REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

CONSTRUÇÃO DE UM PROTÓTIPO DE INTERFACE PARA MICROCOMPUTADOR TIPO PC PARA INTERLIGAR DUAS PLACAS MÃE

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

CLEVERSON DOS SANTOS

BLUMENAU, JUNHO/2002.

2002/1-17

CONSTRUÇÃO DE UM PROTÓTIPO DE INTERFACE PARA MICROCOMPUTADOR TIPO PC PARA INTERLIGAR DUAS PLACAS MÃE

CLEVERSON DOS SANTOS

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

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Antonio Carlos Tavares — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Antonio Carlos Tavares

_____

Prof. Miguel Alexandre Wisintainer

_____

Prof. Francisco Adell Péricas

DEDICATÓRIA

A meu pai João pelo amor, carinho, apoio
e incentivos recebidos ao longo destes anos.

AGRADECIMENTOS

Aos meus pais João e Zeli que me ensinaram a viver, apoiaram e confiaram na minha capacidade.

À minha esposa Ana Paula e minhas filhas Ana Caroline e Ana Beatriz por todo amor, confiança e compreensão em mim depositado.

Ao meu professor e orientador Antonio Carlos Tavares, pela sua dedicação, orientação, e motivação na elaboração deste trabalho.

A todos aqueles que direta ou indiretamente colaboraram para a realização deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

RESUMO

Este trabalho visa a criação de uma interface de comunicação através da construção de um protótipo de uma placa de comunicação, utilizando o barramento ISA de 8 btis, para interligar dois microcomputadores visando a construção de uma máquina paralela. Foram criados dois programas do tipo mestre e escravo onde estes ficam trocando informações entre si a fim de demonstrar como funciona a comunicação entre as placas. Também é discutido o resultado obtido quanto à taxa de comunicação conseguida com esta implementação.

ABSTRACT

This presents the creation of a communication interface through the implementation of a prototype of a communication board that uses 8 bits ISA bus. This board prototype intends to provide the interaction between two computers in order to model a parallel machine. Two software programs ware created a master and slave, to exchange data demonstrating the communication process between the two boards. Finally, it is presented a discussion about the results, mainly about the data transfer rates obtained by the prototype.

1 INTRODUÇÃO

A computação evolui em movimentos pendulares, da mais completa centralização a total descentralização. E fazendo o caminho de volta quando os extremos são alcançados. No início dos anos 80 viu-se o *downsizing* e a descentralização da informática, com a proliferação de redes locais e servidores.

A proliferação dos microcomputadores estabeleceu novos conceitos relacionados ao perfil e exigências dos usuários de computadores. Os mais diversos ambientes profissionais, como também o ambiente denominado de doméstico, têm exigido recursos que permitem a conexão dos microcomputadores, visando o compartilhamento de serviços e informações (Castro Junior, 1999).

Esta proliferação se dá diante do alto avanço de tecnologia dos últimos anos, o que faz com que um microcomputador que hoje é considerado um modelo de última geração, amanhã esteja obsoleto, com isto deixando que muitos destes sejam descartados pelas pessoas, podendo estes serem reutilizados utilizando técnicas de multi-processamento.

Estes computadores pessoais baseados da arquitetura IBM PC AT são compostos de uma “placa mãe” (*Mother Board*), onde nela estão interconectados todos os componentes que formam um microcomputador, entre estes componentes estão: o processador, as memórias e os circuitos responsáveis pela interligação entre estes componentes chamado de barramento (*bus*). Esta também possui locais específicos onde se podem encaixar as placas de expansão, possibilitando assim a extensão das capacidades e funcionalidades do sistema computacional. Esses locais são os *slots* de expansão. As placas de expansão são placas de circuitos impressos que se colocam em slots de expansão de modo a estender as funcionalidades de um computador; por exemplo, placas de rede, de vídeo, de som, de modem, etc.

Há três tipos de barramentos principais em um computador: o barramento de dados, o barramento de endereços e o barramento de controle. O mais conhecido é o barramento de dados, portanto, quando as pessoas falam apenas "barramento", em geral estão querendo referir-se ao barramento de dados (Pelisson, 2001). O barramento de endereços transporta endereços de memória e de periféricos. O seu número de linhas determina o número máximo de endereços que este pode endereçar. Os primeiros PCs tinham um barramento de endereços

de 20 bits, de forma que a CPU podia endereçar 2^{20} bytes, ou 1 MB de dados, hoje, com barramentos de endereços de 32 bits, é possível endereçar 4 GB de memória (Pelisson, 2001).

Para que uma simples placa de vídeo ou disco possa ser utilizada em qualquer microcomputador, independente do processador ou placa-mãe, esta se utiliza de um dos diversos modelos de interface de expansão, ou slots, dentre eles pode-se destacar, o ISA (*Industry Standard Architecture*), o MCA (*Microchannel Architecture*), o EISA (*Extended Industry Standard Architecture*), o VLB (VESA *Local Bus*), o *PCI (Peripheral Component Interconnect*), o AGP *(Accelerated Graphics Port*), o USB (*Universal Serial Bus*), Firewire (também chamado IEEE 1394), IrDa (*Infrared Developers Association*) (Torres, 1998).

O protótipo desenvolvido como objetivo final deste trabalho de conclusão de curso tem como finalidade viabilizar o interfaceamento do microcomputador com o mundo externo, através de uma interface para interligar duas placas-mãe para troca de informações, através do barramento ISA. Esta interligação será feita ponto a ponto e para isto será construída uma interface padrão ISA para estabelecer a comunicação entre dois PCs. Esta comunicação será feita de forma programada, ou seja, não utilizaria interrupção nem DMA. Para avaliar será implementado um *driver* para tratar a entrada e saída entre os dois PCs, que será um programa residente no sistema operacional MS DOS que rodará nas duas pontas, somente controlando o envio e recepção de mensagens entre estas duas interfaces. Algumas características desta interface serão: a transferência de dados será paralela; largura do dado de 8 bits; para ser feita a curta distância.

Para construir as placas será utilizado uma chapa de fibra de uma placa usada que já possui a face padrão ISA e a partir daí fazer as conexões externas. O software ira fazer a leitura num endereço de entrada da placa e transferirá os dados para memória. Esta implementação disponibilizará subsídios para uma futura construção de uma máquina paralela utilizando placas mães de baixo custo.

1.1 OBJETIVOS

O objetivo deste trabalho é desenvolver um protótipo de interface para interligar duas “Placas-mães” (*Mother Board*) para troca de informações, para viabilizar posteriormente à

construção de uma máquina paralela utilizando placas-mães de baixo custo, extremamente populares e com grande diversidade de software.

1.2 RELEVÂNCIA

Tendo em vista proliferação dos microcomputadores tipo PC que hoje encontram-se em qualquer lugar e das constantes evoluções destes equipamentos, a relevância deste trabalho consiste em interligar dois ou mais computadores possibilitando o compartilhamento de seus recursos, bem como um enfoque futuro no processamento paralelo, utilizando-se de placas-mães de baixo custo, extremamente populares e com grande diversidade de software.

1.3 ESTRUTURA DO TRABALHO

Este trabalho está organizado da seguinte maneira:

No primeiro capítulo, é apresentado 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.

O terceiro capítulo apresenta uma visão sobre os tipos de barramentos existentes, suas arquiteturas e técnicas para interfaceamento.

O quarto capítulo trata do desenvolvimento, especificação e implementação da parte de hardware e software que compõem este protótipo.

O quinto capítulo, trás uma análise dos resultados obtidos no protótipo.

E por fim, no sexto capítulo encontram-se as conclusões e sugestões de continuidade do trabalho.

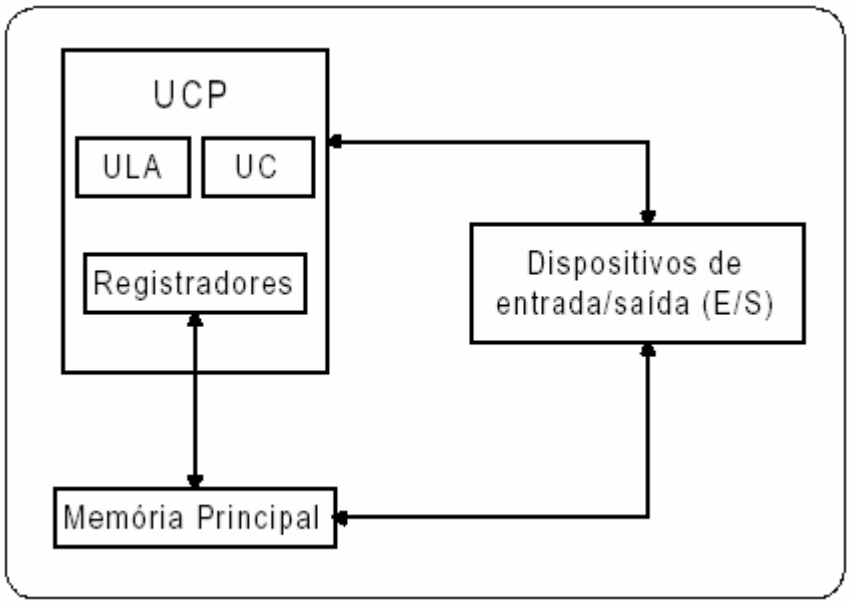
2 ARQUITETURA DE COMPUTADOR

Várias arquiteturas de computadores estão disponíveis na atualidade, para as mais diversas aplicações. A mais utilizada é a arquitetura idealizada por von Neumann. Desde a sua criação até hoje muita coisa mudou: os processadores tornaram-se mais rápidos, 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).

No entanto, está se tornando cada vez mais difícil obter um bom desempenho do modelo de von Neumann. Aplicações que necessitam de uma grande potência computacional, como por exemplo previsão do tempo, simulação de aerodinâmica e sensoriamento remoto, buscam suas soluções em computadores de alto desempenho. Contudo, obter alto desempenho não depende somente do uso dos dispositivos de hardware mais rápidos e mais seguros. Depende também de uma melhoria na arquitetura dos computadores e nas técnicas de processamento (Hwang, 1991).

Todos os componentes de um computador são agrupados basicamente em 3 subsistemas básicos: unidade central de processamento (UCP), memória principal e dispositivos de entrada e saída (E/S). Esses componentes podem ser observados na Figura 1.

Figura 1: Unidades funcionais de um computador



2.1 A UNIDADE CENTRAL DE PROCESSAMENTO (UCP)

A UCP, ou simplesmente processador, tem como função principal unificar todo o sistema controlando as funções realizadas por cada unidade funcional, ou seja, é o “cérebro” do computador. A UCP é também responsável pela execução de todos os programas do sistema, que obrigatoriamente deverão estar armazenados na memória (Machado1992).

A UCP é composta de várias partes distintas. A Unidade de Controle (UC) coordena a execução das instruções através da emissão de pulsos elétricos. A Unidade Lógica e Aritmética (ULA) realiza operações aritméticas (por exemplo, adição e subtração) e operações booleanas (AND, OR) necessárias para realizar as instruções.

A UCP contém também pequenas memórias de alta velocidade usadas para armazenamento temporário de informações. Estas memórias consistem de um número 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. O nome “contador” é enganoso uma vez que esse registrador não funciona como um contador, mas o termo é universalmente usado. Outro registrador importante é o IR

(Instruction Register) ou RI (Registrador de Instrução), o qual armazena a instrução que está sendo executada. A maioria dos computadores tem muitos outros registradores (Tanenbaum 1984).

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

Dentro da UCP existe ainda uma memória interna de alta velocidade. Ela funciona como um cache interno e acelera a movimentação dos dados. Esse cache é muito rápido (e também caro), pois trabalha na velocidade do processador.

Um número bastante elevado de processadores 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

Processador	Voltagem	Tamanho registradores. Internos	Larg. do bar. de dados	Larg. do bar. de endereços	Memória suport.	Cache interna	Processador matemático	Transístores	Lançado em
8088	5v	16 bits	8 bits	20 bits	1MB	Não	Não	29 mil	1979
8086	5v	16 bits	16 bits	20 bits	1MB	Não	Não	29 mil	1978
286	5v	16 bits	16 bits	24 bits	16MB	Não	Não	134 mil	1982
386SX	5v	16 bits	16 bits	24 bits	16MB	Não	Não	275 mil	1988
386SL	3,3v	32 bits	16 bits	24 bits	16MB	0K	Não	855 mil	1990
386DX	5v	32 bits	32 bits	32 bits	4GB	Não	Não	275 mil	1985
486SX	5v	32 bits	32 bits	32 bits	4GB	8KB	Não	1,18 milhões	1991
486SX2	5v	32 bits	32 bits	32 bits	4GB	8KB	Não	1,18 milhões	1994
486DX	5v	32 bits	32 bits	32 bits	4GB	8KB	Sim	1,2 milhões	1989
486SL	3,3v	32 bits	32 bits	32 bits	4GB	8KB	Opcional	1,4 milhões	1992
486DX2	5v	32 bits	32 bits	32 bits	4GB	8KB	Sim	1,1 milhões	1992
486DX4	3,3v	32 bits	32 bits	32 bits	4GB	16KB	Sim	1,6 milhões	1994
Pentium OD	5v	32 bits	32 bits	32 bits	4GB	2x16KB	Sim	3,1 milhões	1995
Pentium 60/66	5v	32 bits	64 bits	32 bits	4GB	2x8KB	Sim	3,1 milhões	1993
Pentium 75+	3,3v	32 bits	64 bits	32 bits	4GB	2x8KB	Sim	3,3 milhões	1994
Pentium Pro	2,9v	32 bits	64 bits	36 bits	64GB	2x8KB	Sim	5,5 milhões	1995

2.2 MEMÓRIA PRINCIPAL

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 secundário;
- Armazenamento primário ou memória principal.

Quando uma informação está na memória secundária ela deve ser primeiramente transferida para a memória principal para ser processada.

A memória principal consiste de um número de células onde cada uma pode armazenar um “pedaço de informação”. Cada célula possui um número, chamado endereço, através do qual elas podem ser referenciadas. Se a memória tem n células, terá endereços de 0 a n-1.

A UCP comunica-se com a memória principal usando 2 registradores: o *Memory Address Register* (MAR) e o *Memory Buffer Register* (MBR). Quando a UCP necessita ler uma célula da memória, seja instrução ou dados, ela coloca o endereço da célula desejada no MAR e envia um sinal de leitura para a memória. A memória inicia a operação e após um ciclo ela coloca a célula desejada no MBR, de onde a UCP pode retirar. Para escrever uma célula na memória, a UCP coloca o endereço de onde a célula será armazenada no MAR e a célula a ser armazenada no MBR, envia um sinal de escrita para a memória e esta realiza a operação (Tanenbaum 1984).

A memória principal pode ser classificada em função de sua volatilidade, que é a capacidade da memória preservar o seu conteúdo mesmo sem uma fonte de alimentação. As memórias denominadas voláteis se caracterizam por permitir tanto a operação de escrita quanto a de leitura. Como exemplo, tem-se a memória RAM (*Random Access Memory*). As memórias denominadas não voláteis, se caracterizam por permitir somente a leitura dos dados, não permitindo a escrita. Como exemplo, tem-se as memórias ROM e EPROM.

2.3 CHIPSET

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

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

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

Os *chipset* são projetados pelo fabricante para operar com determinados conjuntos de processadores. A maioria deles foi projetada para operar com os processadores partir do 486.

Os *chipsets* pertencem à mesma classe de tecnologia dos chips de CPU, a tecnologia VLSI (*Very Large Scale Integration*). Alguns *chipsets* são popularmente conhecidos como "TRITON".

2.4 DISPOSITIVOS DE ENTRADA E SAÍDA (E/S)

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

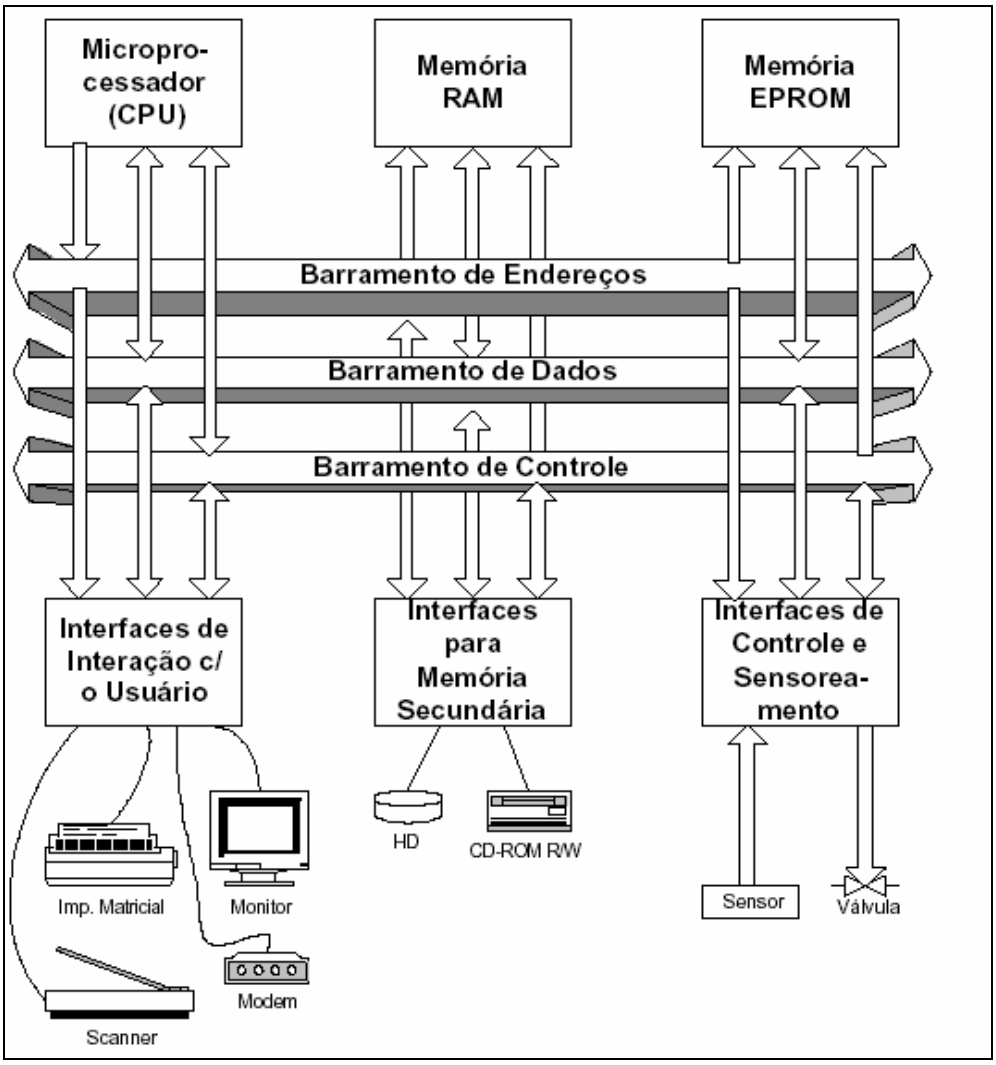
- entrada do programa e dados;
- processamento;
- saída dos resultados.

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

Alguns dispositivos de E/S podem transmitir uma grande quantidade de dados em um curto espaço de tempo. Se a UCP tem que processar cada caracter separadamente, muito tempo de UCP é desperdiçado. Para impedir que a UCP fique preso por muito tempo em E/S, muitos computadores de médio e grande porte têm um ou mais processadores especializados e de baixo custo dedicados a tarefa de E/S. Devido a E/S ser realizada por esses processadores, a UCP fica disponível para outros processamentos. Os processadores de E/S trabalham em paralelo com a UCP, ou seja, enquanto a UCP está processando, esses processadores se dedicam a operações de E/S.

Estes componentes são ligados através de um sistema de barramentos, o qual será explicado no próximo Capítulo.

Figura 2: Sistema de Microcomputador Típico e suas Interfaces



3 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 processador, 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 3 barramentos (Conforme figura 2) independentes em suas funções elétricas: 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 CPU) e define o caminho de comunicação dentro do sistema.

O Barramento de Dados é bidirecional, sendo o meio de comunicação entre os componentes do sistema. Na saída de dados da CPU, estes são gerados pelo microprocessador (CPU) e enviados à uma unidade que é selecionada pelo barramento de endereços. Na entrada de dados, estes são gerados por uma unidade particular 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.

3.1 BARRAMENTO LOCAL

O barramento local é utilizado na comunicação do processador com os circuitos básicos que demandam velocidade (especialmente a memória RAM e a memória cache). Esse barramento é totalmente transparente ao usuário, mesmo quando temos em mãos uma placa-mãe. Também chamado de Frontal Serial Bus (FSB). O barramento local bem como a maioria dos barramentos de expansão podem ser subdivididos 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;
- Frequência 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 – Q*uadruple Data Rate*;
- Acesso a 4GB de memória RAM. Tal resultado é obtido com base no cálculo 2^{32} onde o "2" representa a base ou seja os estados dos bits 0 e 1, e o "32" o número de bits do barramento de endereços, ou seja, o número de bits que combinados identificam as posições de memória que serão acessadas para armazenamento/leitura de informações.

3.2 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.

3.3 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).

3.4 LARGURA DE BANDA DO BARRAMENTO

A Largura de Banda refere-se a quantidade de dados que teoricamente podem ser transferidos no barramento numa dada unidade de tempo. Fazendo a analogia com uma auto-estrada a largura do barramento corresponderá ao número de faixas, a velocidade do barramento a velocidade com que os carros se deslocam, então a largura de banda pode ser entendida como a quantidade de tráfego que a auto-estrada pode suportar por segundo.

A tabela 2, 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).

A largura de banda do PCI standard deveria ser 133 ou seja 32/8 x 33,3 =133,3 Mb/s, como muitas vezes é referenciado, no entanto isto não é tecnicamente correto, devido à definição do Mega assim 1MHz é 1.000.000 Hertz, mas um 1MB, não é 1.000.000 bytes, mas 1.048.576 bytes, então a largura de banda do PCI é de127.2Mbytes/s que corresponde a (32/8) x (1.000.000/1.048.576) x 33,3 = 127,2.

Tabela 2: Taxa máxima de transferência dos barramentos mais conhecidos

Barramento	Largura (bits)	Velocidade (MHz)	Largura de Banda (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)

3.5 INTERFACE DE BARRAMENTO

Num sistema onde existem muitos barramentos distintos, devem ser previstos circuitos pelo *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 vise e versa, o barramento PCI também tem uma *bridge* para o barramento do processador ou *local bus* (Torres, 1999).

3.6 CICLOS DE BARRAMENTO

Existem duas classificações gerais de tipos de ciclos de barramento, o ciclo do processador ou seja ciclos gerados a partir do processador e o ciclo de barramento de DMA gerado pelo controlador de DMA. O processador pode gerar um dos cinco tipos de ciclos: *Memory-read bus cycle, Memory-write bus cycle, I/O-port-read bus cycle, I/O-port-write bus cycl, Interrupt-acknowledge bus cycle*. Ja os gerados pelo controlador de DMA são DMA *read* I/O e DMA *write* I/O. A tabela 3 demonstra os tipos de ciclos de barramentos e seus propósitos:

Tabela 3: Tipos de Ciclos de Barramento

Tipo de Ciclo	Propósito	Fluxo dos dados
Memory read	Processador busca dados ou instrução	Memória para processador
Memory write	Processador escreve dado	Processador para memória
I/O port read	Processador busca dados do dispositivo de Entrada e Saída	Porta de entrada e saída para Processador
I/O port write	Processador envia dado a dispositivo de de Entrada e Saída	Processador para Porta de entrada e saída.
Interrupt acknowledge	Envio de interrupção ao processador	Controlador de interrupção para processador
DMA write I/O	Envio de dados da memória para a interface de entrada e saída	Memória para dispositivo de Entrada e Saída
DMA read I/O	Envido de dados da interface de entrada e saída para a memória	Dispositivo de Entrada e Saída para memória

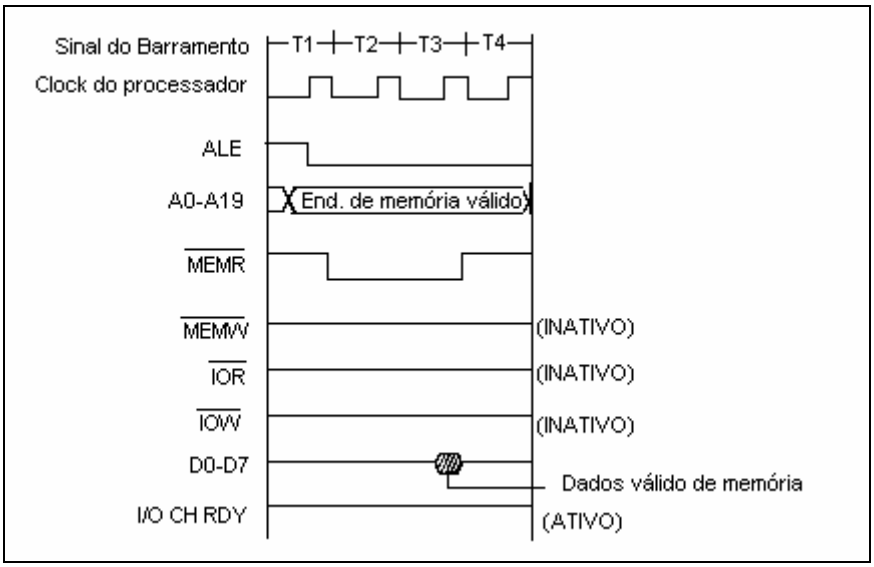
3.6.1 CICLO DE *MEMORY READ*

O ciclo de *Memory read* “Leitura de Memória” é usado para buscar instruções ou dados da memória, que pode ser na memória RAM, memória ROM um uma outra memória endereçável na própria placa mãe. Todos os ciclos utilizam no mínimo 4 *clocks* do

processador. O tempo de cada *clock* varia de acordo com a velocidade de *clock* do processador por exemplo num processador de um PC XT de 8 MHz o tempo é de 125 ns ou seja um ciclo de barramento para um processador de 8 MHz levando quatro *clocks* de processador para efetuar um ciclo de barramento será de 500 ns (Eggebrecht, 1995).

O ciclo de *memory-read* inicia no tempo T1 do *clock* com o sinal de *Address Latch Enable* (ALE) sendo ativado ou posto em alta, no final deste sinal indica que o barramento de endereço conterá um endereço de memória válido. A seguir, aproximadamente no T2, o sinal de barramento de MEMR é ativado, isto indica aos adaptadores de memórias atachados ao barramento que o ciclo é um ciclo de leitura da memória. Também indica que se um adaptador de memória contém um endereço que corresponde ao endereço contido no barramento de endereço, o adaptador deveria enviar ao barramento de dados o conteúdo da posição de memória habilitada pelo barramento de endereço. Todos os dispositivos de memória possuem um decodificador para decodificar o endereço do barramento de endereço e, assim, determina se deverá responder. No começo de T4 do *clock*, o microprocessador captura os dados do barramento de dados. Logo após o começo de T4 do *clock*, é desativado o sinal MEMR do barramento e ao término de T4, se dá o fim do ciclo de leitura de memória, este esquema é demonstrado na figura 3.

Figura 3: Ciclo de *Memory-read*



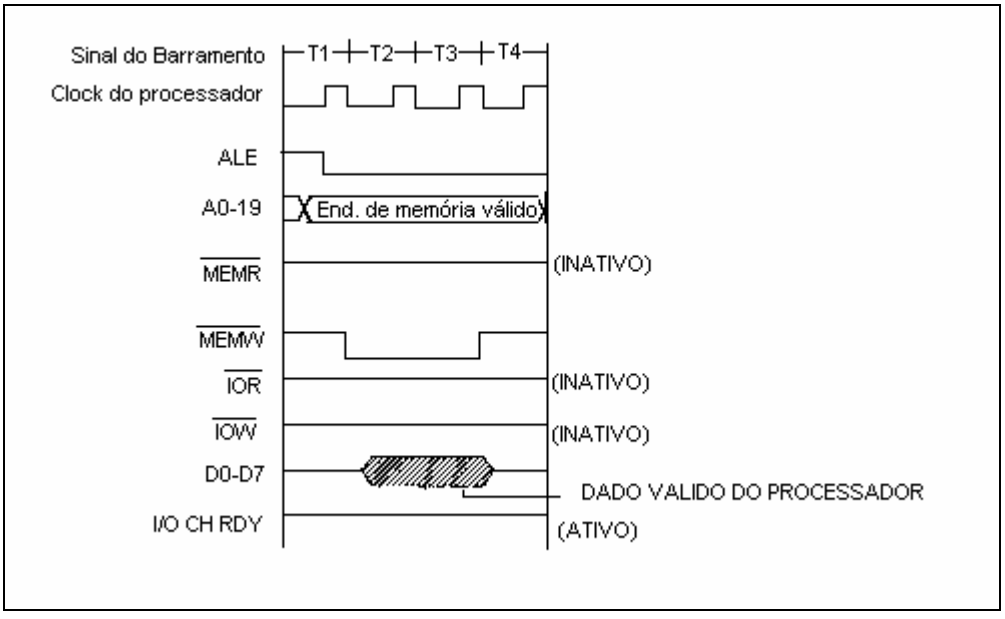
Adaptado de Eggebrecht (1995).

3.6.2 CICLO DE *MEMORY WRITE*

O ciclo de *Memory write* “Escrita em Memória” é usado pelo processador para escrever dados em determinada posição de memória. Como no ciclo de leitura a memória o processador envia ao barramento de endereço a posição de memória de deverá receber o dado, ao barramento de controle o sinal de escrita a memória e também no barramento de dados o dado a ser escrito em memória (Eggebrecht, 1995).

O ciclo de *memory-write* inicia no tempo T1 do *clock* com o sigal de ALE sendo ativado ou posto em alta, isto indica que o barramento de endereço conterá um endereço válido, a seguir aproximadamente no T2, o sinal de barramento de MEMW é ativado, isto indica aos adaptadores de memórias atachados ao barramento que o ciclo é um ciclo de escrita à memória. Logo após ao sinal de MEMW ser ativado o processador envia ao barramento de dados o dado a ser escrito na posição de habilitada pelo barramento de endereço, em T4 o sinal de MEMW é desativado e no final de T4, o ciclo é completado. A figura 4 demostra graficamente este ciclo.

Figura 4: Ciclo de *Memory-write*



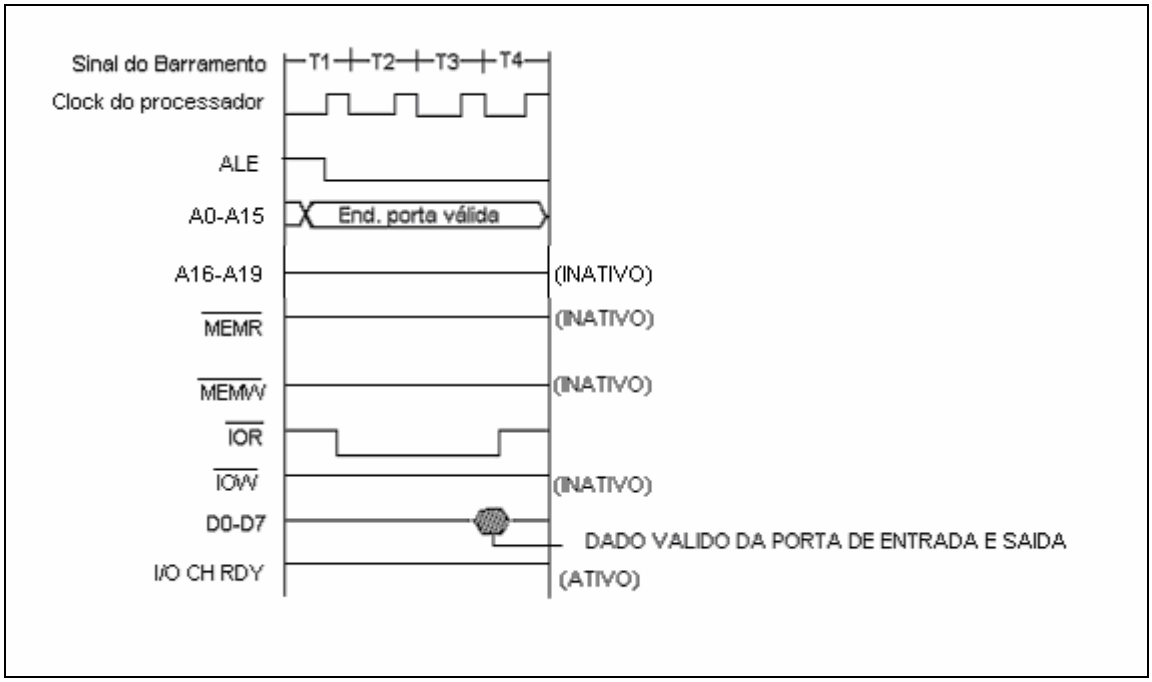
Adaptado de Eggebrecht (1995).

3.6.3 CICLO DE *I/O PORT READ*

O ciclo de *I/O-port read* “Leitura a porta de entrada e saída” é executado quando uma instrução IN e executada pelo processador, este ciclo e similar ao ciclo de leitura a memória descrita acima, seu propósito é de buscar o dado contido no dispositivo de E/S (Entrada e Saída) endereçado no barramento de endereço. Este ciclo possui aproximadamente 5 ciclos de clock, e este dispositivo de E/S pode estender o tamanho do ciclo desativando o sinal de barramento READY (Eggebrecht, 1995).

O ciclo de *I/O-port read* 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 uma porta válida, a seguir aproximadamente no T2, o sinal de IOR é ativado no barramento de controle, que indica que o ciclo é um ciclo de *I/O port read* e que a 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 5 demonstra graficamente este ciclo.

Figura 5: Ciclo de *I/O-port read*



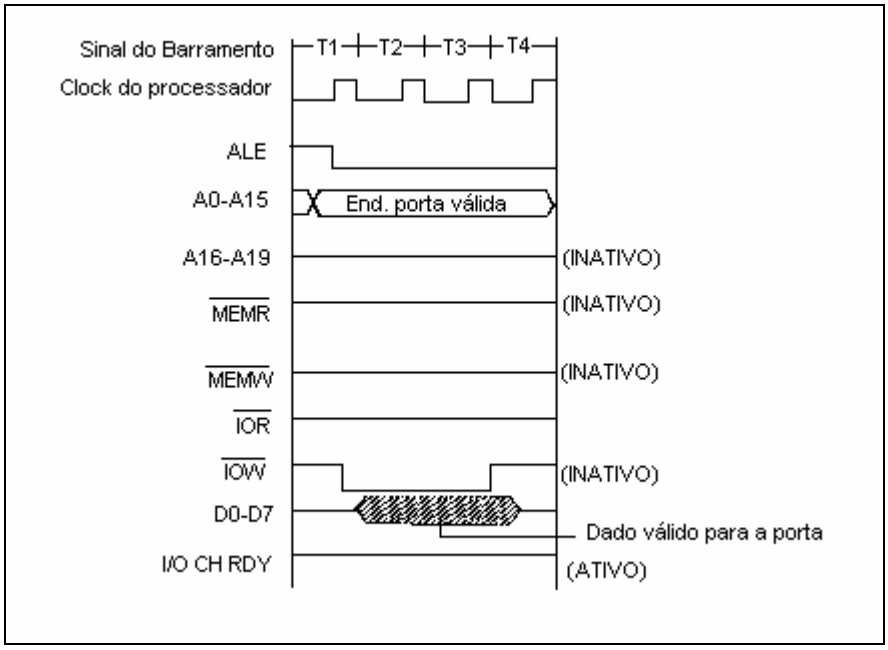
Adaptado de Eggebrecht (1995).

3.6.4 CICLO DE *I/O PORT WRITE*

O ciclo de *I/O-port write* “Escrita na porta de entrada e saída” é executado quando uma instrução OUT e executada pelo processador, este ciclo escreve o dado contido num registrador do processador em uma porta de E/S especifica. Este ciclo possui aproximadamente cinco ciclos de *clock*, e este dispositivo de E/S pode estender o tamanho do ciclo desativando o sinal de barramento READY (Eggebrecht, 1995).

O ciclo de *I/O-port write* 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 uma porta valida, a seguir aproximadamente no T2, o sinal de IOW é ativado no barramento de controle, que indica que o ciclo é um ciclo de *I/O port write* e que a dispositivo de I/O cujo endereço for igual ao endereço contido no barramento de dados está sendo selecionado, Após T2 o processador envia ao barramento de dados o dado que devera ser recebido pelo dispositivo selecionado, no início de T4 o sinal de IOW é desativado e no final de T4 o ciclo estará completo. A figura 6 demonstra graficamente este ciclo.

Figura 6: Ciclo de *I/O-port write*



Adaptado de Eggebrecht (1995).

3.7 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) .

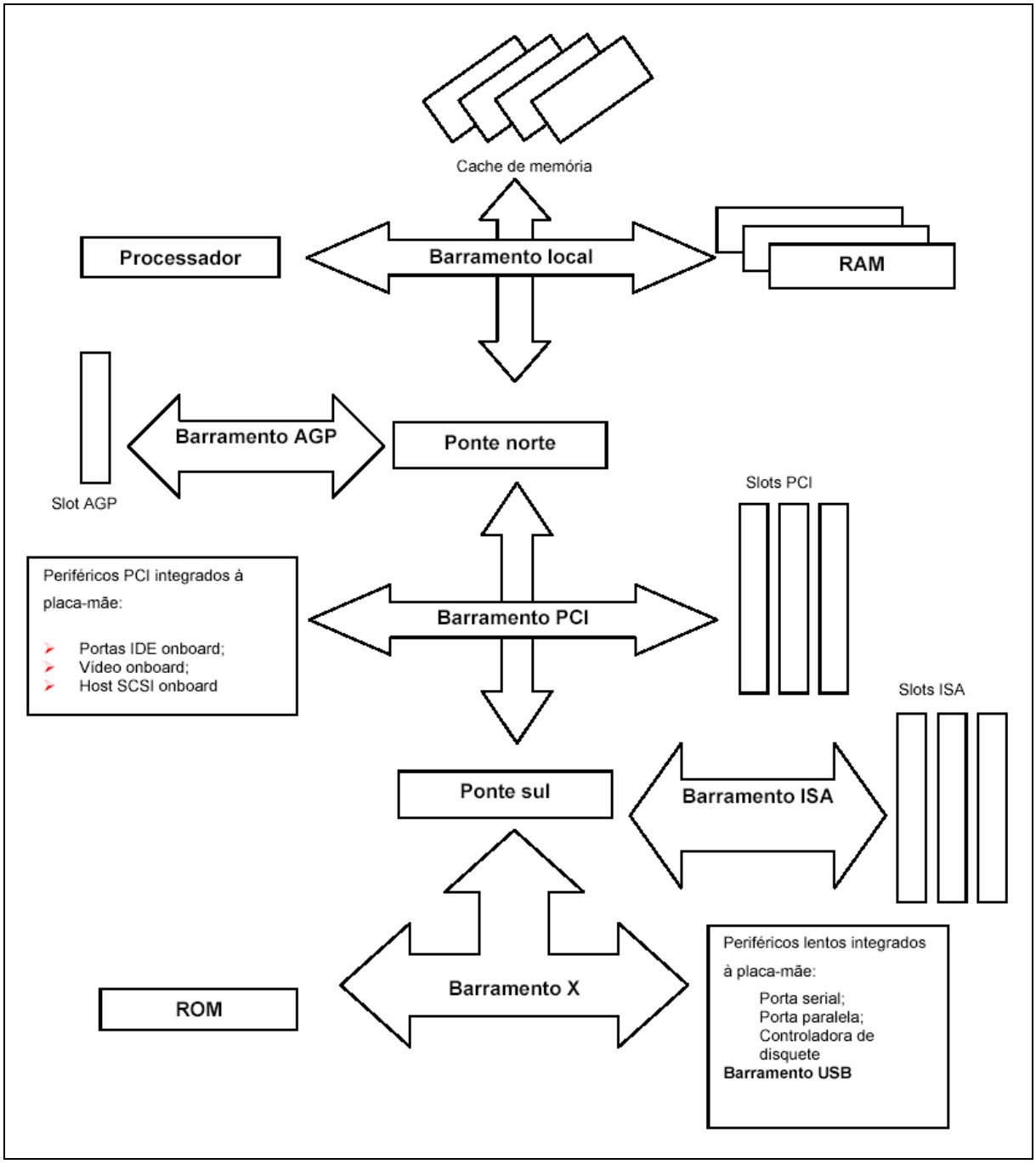
Com a introdução do micro AT, o barramento e o *slot* ISA aumentaram de tamanho de forma a acompanhar as características do processador 80286:

- Barramento de dados de 16 bits
- Barramento de endereços de 24 bits
- Frequência 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 dispositivos 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, na verdade nos microcomputadores de hoje esta conversão não é feita diretamente do barramento local para o barramento ISA e sim do barramento PCI que será discutido na próxima seção para o barramento ISA que é chamando de ponte PCI-ISA (ou ponte sul) conforme esquema ilustrado na figura 7 (Torres, 1999).

Figura 7: Esquema de funcionamento de uma placa-mãe



3.7.1 RECURSOS DO BARRAMENTO ISA

Desde o primeiro PC, se convive com recursos de hardware conhecidos como endereços de E/S, 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 de E/S 220h, interrupção IRQ5 e canais de DMA 1 e 5. O barramento ISA traz estes recursos da seguinte forma:

- Endereços de E/S: 1 KB (de 000h a 3FFh);
- Interrupções: 15 linhas;
- Canais de DMA: 8 canais.

3.7.2 ENDEREÇOS DE E/S

Os endereços de E/S são utilizados na comunicação do processador com um dispositivo através das instruções de E/S "*IN*" e "*OUT*" como descrita nos itens 3.6.3 e 3.6.4. Como por exemplo, a porta paralela utiliza o endereço de E/S 378h, se o processador precisa imprimir um texto, isso será feito enviando dados através desse endereço. A tabela 4 mostra os endereços de E/S tipicamente utilizados do PC AT (Eggebrecht, 1995).

Tabela 4: Mapa de endereços de E/S do PC AT

Faixa de Endereço (em Hexa)	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 coprocessador matemático ocupado
0F1	Reseta o coprocessador matemático
0F8-0FF	Coprocessador 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	Bisincronizador SDLC 2
3A0-3AF	Bisincronizador 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)

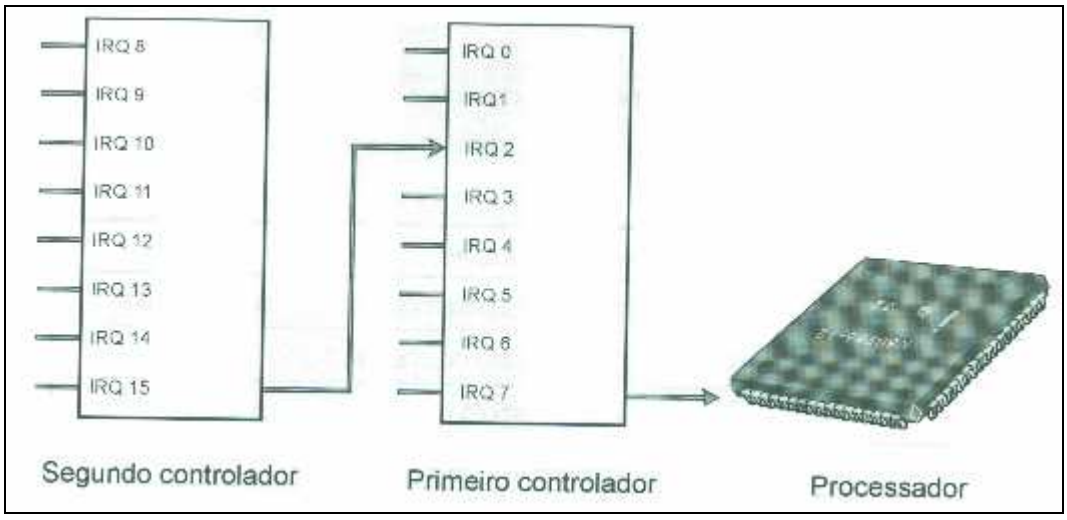
Adaptado de Eggebrecht (1995).

3.7.3 INTERRUPÇÕES

Interrupções são pedidas de atenção feitos por periféricos ao processador. Por exemplo se você mover o mouse, isso gerará uma interrupção no processador que forçará a ler sua nova posição, fazendo as atualizações necessárias aos programas que estão sendo executados.

A primeira versão do barramento ISA e dos micro computadores, o PC XT possuíam 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. Este 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 8 demonstra como esta ligação ocorre (Torres 1999).

Figura 8: Como Funciona o esquema de interrupções



Fonte: Gabriel Torres (1999).

Na realidade, não temos tantas linhas disponíveis assim. Se você reparar no mapa de interrupções (mapa da tabela 5), você verá que há diversas interrupções automaticamente alocadas em todos os micros pela placa-mãe, não podendo ser utilizadas portanto na prática, um PC típico possui apenas 4 interrupções disponíveis caso não utilize nenhum periférico extra.

Tabela 5: 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	Coprocessador 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

Adaptado de Torres (1999)

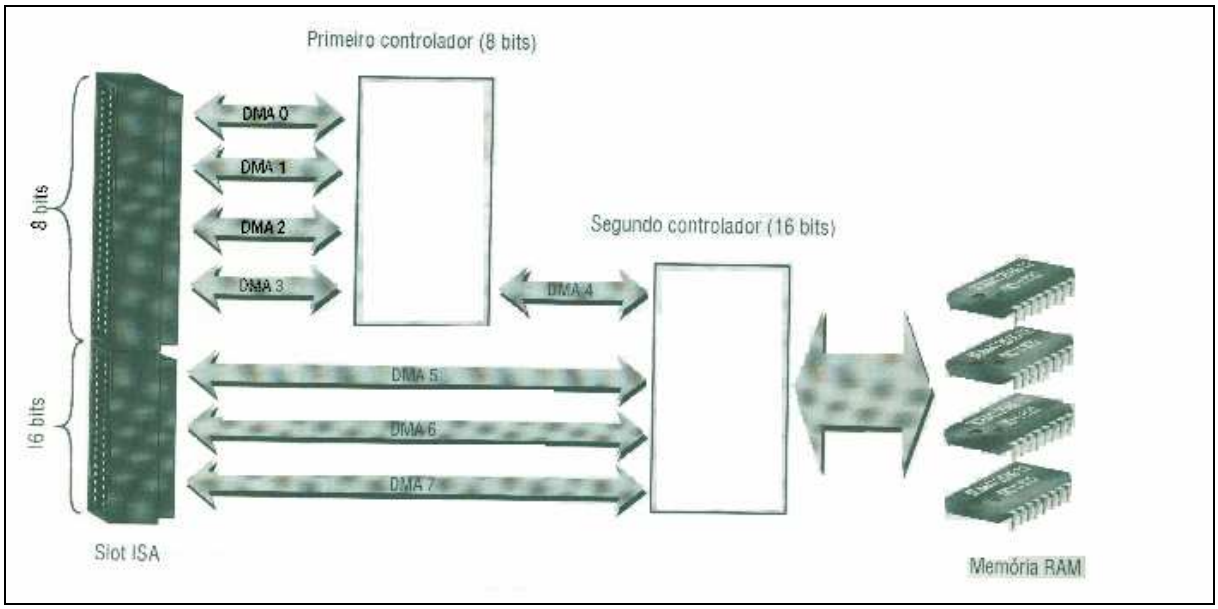
3.7.4 CANAIS DE DMA

Um circuito controlador de DMA faz a transferência de dados entre placas ISA e a memória RAM, sem o uso do processador. Isso faz com que o desempenho do micro aumente consideravelmente, este dispositivo hoje em dia esta integrado ao *chipset* da placa mãe.

Da mesma forma que as interrupções, os periféricos ISA que necessitam usar o controlador de DMA deverão ser conectados a esse dispositivo através de suas linhas (Também chamadas de canais) DMA0 a DMA7. Similarmente ao que ocorreu com o controlador de interrupções, os primeiros PCs possuíam apenas um controlador de 8 bits. A partir do PC AT foi adicionado um segundo controlador, de 16 bits, conectado em cascata com o primeiro controlador. Na figura 9 verifica-se como funciona o controlador de DMA, a

linha DMA4 não pode ser utilizada, pois faz a conexão entre os dois controladores (Torres, 1999).

Figura 9: Funcionamento do Controlador de DMA

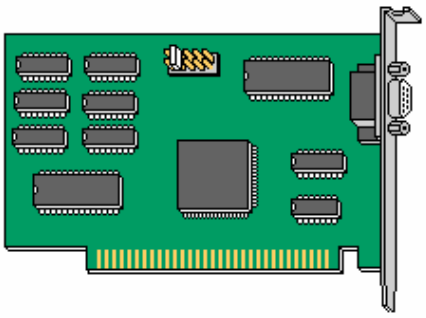


Fonte: Gabriel Torres (1999).

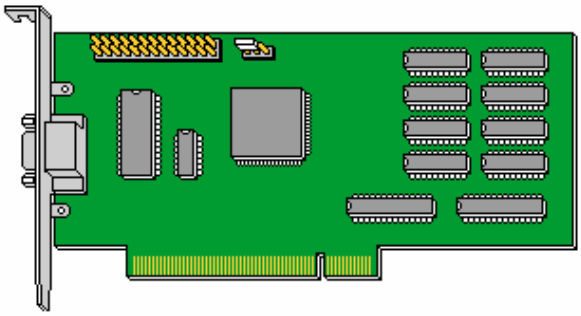
3.7.5 DESCRIÇÃO DOS SINAIS DO BARRAMETO ISA

A interligação do micro computador com a placa de expansão é feita através dos barramentos de dados, controle e endereço como foi descrito acima. A figura 10 mostra os tipos de conectores e placas de expansão ISA que podem ser de 8 ou 16 bits e a tabela 6 apresenta todos os sinais do barramento ISA 8 e 16 bis.

Figura 10: Arquitetura do barramento ISA



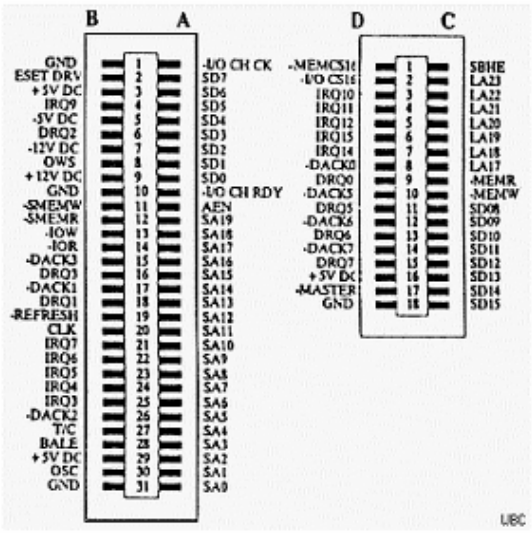
Placa ISA 8 bits



Placa ISA 16 bits

Slot ISA 8 bits

Slot ISA 16 bits



Posição dos Pinos no slot ISA

Tabela 6: Parte A/B do pinos do barramento ISA

Sinal	Nome	Pino	E/S	Descrição
A0-A19	A0-A19	A31 a A12	S	Barramento de endereços (sinal A0 pino A31, sinal A19 pino A12).
AEN	Address Enable	A11	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	A11	S	Indica que há um endereço válido no barramento de endereços. Não é ativado durante operações de DMA.
CLK	Clock	B20	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	A9 a A2	E/S	Barramento de dados (D0 pino A9, D7 pino A2).
/DACK1	DMA Acknowledge 1	B17	S	Indica que o pedido de DMA 1 foi aceito e que o controlador de interrupção irá iniciá-lo.
/DACK2	DMA Acknowledge 2	B26	S	Indica que o pedido de DMA 2 foi aceito e que o controlador de interrupção irá iniciá-lo.
/DACK3	DMA Acknowledge 3	B15	S	Indica que o pedido de DMA 3 foi aceito e que o controlador de interrupção irá iniciá-lo.
DRQ1	DMA Request 1	B18	E	Faz um pedido de DMA nível 1 ao controlador de DMA.
DRQ2	DMA Request 2	B6	E	Faz um pedido de DMA nível 2 ao controlador de DMA.
DRQ3	DMA Request 3	B16	E	Faz um pedido de DMA nível 3 ao controlador de DMA.
/I/O CH CK	I/O Channel Check	A1	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	I/O Channel	A10	E	Gera *wait states* para operações de I/O ou de memória. Se o

RDY	Ready			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	A14	S	Indica que uma leitura em I/O está sendo executada. É ativado também durante ciclos de DMA.
/IOW	I/O Write	A13	S	Indica que uma escrita em I/O está sendo executada. É também ativado durante operações de DMA.
IRQ3	Interrupt Request 3	B25	E	Faz um pedido de interrupção nível 3 ao controlador de interrupções.
IRQ4	Interrupt Request 4	B24	E	Faz um pedido de interrupção nível 4 ao controlador de interrupções.
IRQ5	Interrupt Request 5	B23	E	Faz um pedido de interrupção nível 5 ao controlador de interrupções.
IRQ6	Interrupt Request 6	B22	E	Faz um pedido de interrupção nível 6 ao controlador de interrupções.
IRQ7	Interrupt Request 7	B21	E	Faz um pedido de interrupção nível 7 ao controlador de interrupções.
IRQ9	Interrupt Request 9	B4	E	Faz um pedido de interrupção nível 9 ao controlador de interrupções.
/MEMR	Memory Read	B12	S	Indica que está sendo executada uma operação de leitura em memória. É também ativado durante operações de DMA.
/MEMW	Memory Write	B11	S	Indica que está sendo executada uma operação de escrita em memória. É também ativado durante operações de DMA.
OSC	Oscilador	B30	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	B2	S	Sinal de *reset* do sistema.
TC	Terminal Count	B27	S	Indica que um dos canais de DMA acabou de realizar a transferência de DMA programada.

+5V	+5V	B3, B29	S	+5V
-12V	-12V	B7	S	-12V
-5V	-5V	B5	S	-5V
GND	Terra	B1, B10, B31	S	Terra

Tabela 7: Parte D/C dos pinos do barramento ISA

Sinal	Nome	Pino	E/S	Descrição
A17-A23	A17-A23	C8 a C2	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	C11 a C18	E/S	Barramento de dados, parte alta (sinal D8 pino C11, sinal D15 pino C18).
/DACK0	DMA Acknowledge 0	D8	S	Indica que o pedido de DMA 0 foi aceito e que o controlador de interrupção irá iniciá-lo.
/DACK5	DMA Acknowledge 5	D10	S	Indica que o pedido de DMA 5 foi aceito e que o controlador de interrupção irá iniciá-lo.
/DACK6	DMA Acknowledge 6	D12	S	Indica que o pedido de DMA 6 foi aceito e que o controlador de interrupção irá iniciá-lo.
/DACK7	DMA Acknowledge 7	D14	S	Indica que o pedido de DMA 7 foi aceito e que o controlador de interrupção irá iniciá-lo.
DRQ0	DMA Request 0	D9	E	Faz um pedido de DMA nível 0 ao controlador de DMA.
DRQ5	DMA Request 5	D11	E	Faz um pedido de DMA nível 5 ao controlador de DMA.
DRQ6	DMA Request 6	D13	E	Faz um pedido de DMA nível 6 ao controlador de DMA.
DRQ7	DMA Request 7	D15	E	Faz um pedido de DMA nível 7 ao controlador de DMA.

/IOCS16	I/O Chip Select 16 bits	D2	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	D3	E	Faz um pedido de interrupção nível 10 ao controlador de interrupções.
IRQ11	Interrupt Request 11	D4	E	Faz um pedido de interrupção nível 11 ao controlador de interrupções.
IRQ12	Interrupt Request 12	D5	E	Faz um pedido de interrupção nível 12 ao controlador de interrupções.
IRQ14	Interrupt Request 14	D7	E	Faz um pedido de interrupção nível 14 ao controlador de interrupções.
IRQ15	Interrupt Request 15	D6	E	Faz um pedido de interrupção nível 15 ao controlador de interrupções.
/MASTER	Master	D17	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.
/MEMCS16	Memory Chip Select 16 bits	D1	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).
/MEMR	Memory Read	C9	S	É ativado quando é feita uma leitura em uma posição de memória acima de 1 MB.
/MEMW	Memory Write	C10	S	É ativado quando é feita uma escrita em uma posição de memória acima de 1 MB.
SBHE	System Bus High Enable	C1	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	D16	S	+5 V

Tabela 8: Esquema de funções do sinal de SBHE para determinar tamanho do dado no barramento ISA

SBHE	A0	Função
1	0	Transferência de 16 bits
1	1	Transferência do byte alto
0	0	Transferência do byte baixo
0	1	Inválido

3.8 BARRAMNETO MCA

O barramento MCA (*Micro Channel Architecture*) foi criado em meados dos anos 80 pela IBM, para ser usado nos computadores IBM PS/2, os sucessores do IBM PC/AT. Tratava-se de um barramento de 32 bits que operava com um *clock* de 10 MHz, o que o tornava 2,5 vezes mais veloz que o velho barramento ISA. Apesar de ser veloz para os padrões de sua época, possuía um sério problema: era um barramento proprietário, pertencente à IBM. Isto significa que apenas a IBM podia produzir placas de expansão MCA, bem como placas de CPU equipadas com slots MCA. Além de ser inútil para os demais fabricantes além da IBM, não era nada interessante para o usuário. Como apenas a IBM produzia placas de expansão MCA, ou então empresas credenciadas pela IBM, essas placas acabavam sendo muito raras e caras. O barramento ISA, apesar de ser inferior, continuou sendo usado em larga escala por todos os fabricantes, exceto a IBM.

3.9 BARRAMENTO EISA

O Barramento EISA ("*Extended Industry Standard Architecture*") de 32 bits foi um produto resultante da formação de um consórcio composto de fabricantes de "clones" ("*Gang of Nine*") ou a Gang dos Nove, nominalmente: Wyse, AST Research, Tandy, Compaq, Hewlett-Packard, Zenith, Olivetti, NEC e Epson, mnemonicamente chamado, "WATCHZONE" (Barreto, 2000).

O argumento desses fabricantes era o de que sempre procuraram oferecer aos seus usuários produtos de mais baixo custo e com ampla disponibilidade de placas de expansão, qualidades que na época (e nem hoje) não eram (não são) encontradas na opção pelo barramento MCA (Torres, 1998).

Para que o barramento EISA fosse compatível com a maioria das placas (ISA 8 e ISA 16) já disponíveis no mercado, os conectores da placa mãe (onde se encaixam as placas de expansão) dos microcomputadores foram mudados para conectores EISA/ISA. Ou seja, tanto placas ISA de 8 ou 16 bits, quanto placas EISA podiam e podem ser inseridas nos *slots* do sistema.

O novo padrão criado por esse grupo de fabricantes chamava-se EISA, é totalmente compatível com o antigo ISA.

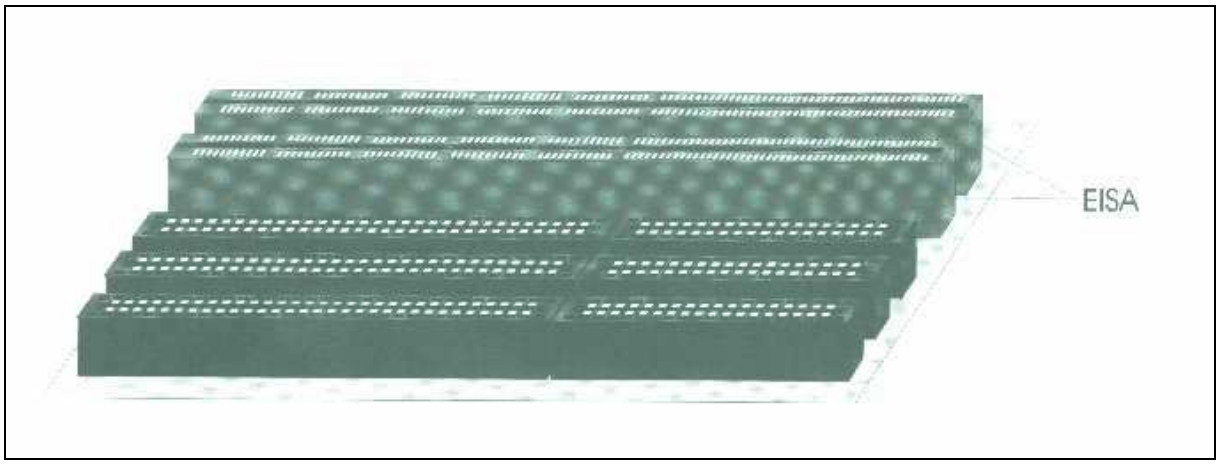
Características:

• Barramento de dados de 32 bits
• Barramento de endereços de 32 bits
• Freqüência de operação de 8 MHz

O problema desse barramento era clássico: para manter a compatibilidade com o barramento ISA o barramento EISA teve que utilizar a mesma freqüência de operação dos sinais do barramento ISA ou seja 8 MHz.

O artifício consistiu de dotar o conector de duas linhas de contactos, uma ISA e outra EISA. Placas ISA penetram até a metade do conector fazendo uso da primeira linha de contatos (compatível ISA 8 e 16 bits). Placas EISA penetram completamente no conector fazendo uso da segunda linha de contatos (compatível EISA). A profundidade de penetração das placas é determinada pelo número de ranhuras das placas. No caso das placas ISA existe somente uma única ranhura e no das placas EISA existem 6 ranhuras (uma ranhura maior e cinco menores). Assim, a profundidade dos contatos do pente das placas de expansão EISA passa dos atuais 0,79 cm para 1,32 cm. A Figura 11 demonstra a diferença física do slot ISA para o slot EISA (Barreto, 2000).

Figura 11: Comparação de um slot ISA com um EISA



Mas havia ainda um problema: para manter total compatibilidade com o barramento ISA, o barramento EISA teve de usar a mesma freqüência de operação dos sinais do barramento ISA. Mesmo tendo a capacidade de trabalhar com dados de 32 bits, endereçar até 4GB de memória e ser uma arquitetura aberta, o EISA não se tornou popular, pois ainda apresentava um gargalo para interfaces que exigiam alto desempenho.

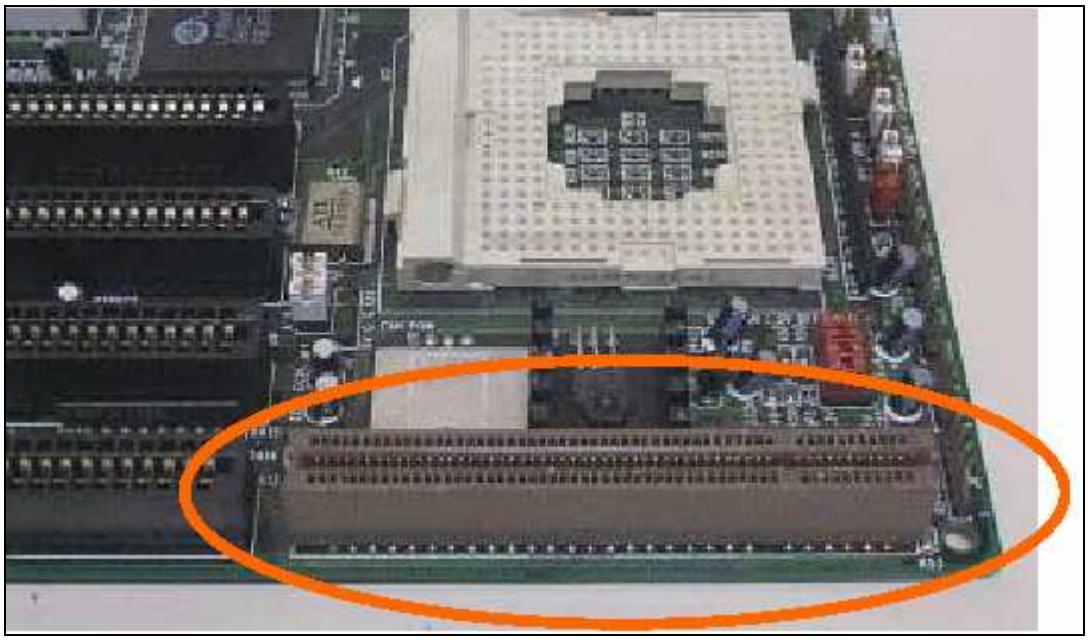
3.10 BARRAMENTO VLB OU VESA

O VLB (VESA Local Bus) foi criado por uma associação dos fabricantes de placa de vídeo denominada VESA com a intenção de aumentar o desempenho do vídeo. O VLB é ligado diretamente ao barramento local do micro. Isso quer dizer que um micro com um processador operando a 33 MHz – a velocidade do barramento local em micros mais antigos era equivalente a velocidade do processador - teria a taxa de transferência de 132 MB/s (Vasconcelos, 2002).

- Barramento de dados igual ao do processador;
- Barramento de endereços de 32 bits;
- Freqüência de operação igual a freqüência do barramento local.

O slot VLB foi amplamente utilizado nos 486, quando poucos processadores trabalhavam com uma freqüência de operação externa acima de 33 MHz. A figura 12 demostra a diferença entre um slot ISA e VLB (Barreto, 2000).

Figura 12: Exemplo de slot VLB de 32 bits.



Fonte: Marcelo Barreto (2000).

No barramento VESA (*Video Eletronics Standard Association*) *Local Bus* 32/64 *Bits*, em geral, a sua velocidade de processamento dos dados é sempre maior para as comunicações entre a CPU e a memória do que para as comunicações entre a CPU e equipamento como disco rígido e monitor de vídeo. A função do *Local Bus* é semelhante a de um trator que alarga uma avenida para que possam trafegar por ela mais carros. Dessa forma, a "conversa" entre a CPU e o monitor passa a ter a mesma velocidade que as comunicações da CPU com a memória principal (Pinto 1998).

O barramento VESA *Local-Bus* é uma extensão física do barramento ISA podendo aceitar placas adaptadoras de 8 ou 16 bits ISA. Desenvolvido principalmente para os processadores 486, não permitem mais que 3 *slots* VL-BUS nas placas mães de seu microcomputador. Além disso, existe uma limitação quanto ao *clock* da placa mãe. Sem a utilização de circuitos adicionais *buffers*, que é de 50 MHz, e neste caso podendo conectar apenas uma placa VL-BUS no micro. Apesar de recente, este barramento foi substituído pelo padrão PCI (Porto, 1999).

O VL-Bus foi designado (1992) para oferecer uma melhor performance de gráficos e outros periféricos, este foi desenvolvido para PC´s 486 e Pentium, para ultrapassar a lentidão (16-bit ISA) encontrada nas máquinas 386. onde era conectado diretamente com o barramento local do microprocessador.

Os slots VLB são compostos de três conectores. Os dois primeiros são inteiramente compatíveis com os slots ISA (por isso, podemos conectar placas ISA de 8 e 16 bits nesses slots, usando a seção ISA), e um terceiro conector no qual é feita a transferência de dados em alta velocidade, e em grupos de 32 bits. A maior parte das transferências de dados é feita através deste terceiro conector.

É importante reconhecer a distinção entre Barramento Local e Barramento de Expansão. O Barramento Local não é algo que pode ser substituído por um barramento de expansão convencional como ISA, EISA ou MCA). Este é projetado fisicamente na placa mãe. Sua finalidade é permitir uma comunicação de alta velocidade entre o processados e periféricos também de alta velocidade como memória principal e memória *cache* (Torres, 1998).

O Barramento de Expansão, por outro lado, permite um número maior de dispositivos físicos, e fornece a compatibilidade inversa, ou seja, uma banda mais lenta e mais curta.

Há algumas limitações em usar uma tecnologia VL-Bus. Entre estas incluem uma limitação no número de dispositivos de VL-Bus, e uma largura de dados de no máximo 32-bits, e de um limite de velocidade de *clock* de 50MHz. Muitos destes problemas foram resolvidos com o surgimento do barramento PCI, que será abortado no próxima sessão. (Barreto, 2000).

3.11 BARRAMENTO PCI

Desenvolvido inicialmente pela Intel, os slots são de 32 bits e só aceitam placas desenvolvidas para esse padrão sendo uma mudança radical no projeto dos barramentos de expansão, abolindo totalmente a dependência de slot ISA. Permite as melhores taxas de transferência estando presente principalmente nos micros com chips Pentium (Porto, 1999).

Este barramento é independente do processador podendo ser implementado em qualquer arquitetura de processamento, ao contrário do VESA Local Bus, que foi desenvolvido especialmente para os 486.

O barramento PCI é um sistema que conecta um microprocessador e os dispositivos ligados a ele nos quais as expansões dos slots são colocados perto do processador para uma maior velocidade de operação. Usando PCI, o computador pode suportar tanto os novos tipos de PCI enquanto continua a suportar o barramento ISA, que é o tipo mais comum de expansão (Torres, 1999).

Desenvolvida pela Intel, o barramento PCI original era similar a Vesa local Bus. Contudo, o barramento PCI 2.0 é projetada para ser independente do processador. O barramento PCI foi projetado para ser sincronizado com a velocidade do relógio do processador, num alcance de 20 a 33 Mhz.

3.11.1 ARQUITETURA PCI

O barramento PCI, o qual está muito difundido é amplamente utilizado e está prestes a sepultar de vez o barramento ISA. Já é comum encontrar placas de som e modems que fazem uso do PCI. Território até então, dominado pelo barramento ISA. A grande vantagem é a taxa da transferência de dados (Porto 1999).

O barramento PCI são os slots (encaixes para placas) brancos (cor padrão) da placa mãe. Possuem uma taxa de transferência de 32 bits e 64 bits à uma velocidade de 33 e 66 MHz (geralmente, pode-se configurar nas placas mãe para ter um barramento de 33 MHz ou metade da freqüência de operação externa (configurações mais comuns), que é geralmente 66 MHz, com exceção do K6-2, Pentium II 300, 350, 400 que operam à 100 MHz e Cyrix M-II que opera à 75 MHz).

A ligação do barramento local com o barramento PCI chama-se *host-PCI bridge*. Já a ligação entre o barramento PCI com o ISA chama-se PCI-ISA *bridge*. Também são conhecidas como ponte norte e ponte sul.

O barramento PCI é o barramento mais utilizado atualmente, já que ele é muito mais rápido que os barramentos ISA, tem como vantagem o fato de trabalhar independentemente

do processador (trabalhar paralelamente a ele), além de possuir uma grande compatibilidade, uma vez que ele não funciona para computadores com processador Pentium, diferentemente dos barramentos intermediários entre ele e o ISA (Vasconcelos, 2002).

Inclusive, nas placas atuais o barramento ISA é acessado através do barramento PCI. As características principais deste barramento são:

- *Bus mastering* - O periférico pode controlar o barramento e acessar memória sem a interferência do processador, como em DMA;
- *Plug-And-Play*.

Existem vários modelos de barramento PCI. Podemos diferenciá-los de acordo com o tamanho de seu barramento de dados e com sua freqüência de operação que são (Torres, 1998):

- 32 bits a 33MHz (mais utilizados - taxa de transferência de 132 MBytes/s);
- 64 bits a 66MHz (264 Mbytes/s);
- 32 bits a 33MHz (264 Mbytes/s);
- 64 bits a 66MHz (528 Mbytes/s);
- Slots de 5V (mais comuns) e 3,3 V (66MHz utiliza sempre esta tensão). Existem placas específicas ou universais.

Outra vantagem ocorre em relação ao tamanho da placa, uma vez que ela se torna menor do que uma placa ISA, desta maneira ocupando um espaço menor dentro do gabinete, o que resulta em uma melhor circulação de ar no interior dele, possibilitando que o processador tenha um menor aquecimento, não só prolongando desta maneira sua vida útil, como também possibilitando uma redução na energia consumida pelo microcomputador (Torres, 1998).

Até pouco tempo, a maioria dos PCs equipados com processadores Pentium e superiores utilizavam placas de vídeo PCI. Depois da criação do barramento AGP, placas de vídeo AGP têm se tornado cada vez mais comuns, sendo que as vantagens deste tipo de barramento serão explicadas adiante.

Além das placas PCI para vídeo(caso de placas VGA e SVGA), podemos utilizar esse barramento outros tipos de placa, como por exemplo: Placa de rede PCI, Digitalizadoras de vídeo PCI, Controladoras SCSI PCI, Placas de som PCI entre outras.

No anexo 1 pode ser visto o esquema da pinagem, a descrição dos sinas e os ciclos do barramento PCI.

3.12 BARRAMETOS AGP

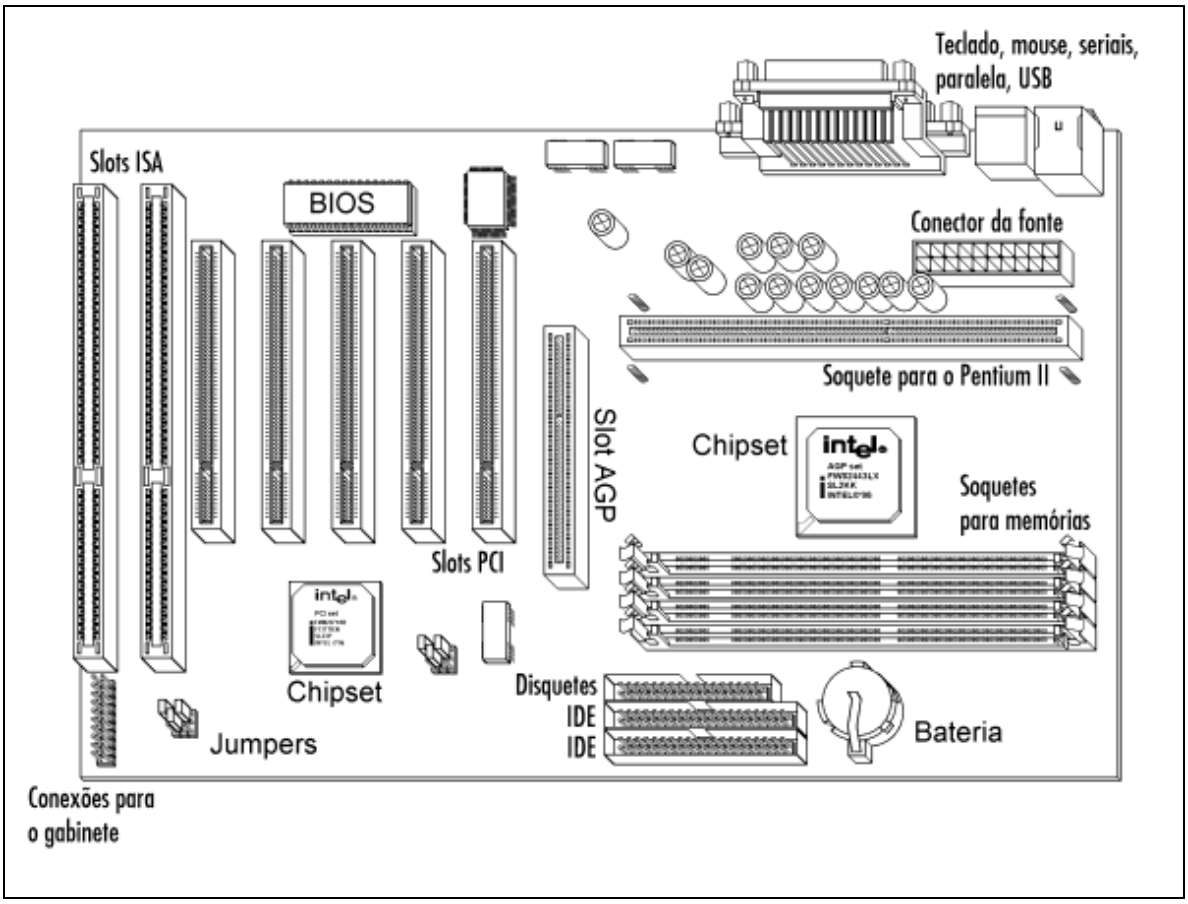
O Barramento *Accelerated Graphics Port* AGP, ou porta gráfica aceleradora foi criada pela Intel para resolver os problemas de desempenho das placas de vídeo 3D que utilizam o barramento PCI, como o uso de softwares e do computador em geral estão cada vez mais explorando áreas antes não utilizadas, como aceleração 3D e *playback* de vídeos de alta qualidade, tanto o processador quanto o *chipset* de vídeo precisam processar mais e mais informações. O barramento PCI está alcançando seus limites de performance nessas aplicações. Há uma demanda crescente por memória de vídeo, não apenas para a imagem na tela mas também para cálculos em 3D, o que exigiria mais memória na placa de vídeo, o que envolve dois problemas (Torres, 1999):

Custo: A memória da placa de vídeo é muito cara se comparada à memória RAM.

Tamanho: Se considerar uma placa de 6 MB, esta precisa de 4 MB para o frame buffer, assim sobrará 2 MB livres para o trabalho de processamento. Assim a única solução é trocar de placa pois não é fácil expandir essa memória, e não se pode usá-la para nenhuma outra função mesmo que não precise dela para o processamento de vídeo.

A Intel somente criou *chipsets* AGP para os microcomputadores Pentium Pro, e posteriores para controlar os *slots* AGP. O *chipsets* AGP tem a função de controlar o *slot* AGP permitindo que a placa de vídeo 3D acesse a memória RAM da placa mãe diretamente permitindo uma maior de taxa de transferência que o *slot* PCI. A figura 13 mostra a localização do *slot* APG na placa-mãe (Pinto, 1998).

Figura 13: Placa-mãe com slot AGP



Fonte: José Simão de Paula Pinto (1998).

A maioria das placas de vídeo modernas estão sendo fabricadas com um *slot* AGP. É cada vez mais raro ver placas de vídeo PCI. A maior vantagem do AGP é ser exclusivo da placa de vídeo, ao contrário do PCI, que é compartilhado por todos os periféricos instalados em slots PCI. O uso do conector AGP libera um *slot* PCI para outro uso nos novos microcomputadores (Torres, 1998).

O barramento AGP transfere dados a 66 MHz e 32 bits no modo x1 e sua taxa de transferência básica é calculada em **264 MB/s** (66.000.000 x 32 / 8).

O modo x2 e conseguido com o barramento externo comunicando-se com a memória RAM a 100 MHz (100.000.000 x 64 / 8 = **800 MB/s**), permitindo uma largura de banda maior que o barramento AGP que é igual a 528 MB/s.

No modo x1, a grande vantagem em relação ao barramento PCI é que o AGP roda com a velocidade total de barramento do sistema, 66 MHz, ao invés da metade, 33 MHz , como roda o PCI.

No modo x2, além dessa vantagem, o AGP consegue dobrar o desempenho de seu modo x1 mandando informação tanto na subida quanto na descida do *clock*.

O desempenho de 528 Mbits/s é apenas o pico teórico, pois como se sabe se a RAM será compartilhada pelo AGP com a CPU e dispositivos DMA, supõe-se que utilizará no máximo 50% da largura de banda do barramento, 264 Mbits/s.

Nos barramentos de 100 MHz, a análise muda significativamente, já que a largura de banda do barramento é calculada em 800 Mbits/s com SDRAM.

A Tabela 9 mostra as taxas de transferências conseguidas com o barramento.

Tabela 9: Taxas de transferências possíveis do barramento AGP

Modo	**Taxa de transferência máxima**	**Fórmula de Cálculo**
Modo x1	264 BM/s	((32 bits * 66 MHz)/8)
Modo x2	528 BM/s	((32 bits * 133 MHz)/8)
Modo x4	1 GB/s	((32 bits * 266 MHz)/8)
Modo x8	2 GB/s	((32 bits * 532 MHz)/8)

4 DESENVOLVIMENTO DO PROJETO

4.1 INTRODUÇÃO

O presente capítulo fornece detalhes sobre a implementação do protótipo da interface para conexão de duas placas-mães através do barramento ISA.

O desenvolvimento do projeto está dividido em duas etapas:

- Projeto de hardware;
- Projeto de Software.

A etapa de projeto de hardware é a criação das placas de interface para interligar os dois microcomputadores, que utiliza a arquitetura do barramento ISA descrito no capitulo anterior, bem como as técnicas utilizadas e detalhamentos das atividades realizadas.

A segunda etapa se caracteriza pela parte de software com a construção de dois módulos, um servidor ou master e outro cliente, onde o servidor controla o fluxo da informação enviada e recebida ao módulo cliente.

4.2 FERRAMENTAS E MATERIAIS UTILIZADOS NO PROJETO DE HARDWARE

A montagem foi realizada utilizando uma Protoboard, que é um suporte para montar protótipos. Os sistemas experimentais em eletrônica, antes de serem montados em placas de circuito impresso e soldados, são testados por software e como protótipos em um protoboard. O Protoboard consiste de uma placa de alumínio sobre a qual são presos conjuntos de estruturas plásticas vazadas. O protoboard apresenta abaixo dos quadrados de plástico, uma série de duas lâminas folheadas a ouro, entre as quais os fios ou os componentes serão conectados, estabelecendo um contato elétrico bastante razoável (Kleinke, 2001).

Para realizar os testes das conexões, correntes e tensões no circuito foi utilizado o multímetro que é um dispositivo capaz de medir intensidade da corrente, voltagem e resistência dos materiais; é um item muito importante no manuseio de equipamentos de eletrônica e informática, pois sua praticidade e capacidade de múltipla de medição faz com que eliminemos vários outros instrumentos, como o amperímetro e o voltímetro (Ribeiro, 2002).

Para verificar os sinais que estavam passando do circuito foi utilizado um osciloscópio, que é um dos instrumentos mais versáteis usados na eletrônica. Com ele podemos verificar um sinal elétrico e suas variações no tempo. O osciloscópio mostra o gráfico da tensão em função do tempo, foi usado aqui neste protótipo para ler os sinais que chegam nos circuitos.

Para a confecção dos esquemas das placas foi utilizado o editor de esquemas Protel Sch que acompanha o pacote de ferramenta do “Protel Design Explorer 99 SE” produto este um dos mais utilizado para montagens de esquemas e projetos de desenvolvimento de circuitos eletrônicos.

4.3 PROJETO DE HARDWARE

Para criar a placa de interface de comunicação primeiramente foi definida a utilização do barramento ISA, pois como o propósito deste trabalho é gerar uma interface de comunicação para futuramente utilizar técnicas de processamento paralelo, e com isto ter se um maior aproveitamento dos micro computadores de baixo custo, muitas vezes abandonados por já estarem ultrapassados, este barramento está presente em todos os modelos de PC deste o PC XT até os computadores atuais como foi descrito no capítulo anterior, e também por se tratar de um barramento mais simples e com vasta bibliografia.

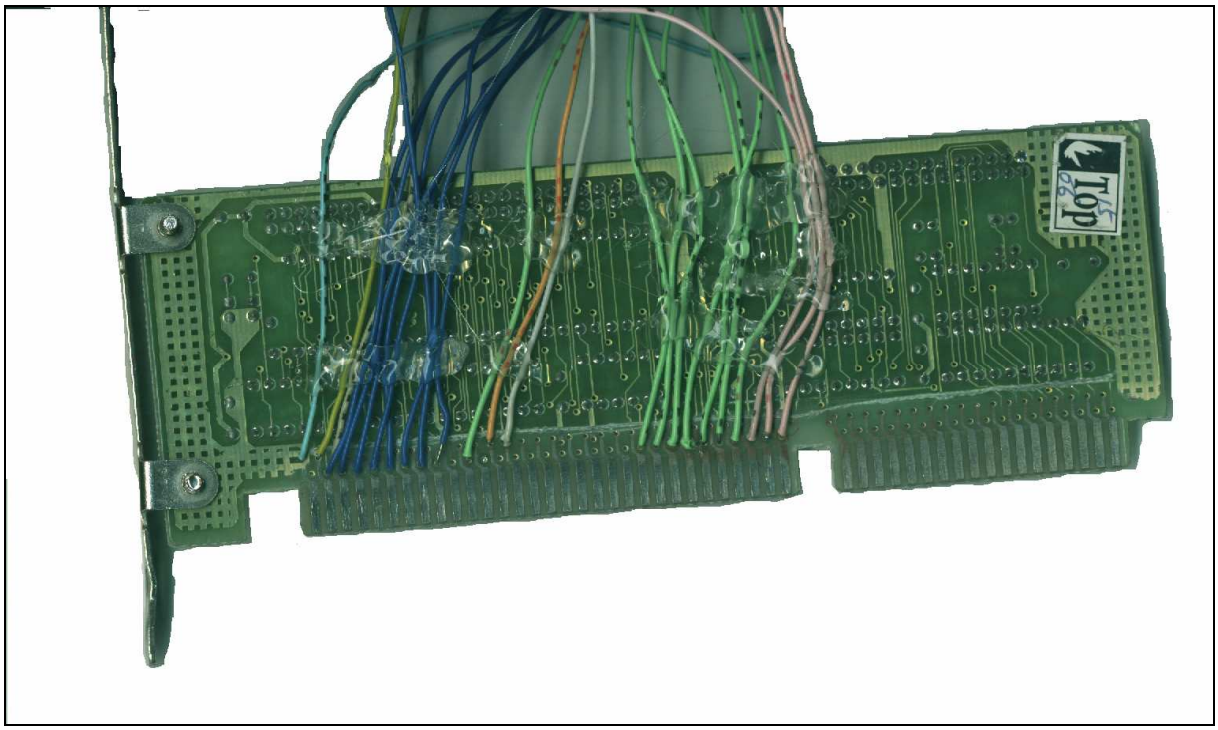
Para isto foi estudado o barramento ISA, seus sinais bem como o funcionamento de suas funções de escrita e leitura nas portas de E/S que foram detalhadas no tópico “3.6 CICLOS DE BARRAMENTO” do capítulo anterior. Também foi definido o endereço de E/S para as placas, pois como foi visto o processador consegue acessar os adaptadores através de suas portas ou endereços. Para este protótipo foi utilizado os endereços 300H e 301H para a placa master ou placa 1 e 310H e 311H para a placa escravo ou placa 2, estas portas são descritas como sendo portas de prototipação como podem ser visto na tabela 4.

Os endereços 300H e 310H foram utilizadas para se obter o status da placa, ou seja, se ela está disponível para escrita e/ou se possui dados para serem lidos, pois como foi dito na proposta este protótipo inicialmente não irá trabalhar com interrupção, e os endereços 301H e 311H foram utilizadas para enviar e receber os pacotes de dados. Foram utilizados dois endereços diferentes para status e dois também para dados porque na primeira etapa foi utilizado um único equipamento para se efetuar os testes e não poderiam ser postas as duas

placas com o mesmo endereço de E/S no mesmo equipamento, com isto teve-se que a placa 1 ficou com o endereço 300H para status e o endereço 301H para dados. E a placa 2 com o endereço 310h para status e 311H para dados.

Para se fazer o protótipo das placas foi pego uma placa do padrão ISA, desabilitados seu circuitos originais comum corte horizontal e soldado fios nos contados dos pinos que foram utilizados para confecção da interface de comunicação conforme figura 14.

Figura 14: Placa padrão ISA com os fios soldados nos contados utilizados pelo protótipo



Os sinais utilizados para confecção do protótipo foram:

- A0 a A9: sinais do barramento de endereço utilizados para endereçar a placa em questão;
- D0 a D7: sinais do barramento de dados do padrão ISA 8 bits utilizados para tratar o dado lido ou escrito no dispositivo;

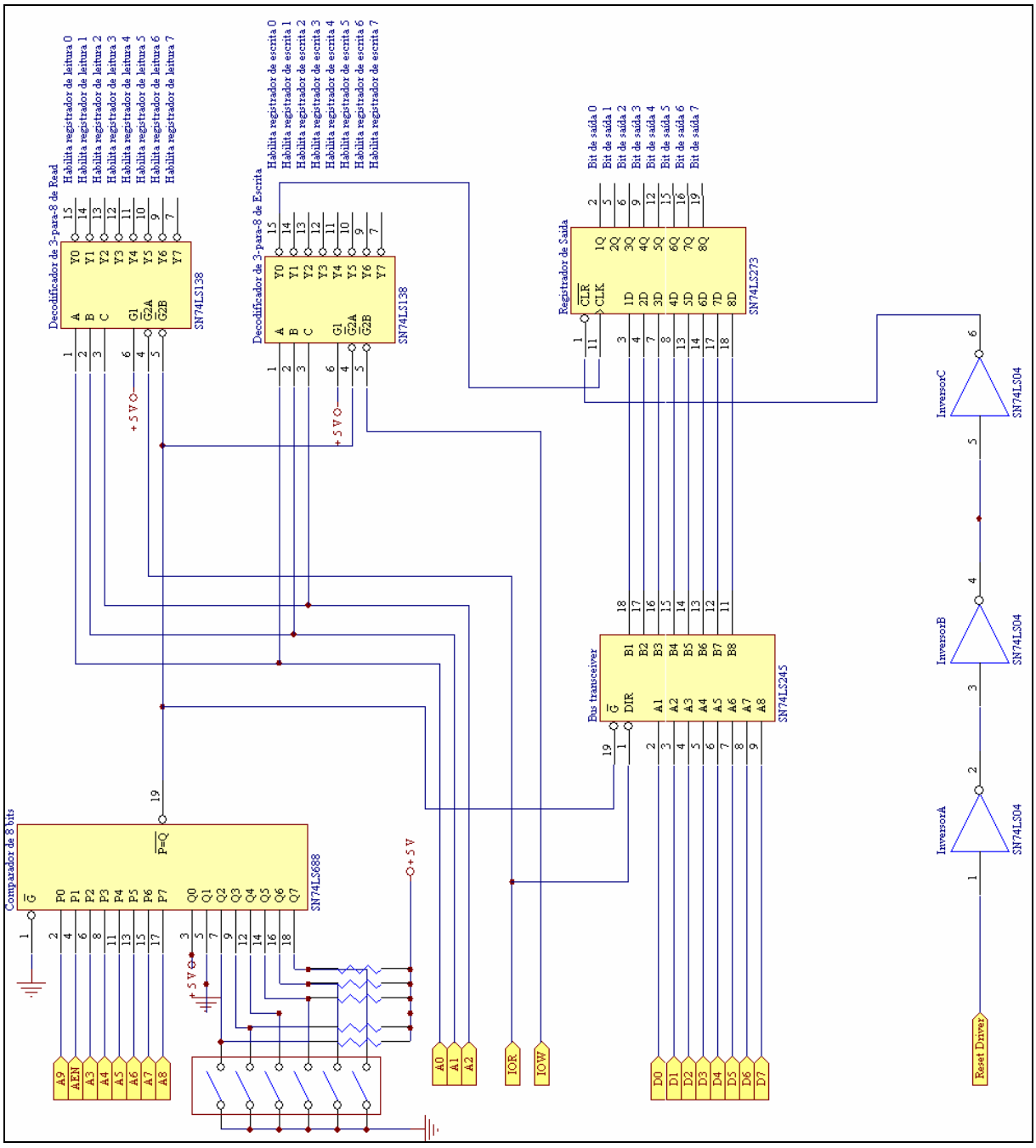
- AEN: sinal que quando vai a HIGH “Alta”, ou 5 volts indica que o barramento de endereço possui um endereço valido para um ciclo de leitura ou escrita;

- 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.

Na primeira etapa da criação da placa foi criado um circuito para teste de envio de dados ou a escrita na porta de dados da placa, conforme figura 15, esquema este retirado de Eggebrecht, 1995.

Figura 15: Diagrama esquemático do circuito da placa para escrita.



Adaptado de Lewis C. Eggebrecht (1995).

Para criação deste circuito foram utilizados os circuitos integrados SN74LS688, SN74LS245, SN74LS138, SN74LS273 e SN74LS04.

O CI SN74LS688 é um comparador de oito bis e foi utilizado para comparar os bits A3 a A9 do barramento de endereço e o sinal de controle AEN que indica que o endereço que está no barramento de endereço é um endereço válido, para habilitar a placa em questão. Para isto os sinais de barramento foram ligados nos pinos de entrada P0..P7 e os valores para comparação indicados no projeto acima por um *Dip-Switch*, Q0...Q7 foi ligado diretamente nos barramentos de alimentação do circuito (vcc + 5 V, e GND), de modo que represente a parte do bit 3 à 9 do valor do endereço da placa que no caso foi o endereço 300h, que em binário fica "1100000000", conforme tabela 10.

Tabela 10: Valores em binário do endereço 300H da placa de comunicação.

Sinal	**A9**	**A8**	**A7**	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
Endereço	1	1	0	0	0	0	0	0	0	0
Configuração do Dip Switch	Vcc	Vcc	GND	GND	GND	GND	GND	-	-	-

Quando o valor do contido no barramento de endereço coincidir com o valor a ser comparado, este circuito habilitará sua saída no pino 19, que está ligado aos circuitos SN74LS138 que são decodificadores do tipo multiplexadores de 3 entradas e 8 saída usadas para habilitar os registradores de dados e de controles utilizados neste projeto.

A saída do comparador está ligada a dois circuitos do tipo SN74LS138 descritos a cima, sendo que um está ligado ao sinal de barramento IOR (leitura na porta de IO), e o outro no IOW (Escrita na porta de IO) e cada uma desta operação poderá utilizar até oito registradores dependendo do valor dos três últimos bits (bits menos significativos), do endereço informado, conforme descritos tabela 11.

Tabela 11: Valores dos endereços possíveis de se endereçar na placa a partir do endereço base 300h.

Endereço	1100000**000**	1100000**001**	1100000**010**	1100000**011**	1100000**100**	1100000**101**	1100000**110**	1100000**111**
Reg habilitado	Reg0	Reg1	Reg2	Reg3	Reg4	Reg5	Reg6	Reg7

Para a parte de dados foi utilizado o componente SN74LS245 que é um *bus transceiver* bidirecional, que está ligado diretamente nos pinos do barramento de dados e serve tanto para mandar como para receber dados vindo do barramento. Para o controle da direção este componente possui um pino chama do de DIR (pino 1) que indica a direção do dado e possui um habilitador chamado de G (pino 19) que indica se o circuito está habilitado para transmitir o dados de um dado para o outro, ou seja, os conjuntos de pinos A1..A8 e B1..B8. onde o pino DIR está ligado ao sinal de barramento IOR que quando está em baixa, e o componente for habilitado por G que está ligado à saída do comparador, que indica que a porta selecionada pertence a esta placa, indica que está sendo executada uma instrução de leitura ou seja o circuito pega os valores que estão entrando no pinos B1...B8 e transfere para a saída de A1...A8, e quando o sinal de IOR estiver em baixa e o circuito for habilitado pelo comparador indica que uma instrução de IOW está sendo executada, e então o circuito pega os valores que estão chegando em A1..A8 e transfere para a saída em B1...B8, que por sua vez está ligado no CI SN74LS273, que é um circuito do tipo Flip-Flop utilizado para capturar o dado que vem do barramento de dados e enviá-lo para sua saída que estará lidado à entrada da outra placa.

Para testar este circuito foi feito um programa em pascal que escrevesse os valores de 0 a 255 da porta 300H e foi confirmado com a sua saída do SN74LS273, para isto cada pino da saída do SN74LS273 foi ligado a um LED para se validar o valor escrito com a saída do SN74LS273.

O programa para teste de escrita na porta está listado no quadro 1.

QUADRO 1: PROGRAMA DE TESTE DE ESCRITA.

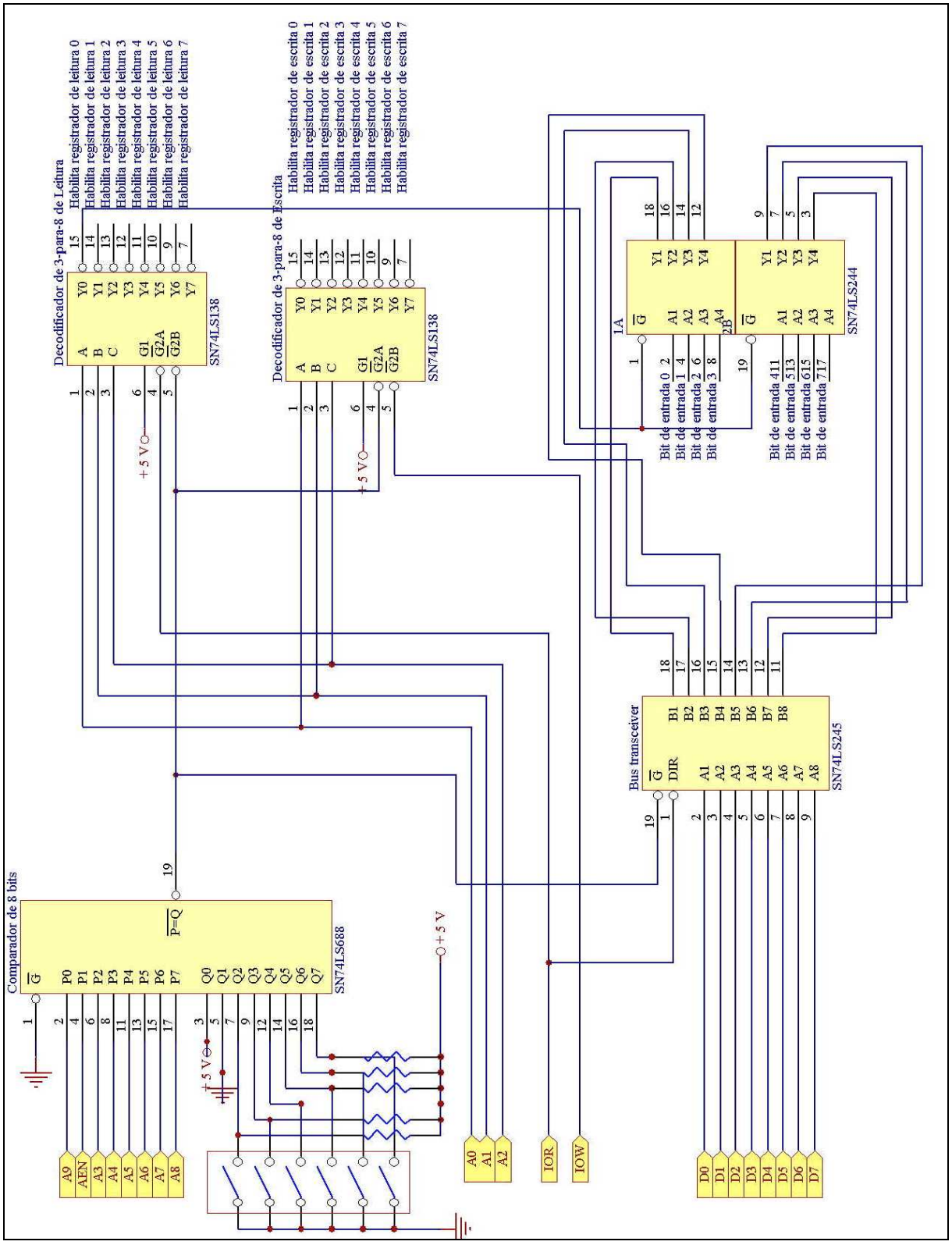
```
Program Testeplaca1;
Uses crt;

Var
   A : byte;

Begin

   For A := 0 to 255 do
   Begin
     Port[$300] := A;
     Delay(500);
   End;
End.
```

Para isto o circuito da placa foi montado em uma *protoboard*, que apresentou problema de mau contato. Então o circuito foi montado em uma placa universal que funcionou por um período mas depois devido a problemas elétricos deixou de funcionar, então foi decidido em conjunto com o orientador que iria se fazer todo o projeto na protoboard e para isto foi pego uma protoboard nova cedida pela FURB, e montado o circuito nesta protoboard que pode ser vista na figura 16.

Figura 16: Projeto do circuito confecção da placa para escrita.

O segundo passo na construção de um circuito para recepção e leitura dos dados escritos no circuito anterior. Para isto foi montado em outra protoboard um circuito semelhante ao anterior só que agora foi utilizado o CI SN74LS244 para receber os sinais de dados vindos da outra placa e enviá-los para o LS245, utilizando a habilitação saída do LS138 referente à leitura para fazer a liberação do dado armazenado no LS244 para o LS245 e daí diretamente para o barramento de dados do microcomputador, como pode ser visto na figura 17. Foi fixado como endereço inicial para esta placa o endereço 310H e podendo ser utilizados até o endereço 317H conforme foi mencionado na tabela 11 para o circuito anterior.

Figura 17: Diagrama esquemático do circuito da placa para leitura.



Para testar este circuito foi feito um programa em PASCAL que se escrevesse os valores de 0 a 255 da porta 300H e em seguida lê-lo no endereço 310H e comparando o valor escrito com o valor lido.

O programa para teste de escrita e leitura nas placas está descrito no quadro 2.

QUADRO 2: PROGRAMA DE TESTE DE ESCRITA E LEITURA.

```
Program Testeplaca1;
Uses crt;

Var
   A : byte;
   B : byte;

Begin

   For A := 0 to 255 do
   Begin
     B := 0;
     Port[$300] := A;
     b := Port[$310];
     Writeln('Valor escrito => ',A, ' Valor lido => ',B);
     Delay(500);
   End;
End.
```

O próximo passo foi a inclusão no circuito, o controle de status da comunicação. Para isto foi inserido em cada uma das placas um registrador de quatro bits, com três estados de saída, o SN74LS173, para indicar se existem dados para serem lidos ou se a interface está livre para escrita. Este registrador foi posto na placa reservando para ele mais um endereço na interface, que foi o endereço 300H na placa um, e 310H na placa dois com isto os registradores de dados da placa um e placa dois passaram para 301H e 311H consecutivamente.

Para isto o valor de D1 do CI foi ligado ao sinal de VCC do circuito. Os pinos M e N que quando estão em alta colocam a saída em alta impedância, impedido assim que o conteúdo do registrador vá para o barramento de dados que foi ligado a saída do registrador LS138 de leitura para a porta 300H ou 310H dependendo da placa, para que o conteúdo deste registrador seja enviado para o barramento somente quando for feita uma instrução de leitura no endereço de status das placas

Já o pino de CLK deste CI foi ligado à saída de escrita da porta 301H com sinal invertido com isto, quando se faz uma escrita na porta 301H (por exemplo na placa um), é

gerado um sinal de *clock* para o registrador LS173 que pega o conteúdo da entrada em D1..D4 e deixa disponível em Q1...Q4 esperando que seja lido com uma instrução de leitura no endereço 300H. Este sinal também é enfiado para o pino CLK do CI LS173 da placa do outro lado para que aja sincronismo nos registradores de status em ambos os lados.

O pino de CLR foi ligado à saída de habilitação de leitura do dado na placa 2 com valor invertido pois seu valor de saída do LS138 para habilitação é em baixa e o sinal do pino CLR para limpar os dados de saída do LS173 deve ser em alta, com isto quando se ler o valor do status em ambos os lados retornará 0, ou seja livre para escrita.

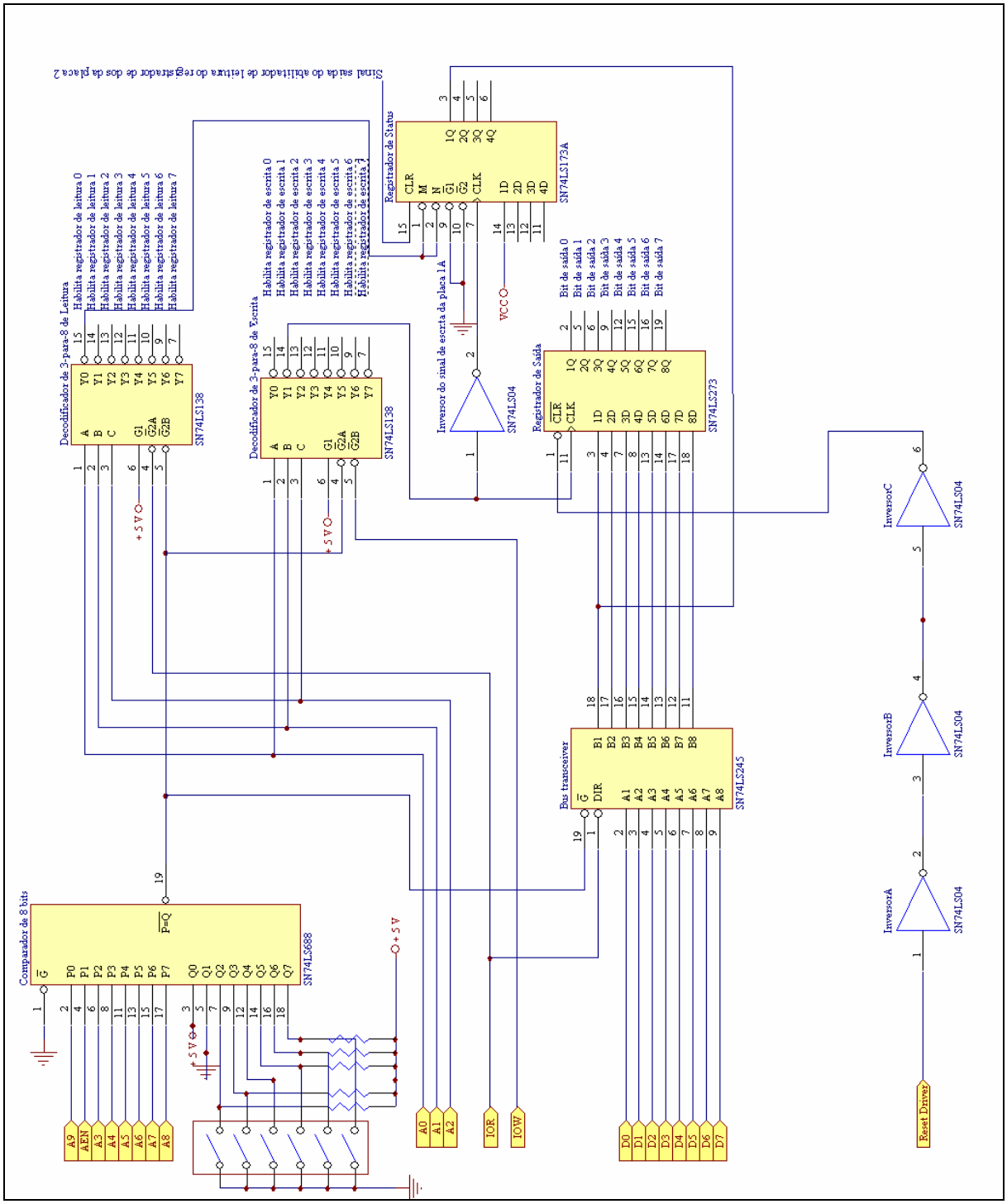
A figura 18 demonstra o esquemático deste circuito. Para testa-lo foi feito um programa em PASCAL que lê o status dos registradores na porta 300H e 310H depois escreve o dado na porta 301H então foram lidos novamente os valores dos registradores em ambas as placas para verificar o sincronismo de ambas, então é feito uma leitura do dado na porta 311H e por ultimo é lido o status dos registradores.

O programa para teste de escrita e leitura nas placas de teste é listado no quadro 3.

QUADRO 3: PROGRAMA DE TESTE DE ESCRITA E LEITURA COM OS SINAS DE STATUS.

```
Program Testeplaca3;
Uses crt;

Var
   A : byte;
   B : byte;
   StatusA : byte;
   StatusB : byte;

Begin

   A := 255;
   while Keypressed do
   Begin
     B := 0;
     {Lido o status das placas antes da escrita na placa 1}
     StatusA := Port[$300];
     StatusB := Port[$310];
     Write('Status A antes escrita: ',StatusA);
     write(' Status B antes escrita',Statusb);
     {Escrevendo o valor da varialvel A na porta 301h}
     Port[$301] := A;
     {Lido o status das placas apos a escrita da escrita na placa 1}
     StatusA := Port[$300];
     StatusB := Port[$310];
     Write('Status A apos escrita: ',StatusA);
     write(' Status B apos escrita',Statusb);
     {Lido o Valor do registrador na porta de dados da placa 2}
     B := Port[$311];
     {Lido o status das placas apos a leitura do dados na placa 2}
     StatusA := Port[$300];
     StatusB := Port[$310];
     Write('Status A apos escrita: ',StatusA);
     write(' Status B apos escrita',Statusb);
     Writeln('Valor escrito => ',A, ' Valor lido => ',B);
     Delay(500);
   End;
End.
```

Figura 18: Diagrama esquemático do circuito da placa de escrita e leitura com status.



O ultimo passo na parte de hardware foi a inclusão do circuito de leitura na porta de dados na placa um, e o circuito para escrita de dados na placa dois, como pode ser no

esquema da figura 19, e a confecção de um cabo para interligar as duas placas de interface, este foi feito utilizando com um cabo de IDE de 30 pinos, onde nas extremidades do mesmo foi soldado num soquete para conexão de CI´s.

Com isto quando foram feitos os primeiros testes, onde verificou-se a necessidade de mais um registrador de *status* em cada uma das placas pois ocorreu a seguinte situação:

> Quando a placa 1 enviou um byte de dados para a placa dois, o registrador de status ficou setado (ligado), indicando que não podia ser enviado outro byte enquanto o programa que utilizava a placa 2 não lesse o byte enviado para liberar o registrador. Mas também neste momento o programa que estava rodando na placa 1, após enviar o byte para a placa 2, precisava receber uma informação vinda da placa 2, e para isto ele precisava ficar checando o *status* do registrador para verificar quando o dado havia chegado. Então neste momento ele não tinham como controlar se o bit do registrador de status estava ligado por causa do dado que este havia enviado, ou se o registrador estava setado indicando que a programa da placa 2 não havia lido o byte enviado para ela.

Para resolver este problema foi inserido mais um LS173 em cada uma das placas, para poder controlar o status tanto de envio como e status de recepção de dados para as placas. Onde na placa 1 este foi colocado no endereço 302H, e para a placa dois como endereço 312H. Então o esquema de endereços das placas ficou da seguinte forma:

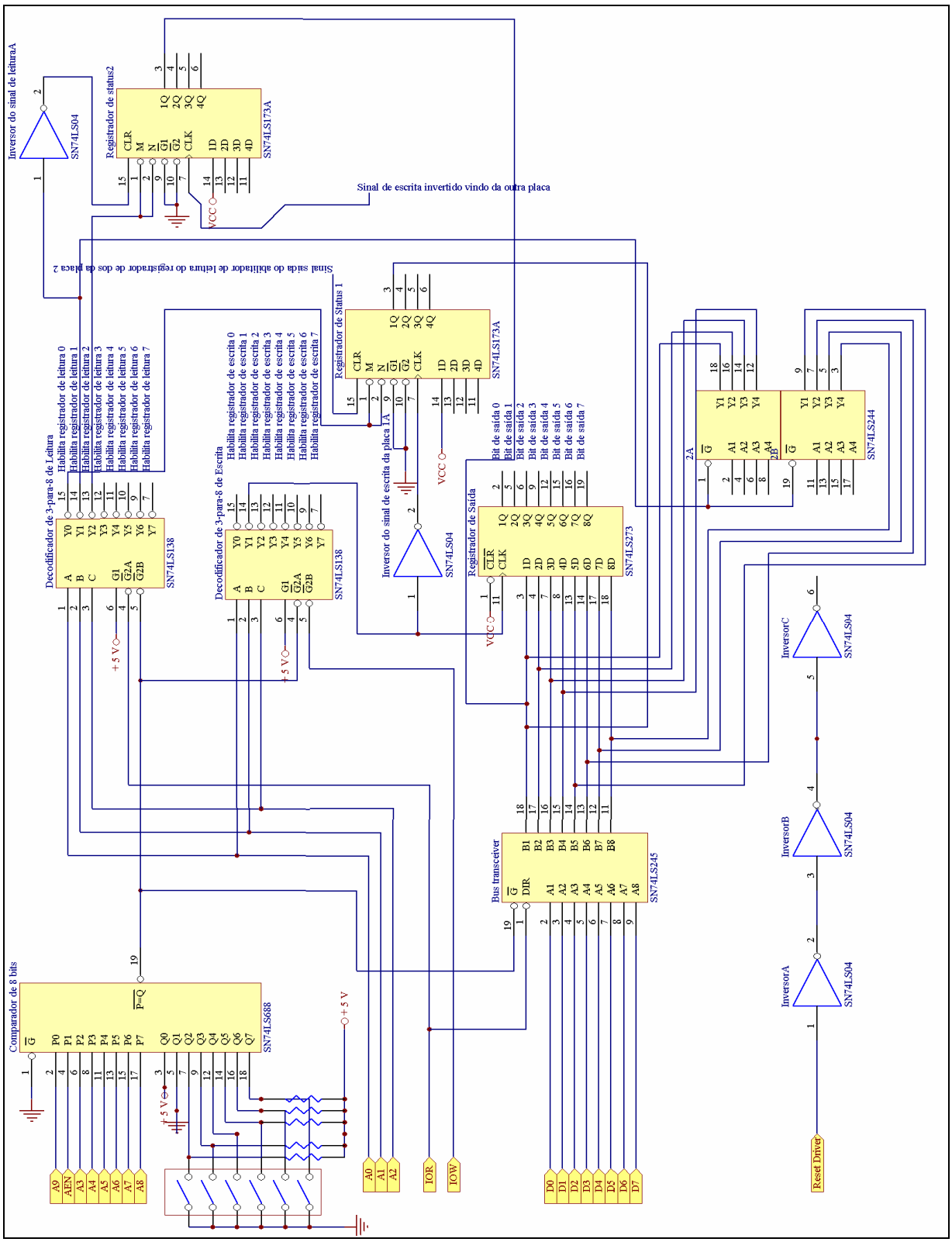
- Placa 1:
  - Endereço de status 300H indica se a interface está disponível para envio de dados do sentido placa 1 para placa 2, ou seja quando a placa 1 escreve um byte no endereço 301H este registrador de status fica com seu valor igual a 1 indicando que a interface do outro ainda não leu o dado, e quando o dado for lido do outro lado este registrador passa a ter seu valor igual a zero indicando que pode ser enviado um novo byte.
  - Endereço 301H é o registrador de dados da placa 1 serve tanto para enviar como para receber informações.

- Endereço 302H: quando este estiver o valor igual a 1, indica que o registrador de dados de leitura da placa 1 possui um novo dado enviado pela placa 2, e quando este for lido o valor deste registrador retorna para 0.

- Placa 2:
  - Endereço 310H: quando este estiver o valor igual a 1, indica que o registrador de dados de leitura da placa 2 possui um novo dado enviado pela placa 1, e quando este for lido o valor deste registrador retorna para 0.
  - Endereço 301H é o registrador de dados da placa 2 serve tanto para envia como para receber informações.
  - Endereço de status 312H indica se a interface esta disponível para envio de dados do sentido placa 2 para placa 1, ou seja quando a placa 2 escreve um byte no endereço 311H este registrador de status fica com seu valor igual a 1 indicando que a interface do outro ainda não leu o dado enviado, e quando o dado for lido do outro lado este registrador passa a ter seu valor igual a zero indicando que pode ser enviado um novo byte.

Também verificou-se quando foram feitos os testes das placas utilizando duas máquinas, que quando a era escrito um dado na placa 1, e o dado não era automaticamente lido na placa 2 o dado era perdido, ou seja quando o outro lado tentava ler o dado lia o valor zero. Estudado o circuito foi constatado que isto acontecia devido ao sinal de *Reset* do barramento estar ligado ao pino de *clear* do SN74LS273. Então foi tirado este sinal do componente e ligado a entrada deste pino no VCC do circuito, como pode ser visto na figura 19.

Figura 19: Diagrama esquemático do circuito da placa para escrita/leitura e status em uma única placa.



4.4 PROJETO DE SOFTWARE

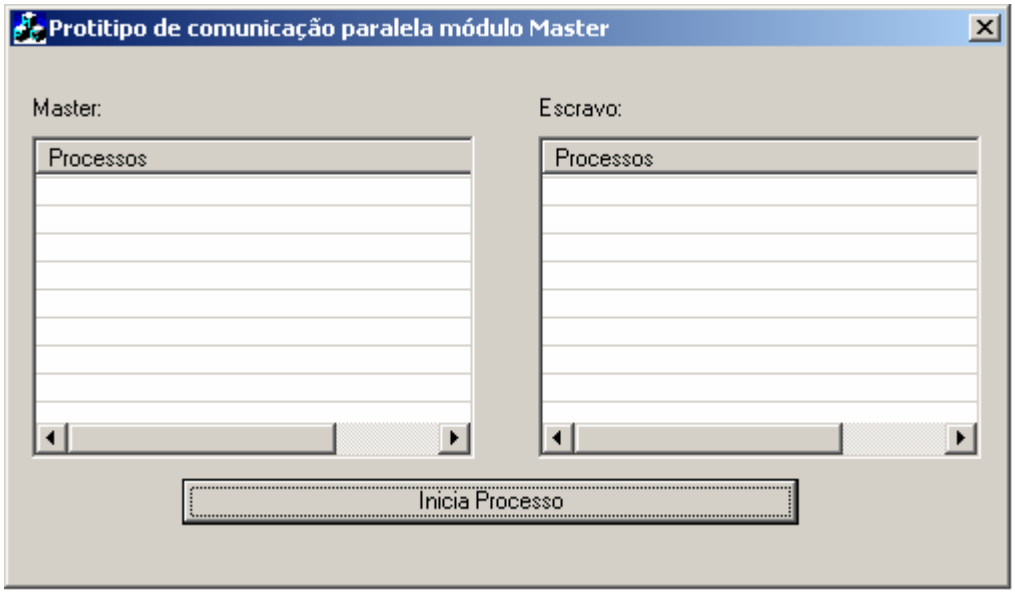
Para demonstrar aplicabilidade deste projeto de hardware inicialmente foi definida a criação de um driver para o controle da comunicação entre as duas placas. Por motivo de falta de tempo devido ao atraso na confecção das placas, foi decidido a criação de dois programas do tipo mestre e escravo, onde o módulo mestre é o que controla o fluxo da comunicação entre os módulos e faz o pedido de execução de uma tarefa pelo módulo escravo e quando este termina a execução da tarefa retorna o resultado para o módulo mestre e este recebe o resultado da tarefa e gera uma nova tarefa para o módulo escravo.

Para isto foi implementado um programa mestre e um programa escravo, feitos em VISUAL C++ 6.0, que é uma ferramenta de desenvolvimento da Microsoft, para a linguagem de programação C e C++, com as facilidades das técnica de programação visual.

4.4.1 PROGRAMA MESTRE

Este programa foi desenvolvido para ser executado no equipamento que estiver instalado a placa 1, que trabalha com os endereços de E/S 300H a 302H, descritos anteriormente no projeto de hardware, e possui a interface conforme figura 20.

Figura 20: Interface do programa Mestre.



A lista de processo da esquerda são as tarefas enviadas do programa mestre para o programa escravo, a lista da direita são as tarefas cumpridas pelo programa escravo e o botão “Inicia Processo” inicia a transmissão das tarefas que só é terminada com a finalização do programa.

Algumas considerações quanto as tarefas. O programa mestre somente manda uma tarefa por vez para o programa escravo, ou seja, ele manda a tarefa 1 e continua seu processo normal e quando este recebe a confirmação do término da tarefa então este manda a próxima tarefa. Para esta implementação a tarefa é somente uma seqüência de 255 caracteres ASCII tipo texto visível sem utilizar os caracteres de controle e formatação, o valor de 255 caracteres para o tamanho da tarefa é porque este é o tamanho máximo do pacote de dados a ser transmitido por vez para o programa escravo, pois a confirmação da quantidade de bytes enviados com a quantidade de bytes recebidos é de um byte ou seja de 0 a 255.

Para esta comunicação foram definidos alguns caracteres de controles para iniciar a conversa entre os programas e para controlar início e fim de transmissão descritos na tabela 12, e também foi definido um número máximo de tentativas que o processo ficará lendo uma determinada porta de status a espera para envio e recepção de dados antes que abandone o processo que foi de 10000 vezes.

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

Caracter (código ASCII)	Funcionalidade
01	Pedido de início de sincronismo para transmissão de pacote
02	Início de sincronismo para transmissão de pacotes aceito
03	Caracter que indica inicio do pacote de dados
04	Caracter que indica o final do pacote de dados
05	Tamanho do pacote recebido confere com o tamanho do pacote enviado
06	Tamanho do pacote recebido não confere com o tamanho do pacote enviado, pacote será retransmitido.

Ao se clicar no botão “Inicia Processo” este dispara as threads “TransmiteTarefa()” e “EsperaFinalTarefa(..)”.

A *thread* “TransmiteTarefa()” é a que faz a transmissão das tarefas para o programa escravo. Este processo inicia enviando para o programa escravo um caracter de pedido de início de sincronismo para transmissão de pacote, então fica esperando a confirmação do programa escravo com o recebimento do byte confirmação, byte 2, após o recebimento da confirmação envia byte de indicação de inicio de pacote, e em seguida inicia a transmissão do pacote, byte a byte. Após o término da transmissão do pacote envia o caracter de final de pacote, e aguarda o recebimento do byte que indica o tamanho do pacote enviado, quando recebe esta informação compara o tamanho do pacote enviado com o tamanho recebido pelo programa escravo caso os dois não sejam iguais envia o caracter “6” que indica que os tamanhos não conferem e inicia uma nova tentativa de transmissão, caso contrário envia do caracter que confirma tamanho do pacote.

Após a transmissão da tarefa para o programa escravo a *thread* seta a variável de controle do programa que indica que existe tarefa pendente com o programa escravo (m_boTaretaAtiva), e insere a tarefa na lista de processos enviados.

A *thread* EsperaFinalTarefa(..), é um laço eterno e só é encerrada com o final do programa mestre. Esta *thread* a cada passo do laço verifica se a variável de controle do programa que indica que existe tarefa pendente com o programa escravo, está habilitada, caso não esteja, espera 100 milisegundos e testa a variável novamente, caso contrário, fica lendo o endereço 302 para esperar até que venha a resposta do término da tarefa pelo escravo. Quando receber um dado verifica se o dado é o caracter de início de sincronismo de transmissão o que indica que o programa escravo terminou a tarefa, caso contrário ignora o dado recebido e volta a esperar dado para leitura.

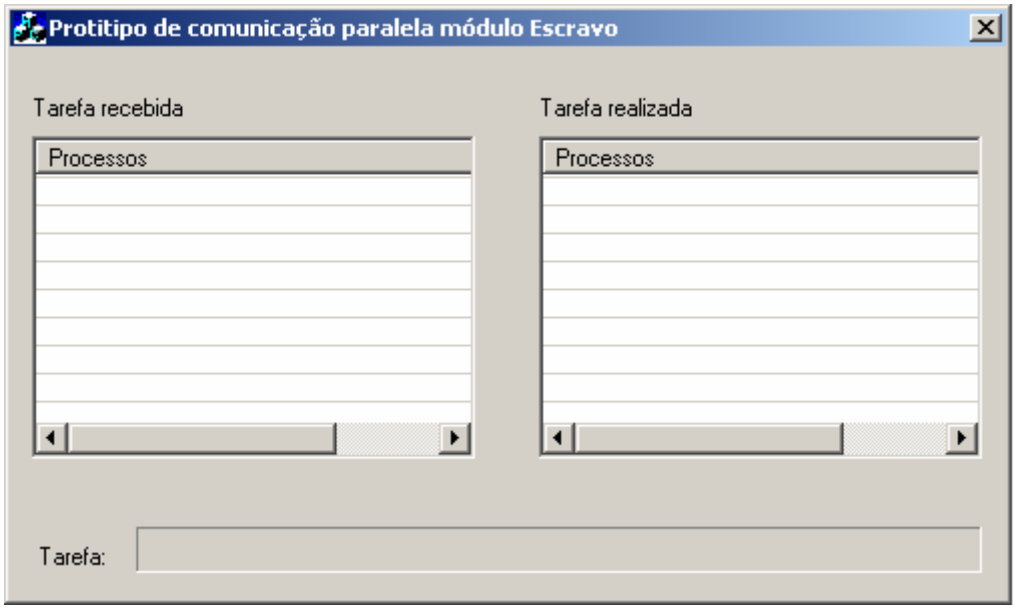
Depois de recebido o caracter de início de sincronismo envia o caracter de aceitação de início de sincronismo. Então entra em novo laço para o recebimento do pacote de dados que é a resposta do programa escravo, até receber o caracter de final de pacote, onde então envia a quantidade de bytes recebidos no pacote, e espera a confirmação do tamanho do

pacote recebido. Se a confirmação for negativa inicia o processo para recebimento do pacote novamente, e caso seja uma resposta positiva, seta a variável de controle de tarefa pendente no escravo, insere o conteúdo do pacote na lista de tarefas realizadas pelo programa escravo, e manda uma mensagem para o módulo principal do programa mestre para disparar o evento de Tarefa cumprida, que por sua vez dispara a *thread* TransmiteTarefa() novamente.

4.4.2 PROGRAMA ESCRAVO

O programa escravo possui uma interface similar ao programa mestre como pode ser visto na figura 21.

Figura 21: Tela programa escravo



Este programa possui duas listas, uma a esquerda que mostra as tarefas recebidas pelo programa mestre e outra a direita que mostra as tarefas realizadas. Também possui uma barra de processo que é utilizada pelo evento “OnExecutaTarefa”, que é ativado quando o programa recebe uma tarefa do programa mestre. Este evento que ativa a barra de tarefas para simulando a execução de uma tarefa e após o final deste processo envia para o programa mestre o pacote recebido como se fosse a conclusão da tarefa.

Quando este programa é inicializado ele dispara a *thread* “EsperaTarefa()” que é um laço eterno que só é encerrado com o término da aplicação. Esta *thread* fica checando o

recebimento de uma tarefa caso não aja nenhuma tarefa em execução, isto é controlado pela variável de controle do programa m_boTaretaAtiva.

Esta *thread* segue o mesmo algoritmo da *thread* EsperaFinalTarefa(..) do programa mestre somente modificando os endereços de E/S e que quando este recebe o pacote indica que uma tarefa foi recebida e não encerrada como no outro programa. Então após o recebimento da tarefa esta *thread* dispara o evento ”OnExecutaTarefa” e seta a variável m_boTaretaAtiva.

As figuras 22 e 23 mostram o fluxograma das rotinas de transmissão e recepção das tarefas do programa mestre para o programa escravo, onde estes também são muitos semelhantes para o programa escravo somente mudando os endereços de I/O.

Figura 22: Fluxograma da rotina de transmissão da tarefa

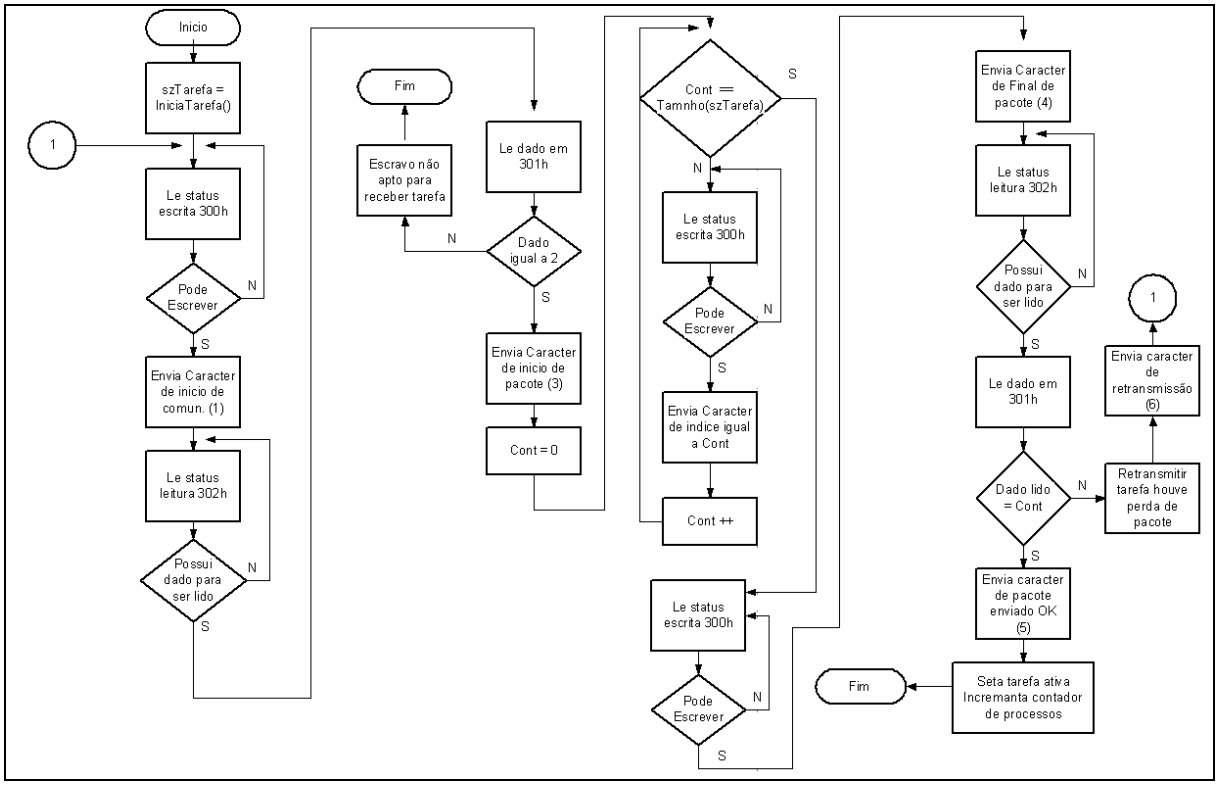
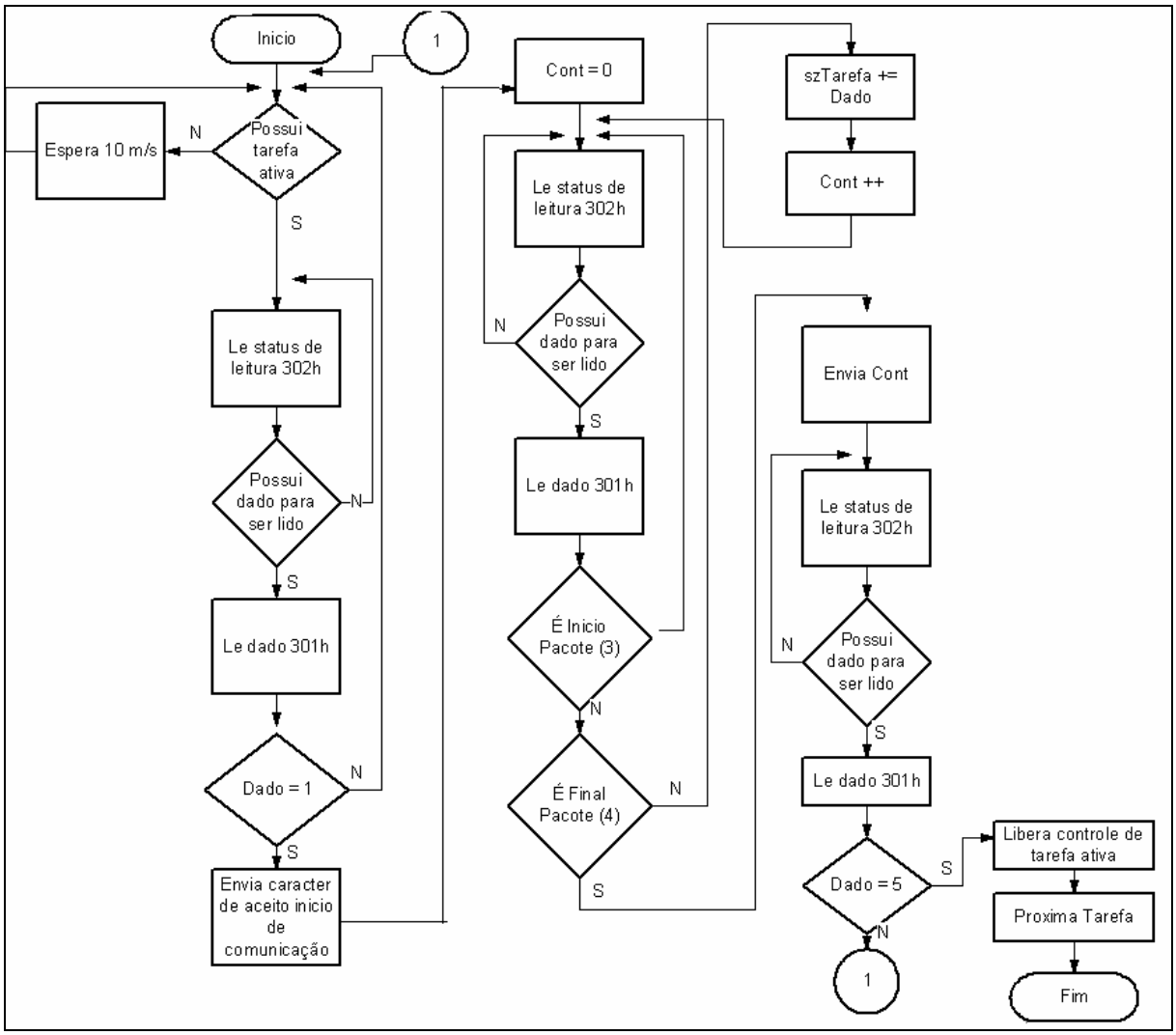


Figura 23: Fluxograma da rotina de recepção da tarefa



Nos Apêndices 1 e 2 pode ser visto o código fonte dos programas Mestre e Escravo, respectivamente.

5 RESULTADOS E DISCUSSÃO

Os testes iniciais durante o desenvolvimento da interface foram realizados com um programa escrito em Borland PASCAL 7.0 para sistema operacional MS-DOS, para não haver, influências de outros processos do sistema operacional como ocorreria no caso do Windows.

5.1 TAXA MÁXIMA DE TRANSFERÊNCIA ALCANÇADA

Desenvolveu-se o programa de testes descrito no quadro 4 que somente escreve os bytes na porta de endereço 301H para obter a banda máxima alcançada pela interface.

QUADRO 4: PROGRAMA TESTE DE BANDA DA PLACA DE TRANSMISSÃO.

```
program teste3;
uses crt, Dos;
var
   A : byte;
   b : byte;
   hi, hf, mi, mf, si, sf, hund, hundf : word;
   cont, conterro : longint;
begin
     ClrScr;
     A := 255;
     b := 0;
     hi := 0;
     hf := 0;
     mi := 0;
     mf := 0;
     si := 0;
     sf := 0;
     hund := 0;
     hundf :=0;
     GetTime(hi,mi,si,hund);
     for cont  := 1 to 10485760 do
     begin
       b := 0;
       port[$301] := a;
     end;
     GetTime(hf,mf,sf,hundf);
     writeln('hora inicial=> ',hi,':',mi,':',si,':',hund);
     writeln('hora final => ',hf,':',mf,':',sf,':',hundf);
     while not keypressed do;
end.
```

O resultado alcançado pelo programa é descrito na tabela 13. Para chegar a estes tempos foi primeiramente retirado a instrução de escrita interface e pego o tempo que o programa leva para ser executado, então foi habilitado a instrução de escrita e tomado os tempos então este tempo foi extraído o tempo de execução do programa sem a instrução de escrita e calculado a taxa de transferência.

Tabela 13: Taxa máxima de transferência da interface de comunicação.

Quantidade dados	Tempo/s	Bytes/s	Bits/s
1MB	1,37	747,45 KB/s	5,8 Mb/s
2MB	2,74	734 KB/s	5,8 Mb/s
3MB	4,12	745,63 KB/s	5,8 Mb/s
5MB	6,81	751,84 KB/s	5,8 Mb/s
10MB	13,63	751,28 KB/s	5,8 Mb/s

Estes valores ficaram abaixo do descrito na fundamentação teórica do barramento ISA, que como possui um freqüência de 8Mhz e uma banda de 8 bits, utilizando 4 ciclos para uma instrução de I/O deveria chegar em um valor aproximado de 2MB/s. Pesquisando as referencias viu-se em Eddggebrecht (1995) em PC-AT os tempos são diferentes dos encontrados no PC-XT. Segundo este mesmo autor o PC-AT com barramento do 8 bits consome 10 *clocks* para instruções de escrita e leitura em portas de I/O. Valor este que foi verificado nos testes realizados.

6 CONCLUSÕES

Este trabalho alcançou seu objetivo principal que era o de estudar os tipos de barramentos, e a criação de uma interface para interligar dois microcomputadores utilizando o barramento ISA.

Através dos resultados obtidos foi mostrado que e possível viabilizar a construção de uma máquina paralela utilizando placas-mães de baixo custo, utilizando a comunicação via extensão de barramento.

6.1 DIFICULDADES ENCONTRADAS

No decorrer do desenvolvimento deste trabalho, foram encontradas algumas dificuldades, podendo-se citar as seguintes:

- falta de conhecimento pessoal nos componentes utilizados para confecção dos circuitos.

- um baixo conhecimento em eletrônica o que veio a acarretar um pesquisa inicial pessoal nos conceitos de eletrônica e componentes eletrônicos.

6.2 SUGESTÕES PARA TRABALHOS FUTUROS

Trabalhos futuros poderão ser feitos utilizando as técnicas de interrupção, DMA e o padrão ISA de 16 bits o que não teria uma mudança muito grande em termos de projeto de interfaces e conhecimentos aqui relacionados, mais com ganhos significativos em termo de performance.

Um outro caminho a ser seguido é a criação desta interface utilizando o barramento PCI, também descrito aqui neste trabalho que também é um barramento que está presente na maioria dos equipamentos a partir do 386 e que possui uma taxa de transferência significativamente maior que a do barramento ISA.

E por último a construção de uma biblioteca de funções para a uso das técnicas de processamento paralelo utilizando esta interface.

REFERÊNCIAS BIBLIOGRÁFICAS

AMORIM, Cláudio L. de; BARBOSA, Valmir C.; FERNADES, Edil S. T. **Uma introdução à computação paralela e distribuída**. Campinas: R. Vieira Gráfica e Editora Ltda, 1988. 258p.

BARRETTO, Marcelo. **Microprocessadores**. Rio de Janeiro 2000. Disponível em <http://www.prodepa.gov.br/marcelo/Home%20Page/DocsMicro/>. Acesso em: 15 Jan. 2002.

EGGEBRECHT, Lewis C. **Interfacing to the IBM personal computer**. 2.ed. United States of America: SAMS, 1995. 345p.

CASTRO JUNIOR, Antonio Pires de; BUENO, Pedro Paulo Ferreira . **Grupo de usuários linux de goiás**. Goiás, set. 1999. Disponível em: <http://linuxgo.persogo.com.br/socket.html>. Acesso em: 18 Nov. 2001.

CIPELLI, Antonio Marco Vicari. **Teoria e desenvolvimento de projetos de circuitos eletrônico**s. 15.ed. São Paulo: Érica, 1990. 404p.

HWANG, kay; BRIGGS, Fayé A.. **Computer architecture and parallel processing.** New York: McGraw-Hill Book CO, 1991. 846 p.

KLEINKE, Maurício Urban. **Cusrso de Eletrônica**. Campinas, março de 2001. Disponível em: <http://www.ifi.unicamp.br/~kleinke/f540/e_dio1.htm#proto>. Acesso em: 11 Jun. 2002.

MACHADO, F. B., MAIA, L. P. **Introdução à Arquitetura de Sistemas Operacionais**. Rio de Janeiro: Ed. LTC, 1992.

MARCELINO, Rui. **Arquitetura dos PC´s**. Lisboa, [2002]. Disponível em: <http://w3.ualg.pt/~rmarcelino/ >. Acesso em: 01 jun. 2002.

MENDONÇA, Alexandre; ZELENOVSKY, Ricardo. **USB - estrutura de software**, São Paulo, jun de 1998 Disponível em: <http://www.gabrieltorres.com.br/usb1.html>. Acesso em: 17 Nov. 2001.

OGREN, Joakim et al. **The Hardware Book**. Jun. 2001. Disponivel em: <http://www.hardwarebook.net/ >. Acesso em: 01 maio 2002.

PELISSON, Luiz Augusto. **Fundamentos da informática**. São Paulo, maio 2001. Disponível em: < http://www.dainf.cefetpr.br/~pelisson/barram01.htm>. Acesso em: 15 Nov. 2001.

PINTO, José Simão de Paula. Arquitetura de micros IBM PC. Curitiba, fevereiro 1998. Disponível em: < http://www.grotta.8k.com/tut/arquitetura/ >. Acesso em: 14 abr. 2002.

PORTO, Marcelo Franco. **Periféricos e Interfaces.** Belo Horizonte [1999]. Disponível em: <http://www.dcc.pucminas.br/computacao/disciplinas/pi/semestre_99/usb/index.htm>. Acesso em: 15 Nov. 2001.

RIBEIRO, Marcio R.. **Mini-curso de eletricidade para alunos de 2º grau**. São Paulo [2002]. Disponível em: <http://fisica.cdcc.sc.usp.br/ce/aula03/aula003.htm>. Acesso em 10 Jun. 2002.

SERAGGI, Márcio Roberto**. Entenda a memória do seu computador**, São Paulo, fev. 2001. Disponível em: < http://www.seraggi.page.com.br/html/memoria.html>. Acesso em: 24 Nov. 2001.

TORRES, Gabriel. **Hardware curso completo.** 2.ed. - Rio de Janeiro : Axcel Books, 1998. 894p.

TORRES, Gabriel. **Hardware curso completo.** 4. ed. Rio de Janeiro: Axcel Books, 1999. 893 p.

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

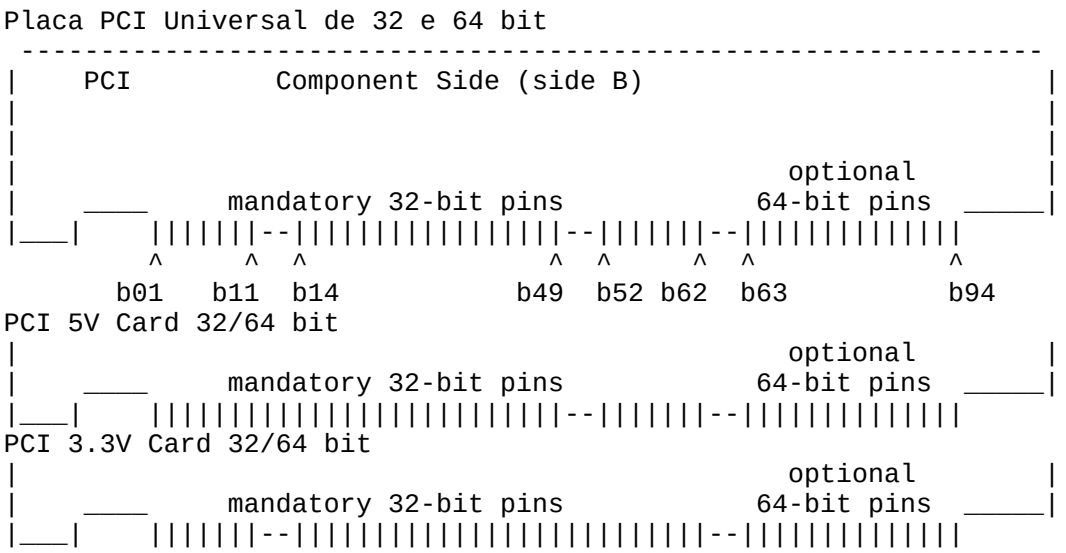
WIRTH, Niklaus. **Digital circuit design for computer science students : an introductory textbook**. Berlin : Springer, 1995. xiii, 204p.

TANENBAUM, Andrew S. **Structured Computer Organization.** 4th ed. Prentice-Hall Inc, 1999. xviii, 669p.

VASCONCELOS, Laércio. **Placas de CPU e barramentos**. São Paulo [2002] Disponível em: <http://www.laercio.com.br/>. Acesso em 10 Jun. 2002.

ZEFERINO, Cesar Albenes et al. Um multicomputador com sistema experimental de comunicação. In Simpósio Brasileiro de Arquitetura de Computadores – Processamento de Alto Desempenho, 7. 1995, Canela. **Anais** ... Canela: UFRGS, 1995. p. 137-150.

ANEXO 1: PINAGEM DO BARRAMENTO PCI



Características dos pinos:

Descrição da pinagem do slot PIC

Pin	+5V	+3.3V	Universal	Description
A1	TRST			Test Logic Reset
A2	+12V			+12 VDC
A3	TMS			Test Mde Select
A4	TDI			Test Data Input
A5	+5V			+5 VDC
A6	INTA			Interrupt A
A7	INTC			Interrupt C
A8	+5V			+5 VDC
A9	RESV01			Reserved VDC
A10	+5V	+3.3V	Signal Rail	+V I/O (+5 V or +3.3 V)
A11	RESV03			Reserved VDC
A12	GND03	(OPEN)	(OPEN)	Ground or Open (Key)
A13	GND05	(OPEN)	(OPEN)	Ground or Open (Key)
A14	RESV05			Reserved VDC
A15	RESET			Reset

A16	+5V	+3.3V	Signal Rail	+V I/O (+5 V or +3.3 V)
A17	GNT			Grant PCI use
A18	GND08			Ground
A19	RESV06			Reserved VDC
A20	AD30			Address/Data 30
A21	+3.3V01			+3.3 VDC
A22	AD28			Address/Data 28
A23	AD26			Address/Data 26
A24	GND10			Ground
A25	AD24			Address/Data 24
A26	IDSEL			Initialization Device Select
A27	+3.3V03			+3.3 VDC
A28	AD22			Address/Data 22
A29	AD20			Address/Data 20
A30	GND12			Ground
A31	AD18			Address/Data 18
A32	AD16			Address/Data 16
A33	+3.3V05			+3.3 VDC
A34	FRAME			Address or Data phase
A35	GND14			Ground
A36	TRDY			Target Ready
A37	GND15			Ground
A38	STOP			Stop Transfer Cycle
A39	+3.3V07			+3.3 VDC
A40	SDONE			Snoop Done
A41	SBO			Snoop Backoff
A42	GND17			Ground
A43	PAR			Parity
A44	AD15			Address/Data 15
A45	+3.3V10			+3.3 VDC
A46	AD13			Address/Data 13
A47	AD11			Address/Data 11

A48	GND19			Ground
A49	AD9			Address/Data 9
A52	C/BE0			Command, Byte Enable 0
A53	+3.3V11			+3.3 VDC
A54	AD6			Address/Data 6
A55	AD4			Address/Data 4
A56	GND21			Ground
A57	AD2			Address/Data 2
A58	AD0			Address/Data 0
A59	+5V	+3.3V	Signal Rail	+V I/O (+5 V or +3.3 V)
A60	REQ64			Request 64 bit ???
A61	VCC11			+5 VDC
A62	VCC13			+5 VDC
A63	GND			Ground
A64	C/BE[7]#			Command, Byte Enable 7
A65	C/BE[5]#			Command, Byte Enable 5
A66	+5V	+3.3V	Signal Rail	+V I/O (+5 V or +3.3 V)
A67	PAR64			Parity 64 ???
A68	AD62			Address/Data 62
A69	GND			Ground
A70	AD60			Address/Data 60
A71	AD58			Address/Data 58
A72	GND			Ground
A73	AD56			Address/Data 56
A74	AD54			Address/Data 54
A75	+5V	+3.3V	Signal Rail	+V I/O (+5 V or +3.3 V)
A76	AD52			Address/Data 52
A77	AD50			Address/Data 50
A78	GND			Ground
A79	AD48			Address/Data 48
A80	AD46			Address/Data 46

A81	GND			Ground
A82	AD44			Address/Data 44
A83	AD42			Address/Data 42
A84	+5V	+3.3V	Signal Rail	+V I/O (+5 V or +3.3 V)
A85	AD40			Address/Data 40
A86	AD38			Address/Data 38
A87	GND			Ground
A88	AD36			Address/Data 36
A89	AD34			Address/Data 34
A90	GND			Ground
A91	AD32			Address/Data 32
A92	RES			Reserved
A93	GND			Ground
A94	RES			Reserved
B1	-12V			-12 VDC
B2	TCK			Test Clock
B3	GND			Ground
B4	TDO			Test Data Output
B5	+5V			+5 VDC
B6	+5V			+5 VDC
B7	INTB			Interrupt B
B8	INTD			Interrupt D
B9	PRSNT1			Reserved
B10	RES			+V I/O (+5 V or +3.3 V)
B11	PRSNT2			??
B12	GND	(OPEN)	(OPEN)	Ground or Open (Key)
B13	GND	(OPEN)	(OPEN)	Ground or Open (Key)
B14	RES			Reserved VDC
B15	GND			Reset
B16	CLK			Clock
B17	GND			Ground

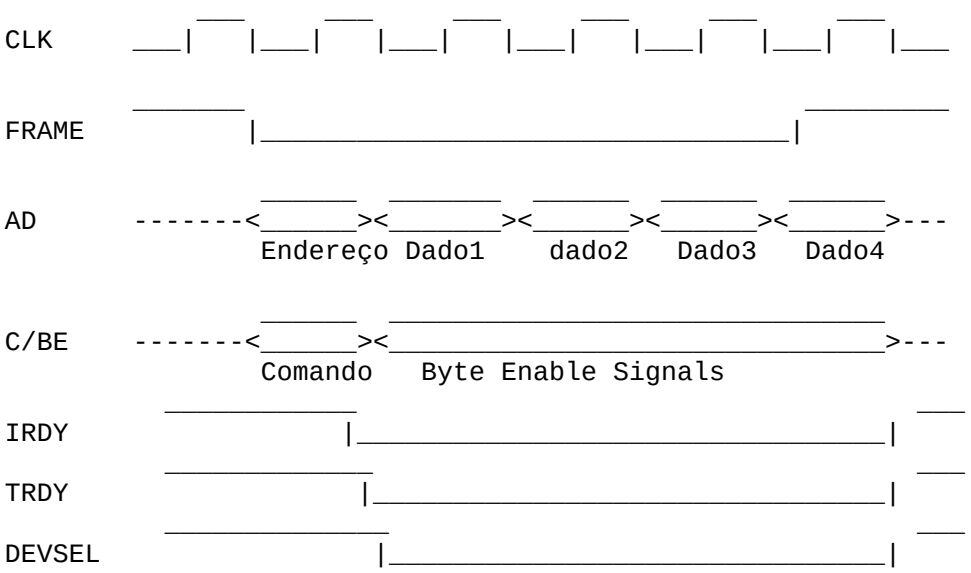
B18	REQ			Request
B19	+5V	+3.3V	Signal Rail	+V I/O (+5 V or +3.3 V)
B20	AD31			Address/Data 31
B21	AD29			Address/Data 29
B22	GND			Ground
B23	AD27			Address/Data 27
B24	AD25			Address/Data 25
B25	+3.3V			+3.3VDC
B26	C/BE3			Command, Byte Enable 3
B27	AD23			Address/Data 23
B28	GND			Ground
B29	AD21			Address/Data 21
B30	AD19			Address/Data 19
B31	+3.3V			+3.3 VDC
B32	AD17			Address/Data 17
B33	C/BE2			Command, Byte Enable 2
B34	GND13			Ground
B35	IRDY			Initiator Ready
B36	+3.3V06			+3.3 VDC
B37	DEVSEL			Device Select
B38	GND16			Ground
B39	LOCK			Lock bus
B40	PERR			Parity Error
B41	+3.3V08			+3.3 VDC
B42	SERR			System Error
B43	+3.3V09			+3.3 VDC
B44	C/BE1			Command, Byte Enable 1
B45	AD14			Address/Data 14
B46	GND18			Ground
B47	AD12			Address/Data 12
B48	AD10			Address/Data 10
B49	GND20			Ground

B50	(OPEN)	GND	(OPEN)	Ground or Open (Key)
B51	(OPEN)	GND	(OPEN)	Ground or Open (Key)
B52	AD8			Address/Data 8
B53	AD7			Address/Data 7
B54	+3.3V12			+3.3 VDC
B55	AD5			Address/Data 5
B56	AD3			Address/Data 3
B57	GND22			Ground
B58	AD1			Address/Data 1
B59	VCC08			+5 VDC
B60	ACK64			Acknowledge 64 bit ???
B61	VCC10			+5 VDC
B62	VCC12			+5 VDC
B63	RES			Reserved
B64	GND			Ground
B65	C/BE[6]#			Command, Byte Enable 6
B66	C/BE[4]#			Command, Byte Enable 4
B67	GND			Ground
B68	AD63			Address/Data 63
B69	AD61			Address/Data 61
B70	+5V	+3.3V	Signal Rail	+V I/O (+5 V or +3.3 V)
B71	AD59			Address/Data 59
B72	AD57			Address/Data 57
B73	GND			Ground
B74	AD55			Address/Data 55
B75	AD53			Address/Data 53
B76	GND			Ground
B77	AD51			Address/Data 51
B78	AD49			Address/Data 49
B79	+5V	+3.3V	Signal Rail	+V I/O (+5 V or +3.3 V)
B80	AD47			Address/Data 47

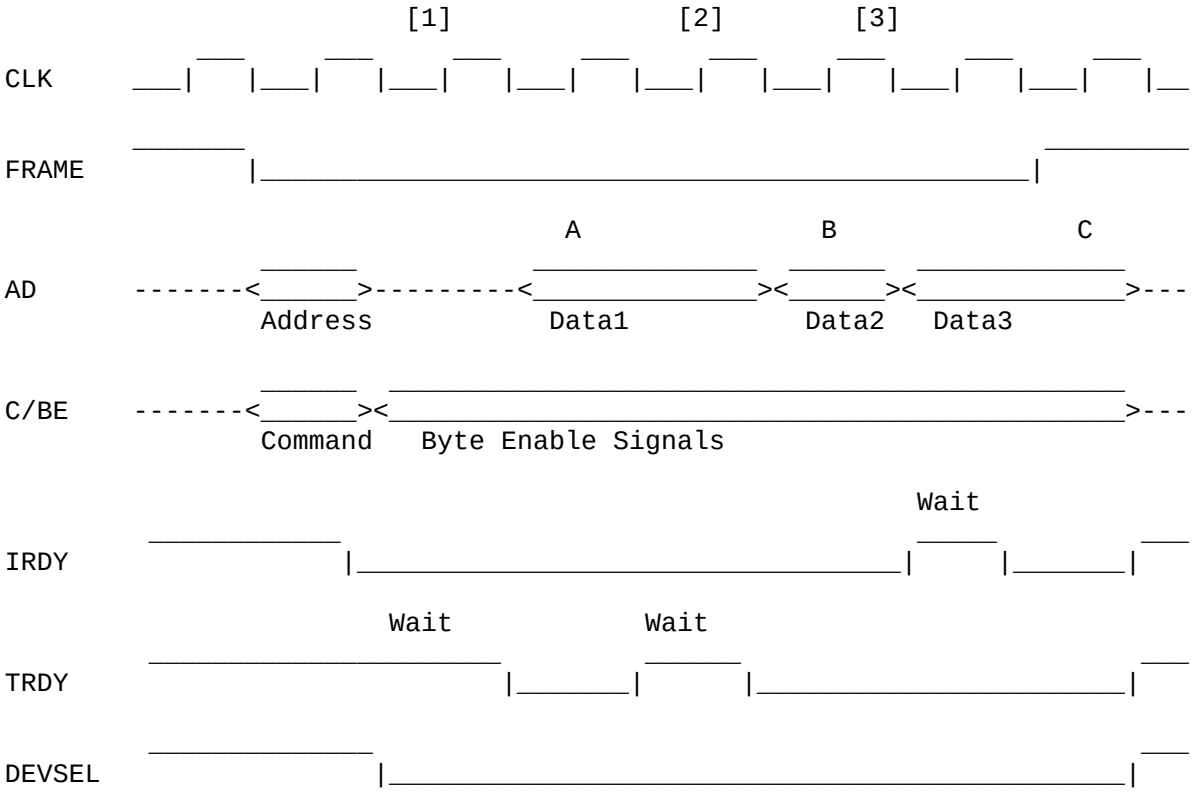
B81	AD45			Address/Data 45
B82	GND			Ground
B83	AD43			Address/Data 43
B84	AD41			Address/Data 41
B85	GND			Ground
B86	AD39			Address/Data 39
B87	AD37			Address/Data 37
B88	+5V	+3.3V	Signal Rail	+V I/O (+5 V or +3.3 V)
B89	AD35			Address/Data 35
B90	AD33			Address/Data 33
B91	GND			Ground
B92	RES			Reserved
B93	RES			Reserved
B94	GND			Ground

Diagrama de tempo:

Será mostrado o comportamento dos sinais desse dispositivo:



PCI transfer cycle, 4 data phases, no wait states. Data is transferred on the rising edge of CLK.



PCI transfer cycle, with wait states.

Fonte: (Ogren, 1997)

APÊNDICE 1: CÓDIGO FONTE DO PROGRAMA MESTRE

```
// MasterDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Master.h"
#include "MasterDlg.h"
#include "conio.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define TEMPO_MAX_ESPERA 10000
#define MSG_PROXIMA_TAREFA (WM_APP + 101)

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}
```

```
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMasterDlg dialog

CMasterDlg::CMasterDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CMasterDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CMasterDlg)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
	m_boTaretaAtiva	= false;
	m_iIdProcesso= 1;

	m_hThread		= NULL;
	m_dwIDThread = NULL;

	m_hThreadEspera		= NULL;
	m_dwIDThreadEspera	= NULL;

	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

/////////////////////////////////////////////////////////////////////////////
//	Função...:	Destrutor
//
//	Descrição:	Quando a tela/classe for destruída, irá testar novamente
//		se as threads estão rodando, Caso afirmativo, elas serão
//		eliminadas.
//
//
/////////////////////////////////////////////////////////////////////////////
CMasterDlg::~CMasterDlg()
{
	DWORD dwExit;
	::GetExitCodeThread(m_hThread,&dwExit);
	while (dwExit == STILL_ACTIVE)
		::GetExitCodeThread(m_hThread,&dwExit);
	if (m_hThread)
		::CloseHandle(m_hThread);
```

```
	::GetExitCodeThread(m_hThreadEspera,&dwExit);
	while (dwExit == STILL_ACTIVE)
		::GetExitCodeThread(m_hThreadEspera,&dwExit);
	if (m_hThreadEspera)
		::CloseHandle(m_hThreadEspera);
}


void CMasterDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CMasterDlg)
	DDX_Control(pDX, IDC_BNT_INICIA_PROCESSO, m_bntInitProcesso);
	DDX_Control(pDX, IDC_LIST_MASTER, m_lstListaMaster);
	DDX_Control(pDX, IDC_LIST_ESCRAVO, m_lstListaEscravo);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CMasterDlg, CDialog)
	//{{AFX_MSG_MAP(CMasterDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BNT_INICIA_PROCESSO, OnBntIniciaProcesso)
	//}}AFX_MSG_MAP
	ON_MESSAGE(MSG_PROXIMA_TAREFA, OnProximaTarefa)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMasterDlg message handlers

BOOL CMasterDlg::OnInitDialog()
{
	CDialog::OnInitDialog();


	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
```

```
                pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
            }
        }

        // Set the icon for this dialog.  The framework does this automatically
        //  when the application's main window is not a dialog
        SetIcon(m_hIcon, TRUE);                 // Set big icon
        SetIcon(m_hIcon, FALSE);          // Set small icon

        m_lstListaMaster.DeleteAllItems();
        m_lstListaEscravo.DeleteAllItems();

        // função usada para deixar selecionar tota a linha no list
        ListView_SetExtendedListViewStyle((HWND)                    m_lstListaMaster,
LVS_EX_FULLROWSELECT | LVS_EX_GRIDLINES);
        ListView_SetExtendedListViewStyle((HWND)                   m_lstListaEscravo,
LVS_EX_FULLROWSELECT | LVS_EX_GRIDLINES);

        // Cleverson dos Santos Inicializa o gride da tela [20/6/2002]
        while (m_lstListaMaster.DeleteColumn(0));
        while (m_lstListaEscravo.DeleteColumn(0));


        // insere coluna do gride para os vertices da lista de adjacencia
        LVCOLUMN col;
        col.mask = LVCF_FMT | LVCF_TEXT | LVCF_WIDTH;
        col.fmt  = LVCFMT_LEFT;
        col.pszText = _T("Processos");
        col.cx = 300;
        m_lstListaMaster.InsertColumn(0, &col);

        m_lstListaEscravo.InsertColumn(0, &col);


        // TODO: Add extra initialization here

        return TRUE;  // return TRUE  unless you set the focus to a control
}

void CMasterDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
                CAboutDlg dlgAbout;
                dlgAbout.DoModal();
        }
        else
        {
                CDialog::OnSysCommand(nID, lParam);
        }
}
```

```
// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CMasterDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CMasterDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}

void CMasterDlg::OnBntIniciaProcesso()
{
	if (!m_boTaretaAtiva)
	{
		m_hThread                                                                    =
::CreateThread(NULL,NULL,(LPTHREAD_START_ROUTINE)TransmiteTarefa,static_cast<LPVOID
>(this),CREATE_SUSPENDED,&m_dwIDThread);
		::SetThreadPriority(m_hThread,THREAD_PRIORITY_NORMAL);
		::ResumeThread(m_hThread);

		m_hThreadEspera                                                              =
::CreateThread(NULL,NULL,(LPTHREAD_START_ROUTINE)EsperaFinalTarefa,static_cast<LPVO
ID>(this),CREATE_SUSPENDED,&m_dwIDThreadEspera);
		::SetThreadPriority(m_hThreadEspera,THREAD_PRIORITY_NORMAL);
		::ResumeThread(m_hThreadEspera);
```

```
                m_bntInitProcesso.EnableWindow(FALSE);
        }

}


//////////////////////////////////////////////////////////////////////
//      Função...:    TransmiteTarefa(LPVOID pParam)
//
//      Descrição:    É uma thread que Transmite a tabera
//
//      Data.....:    [26/06/2002]
//
//////////////////////////////////////////////////////////////////////
DWORD CMasterDlg::TransmiteTarefa(LPVOID pParam)
{
        long lTempoMaxExpera;
        // Ponteiro para ter contato com a classe CCMasterDlg.
        CMasterDlg* pDlgMaster = static_cast<CMasterDlg*>(pParam);

        CString sTarefa;

        sTarefa.Format(_T("Dados da Tarefa %d => "),pDlgMaster->m_iIdProcesso);

        for(int i = sTarefa.GetLength(); i <= 255; i++)
                sTarefa += _T("D");

        int iRet;
        lTempoMaxExpera = 0;
        do
        {
                // verifica status da porta para escrita
                do
                {
                        iRet = _inp(0x300);

                        lTempoMaxExpera ++;
                        if (lTempoMaxExpera == TEMPO_MAX_ESPERA)
                        {
                                AfxMessageBox(_T("Tempo  limite  para  iniciar  a  TX  da
Tarefa esgotado"));
                                pDlgMaster->m_boTaretaAtiva = false;
                                return 0;
                        }
                }
                while (iRet);
                // Transmite caracter de inicio de Transmição
                _outp(0x301,1);
                // espera confimação para transmição da tarefa
                iRet                = 0;
```

```
		lTempoMaxExpera = 0;
		// verifica status da porta para escrita
		do
		{
			// le status da porta de dado recebido
			iRet = _inp(0x302);
			lTempoMaxExpera ++;
			if (lTempoMaxExpera == TEMPO_MAX_ESPERA)
			{
				AfxMessageBox(_T("Tempo limite para receber confimação de comunicação estourado"));
				pDlgMaster->m_boTaretaAtiva = false;
				return 0;
			}
		}
		while (iRet);
		// Le dado de confimação para inicio de transmissão
		iRet = _inp(0x301);
		if (iRet != 2)
		{
			AfxMessageBox(_T("Escravo não apto para receber tarefa"));
			pDlgMaster->m_boTaretaAtiva = false;
			return 0;
		}
		// Envia caracter de inicio do pacote
		_outp(0x301,3);
		// Envia pacote de dados para o escravo
		for(i = 0; i < sTarefa.GetLength(); i++)
		{
			// verifica status da porta para escrita
			iRet = 0;
			lTempoMaxExpera = 0;
			do
			{
				iRet = _inp(0x300);
				lTempoMaxExpera ++;
				if (lTempoMaxExpera == TEMPO_MAX_ESPERA)
				{
					AfxMessageBox(_T("Tempo limite para TX de dado esgotado"));
					pDlgMaster->m_boTaretaAtiva = false;
					return 0;
				}
			}
			while (iRet);
			_outp(0x301, (int)sTarefa.GetAt(i));
		}

		// Envia caracter de fim de Pacote
		iRet = 0;
		lTempoMaxExpera = 0;
```

```
                do
                {
                        iRet = _inp(0x300);
                        lTempoMaxExpera ++;
                        if (lTempoMaxExpera == TEMPO_MAX_ESPERA)
                        {
                                AfxMessageBox(_T("Tempo   limite   para   TX   de   dado
esgotado"));
                                pDlgMaster->m_boTaretaAtiva = false;
                                return 0;
                        }
                }
                while (iRet);
                _outp(0x301, 4);

                // Espera caracter de confimação de tamanho de pacote
                iRet                = 0;
                lTempoMaxExpera = 0;
                // verifica status da porta para escrita
                do
                {
                        // le status da porta de dado recebido
                        iRet = _inp(0x302);
                        lTempoMaxExpera ++;
                        if (lTempoMaxExpera == TEMPO_MAX_ESPERA)
                        {
                                AfxMessageBox(_T("Tempo limite para receber confrmação de
Pacote esgotado"));
                                pDlgMaster->m_boTaretaAtiva = false;
                                return 0;
                        }
                }
                while (iRet);
                // Tamanho do Pacote enviado
                iRet = _inp(0x301);
                if (iRet != sTarefa.GetLength())
                {
                        AfxMessageBox(_T("Retransmitir tarefa ouve perda de pacote"));
                        pDlgMaster->m_boTaretaAtiva = false;
                        // Envia sinal de renvio de pacote
                        _outp(0x301,6);

                }
                else
                {
                        // Envia caracter de fim TX
                        _outp(0x301,5);
                }
        }
        while(pDlgMaster->m_boTaretaAtiva);
        sTarefa.Format("Tarefa %d enviada para o escravo");
```

```
	pDlgMaster->m_lstListaMaster.InsertItem(LVIF_TEXT, i, (LPCTSTR) sTarefa, 0,
0, 0, NULL);
	// indica a thread de espera de final de tarefa que existe tarefa ativa
	pDlgMaster->m_boTaretaAtiva = true;
	pDlgMaster->m_iIdProcesso ++;
	return 0;
}



//////////////////////////////////////////////////////////////////////
//	Função...:	EsperaFinalTarefa(LPVOID pParam)
//
//	Descrição:	É uma thread que Transmite a tabera
//
//	Data.....:	[26/06/2002]
//
//////////////////////////////////////////////////////////////////////
DWORD CMasterDlg::EsperaFinalTarefa(LPVOID pParam)
{
	int iTamPacote = 0;

	char szTarefa[255] = {0};

	// Ponteiro para ter contato com a classe CCMasterDlg.
	CMasterDlg* pDlgMaster = static_cast<CMasterDlg*>(pParam);

	int iRet;
	do
	{
		// verifica se existe tarefa aberta
		if (pDlgMaster->m_boTaretaAtiva)
		{
			// verifica status da porta de leitura
			do
			{
				iRet = _inp(0x302);
			}
			while (iRet);
			// recebe caracter da porta de dados
			iRet = _inp(0x301);
			// verifica se sinal é sinal de final de tarefa, ou seja inicio
de transmição do sentido
			// escravo mestre caso não seja continua esperando
			if (iRet != 1)
				continue;

			 // envia confirmação de inicio de transmição
			_outp(0x301,2);

			bool boFimPacote = false;
```

```
                        // recebe pacode de dados
                        do
                        {
                                // verifica status da porta de leitura do dado
                                do
                                {
                                        iRet = _inp(0x302);
                                }
                                while (iRet);

                                // recebe caracter de inicio de pacote
                                iRet = _inp(0x301);
                                // verifica se recebeu caracter de fim de pacote
                                if (iRet == 4)
                                {
                                        boFimPacote = true;
                                }
                                else
                                {
                                        //  verifica  se  recebeu  caracter  de  inicio  de
pacote
                                        if (iRet != 3)
                                        {
                                                szTarefa[iTamPacote] = (char) iRet;
                                                iTamPacote++;
                                        }
                                }
                        }
                        while(boFimPacote);

                        // envia tamanho do pacote recebido
                        _outp(0x301,iTamPacote);

                        // espera confirmação do pacote
                        do
                        {
                                iRet = _inp(0x302);
                        }
                        while (iRet);
                        // recebe caracter da porta de dados
                        iRet = _inp(0x301);

                        // verifica se pacote foi recebido com sucesso
                        if (iRet == 5)
                        {
                                // finaliza status de tarefa ativa
                                pDlgMaster->m_boTaretaAtiva = false;
                                CString sTarefa(*szTarefa);
                                sTarefa = _T("Tarefa finalizada => ") + sTarefa;
                                pDlgMaster->m_lstListaEscravo.InsertItem(LVIF_TEXT,    0,
(LPCTSTR) sTarefa, 0, 0, 0, NULL);
```

```
                        ::SendMessage(pDlgMaster-
>m_hWnd,MSG_PROXIMA_TAREFA,NULL,NULL);
                    }
                }  // Fim if tarefa ativa
                // espera 100 milesegundos
                Sleep(100);
        }
        while (true);
        return 0;
}


LRESULT CMasterDlg::OnProximaTarefa(WPARAM wParam, LPARAM lParam)
{
        if (m_hThread)
        {
                DWORD exitCode;
                ::GetExitCodeThread(m_hThread,&exitCode);
                if(exitCode != 0)
                        ::TerminateThread(m_hThread,0);

                ::CloseHandle((HANDLE)m_hThread);
                m_hThread = NULL;
        }
        // inicia nova tarefa
        m_hThread                                                                    =
::CreateThread(NULL,NULL,(LPTHREAD_START_ROUTINE)TransmiteTarefa,static_cast<LPVOID
>(this),CREATE_SUSPENDED,&m_dwIDThread);
        ::SetThreadPriority(m_hThread,THREAD_PRIORITY_NORMAL);
        ::ResumeThread(m_hThread);
        return 0L;
}
```

APÊNDICE 2: CÓDIGO FONTE DO PROGRAMA ESCRAVO

```
// ESCRAVODlg.cpp : implementation file
//

#include "stdafx.h"
#include "ESCRAVO.h"
#include "ESCRAVODlg.h"
#include "conio.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define TEMPO_MAX_ESPERA 10000
#define MSG_EXECUTA_TAREFA (WM_APP + 102)

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About
class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CESCRAVODlg dialog

CESCRAVODlg::CESCRAVODlg(CWnd* pParent /*=NULL*/)
	: CDialog(CESCRAVODlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CESCRAVODlg)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hThread           = NULL;
	m_dwIDThread = NULL;
	m_boTaretaAtiva = false;

	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

CESCRAVODlg::~CESCRAVODlg()
{
	DWORD dwExit;
	::GetExitCodeThread(m_hThread,&dwExit);
	while (dwExit == STILL_ACTIVE)
		::GetExitCodeThread(m_hThread,&dwExit);
	if (m_hThread)
		::CloseHandle(m_hThread);
}

void CESCRAVODlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CESCRAVODlg)
	DDX_Control(pDX, IDC_PB_PROCESSO, m_pbProcesso);
	DDX_Control(pDX, IDC_LIST_ESCRAVO, m_lstListaEscravo);
	DDX_Control(pDX, IDC_LIST_MASTER, m_lstListaMaster);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CESCRAVODlg, CDialog)
	//{{AFX_MSG_MAP(CESCRAVODlg)
```

```
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	//}}AFX_MSG_MAP
	ON_MESSAGE(MSG_EXECUTA_TAREFA, OnExecutaTarefa)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CESCRAVODlg message handlers

BOOL CESCRAVODlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	m_lstListaMaster.DeleteAllItems();
	m_lstListaEscravo.DeleteAllItems();

	// função usada para deixar selecionar tota a linha no list
	ListView_SetExtendedListViewStyle((HWND)                    m_lstListaMaster,
LVS_EX_FULLROWSELECT | LVS_EX_GRIDLINES);
	ListView_SetExtendedListViewStyle((HWND)                   m_lstListaEscravo,
LVS_EX_FULLROWSELECT | LVS_EX_GRIDLINES);

	// Cleverson dos Santos Inicializa o gride da tela [20/6/2002]
	while (m_lstListaMaster.DeleteColumn(0));
	while (m_lstListaEscravo.DeleteColumn(0));
```

```
	// insere coluna do gride para os vertices da lista de adjacencia
	LVCOLUMN col;
	col.mask = LVCF_FMT | LVCF_TEXT | LVCF_WIDTH;
	col.fmt  = LVCFMT_LEFT;
	col.pszText = _T("Processos");
	col.cx = 300;
	m_lstListaMaster.InsertColumn(0, &col);

	m_lstListaEscravo.InsertColumn(0, &col);

	m_hThread                                                                    =
::CreateThread(NULL,NULL,(LPTHREAD_START_ROUTINE)EsperaTarefa,static_cast<LPVOID>(t
his),CREATE_SUSPENDED,&m_dwIDThread);
	::SetThreadPriority(m_hThread,THREAD_PRIORITY_NORMAL);
	::ResumeThread(m_hThread);

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CESCRAVODlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CESCRAVODlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;
```

```
		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CESCRAVODlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}


//////////////////////////////////////////////////////////////////////
//	Função...:	EsperaFinalTarefa(LPVOID pParam)
//
//	Descrição:	É uma thread que Transmite a tabera
//
//	Data.....:	[26/06/2002]
//
//////////////////////////////////////////////////////////////////////
DWORD CESCRAVODlg::EsperaTarefa(LPVOID pParam)
{
	int iTamPacote = 0;

	char szTarefa[255] = {0};

	// Ponteiro para ter contato com a classe CCMasterDlg.
	CESCRAVODlg* pDlgEscravo = static_cast<CESCRAVODlg*>(pParam);

	int iRet;
	do
	{
		// verifica se existe tarefa aberta
		if (!pDlgEscravo->m_boTaretaAtiva)
		{
			// verifica status da porta de leitura
			do
			{
				iRet = _inp(0x310);
			}
			while (iRet);
			// recebe caracter da porta de dados
			iRet = _inp(0x311);
			// verifica se sinal é sinal de inicio de transmição do sentido
			// mestre escravo caso não seja continua esperando
```

```
if (iRet != 1)
	continue;

 // envia confirmação de inicio de transmição
_outp(0x311,2);

bool boFimPacote = false;
// recebe pacode de dados
do
{
	// verifica status da porta de leitura do dado
	do
	{
		iRet = _inp(0x310);
	}
	while (iRet);

	// recebe caracter de inicio de pacote
	iRet = _inp(0x311);
	// verifica se recebeu caracter de fim de pacote
	if (iRet == 4)
	{
		boFimPacote = true;
	}
	else
	{
		//  verifica  se  recebeu  caracter  de  inicio  de
pacote
		if (iRet != 3)
		{
			szTarefa[iTamPacote] = (char) iRet;
			iTamPacote++;
		}
	}
}
while(boFimPacote);

// envia tamanho do pacote recebido
_outp(0x311,iTamPacote);

// espera confirmação do pacote
do
{
	iRet = _inp(0x310);
}
while (iRet);
// recebe caracter da porta de dados
iRet = _inp(0x311);

// verifica se pacote foi recebido com sucesso
if (iRet == 5)
```

```
                    {
                           // finaliza status de tarefa ativa
                           pDlgEscravo->m_boTaretaAtiva = true;
                           CString sTarefa(*szTarefa);
                           pDlgEscravo->m_sTarefaRec = sTarefa;
                           sTarefa = _T("Tarefa Recebida => ") + sTarefa;
                           pDlgEscravo->m_lstListaMaster.InsertItem(LVIF_TEXT,     0,
(LPCTSTR) sTarefa, 0, 0, 0, NULL);
                           ::SendMessage(pDlgEscravo-
>m_hWnd,MSG_EXECUTA_TAREFA,NULL,NULL);
                    }
             }  // Fim if tarefa ativa
             // espera 100 milesegundos
             Sleep(100);
      }
      while (true);
      return 0;
}


LRESULT CESCRAVODlg::OnExecutaTarefa(WPARAM wParam, LPARAM lParam)
{
      for(int iPosic =1; iPosic <= 100; iPosic++)
      {
             Sleep(10);
             m_pbProcesso.SetPos(iPosic);
      }
      m_lstListaEscravo.InsertItem(LVIF_TEXT, 0, (LPCTSTR) m_sTarefaRec, 0, 0, 0,
NULL);
      m_pbProcesso.SetPos(0);

      while (!TransmiteResultadoTarefa());
      m_boTaretaAtiva = false;
      m_sTarefaRec.Empty();
      return 0L;
}


bool CESCRAVODlg::TransmiteResultadoTarefa()
{
      long lTempoMaxExpera;

      int iRet;
      lTempoMaxExpera = 0;

      // verifica status da porta para escrita
      do
      {
             iRet = _inp(0x312);
             lTempoMaxExpera ++;
             if (lTempoMaxExpera == TEMPO_MAX_ESPERA)
```

```
		{
			AfxMessageBox(_T("Tempo limite para iniciar a TX da resposta tarefa esgotado"));
			return false;
		}
	}
	while (iRet);
	// Transmite caracter de inicio de Transmição
	_outp(0x311,1);
	// espera confimação para transmição da tarefa
	iRet		= 0;
	lTempoMaxExpera = 0;
	// verifica status da porta para escrita
	do
	{
		// le status da porta de dado recebido
		iRet = _inp(0x310);
		lTempoMaxExpera ++;
		if (lTempoMaxExpera == TEMPO_MAX_ESPERA)
		{
			AfxMessageBox(_T("Tempo limite para receber confimação de comunicação estourado"));
			return false;
		}
	}
	while (iRet);

	// Le dado de confimação para inicio de transmissão da reposta da tarefa
	iRet = _inp(0x311);
	if (iRet != 2)
	{
		AfxMessageBox(_T("Mestre não apto para receber tarefa"));
		return false;
	}

	// Envia caracter de inicio do pacote
	_outp(0x311,3);
	// Envia pacote de dados para o master
	for(int i = 0; i < m_sTarefaRec.GetLength(); i++)
	{
		// verifica status da porta para escrita
		iRet = 0;
		lTempoMaxExpera = 0;
		do
		{
			iRet = _inp(0x312);
			lTempoMaxExpera ++;
			if (lTempoMaxExpera == TEMPO_MAX_ESPERA)
			{
				AfxMessageBox(_T("Tempo limite para TX de dado esgotado"));
```

```
                        return false;
                }
        }
        while (iRet);
        _outp(0x311, (int)m_sTarefaRec.GetAt(i));
}

// Envia caracter de fim de Pacote
iRet = 0;
lTempoMaxExpera = 0;
do
{
        iRet = _inp(0x312);
        lTempoMaxExpera ++;
        if (lTempoMaxExpera == TEMPO_MAX_ESPERA)
        {
                AfxMessageBox(_T("Tempo limite para TX de dado esgotado"));
                return false;
        }
}
while (iRet);
_outp(0x311, 4);

// Espera caracter de confimação de tamanho de pacote
iRet                = 0;
lTempoMaxExpera = 0;
// verifica status da porta para escrita
do
{
        // le status da porta de dado recebido
        iRet = _inp(0x310);
        lTempoMaxExpera ++;
        if (lTempoMaxExpera == TEMPO_MAX_ESPERA)
        {
                AfxMessageBox(_T("Tempo limite para receber confrmação de
Pacote esgotado"));
                return false;
        }
}
while (iRet);

// Tamanho do Pacote enviado
iRet = _inp(0x311);
if (iRet != m_sTarefaRec.GetLength())
{
        AfxMessageBox(_T("Retransmitir resposta tarefa ouve perda de
pacote"));
        // Envia sinal de renvio de pacote
        _outp(0x311,6);
        return false;
}
```

```
        else
        {
                // Envia caracter de fim TX
                _outp(0x311,5);
        }
        return true;
}
```