
09	A0	sinal de endereço do barramento ISA.
10	não utilizado	
11	não utilizado	
12	não utilizado	
13	não utilizado	
14	não utilizado	Saída do sinal que ativará a interrupção na outra placa.
15	não utilizado	
16	não utilizado	
17	não utilizado	
18	não utilizado	bit 8 de transmissão de status para a outra placa.
19	não utilizado	bit 9 de transmissão de status para a outra placa.
20	não utilizado	bit 10 de transmissão de status para a outra placa.
21	não utilizado	bit 11 de transmissão de status para a outra placa.
22	não utilizado	bit 12 de transmissão de status para a outra placa.
23	não utilizado	bit 13 de transmissão de status para a outra placa.
24	não utilizado	bit 14 de transmissão de status para a outra placa.
25	não utilizado	bit 15 de transmissão de status para a outra placa.
26	D16	sinal de 5 volts do barramento.
27	C18	bit 15 da parte baixa do barramento de dados.
28	C17	bit 14 da parte baixa do barramento de dados.
29	C16	bit 13 da parte baixa do barramento de dados.
30	C15	bit 12 da parte baixa do barramento de dados.
31	C14	bit 11 da parte baixa do barramento de dados.
32	C13	bit 10 da parte baixa do barramento de dados.
33	C12	bit 9 da parte baixa do barramento de dados.
34	C11	bit 8 da parte baixa do barramento de dados.
35	B2	*Reset*
36	B13	WR (sinal de escrita no barramento)
37	não utilizado	bit 15 de transmissão de dados para a outra placa.
38	não utilizado	bit 14 de transmissão de dados para a outra placa.
39	não utilizado	bit 13 de transmissão de dados para a outra placa.
40	não utilizado	bit 12 de transmissão de dados para a outra placa.

Os modos de operação do 82C55A resumem-se em 3 modos distintos:

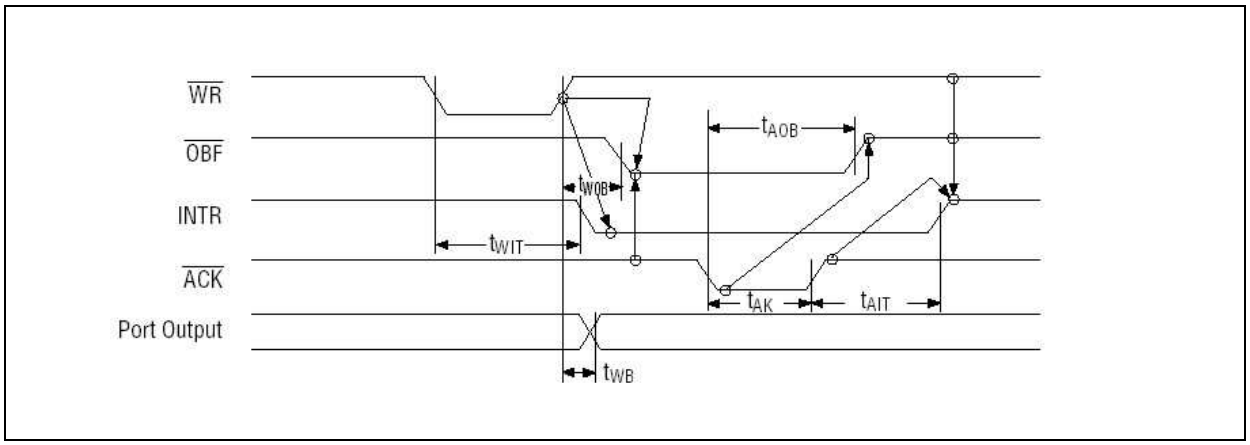
- Modo 0 = Operações básicas de entrada e saída utilizando as três portas (A, B e C) do circuito integrado, sem sinais de controle.
- Modo 1 = A porta A e a porta B com sinal de controle (*strobe*) para entrada e saída (porém não bidirecional). A porta C fica dividida em duas; sendo a metade para controle da porta A e a outra metade para controle da porta B.
- Modo 2 = A porta A fica bidirecional e parte da porta C fica como controle da A.

Para elaboração deste trabalho, foi escolhido o modo 1 de operação do circuito integrado 82C55A. As Figuras 12 e 13 mostram o fluxo elétrico de entrada e saída. Foi utilizado o pino 10 para indicar ao receptor que já contém informação carregada na porta de saída do transmissor. Este sinal alimenta o pino 13 do receptor, ordenando o 82C55A a colocar no buffer o conteúdo da porta de entrada.



Fonte: Oki (1998).

Figura 12: Modo 1 – operação de entrada



Fonte: Oki (1998).

Figura 13: Modo 1 – operação de saída

3.2.3 O SN74LS32

Esse circuito integrado é do tipo OR. Este integrado será utilizado para unificar os sinais de interrupção dos dois 82C55A provenientes da outra placa. Ele garantirá que ocorrerá interrupção com valores escritos no 82C55A da parte alta ou no 82C55A da parte baixa. Utilizaremos o pino 1 alimentado pelo pino 14 do 82C55A da parte alta e o pino 2 com o pino 14 do 82C55A da parte baixa. O sinal de saída no pino 3 irá provocar a interrupção no outra placa.

3.3 IMPLEMENTAÇÃO

3.3.1 Projeto de Hardware

Com base no desenho esquemático da Figura 10, foi desenvolvido o *lay-out* dos componentes da placa. Podemos observar detalhes na Figura 14.

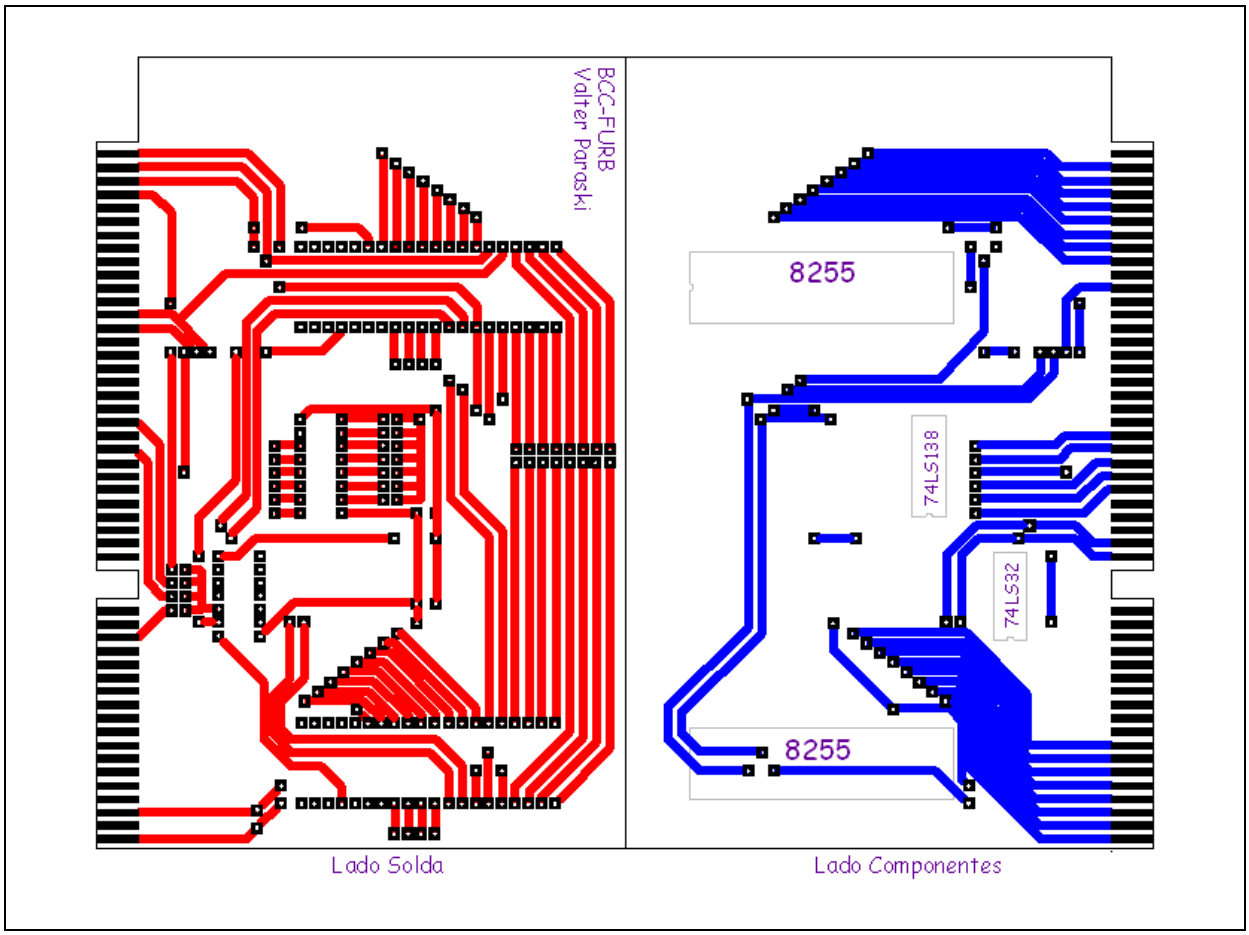


Figura 14: Desenho do *Lay-out* da placa de circuito impresso.

O passo seguinte foi estampar numa placa de circuito impresso o conteúdo do desenho, corroer e efetuar as perfurações. E, para em seguida, e solda dos componentes. O resultado pode ser visto na Figura 15.

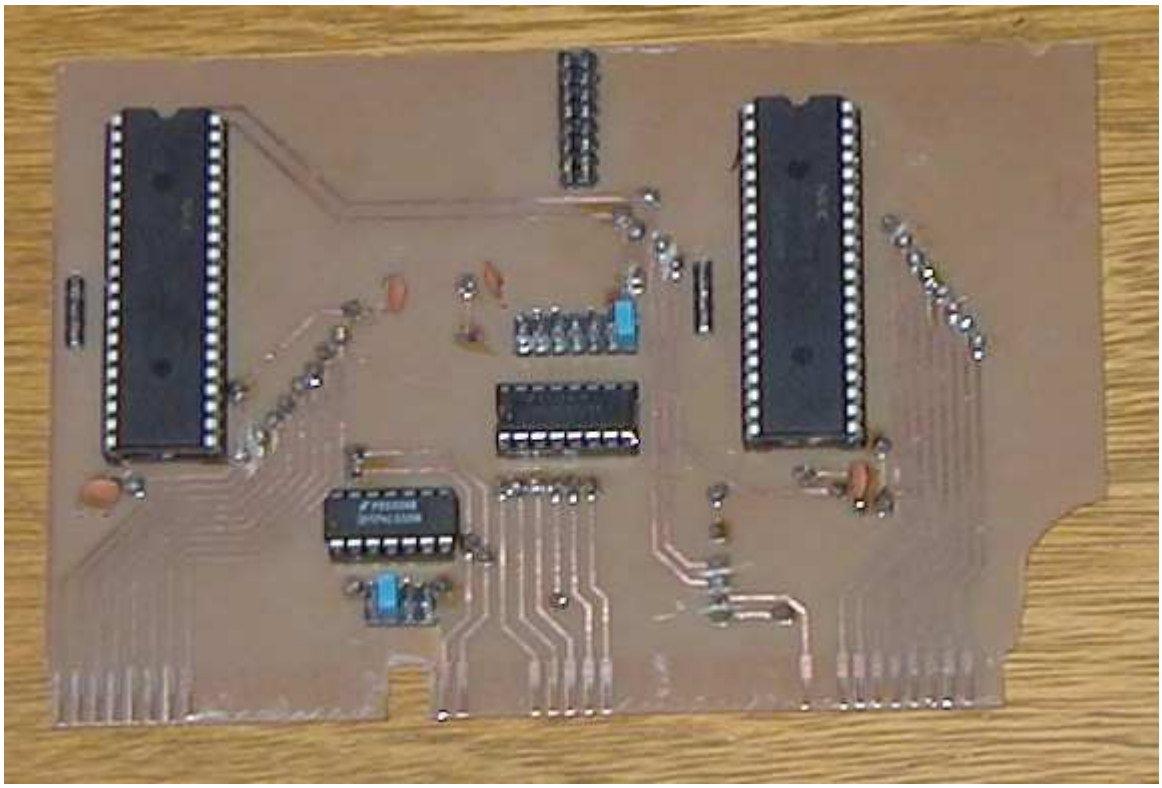


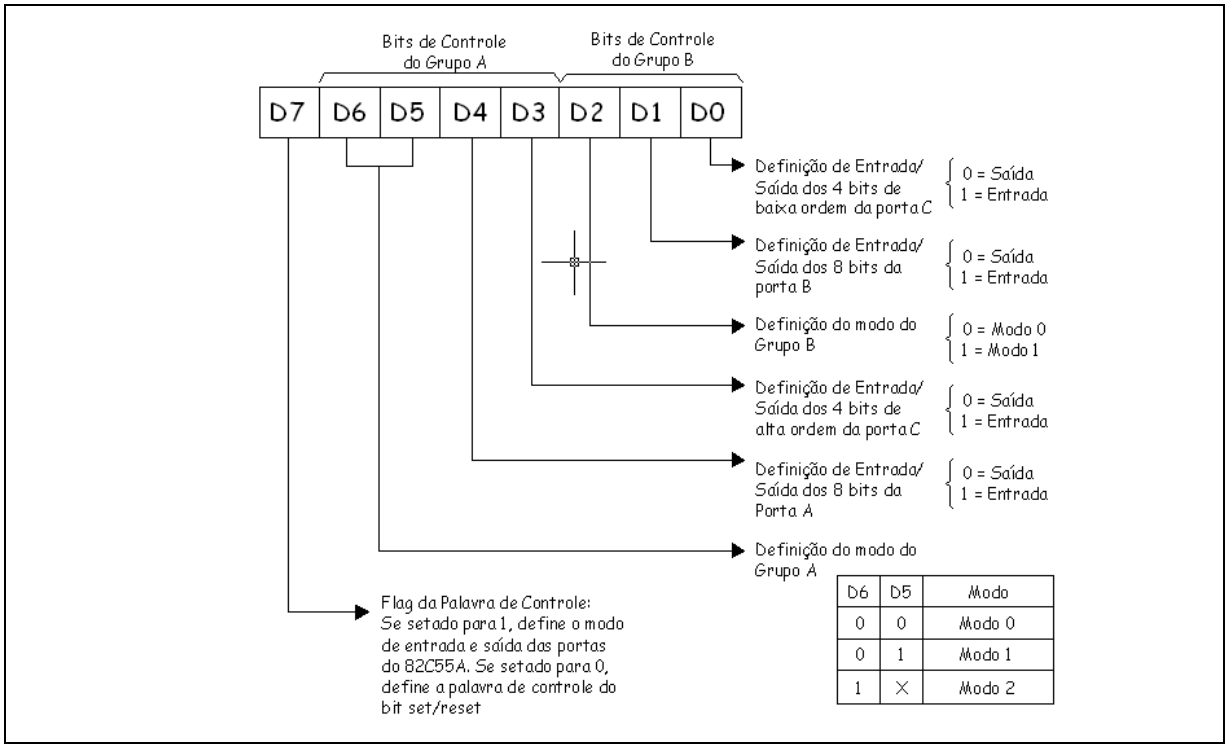
Figura 15: Fotografia da placa pronta.

3.3.2 Projeto de Software

Para demonstrar aplicabilidade deste projeto de hardware foi criado um programa de computador que fará a troca de informações entre as duas placas controladoras. Este programa varre o diretório atual e permite que o usuário escolha um arquivo entre outros para ser transferido. O programa lê o arquivo em blocos e coloca-o num *buffer* (fila) de pacotes no receptor. Estando no *buffer*, o usuário poderá optar por gravar ou não o arquivo em sua unidade de armazenamento.

No software, necessitamos configurar o 82C55A, utilizando uma palavra de controle, para que o mesmo trabalhe da forma mais adequada às necessidades de seu emprego.

Podemos observar na Figura 16, a forma de definição da palavra de controle. Neste caso, foi adotado como modo de saída o modo 1. Logo, a palavra de controle foi definida desta forma: D7=1, para entrar no modo de configuração do 82C55A. D6=0 e D5=1 para definir o modo 1 de operação do grupo A. D4=0 para definir a porta A como sendo de saída e o D3=0 para definição dos bits de alta ordem da porta C como sendo de saída também. O D2=1, D1=0 e D0=0, apenas para configurar, pois a porta B não terá utilidade no nosso projeto. Desta forma a palavra de controle ficou: 10100100. Adotaremos então, o A4 que é a palavra de controle convertida para o hexadecimal.

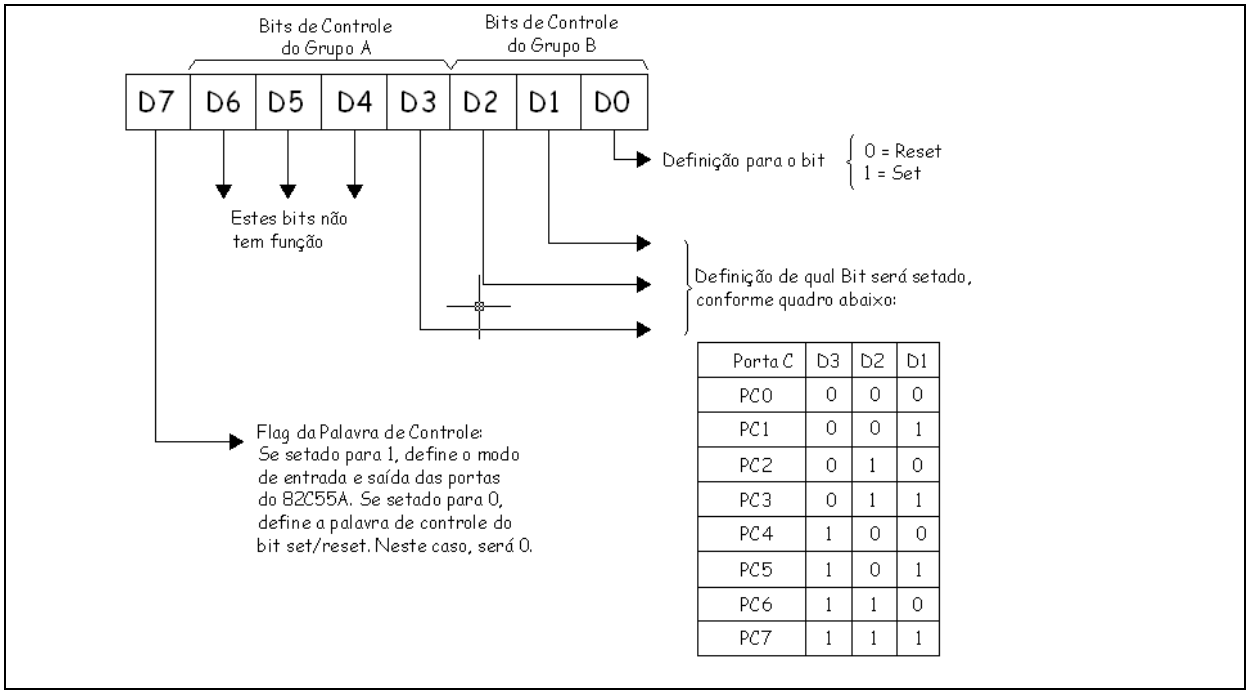


Fonte: Oki (1998).

Figura 16: Configuração do 82C55A.

Para o modo de recepção, ou entrada, foi adotado o modo 1. Logo, a palavra de controle foi definida desta forma: D7=1, para entrar no modo de configuração do 82C55A. D6=0 e D5=1 para definir o modo 1 de operação do grupo A. D4=1 para definir a porta A como sendo de entrada e o D3=0 para definição dos bits de alta ordem da porta C como sendo de saída também. O D2=1, D1=1 e D0=0, apenas para configurar e poderão receber qualquer valor binário, pois a porta B não terá utilidade no nosso projeto. Desta forma a palavra de controle ficou: 10110110, ou seja, B6 em hexadecimal.

Estando o 82C55A configurado, teremos que setar o sinal INT A, que é um registrador interno do 88C55A, representado pelo bit 6 (PC6) da porta C. A Figura 17 demonstra como pode ser setado este bit.



Porta C	D3	D2	D1
PC0	0	0	0
PC1	0	0	1
PC2	0	1	0
PC3	0	1	1
PC4	1	0	0
PC5	1	0	1
PC6	1	1	0
PC7	1	1	1

Fonte: Oki (1998).

Figura 17: Configuração do Bit set/reset da porta C.

Para ativarmos a interrupção quando chega alguma informação na porta A teremos que setar o bit 4 da porta C. Para isto utilizaremos o seguinte byte em binário: 1001, sendo que o "100" refere-se ao PC4 (bit 4) da porta C que iremos setar, conforme pode ser visto na Figura 17 e o outro bit é o comando set da porta PC4.

Agora, tendo definida a configuração do 82C55A, estas palavras de controle serão utilizadas no programa para fazer a comunicação entre os dois microcomputadores.

O fluxo das informações de um microcomputador para outro bem como os sinais de controle envolvidos podem ser visto na Figura 18.

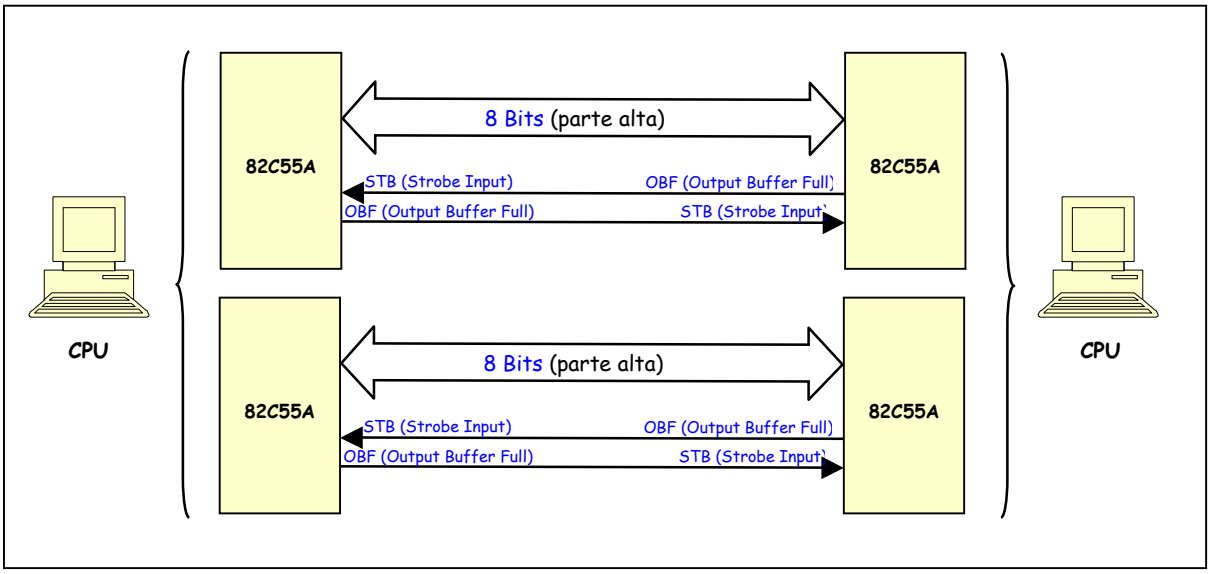


Figura 18: Forma de comunicação entre as controladoras

A Figuras 19 mostra o fluxograma das rotinas de transmissão do *buffer* de saída dos dados para o receptor. A Figura 20 mostra o fluxograma das rotinas de recepção dos dados no *buffer* pelo computador receptor.

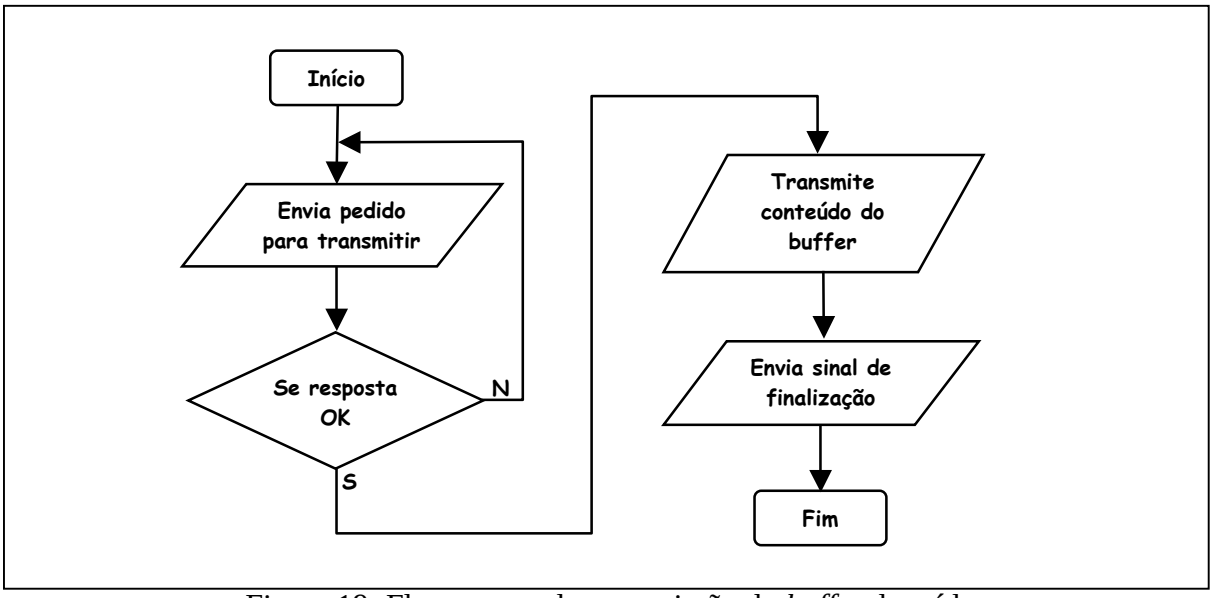


Figura 19: Fluxograma de transmissão do *buffer* de saída

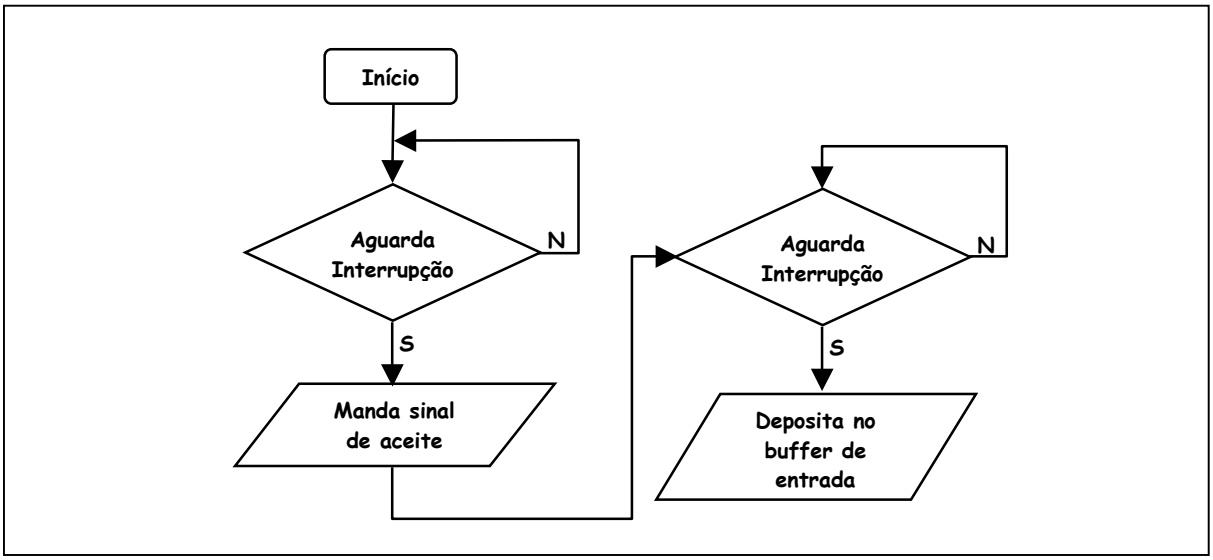


Figura 20: Fluxograma de recepção no *buffer* de entrada

Para os testes iniciais, foi criada uma rotina que lê e mostra o valor da porta 2C0H quando ocorre uma interrupção. Neste programa, os computadores foram configurados para o modo de leitura da porta A através do comando port[$2C3]=$B6. Caso o usuário desejasse transmitir alguma informação (no caso o valor de um contador), teria que pressionar o 1 do menu. Neste caso, a configuração do 82C55A mudaria para o modo de saída com o comando port[$2C3]=$A4, seguido pela gravação na porta de saída do valor de um contador. Em seguida, retorna a configuração para o modo de leitura a porta A. No Quadro 1, encontra-se o programa elaborado para efetuar os teste iniciais com as portas e com as interrupções.

```
{$M 2048, 0, 8000}
program interrupcao;
uses
    crt, dos;

const EIO : byte=$20;

var
   antiga         : procedure; {rotina de tratamento da interrupção antiga}
   a              : char;
   t              : byte;

{$F+}
procedure comunicacao; interrupt; {rotina para tratamento da interrupção}
begin
   writeln('2c0: ',port[$2c0]); {lê e mostra o conteúdo da porta A}
   port[$20]:=$20;              {finaliza a interrupção}
end;
{$F-}

begin
   clrscr;
   t:=0;
   PORT[$21]:=port[$21] and $DF; {ativa a int 5 do controlador de interrup}
   getintvec(13, addr(antiga)); {salva o endereço da rotina antiga}
```

```
   setintvec(13, addr(comunicacao)); {atribui o endereco da rotina nova}
   port[$2c3]:=$b6; port[$2c3]:=$9; {configura o 82C55A como leitura}
   while not keypressed do
   begin
     writeln('1 – Transmitir / 2 – Sair: ');
     readln(a);
     if a='1' then
     begin
       port[$2c3]:=$a4; {configura o 82C55A como escrita}
       inc(t);
       port[$2c0]:=t; {escreve o valor na porta A de saída}
       writeln(t,' - ',port[$2c0]); {mostra o conteúdo gravado na porta}
       port[$2c3]:=$b6; port[$2c3]:=$9; {configura o 82C55A como leitura}
     end;
     if a='2' then exit; {finaliza o programa}
   end;
end.
```

QUADRO 1 – Programa para teste das portas.

No Quadro 2, consta a listagem do programa principal cujo objetivo é transferir arquivos para o microcomputador receptor e vice-versa. Para que essa transferência ocorra, será necessário utilizar caracteres de controle para que a comunicação seja compreendida por transmissor e receptor. A Tabela 12 montra os caracteres de controle utilizados.

Tabela 12: Caracteres de controle utilizados para a comunicação dos programas

Caracter (código ASCII)	**Funcionalidade**
*	Pedido para início da transmissão de um arquivo
&	Início da transmissão de um arquivo aceito
%	Tamanho do arquivo a ser transferido
!	Caracter que indica o final da transmissão do arquivo
?	Caracter que indica erro e pede para retransmitir

```
{$M 2048, 0, 8000}
{$G+}
program TCC;
uses
    crt, dos, auxilio;

const EIO : byte=$20;

var
   antiga  : procedure;
   c,t     : byte;
   a       : char;
   fila    : text;
   i       : integer;
   s       : string;


procedure comunicacao; interrupt;
begin
   c:=port[$2c0];
   if (grava) then writeln(fila,port[$2c0]);
```

```
   if (c=ord('*')) and (inicio=false) then
   begin
      assign(fila,'valter.tmp');
      grava:=true;
      rewrite(fila);
      port[$2c0]:=$A4;
      port[$2c0]:=ord('&');
      port[$2c0]:=$b6; port[$2c0]:=$9;
   end;
   if (c=ord('&')) and (inicio=false) then
   begin
      inicio:=true;
   end;
   port[$20]:=EIO;
end;

begin
   clrscr;
   i:=0; s:='';
   port[$21]:=port[$21] and $DF;
   getintvec(13, addr(antiga));
   setintvec(13, addr(comunicacao));
   port[$2c3]:=$b6; port[$2c3]:=$9;
   buffer:=false;
   grava:=false;
   inicio:=false;
   t:=0;l:=5;max:=0;
   le_arvore;
   tela;
   pos:=5;iarv:=1;
   t:=1;
   while true do
   begin
      a:=readkey;
      if ord(a)=72 then {para cima}
      begin
         if (pos=5) and (iarv>1) then dec(iarv)
         else
            if pos>5 then dec(pos);
         atualiza(pos,iarv);
      end;
      if ord(a)=80 then {para baixo}
      begin
         if (pos=22) and (iarv<22) then inc(iarv)
         else
            if pos<22 then inc(pos);
         atualiza(pos,iarv);
      end;
      if a='1' then grava_buffer(nome);
      if a='2' then
      begin
         Textcolor(white); TextBackGround(black);
         clrscr;
         close(fila);
         le_buffer;
         tela;
         le_arvore;
         atualiza(pos,iarv);
      end;
      if a='3' then
```

```
         halt;
      if a='5' then {Sair}
      begin
         halt;
      end;
      if buffer then
      begin
         manda_buffer;
      end;
   end;
end.
```

QUADRO 2 – Programa Principal

```
unit auxilio;

interface

uses
    crt, dos;

var
   nomearq    : array [1..2000] of string[12];
   tamarq     : array [1..2000] of longint;
   c,t        : byte;
   l,max,z,k  : integer;
   nome,
   caminho    : string;
   pos,iarv,i : integer;
   r          : string;
   buffer,
   inicio,
   grava      : boolean;
   arq        : file;
   fila1      : text;
   procedure atualiza(pos1,iarv1:integer);
   procedure tela;
   procedure le_arvore;
   procedure le_arquivo(nomearq: string);
   procedure le_buffer;
   procedure grava_buffer(nomearq: string);
   procedure manda_buffer;

Implementation

procedure atualiza(pos1,iarv1:integer);
var
   l,m    : integer;
   linha  : string;

begin
   l:=5;
   for t:=iarv1 to iarv1+17 do
   begin
      Textcolor(white); TextBackGround(black);
      GotoXY(10,l);write('                                        ');
      if l=pos1 then
      begin
         Textcolor(red); TextBackGround(yellow);
```

```
         nome:=nomearq[t];
      end;
      linha:=nomearq[t];
      if length(linha)<20 then
         for m:=length(linha) to 20 do linha:=linha+' ';
      linha:=linha+'-';
      GotoXY(10,l);write(linha);
      GotoXY(30,l);write('-         bytes');
      Str(tamarq[t],r);
      z:=length(r);
      GotoXY(39-z,l);write(r);
      inc(l);
      Textcolor(blue); TextBackGround(blue);
   end;
end;


procedure tela;
var
   x,y: integer;

begin
   for y:=0 to 27 do
   begin
      Textcolor(blue); TextBackGround(blue);
      gotoxy(0,y);writeln('
');
   end;
   Textcolor(white); TextBackGround(black);
   gotoxy(8,4);write('ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»');
   gotoxy(8,5);write('º                                     º');
   gotoxy(8,6);write('º                                     º');
   gotoxy(8,7);write('º                                     º');
   gotoxy(8,8);write('º                                     º');
   gotoxy(8,9);write('º                                     º');
   gotoxy(8,10);write('º                                     º');
   gotoxy(8,11);write('º                                     º');
   gotoxy(8,12);write('º                                     º');
   gotoxy(8,13);write('º                                     º');
   gotoxy(8,14);write('º                                     º');
   gotoxy(8,15);write('º                                     º');
   gotoxy(8,16);write('º                                     º');
   gotoxy(8,17);write('º                                     º');
   gotoxy(8,18);write('º                                     º');
   gotoxy(8,19);write('º                                     º');
   gotoxy(8,20);write('º                                     º');
   gotoxy(8,21);write('º                                     º');
   gotoxy(8,22);write('º                                     º');
   gotoxy(8,23);write('ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼');
   Textcolor(lightred); TextBackGround(red);
   gotoxy(1,1);write('-=-=-=-=-=-=-=-=-=-=-==- TRANSFERENCIA DE ARQUIVOS
-=-=-=-=-=-=-=-=-=-=-==-=');
   r:=caminho;
   z:=length(r);
   if z<26 then z:=26-z;
   for y:=1 to z do r:=r+' ';
   gotoxy(8,3);write(' Local ----> '+r);
   TextColor(black); TextBackGround(cyan);
   gotoxy(50,8);write('                     ');
   gotoxy(50,9);write(' 1 - Enviar Arquivo  ');
```

```
   gotoxy(50,10);write('                       ');
   gotoxy(50,11);write(' 2 - Gravar Buffer   ');
   gotoxy(50,12);write('                       ');
   gotoxy(50,13);write(' 3 - Apagar Arquivo  ');
   gotoxy(50,14);write('                       ');
   gotoxy(50,15);write(' 4 - copiar Arquivo  ');
   gotoxy(50,16);write('                       ');
   gotoxy(50,17);write(' 5 - Sair            ');
   gotoxy(50,18);write('                       ');
   atualiza(pos,iarv);
   TextColor(red); TextBackGround(white);
   gotoxy(1,25);write(' Academico: Valter Paraski      -      Orientador:
Prof§ Antonio Carlos Tavares');
end;

procedure le_arvore;
var
   DirInfo: SearchRec;

begin
   GetDir(0,caminho); { 0 = Current drive }
   caminho:=caminho+'\*.*';
   FindFirst(caminho, Archive, DirInfo); { mesmo que DIR *.PAS }
   Textcolor(white); TextBackGround(black);
   max:=1;
   while DosError = 0 do
   begin
      nomearq[max]:=DirInfo.Name;
      tamarq[max]:=(DirInfo.Size);
      FindNext(DirInfo);
      inc(max);
   end;
end;

procedure le_arquivo(nomearq: string);
var
   buf : byte;
   lidos: word;
begin
   assign(fila1,'temp.dat');
   rewrite(fila1);
   assign(arq,nome);
   reset(arq,1);
   port[$2c3]:=$a4;
   for i:=1 to 3 do
   begin
      port[$2c0]:=ord('~');
      delay(5);
      writeln(fila1,ord('~'));
   end;
   for i:=1 to length(nome) do
   begin
      port[$2c0]:=ord(nome[i]);
      delay(5);
      writeln(fila1,ord(nome[i]));
   end;
   for i:=1 to 3 do
   begin
      port[$2c0]:=ord('^');
      delay(5);
```

```
      writeln(fila1,ord('^'));
   end;
   REPEAT
      blockread(arq,buf,2,lidos);
      writeln(fila1,buf);
      port[$2c0]:=buf;
      delay(5);
   until (lidos=0) {or (gravados<>lidos)};
   for i:=1 to 3 do
   begin
      port[$2c0]:=ord('+');
      delay(5);
      writeln(fila1,ord('+'));
   end;
   port[$2c3]:=$b6; port[$2c3]:=$9;
   close(arq);
   close(fila1);
end;

procedure grava_buffer(nomearq: string);
var
   buf : byte;
   lidos: word;
   gravados: word;
begin
   assign(fila1,'temp.dat');
   rewrite(fila1);
   assign(arq,nome);
   reset(arq,1);

   writeln(fila1,ord('~'));
   for i:=1 to length(nome) do  writeln(fila1,ord(nome[i]));
   writeln(fila1,ord('^'));
   lidos:=0;
   gravados:=sizeof(arq);
   REPEAT
      blockread(arq,buf,1,lidos);
      writeln(fila1,buf);
   until (lidos=0) {or (gravados<>lidos)};
   writeln(fila1,ord('$'));
   buffer:=true;
   close(arq);
   close(fila1);
end;

procedure manda_buffer;
var
   buf : byte;
   lidos: word;
begin
   assign(fila1,'temp.dat');
   reset(fila1);
   i:=0;
   port[$2c3]:=$a4;
   port[$2c0]:=ord('*');
   port[$2c3]:=$b6; port[$2c3]:=$9;
   repeat
      inc(i);
   until (inicio) or (i>1000);
   if (i<1000) and (inicio) then
```

```
   begin
      port[$2c3]:=$a4;
      while not eof(fila1) do
      begin
         readln(fila1,buf);
         port[$2c0]:=buf;
         delay(5);
      end;
      port[$2c3]:=$b6; port[$2c3]:=$9;
   end;
   close(fila1);
end;

procedure le_buffer;
var
   palavra: byte;
   ff: char;
   nomea: string;
   cabeca, corpo: integer;
   vnome, varq: boolean;
   lidos, gravados : word;

begin
   assign(fila1,'valter.tmp');
   reset(fila1);
   cabeca:=0;corpo:=0;
   vnome:=false;varq:=false;
   readln(fila1,palavra);
   if (palavra=126) then
   begin
      readln(fila1,palavra);
      if (palavra=126) then
      begin
         readln(fila1,palavra);
         if (palavra=126) then vnome:=true
         else
         begin
            gotoxy(22,50); write('buffer invalido!!!!');
         end;
      end;
   end;
   if vnome then
      while not eof(fila1) do
      begin
         readln(fila1,palavra);
         if (varq=false) and (palavra=94) then
         begin
            readln(fila1,palavra);
            if palavra=94 then
            begin
               readln(fila1,palavra);
               if palavra=94 then
               begin
                  varq:=true;
                  assign(arq,nomea);
                  rewrite(arq);
               end;
            end;
         end else
         begin
```

```
            nomea:=nomea+chr(palavra);
            writeln(nomea);
         end;
         if varq then
         begin
            readln(fila1,palavra);
            if palavra=43 then
            begin
               readln(fila1,palavra);
               if palavra<>43 then
                  blockwrite(arq,palavra,lidos,gravados);
            end;
         end;
      end;
  end;
end.
```

QUADRO 3 – Unidade do programa principal.

A Figura 19 apresenta a tela principal da aplicação desenvolvida.

Figura 19: Tela principal do programa.

3.4 RESULTADOS E DISCUSSÃO

Para verificar os resultados, foi feita a transmissão de um valor inteiro, incrementado até 10.000. Este valor do *buffer* de entrada no micro receptor foi salvo num arquivo tipo texto e conferido com uma variável que foi incrementada até 10.000 e conferido com cada valor lido na porta. O resultado foi que não houve ocorrência de valores alterados.

4 CONCLUSÕES

Este trabalho alcançou seu objetivo principal de estudar o barramento ISA, as interrupções por hardware, e a criação de duas placas controladoras para efetuar a troca de informações entre dois microcomputadores.

Através dos resultados obtidos ficou demonstrado que ainda é possível desenvolver soluções de hardware para barramento ISA, pelo seu baixo custo. Uma indicação para seu uso é na automação de processos na indústria.

4.1 DIFICULDADES ENCONTRADAS

As dificuldades encontradas foram no sentido de confeccionar as placas controladoras. O protótipo foi confeccionado artesanalmente, devido à ausência de equipamentos adequados para esse fim no laboratório da Furb. Isto trouxe grandes atrasos no desenvolvimento do software, por estar trabalhando na solução dos problemas de mau-contato e mau funcionamento do protótipo. Por esse motivo, o protótipo teve de ser simplificado o máximo possível para reduzir as trilhas e ilhas traçadas em sua superfície.

4.2 EXTENSÕES

Os trabalhos futuros seguindo como base este, poderão ser feitos utilizando as técnicas de DMA, e também criar em hardware um *latch* para representar um sinal enviado do receptor para o transmissor informando que o dado já foi lido na porta de entrada. Isto aumentaria significativamente a velocidade de transmissão no protótipo.

Um outro caminho a ser seguido é a adaptação deste trabalho para o barramento PCI ou *firewire*, por serem um tipo de barramento presente na maioria dos microcomputadores novos, e que possuem uma taxa de transferência significativamente maior que a do barramento ISA.

REFERÊNCIAS BIBLIOGRÁFICAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 6023: informação e documentação: referências - elaboração. Rio de Janeiro, 2002a.

BAKKER, Jacobus Willem. Languages for parallel architectures: design, semantics, implementation models. Chichester: John Wiley, 1989. xv, 273 p.

CANTU, Marco. **Dominando o Delphi 6:** a bíblia. São Paulo: Makron Books, 2002. 934 p.

CIPELLI, Antonio Marco Vicari et al. **Teoria e desenvolvimento de projetos de circuitos eletrônicos.** 18. ed. Sao Paulo: Érica, 2001. 445 p.

DASGUPTA, Subrata. **Computer architecture:** a modern synthesis. New York: J. Wiley, 1989. 2 v.

DEMARCO, Tom. **Análise estruturada e especificação de sistema.** Rio de Janeiro: Campus, 1989. 333 p.

KARBO, Michael B. **A complete illustrated guide to the pc hardware**. Denmark: Mkdata, 1998.

OGREN, Joakim. **The hardware book v1.3.** The hardware book team, [S.L.], 2001. Disponível em: <http://www.hwb.acc.umu.se/>. Acesso em: 07 ago. 2003.

PELISSON, Luiz Augusto. **Fundamentos da informática,** São Paulo, maio 2001. Disponível em: <http://www.dainf.cefetpr.br/~pelisson>. Acesso em: 07 ago. 2003.

SANTOS, Cleverson dos. **Construção de um protótipo de interface para microcomputador tipo PC para interligar duas placas mãe.** 2002. 97 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SCHULER, João Paulo S. **Tutorial de Delphi**, Porto Alegre, dez. 1999. Disponível em: <http://www.schulers.com/jpss/pascal/dtut/>. Acesso em: 07 ago. 2003.

TORRES, Gabriel. **Hardware curso completo.** 4. ed. Rio de Janeiro: Axcel Books do Brasil, 1998. 893 p.

WEBER, Raul Fernando. **Arquitetura de computadores pessoais.** 2. ed. Porto Alegre: Instituto de Informática da UFRGS, Sagra Luzzatto, 2000. 267 p.

WIRTH, Niklaus. **Digital circuit design for computer science students:** an introductory textbook. Berlin: Springer, 1995. xiii, 204 p.