PROPOSTA E AVALIAÇÃO DE UMA ARQUITETURA A MULTIPROCESSADOR

Luiz Fernando Magalhães Cordeiro

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Valmir Carneiro Barbosa, Ph.D.
(Presidente)

Prof. Edil Severiano Tavares Fernandes, Ph.D.

Prof. Adriano Joaquim de Oliveira Cruz, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

ABRIL DE 1990

CORDEIRO, LUIZ FERNANDO MAGALHÃES

Proposta e avaliação de uma arquitetura a multiprocessador [Rio de Janeiro] 1990

xi, 148 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas,1990)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Arquitetura de Computadores, Multiprocessadores, Máquinas Paralelas. I. COPPE/UFRJ II. Título (série)

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para obtenção do grau de Mestre em Ciências (M.Sc.).

PROPOSTA E AVALIAÇÃO DE UMA ARQUITETURA A MULTIPROCESSADOR

Luiz Fernando Magalhães Cordeiro

Abril, 1990

Orientadores: Professor Valmir Carneiro Barbosa
Professor Edil Severiano Tavares Fernandes

Programa: Engenharia de Sistemas e Computação

Este trabalho trata da definição e validação de uma arquitetura para máquinas de alto desempenho em topologia de multiprocessadores. A idéia básica da arquitetura surgiu ao se estudar os pontos de estrangulamento das arquiteturas já existentes e ao se analisar criticamente as técnicas normalmente utilizadas para se aumentar o desempenho.

A arquitetura é do tipo fortemente conectada em que os processadores são ligados através de um barramento único e compartilham uma memória comum. A arquitetura per-

mite uma implementação com crescimento gradual e com o custo proporcional ao desempenho alcançável. Permite uma total compatibilidade entre máquinas com diferentes capacidades e permite uma programação em multitarefa. Apresenta uma característica de modularidade que se presta otimamente para a integração em circuitos integrados LSI.

A avaliação de seu desempenho, feita através de simulações por programa, mostra o comportamento da arquitetura em problemas envolvendo muito processamento matemático. Verifica-se que o desempenho da máquina pode sempre aproximar-se de um máximo, definido pela sua capacidade instalada. Não apresenta diferenças significativas de desempenho entre processamento escalar e vetorial para uma quantidade suficiente de carga computacional.

A estrutura interna dos processadores e seu conjunto de instruções foram definidos visando otimizar o processamento matemático.

Os princípios básicos da arquitetura, suas características específicas, sua forma de operar e algumas possíveis formas de implementação são o escopo deste trabalho.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M. Sc.).

PROPOSAL AND EVALUATION OF A MULTIPROCESSOR ARCHITECTURE

Luiz Fernando Magalhães Cordeiro

April, 1990

Thesis

Supervisors: Professor Valmir Carneiro Barbosa
Professor Edil Severiano Tavares Fernandes

Department: Systems Engineering and Computer Science

This work is concerned with the definition and validation of an architecture for high-performance multiprocessor machines. The architecture's basic concepts appeared from a study of the bottlenecks in existing architectures, and from a critical analysis of the techniques normally used to enhance performance.

The architecture is tightly coupled, and it processors are connected to a single bus and share a common memory. It allows an implementation that scales easily with a cost proportional to the expected performance. It allows total compatibility among machines with diferent capacities and can be multiprogrammed. Its modularity characteristics are optimal to LSI integration.

The architecture's performance has been evaluated via simulations, and show its behavior in problems with considerable mathematical processing. We show that the machine's performance can always be as close as possible to a maximum allowed by its installed capacity, and that there is no significant difference between scalar and vetor processing for a sufficientes high computational load.

The processors' internal structure and their instruction set were defined to optimize mathematical processing.

The architecture's basic conception, its specific characteristics, its operation mode, and some possible implementation forms are the scope of this present work.

Agradecimentos

Aos meus orientador e co-orientador, que souberam tão bem me orientar no caminho da pesquisa, e sem os quais este trabalho não seria possível.

Ao meu amigo Jorge, pela ajuda e incentivo.

À minha irmã Claudia pelas longas horas digitando e corrigindo os textos.

À amiga Edite, pelo interesse e pelo auxílio na criação das partes gráficas.

Ao Mário Vaz, pelos conselhos sempre adequados.

Aos meus pais e irmãos pela paciência e compreensão.

À minha esposa pelo apoio constante e pelo carinho durante as longas horas de esforço intenso.

Ao meu filho Vítor,
por me lembrar sempre quais são as coisas realmente importantes.

ÍNDICE

CAPíTULO IV

CAPíTULO V

CAPíTULO VI

CAPÍTULO I

INTRODUÇÃO

A Arquitetura de computadores pode ser vista como o ramo da ciência da computação que estuda as possíveis formas de se implementar uma máquina de processar dados. Os trabalhos desta disciplina visam a obtenção de novos blocos funcionais ou de novas formas de conexão dos blocos já existentes. Propõe-se aqui uma arquitetura paralela, de multiprocessadores fortemente conectados através de um barramento compartilhado.

Normalmente quando se propõe uma nova arquitetura pretende-se que esta apresente vantagens sobre as outras já existentes. Pode-se avaliar essas vantagens considerando-se o desempenho da arquitetura, seu custo de implementação, sua adaptabilidade ou o compromisso desses três fatores. A facilidade de programação, a tolerância a falhas e a capacidade de endereçamento de memória podem ser incluídos na capacidade da arquitetura de se adaptar ao ambiente em que será efetivamente usada.

Busca-se aqui uma arquitetura com um bom compromisso entre os fatores acima. Sua avaliação deverá ser feita, portanto, considerando-se esse compromisso.

A pesquisa que culminou nesse trabalho foi motivada principalmente pelo desejo de se definir uma máquina de uso geral e que alcançasse um alto desempenho com uma boa relação custo/desempenho.

O desempenho será avaliado pela capacidade computacional alcançável pela arquitetura, dentro de uma dada

tecnologia de componentes. Uma estimativa normalizada dessa capacidade é feita considerando-se seu desempenho na resolução de problemas matemáticos. Tal estimativa baseia-se nos resultados de simulações feitas sobre uma máquina hipotética de 32 bits, definida sob essa arquitetura.

A boa adaptabilidade da arquitetura deriva de sua característica de alta modularidade, que permite a implementação com crescimento gradual de: desempenho, capacidade de memória e confiabilidade.

A modularidade de baixa complexidade, aliada à regularidade apresentada pela forma de operar dos módulos, leva a características de baixo custo de implementação.

A facilidade de programação pode ser alcançada pelo fato do desempenho da arquitetura não depender do tipo de instruções implementado. A inexistência de hierarquização de memória permite uma transparência de contexto que simplifica também sua programação e operação.

Trata-se de uma arquitetura de processamento paralelo. Dentre suas três formas básicas (pipelining, processamento matricial/vetorial e multiprocessadores), preferiu-se a última por se julgar que ela apresente melhores condições de atingir os compromissos acima (JONES, 1980 e LUNDSTROM, 1987).

O sistema fortemente conectado foi o escolhido para obter-se um baixo custo de comunicação entre os processos. Dentre as várias formas de conexão possíveis optou-se pelo barramento compartilhado por se conseguir com ele uma relação quase linear entre o custo de implementação e o desempenho alcançável. Os processadores são ligados a uma me-

mória comum, através de um barramento compartilhado especial. A forma particular de operar desse barramento é o cerne da arquitetura; sua descrição pormenorizada e a avaliação dos fatores a ele relacionados são os pontos principais deste trabalho.

A idéia básica da arquitetura surgiu ao se analisar os pontos de estrangulamento das arquiteturas já existentes e ao se fazer uma análise crítica das técnicas normalmente utilizadas para aumentar o desempenho.

No capítulo 2 faz-se um breve resumo dos pontos de estrangulamento das outras arquiteturas, das técnicas de aceleração normalmente empregadas e uma revisão de algumas máquinas multiprocessadoras implementadas com algum sucesso.

O capítulo 3 mostra a abordagem adotada na definição da arquitetura, quais os princípios de operação da máquina, quais suas vantagens e como se fará a validação de seu desempenho.

No capítulo 4, os aspectos práticos da avaliação do desempenho da arquitetura são apresentados: os sistemas utilizados, a metodologia e as premissas e simplificações assumidas.

No capítulo 5 (e no apêndice IV) apresenta-se os resultados obtidos nas simulações: tabelas, gráficos, valores e relações de dependência .

No capítulo 6 faz-se uma discussão dos dados do capítulo anterior, avaliando-os em relação às premissas assumidas e analisando seus possíveis desdobramentos. Uma proposta de uma forma derivada dessa arquitetura é apresen-

tada.

No capítulo 7 conclui-se o trabalho.

No apêndice I é feita uma descrição pormenorizada da programação dos problemas matemáticos usados nas simulações; no apêndice II descreve-se o programa simulador, sua forma de operar e seus princípios de funcionamento; traz também uma listagem completa do programa fonte; no apêndice III tem-se uma lista de gráficos e figuras existentes no trabalho e, no apêndice IV, a relação de todas as simulações efetuadas e usadas para gerar os gráficos.

CAPÍTULO II

REVISÃO

Faz-se aqui uma revisão de alguns conceitos utilizados ao longo deste trabalho. Algumas máquinas multiprocessadoras representativas do atual estado tecnológico serão resumidamente descritas para que se possa melhor situar a arquitetura proposta.

II.1 - MODELO DE VON NEUMANN

No modelo clássico de von Neumann, um computador pode ser dividido em dois módulos principais: processador e memória. Um programa, nesse modelo, é composto de um conjunto de instruções, colocadas sequencialmente na memória, e executadas uma a uma, também sequencialmente.

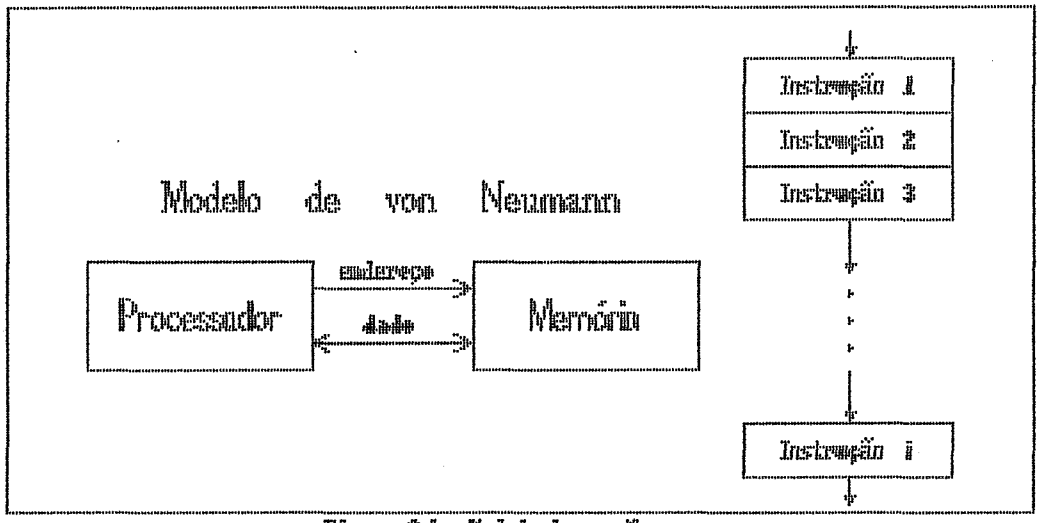


Figura 2.1 - Modelo de von Neumann

A unidade básica de um programa (a instrução) é composta de várias partes:

- busca do código da instrução
- decodificação do tipo de instrução
- busca (eventual) de operandos
- execução propriamente dita da instrução
- armazenamento do resultado (opcional)

Por exemplo, a execução de uma instrução do tipo: R := A + B, teria os passos abaixo, visto na forma de diagrama de tempo:

Figura 2.2 – Etapas de uma instrução

As etapas 1, 3, 4 e 6 são operações de acesso à memória, as três primeiras de leitura e a última de escrita; as etapas 2 e 5 são realizadas internamente ao processador.

Por esse exemplo, pode-se ver que uma substancial parte do tempo de execução de uma instrução é dispendido em transações entre o processador e a memória.

Os projetistas de computadores, desde o início da história da computação, vêm buscando formas de torná-los mais rápidos. Pouca coisa se conseguiu mudar, efetivamente, na estrutura básica de uma instrução; os maiores progressos verificaram-se na forma com que se realizam as etapas fun-

damentais da instrução. Os esforços nesse sentido podem ser compreendidos, estudando-se as técnicas de aceleração de processamento; veremos um resumo das principais mais a frente.

II.2 - PARALELISMO

Pode-se entender paralelismo como o processamento de dados de forma não estritamente sequencial; assim, sempre que uma determinada tarefa for executada, total ou em parte, por mais de uma unidade operadora, ao mesmo tempo, teremos paralelismo (HWANG, 1984). Pode-se dividir os computadores paralelos em três categorias em função da forma como exploram o paralelismo:

1- Paralelismo temporal, realizado por computadores pipeline; embora um processo seja executado sequencialmente, existe uma superposição, no tempo, das partes desse processo. Processamentos de tipos diferentes são realizados por unidades diferentes, simultaneamente, de tal forma que o tempo total de processamento fica substancialmente reduzido (RAMAMORTLHY, 1977).

2- Paralelismo espacial; ocorre quando se replicam as unidades operadoras; operações idênticas, sobre operandos distintos, são executadas simultaneamente.

3- Paralelismo assíncrono; o próprio processador é replicado; um processo é dividido em segmentos, parcial ou totalmente independentes entre si, de tal forma que cada processador executará um conjunto de segmentos distintos. Como segmento, entende-se não só rotinas distintas, como também uma mesma rotina operando sobre dados distintos.

As máquinas que exploram o paralelismo temporal, normalmente o fazem em operações vetoriais; são os chamados processadores vetoriais (Vector Processors). O paralelismo espacial é observado, principalmente, nos processadores matriciais (Array Processors).

O produto interno de dois vetores é uma atividade que serve para ilustrar o funcionamento dos dois primeiros tipos de processamento paralelo, e permite visualizar suas diferenças. Como se sabe, o produto interno consiste em se somar os produtos parciais de cada par de termos de dois vetores:

$$R = \sum_{i=1}^{n} V_i \times W_i$$

No paralelismo temporal (pipelining), enquanto uma unidade multiplicadora efetua o produto parcial dos termos de ordem 4 dos dois vetores, uma unidade somadora realiza a soma do produto parcial dos termos de ordem 3, com a soma dos produtos anteriores. Esquematicamente:

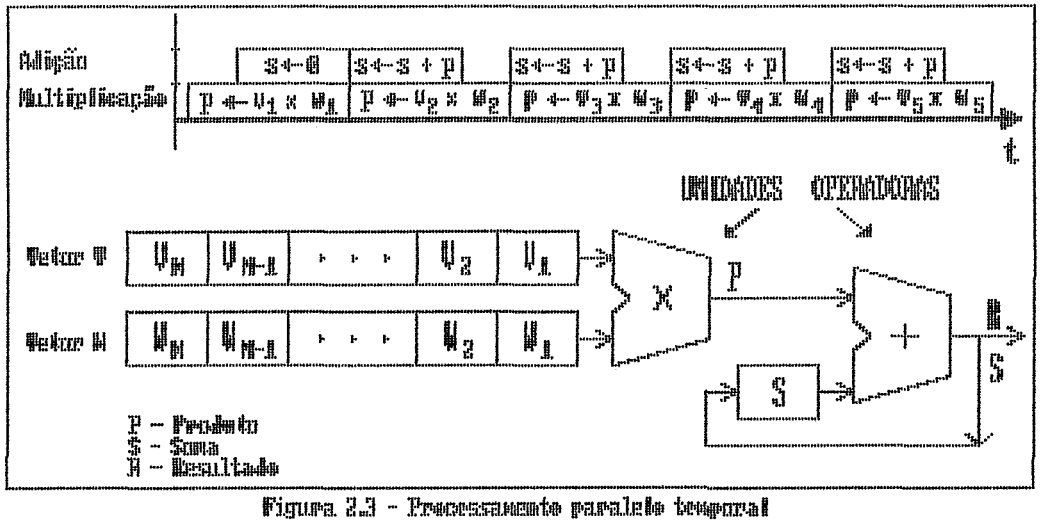


Figura 2.3 - Processamento paralelo temporal

No paralelismo espacial, todos os produtos parciais são efetuados ao mesmo tempo, em N multiplicadores distintos e, ao final, são somados.

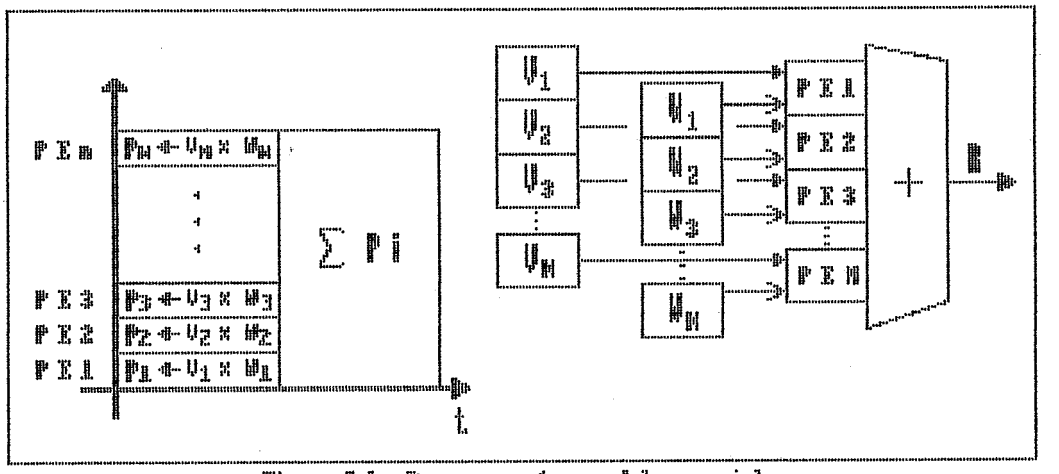


Figura 2.4 – Processamento paralelo espacial

No terceiro caso, o paralelismo assíncrono, realizado pelos multiprocessadores, o produto interno não é uma atividade que ilustre seu funcinamento, devido a um fator denominado "granularidade do paralelismo".

A granularidade consiste na relação entre o tamanho do processo total e o tamanho dos sub-processos em que este foi dividido (WACHS, 1986). Pode-se dizer que um grão é grande, médio ou pequeno considerando quanto do seu tempo de processamento é utilizado de forma "útil" e quanto é dispendido em inicialização e sincronização. Pode-se, simplificadamente, dizer que um grão é pequeno se o tempo útil é da mesma ordem de grandeza do tempo "inútil"; caso o tempo útil seja de uma a duas ordens de grandeza maior, dizemos que existe uma granulação média e, caso esse tempo seja muito maior, dizemos que o grão é pequeno.

Um multiprocessador trabalha bem em uma granularidade com grãos médios ou grandes (STONE, 1987). Essa característica se deve ao fato de que cada processador, além de executar um sub-processo, tem um custo de inicialização e finalização desse sub-processo e um custo eventual de comunicação com outros sub-processos. No caso do produto interno, se cada processador realizasse apenas um produto parcial, seu tempo de inicialização seria provavelmente da mesma ordem de grandeza que o tempo de executar o produto. Além disso, a soma de todos os produtos parciais, ou seria feita por um único processo (o que acabaria com o paralelismo) ou seria executada por todos eles, o que implicaria num alto custo de comunicação pois, ao terminar o produto, um processador teria que bloquear uma variável "somatório", somar a ela o seu produto parcial, e só então liberá-la. Enquanto fizesse isto, outros processos estariam esperando para fazer a mesma coisa. O tempo efetivamente útil de processamento chega a ficar irrelevante ante os outros tempos todos; pode-se facilmente imaginar que o caso piora na medida que se aumenta o número de processadores no sistema.

Se, por outro lado, aumentar-se o tamanho dos grãos, de tal forma que o tempo útil seja muito maior que os outros tempos, a perda será pequena e o número de processadores presentes (e ativamente úteis) poderá ser maior.

Seja tt o tempo total médio; ti o tempo de inicialização e ts o tempo de atualizar uma variável comum aos processos. Esquematicamente:

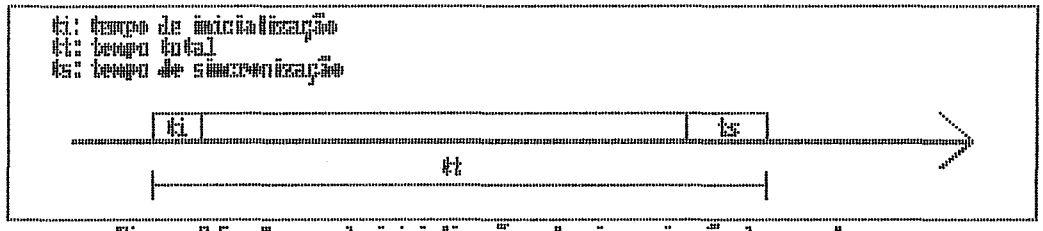


Figura 2.5 – Tempos de inicialização e de sincronização de um sub-processo

No sistema temos P processadores e N processos. O Número N deve ser muito maior que P para permitir uma distribuição uniforme dos processos entre os processadores, de forma que possamos abstrair da relação entre N e P

Normalmente, para P processadores no sistema, considera-se que a relação que estima a probabilidade da ocorrência de conflitos na atualização de uma variável comum é a seguinte:

$$\text{Probabilidade} = \sum_{k=1}^{P} \binom{k}{p} \left(\frac{ts}{tt}\right)^{k} \left(1 - \frac{ts}{tt}\right)^{(P-k)}$$

A relação acima não considera contudo que a sincronização ocorre sempre no final de um processo. Ela supõe que a região de sincronização possa ocorrer em qualquer parte do processo com igual probabilidade. Nos problemas utilizados para a avaliação dessa arquitetura, a paralelização leva à geração de processos com a região de sincronização localizada sempre no início. A relaçào acima também avalia apenas a probabilidade de ocorrência simultânea de de regiões de sincronização. Não calcula os custos advindos dessas "colisões".

Quando dois processadores competem por uma região crítica um deles é atrasado, gerando o custo de sincronização. Pode-se observar contudo, que a próxima "competição"

pela região crítica entre os dois mesmos processadores terá uma probabilidade menor de ocorrer já que houve um atraso, gerando uma defasagem no início dos dois processos, levando a uma provável defasagem no final dos mesmos. Ao iniciar novos processos os dois processadores deverão estar já meio defasados, reduzindo a ocorrência de conflitos.

O cálculo desses atrasos é bastante complexo pois depende de como varia o tempo total de um processo. Pode-se esperar contudo que o custo de comunicação seja de alguma forma diretamente proporcional ao número de processadores e inversamente proporcional à relação ts/tt.

Parece razoável que, devido às defasagens introduzidas durante os conflitos, o custo situe-se em valores aceitáveis se o produto de P por ts/tt for menor que a unidade. Se P(ts/tt) for muito menor que a unidade, espera-se que o custo de sincronização não seja relevante. Para conseguir isso, sem que se limite demais o número de processadores, o "grão" deve ser grande. Considerando-se a dificuldade de formalizar, e até de calcular o custo de sincronização, passa-se às simulações a tarefa de avaliar sua dependência de P e de ts/tt.

II.3 - TÉCNICAS DE ACELERAÇÃO DE PROCESSAMENTO

Para se aumentar a velocidade de um computador diversas técnicas são empregadas; cada uma trabalha sobre determinada etapa da execução de uma instrução. O que se vê, na prática, é o uso de uma combinação dessas técnicas (SLEWLOREC, 1982); a seguir, pode-se ver uma breve descrição das principais.

1 - Redução do tempo de acesso à memória:

- Hierarquização de mémoria - Consiste em se colocar, entre a memória principal e o processador, uma outra menor e mais rápida (memória CACHE); menor para que seu elevado custo não seja significativo no custo total do sistema e mais rápida para que se agilize uma boa parte dos acessos. Aquelas posições da memória primária de uso mais frequente tendem a ter uma cópia na Cache. Tal mecanismo apresenta uma boa relação custo/desempenho porém tem um comportamento muito influenciável pela "localidade" do código gerado.

  A utilização da Cache é transparente ao programador. Uma forma variante de hierarquização de memória é conseguida através do aumento substancial do número de registradores usados na máquina. Tem o inconveniente de aumentar o tamanho do processador e tornar uma troca de contexto bastante demorada.

- Acesso múltiplo à memória (Interleaving) - traz-se para o processador várias posições consecutivas da memória de cada vez, de forma que, ao se terminar uma instrução, se não houver desvio no fluxo do programa, o código da próxima instrução já estará no processador; é realizado pela replicação da barra de dados e pela divisão da memória em vários bancos. Apresenta o inconveniente de aumentar enormemente a quantidade de conexões nos barramentos, elevando o custo e reduzindo a confiabilidade do sistema.

2 - Execução simultânea de micro tarefas (Pipelining) - quando algumas das etapas da execução de uma instrução

são realizadas por dispositivos diferentes, e não têm precedência umas sobre as outras, pode-se conseguir encadear as instruções de forma que o tempo de execução de um conjunto de instruções fique bem menor;

3 - Redução do tempo de decodificação - Máquinas RISC - As máquinas RISC (Reduced Instruction Set Computers) baseiam-se na premissa de que um conjunto pequeno e simples de instruções torna a decodificação e a execução de uma instrução mais rápida; a decodificação de uma instrução não precisa ser feita por um micro-programa interno ao processador, o que reduz drasticamente a complexidade e o tamanho do circuito do processador, liberando espaço também para a colocação de um conjunto grande de registradores internos, viabilizando a técnica de Múltiplos Registradores apresentada acima;

II.4 - MÁQUINAS MULTIPROCESSADORAS

Um multiprocessador difere radicalmente, na abordagem, das técnicas de aceleração vistas anteriormente; consiste na replicação do próprio processador; o problema a resolver deve ser dividido em sub-problemas, mais ou menos independentes entre si, de forma que se atribui a vários processadores a execução desses sub-problemas.

Os Multiprocessadores podem ser divididos em duas categorias pricipais:

- Multiprocessadores fracamente conectados - são aqueles em que os processadores comunicam-se entre si através de mensagens enviadas por interfaces de comunicação;

- Multiprocessadores fortemente conectados - são aqueles em que os processadores comunicam-se através de uma memória comum; os processadores podem ter ou não uma memória de uso particular. Existem vários tipos de conexão possíveis entre os processadores e a memória comum:
  - Conexão espacial por matriz de chaves (Crossbar switch) - Um conjunto de P processadores e M módulos de memória são conectados por uma matriz P x M de chaves. Normalmente M é igual a P. Os processadores podem realizar até M acessos à memória simultaneamente; a matriz de chaves fará as conexões adequadas aos pedidos de acesso. Mais de um pedido simultâneo ao mesmo módulo de memória leva a um conflito, que é decidido beneficiando um dos processadores e atrasando todos os outros.

    O custo da matriz é uma função quadrática do número de processadores, e o custo unitário de cada chave é muito alto quando comparado ao dos processadores; isso faz com que o grau de paralelização atingível seja pequeno.
  - Conexão espacial por chaves multi-estágios - A conexão se faz também através de chaves, só que em menor quantidade que no sistema visto acima; não é totalmente em paralelo. A conexão entre um processador e um módulo de memória não é feita por apenas uma chave mas por um "caminho" de chaves. Os conflitos são resolvidos como no caso anterior. Consegue-se uma melhor relação custo/desempenho com esse sistema porém políticas complexas de otimização de traçado de rotas

são necessárias.

- Conexão temporal por barramento compartilhado - Todos os processadores e módulos de memória são ligados a um mesmo barramento. O compartilhamento é feito no tempo. Esse sistema não permite simultaneidade de acessos, porém é muitíssimo mais simples e barato que os do conexão espacial. Uma variante desse tipo de conexão é usada nesta arquitetura.

Far-se-á aqui uma breve descrição de algumas máquinas multiprocessadoras implementadas nos últimos anos. Esse resumo serve, basicamente para que se possa avaliar o tipo de abordagem que vem sendo dada ao projeto de máquinas multiprocessadoras.

II.4.1 - SISTEMA Cmmp

A arquitetura do sistema Cmmp, criada na década de 1970, é baseada num conjunto de processadores Digital PDP 11/40E, ligeiramente modificados.

Esta arquitetura, apresentada na figura 2.6 abaixo, consiste basicamente de 16 módulos processadores, conectados a 16 módulos de memória compartilhada através de uma matriz (crossbar) de 16 x 16 chaves.

Pode-se também observar nessa figura, a estrutura funcional de um módulo processador típico. A memória compartilhada provê um espaço de endereçamento de 32 megabytes. Foram feitas modificações nos processadores, basicamente para criar restrições para o acesso dos usuários a certas instruções privilegiadas (Halt, Reset, Wait, RTI,

RTT). Também foram criadas formas de verificação de endereçamentos ilegais. Estas modificações eram necessárias para permitir a proteção dos programas.

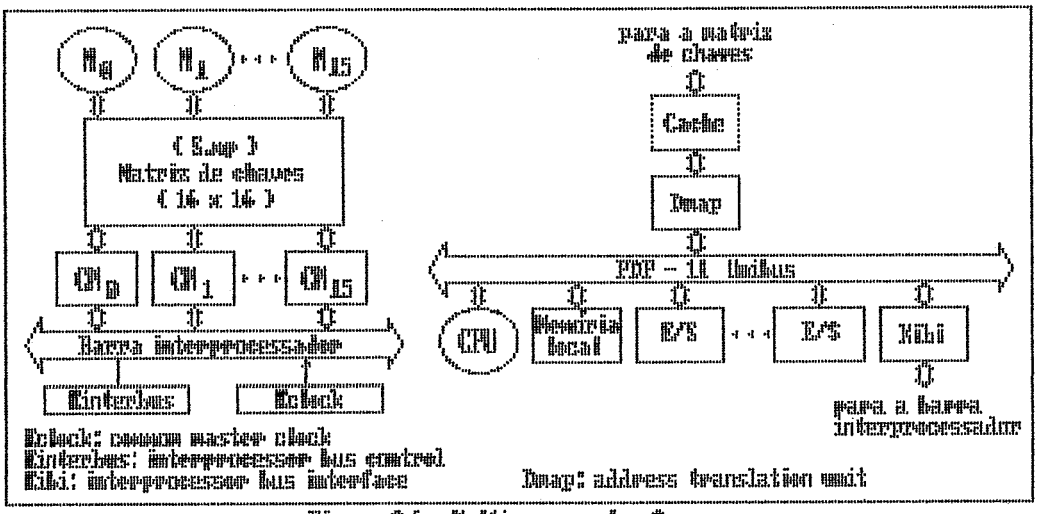


Figura 2.6 - Multiprocessador Comp

Nesta arquitetura, cada processador possui uma memória local (8k bytes), que é usada primariamente para funções do sistema operacional. Os dispositivos periféricos, são conectados individualmente a cada processador (através do seu Unibus).

Não existe, portanto, compartilhamento físico dos periféricos; desta forma, um processador não pode iniciar uma operação de E/S em um periférico que não esteja no seu Unibus. Essa é uma tarefa do sistema operacional: esconder do usuário as assimetrias do subsistema de E/S.

Um barramento de interconexão de processadores é usado para a realização das funções gerais de comunicação interprocesso (Kinterbus e Kclock). Cada processador pode interromper, parar, reiniciar ou iniciar qualquer outro, incluindo ele próprio. Estas funções são usadas em situa-

ções drásticas como, por exemplo, na reinicialização do sistema.

Um dos fatores limitantes na criação de sistemas usando estas máquinas é o limitado espaço de endereçamento oferecido por estes minicomputadores (16 bits).

Um sistema de relocação de endereço (Dmap) foi criado para expandir o espaço de endereçamento da memória compartilhada (que é de 25 bits). Neste sistema o tempo da translação de endereços, somado ao do mecanismo de chaveamento, era bastante alto (da ordem de 1 microsegundo), sendo maior que o tempo de acesso da memória.

II.4.2 - SISTEMA S1

O S1 é implementado utilizando uniprocessadores Mark IIA. Segundo seus projetistas, para uma grande classe de problemas numéricos o Mark IIA deve atingir uma capacidade computacional aproximadamente uma ordem de grandeza acima daquela alcançada pelo Cray I. Na figura abaixo pode-se ter uma visão do multiprocessador S1.

Essa estrutura inclui 16 processadores Mark IIA, que compartilham 16 bancos de memória, através de uma matriz de chaves (crossbar). Cada banco de memória pode conter até 1 gigabyte e, portanto, seu espaço de endereçamento físico total é de 16 gigabytes. Esta capacidade permite ao S1 eliminar o custo de programas de gerenciamento de múltiplos sistemas de armazenamento.

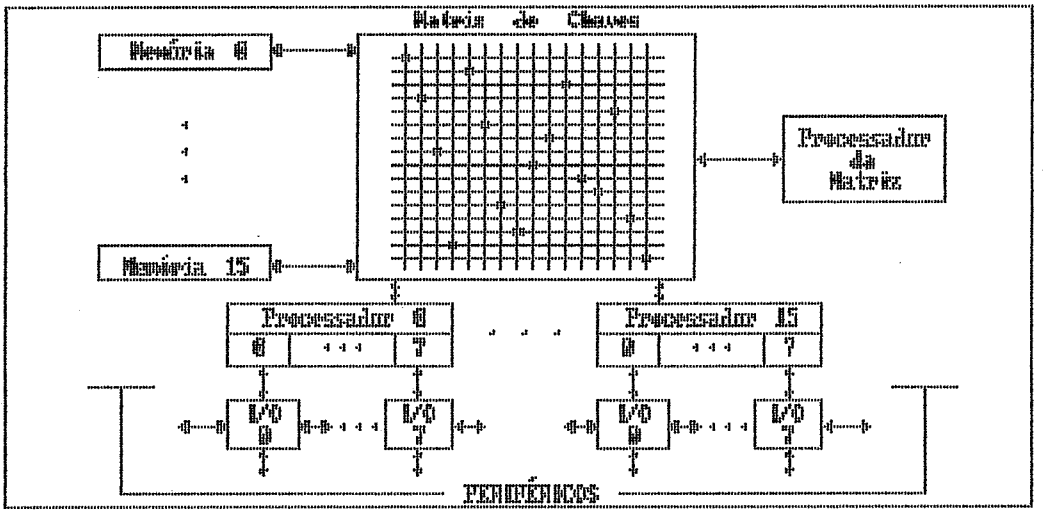


Figura 2.7 - Multiprocessador S1

Cada conexão processador-memória pode transferir uma palavra em 50 nanosegundos, resultando numa taxa de transferência, no melhor caso, de 320 milhões de palavras por segundo. Uma memória Cache em cada processador permite aumentar essa taxa de transferência. A matriz de chaves foi projetada para proporcionar acesso a pedidos múltiplos de memória. A disciplina de serviço para pedidos de memória é tal que nenhum processador obtém dois acessos a um banco de memória enquanto algum outro está tentando acesso a esse mesmo banco.

A comunicação interprocessadores também é feita pela matriz de chaves. O sistema S1 pode utilizar dois conjuntos de matriz de chaves, ambos controlados por um processador dedicado, que substitui uma chave com defeito por uma alternada, permitindo, desta forma, um aumento de confiabilidade. Esta abordagem de tolerância a falhas foi amplamente utilizada no projeto da arquitetura do S1, visando atingir alta confiabilidade e alta disponibilidade.

A razão de crescimento para esse tipo de matriz "quadrada" de chaves é de ordem N ao quadrado (O (N2)). Estima-se que a matriz de chaves do S1 custe menos que um único processador. Além disto, menos de 25 por cento das chaves (crossbar), ou 0,8 por cento do custo total do sistema, deverão ter esta razão quadrática de crescimento, sendo que o resto do sistema cresce apenas com uma razão direta. Isto foi calculado levando em conta que o custo da memória corresponde a metade do custo do sistema. Estas informações levariam a pensar-se na possibilidade de se construir multiprocessadores economicamente viáveis com mais de 16 processadores, entretanto isto não será válido para o caso em que os processadores usados no multiprocessador sejam de baixo custo, pois a matriz de chaves terá o custo dominante.

II.4.3 - HEP (Heterogeneous Element Processor)

Este é um sistema de multiprocessadores de larga escala, que pode executar programas sequenciais e paralelos simultaneamente. O sistema contém até 16 módulos de execução de processos (PEM) e até 128 módulos de memória (DMM). Estes (PEMs e DMMs) são conectados aos subsistemas de controle e de entrada e saída, através de uma rêde comutadora de alta velocidade. Um exemplo de configuração do HEP, com 28 nós de comutação, é mostrada na figura abaixo.

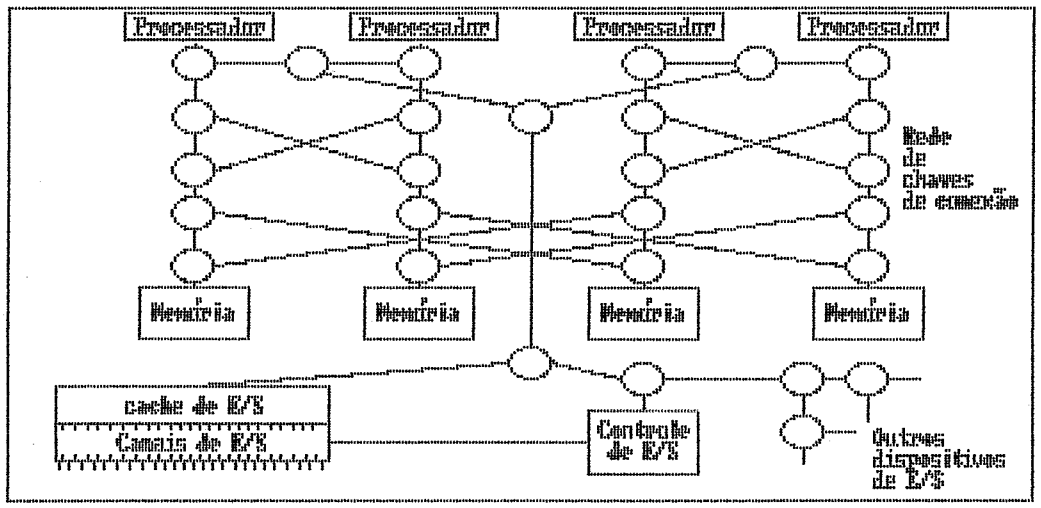


Figura 2.8 - Multiprocessador HEP

Essa configuração consiste de 4 PEMs, 4 DMMs, um subsistema de armazenamento de massa, o processador de controle de entrada/saída, e os nós de conexão para 4 dispositivos. Cada um dos nós consiste de 3 portas de dupla via (full duplex) que o conecta aos elementos vizinhos. Estes vizinhos podem ser PEMs, DMMs, subsistemas, ou então outros nós. Cada nó recebe 3 pacotes de mensagem em cada uma das suas portas, a cada 100 nanosegundos, e tenta direcionar as mensagens de forma que a distância para a seu destino final seja a menor possível. Isto pode ser feito pois em cada nó foram incorporadas tabelas de rotas otimizadas (para cada porta). Também foi implementada uma opção de rota alternativa para permitir o contorno de elementos defeituosos. Desta forma, a rota utilizada é sempre a melhor possível, existindo também um esquema de prioridades para a solução de conflitos. A prioridade é formada a partir de contadores de idade, que são incrementados a cada rota não-ótima utilizada. Instruções sobre processos ativos, alocadas para um

PEM, são armazenadas nesta memória. Entretanto, a unidade de busca de instrução não permite simultaneidade entre a busca e a decodificação de intruções. Por existir apenas uma unidade de busca de instruções na PEM a performance do multiprocessador HEP fica limitada a apenas um ciclo de instrução para cada 100 nanosegundos. Isto limita a utilização efetiva das unidades funcionais.

II.4.4 - SISTEMA Cm*

Este é um exemplo de sistema hierárquico frouxamente conectado, e foi desenvolvido na Carnegie Mellon University. Antecipando a era do computador em um circuito integrado, este multiprocessador era composto por um conjunto de 50 módulos computacionais, cada um dos quais se compunha de um microcomputador Digital LSI 11 com o seu barramento, memória e dispositivos (AQUIZERAT, 1984). Toda a memória primária no sistema é acessível a cada processador. Uma visão do sistema pode ser obtida da figura abaixo.

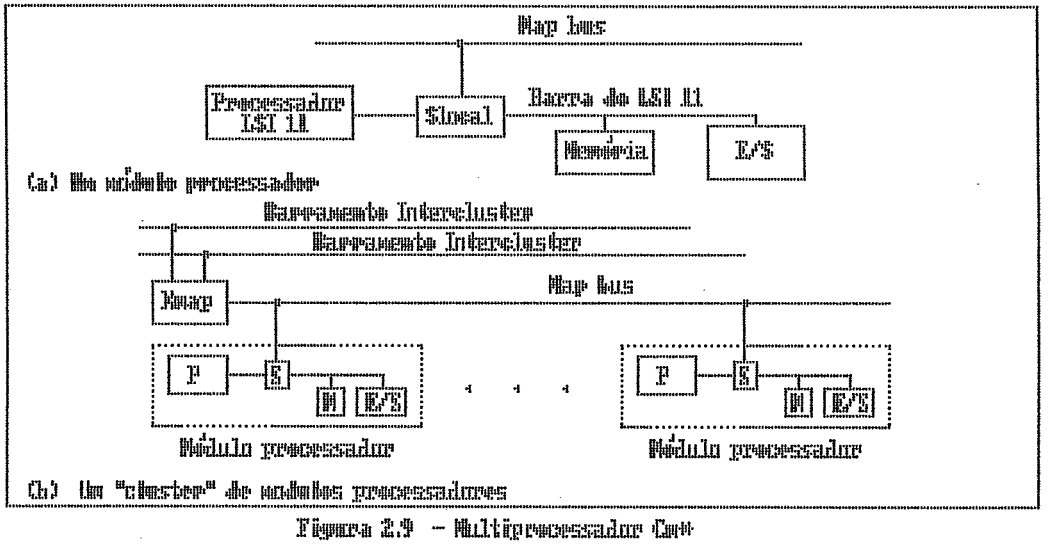


Figura 2.9 - Multiprocessador Cm*

Contrastando com o sistema Cmmp, onde os processadores e memórias são conectados a uma chave centralizada, neste sistema temos chaves distribuídas, ou seja, cada módulo computacional do Cm* inclui uma chave local denominada Slocal; essa chave intercepta e dirige os pedidos do processador para a memória e para os dispositivos de E/S fora do módulo do computador através de uma via denominada de "Map Bus". Também permite que referências de outros módulos do computador sejam aceitas na sua memória e nos seus dispositivos locais de E/S. A responsabilidade pelo mapeamento de endereços e do direcionamento de dados entre Slocals é de um processador denominado Kmap. Os módulos do computador são conectados na forma de módulos hierarquizados, com a utilização de dois barramentos.

Conjuntamente, os Kmaps e os Slocals formam a chave de memória distribuída. Servem para mediar cada referência não local, mantendo a aparência de uma memória única, uniformemente endereçável. Entretanto existem custos de referenciamento diferentes para cada tipo de acesso, sendo que as referências locais são de menor custo.

II.4.5 - SISTEMA BBN PluriBus

Este sistema é composto de uma coleção configurável de 3 tipos de módulos chamados de blocos (buses), cada um dos quais possui uma barra Lockheed SUE como componente central. Cada bloco tipo processador contém dois processadores de 16 bits Lockheed SUE com a sua memória local (8kb). A barra tipo memória contém 32 kb de memória: esta pode ser compartilhadas entre diversos blocos processado-

res. O terceiro bloco, de E/S, contém os diversos dispositivos necessários para E/S que podem ser compartilhados entre diversos blocos processadores. Os diversos tipos de módulos que compõem o sistema, são ligados por acopladores de blocos (buss couplers), que têm a capacidade de relocar os endereços do processador fonte para a memória ou para o bloco de E/S destino.

O protótipo do Pluribus tem uma estrutura de máquina funcionalmente similar a do Cmmp, no entanto existem importantes diferenças:

- A estrutura de interconexão física é completamente distribuída, não possuindo componentes críticos únicos;
- Os registros de controle dos dispositivos de E/S são endereçáveis de qualquer processador do sistema;
- Não existe mecanismo de interrupção para os dispositivos de E/S ou para comunicação interprocesso. Os dispositivos, para serem atendidos, devem colocar uma marca em uma fila especial denominada de dispositivo de pseudo-interrupção (PID).

CAPÍTULO III

FUNDAMENTOS

Normalmente é bastante difícil explicar uma opção de compromisso. É um processo gradual que ocorre com os vários fatores relevantes (e alguns irrelevantes) assumindo diferentes e variáveis pesos ao longo do processo. As vezes, só ao final se consegue ter uma idéia realmente clara dos motivos para cada decisão.

O desejo de definir uma máquina de uso geral que pudesse alcançar um alto desempenho com uma boa relação custo/desempenho foi o que norteou o confuso caminho de seleção de características para a arquitetura.

Alcançar alto desempenho SEM processamento paralelo leva apenas ao uso de componentes mais rápidos.

Dentre os três tipos de processamento paralelo (temporal, espacial e assíncrono), descartou-se o primeiro pela sua incapacidade de permitir um crescimento gradual. O alto desempenho nessa topologia só é alcançável a partir de um patamar de complexidade muito elevado. O segundo tipo permite um crescimento gradual com limites bem flexíveis, porém apenas em operações vetoriais e matriciais; é portanto de uma aplicação mais específica.

Os multiprocessadores apresentam uma promessa de total flexibilidade, possibilidade de fácil crescimento gradual, disponibilidade, confiabilidade etc. Assustam porém quando se pensa em consistência de memória, forma de conexão, paralelização de processos, problemas de comunicação etc.

Os problemas de comunicação e consistência da informação (múltiplos contextos) levaram à adoção do modelo fortemente conectado e com uma ÚNICA memória comum.

Os custos de implementação fizeram com que se adotasse a conexão por barramento compartilhado (o mais barato e lento dos vários tipos possíveis).

O crescimento gradual buscado nos multiprocessadores pode ser acentuado ao se modularizar mais ainda a arquitetura. Isso foi permitido pelo barramento compartilhado que possibilitou ver o sistema como uma via única, onde pedidos, habilitações e transferências são feitos entre módulos distintos.

A pequena taxa de transferência alcançável por esse tipo de conexão levou a se criar um barramento especial, que será esplicado mais adiante. A aceleração desse barramento permitiu que se aumentasse a quantidade de módulos a ele ligados. Isso levou à busca de simplificação e homogeneização dos módulos que permitiu um ganho acentuado no custo de implementação.

Sem considerarmos o sub-sistema de entrada e saída, chegou-se a apenas 5 tipos diferentes de módulos, todos eles de baixa complexidade e pequeno tamanho.

Devida a conexão totalmente paralela (em barramento), a implementação dos módulos em circuitos integrados permite a colocação de qualquer quantidade deles numa pastilha; pode-se, inclusive, "misturar" tipos diferentes de módulos numa pastilha.

Tem-se então:

- baixo custo de integração pelas características de baixa complexidade, pequeno tamanho dos circuitos, possibilidade de repetição de um mesmo circuito numa mesma pastilha e pequeno número de módulos distintos;
- crescimento gradual devido à modularidade;
- limite superior do desempenho estabelecido apenas pela velocidade (aumentada) do barramento compartilhado;
- adaptabilidade, provida pela possibilidade de mudar a programação da máquina alterando-se apenas uma parte de seus componentes;
- possibilidade de recuperação de falhas; um módulo defeituoso pode ser desabilitado com uma pequena perda de desempenho.

A seguir pode-se ver uma explicação pormenorizada da aceleração do barramento que permitiu alcançar as características acima.

III.1 - PRINCÍPIO DE OPERAÇÃO

Uma máquina implementada nesta arquitetura deverá operar, do ponto de vista do usuário, como uma máquina multiprocessadora com um desempenho máximo definido pela sua configuração instalada, alcançável se houver um número suficiente de processos a executar. O desempenho máximo alcançável pela máquina será limitado apenas pela velocidade desse barramento. Para se alcançar esse desempenho o número de processadores deve ser adequado; esse número "adequado" de processadores depende, em certo grau, do tipo de "carga" computacional a que a máquina será submetida.

Ao longo do texto e, principalmente, no capítulo sobre testagem, pode-se ver referências que permitem entender o porquê dessa dependência.

A seguir, pode-se ver uma análise crítica do modelo clássico de von Neumann, que permitirá entender qual é, de fato, a idéia básica que caracteriza essa arquitetura.

Conforme viu-se no item II.1, a unidade básica de um programa no modelo clássico de von Neumann é a instrução; quando um processador executa uma instrução, várias etapas sequenciais podem ser determinadas:

1 - busca, na memória, do código de operação;

2 - decodificação do tipo de instrução;

3 - busca do operando A na memória;

4 - busca do operando B na memória;

5 - realização da operação;

6 - armazenamento do resultado na memória;

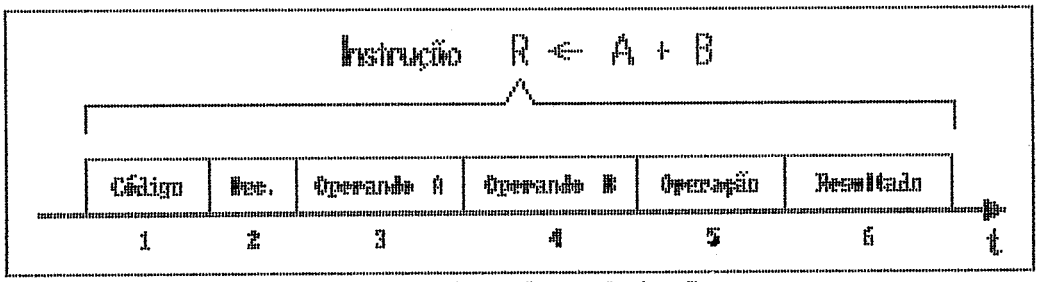


Figura 3.1 - Etapas de uma instrução

As etapas 1, 3, 4 e 6 implicam em acessos à memória enquanto que as outras duas são apenas de atividade interna ao processador.

Se as etapas 2 e 5 não demandassem nenhum tempo de execução, a velocidade da máquina seria definida basicamente pelo tempo de acesso à memória. Uma operação de soma, por exemplo, fará quatro acessos à memória: um para a busca

do código de operação, duas para a busca dos dois operandos e uma para o armazenamento do resultado.

Por outro lado, se apenas a etapa 5 gastasse algum tempo para ser executada, a velocidade de processamento seria definida pela velocidade de se realizar a operação desejada.

Pode-se ver abaixo uma comparação entre os tempos:

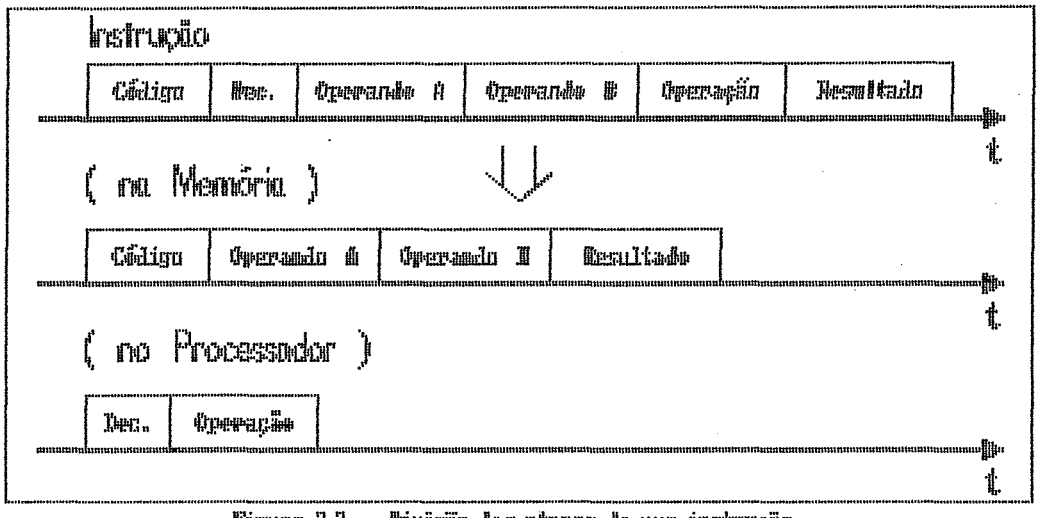


Figura 3.2 - Divisão das etapas de uma instrução

Se fosse possível superpor os dois tempos acima, de forma que eles fossem executados simultaneamente e não em sequência, a velocidade da máquina seria definida pelo mais lento dos dois. Essa é a abordagem utilizada, na maior parte das arquiteturas existentes, para se acelerar o processamento.

No caso desta arquitetura, optou-se por uma abordagem diferente. Como um processador não usa a memória enquanto está realizando a operação e não usa a unidade operadora quando está acessando a memória, tanto a memória quanto a unidade operadora não são "presas" a um processa-

dor mas são consideradas como "recursos" do sistema. Esses recursos são alocados para uso apenas o tempo mínimo necessário e são liberados tão logo executem sua tarefa, de forma a poderem ser usados por outros processadores.

A execução de um processo em um processador será feita passando-se sequencialmente pelas seis etapas acima, só que o processador não "prende" um recurso se não o estiver utilizando. Desta forma, a velocidade do sistema de processadores pode ser, de fato, avaliada pela velocidade do recurso mais lento (memória ou operador) e pode-se balancear a quantidade de módulos de memória e de operadores de forma que, para uma dada aplicação, praticamente não haja ociosidade em nenhum deles.

Naturalmente, ao se fazer apenas isso, a velocidade máxima do sistema ainda ficará limitada à velocidade do barramento em que estão conectados os processadores e os módulos de memória (KINNEY, 1978). Podemos ver, contudo, que o processo de acessar a memória também pode ser decomposto em várias etapas sequenciais:

1 - Colocação do endereço na barra e sua estabilização elétrica;
2 - Reconhecimento e decodificação do endereço pelo módulo referido no endereço;
3 - Acesso propriamente dito à "pastilha" de memória;
4 - Colocação do dado lido na barra de dados e sua estabilização elétrica;

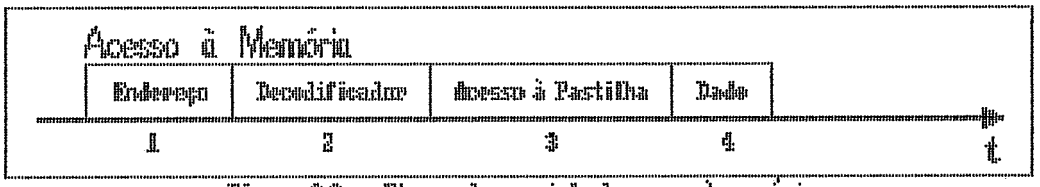


Figura 3.3 - Etapas de um ciclo de acesso à memória

Normalmente otimiza-se todas as etapas acima pois o tempo de acesso à memória é sempre o tempo total. Pode-se ver, contudo, que só se usa o barramento de endereço na etapa 1 e o de dados na etapa 4. Durante as outras etapas, esses barramentos ficam ociosos, podendo ser usados por outros processadores.

Se um processador só colocar o endereço na barra durante o tempo apenas necessário para que esse estabilize eletricamente e seja armazenado nos módulos de memória, outros processadores poderão usar a barra logo após; e o tempo de acesso a um dado na memória não será mais o limitante da velocidade nem da barra de endereços nem da barra de dados. O caso é similar ao mostrado para a busca e execução de uma instrução.

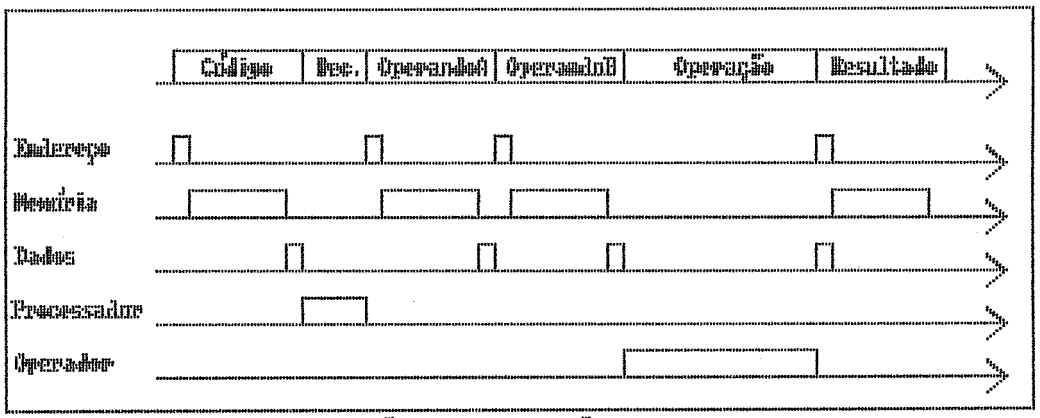


Figura 3.4 - Divisão de uma instrução em suas micro-partes

Assim, "pulverizou-se" a execução de um processo em um número grande de etapas e em vários módulos diferentes. Um processo é executado de forma estritamente sequencial: todos os atrasos serão somados para a obtenção do tempo total; um processo, porém, sempre usa um "recurso" de forma útil, sem prendê-lo mais do que o tempo necessário.

A existência de uma quantidade grande de "interfaces" entre as diversas etapas causa um aumento do tempo de execução de uma intrução em um processador, porém, se a partição em etapas for convenientemente executada, o aumento do tempo será compensado pela criação da possibilidade de conexão de um número grande de processadores ao sistema. Na figura anterior pode-se ver como ficariam os vários "recursos" do sistema durante a execução de uma instrução do tipo R <- A + B. A predominância de "lacunas" nos diagramas de tempo indica um alto grau de ociosidade nos recursos. O "recurso" memória é o que apresenta um menor grau de ociosidade; se se dividir a memória em vários bancos, cada qual com uma parte do espaço endereçável, a ociosidade deverá aumentar e se tornar comparável à dos outros recursos.

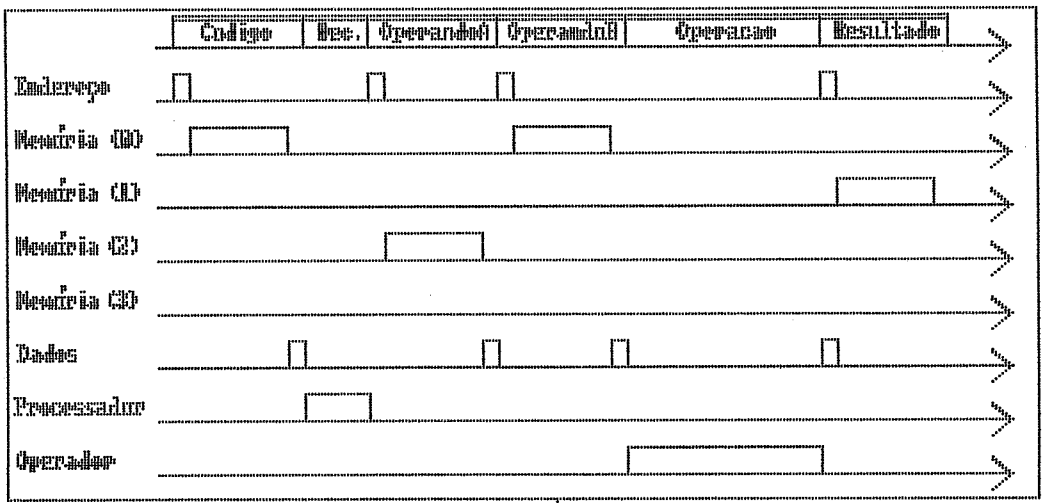


Figura 3.5 - Divisão da memória em quatro bancos

Agora, aproveitando a alta taxa de ociosidade dos recursos, coloca-se mais um processador no sistema; pode-se ver na figura adiante, como ficam então os tempos de execução. Na superposição, caso os dois processadores desejem usar um mesmo recurso ao mesmo tempo, aquele que tiver uma prioridade maior será o primeiro a fazê-lo, atrasando o outro de, pelo menos, um ciclo. Essa prioridade entre os processadores, por simplicidade, é pré-definida e estática.

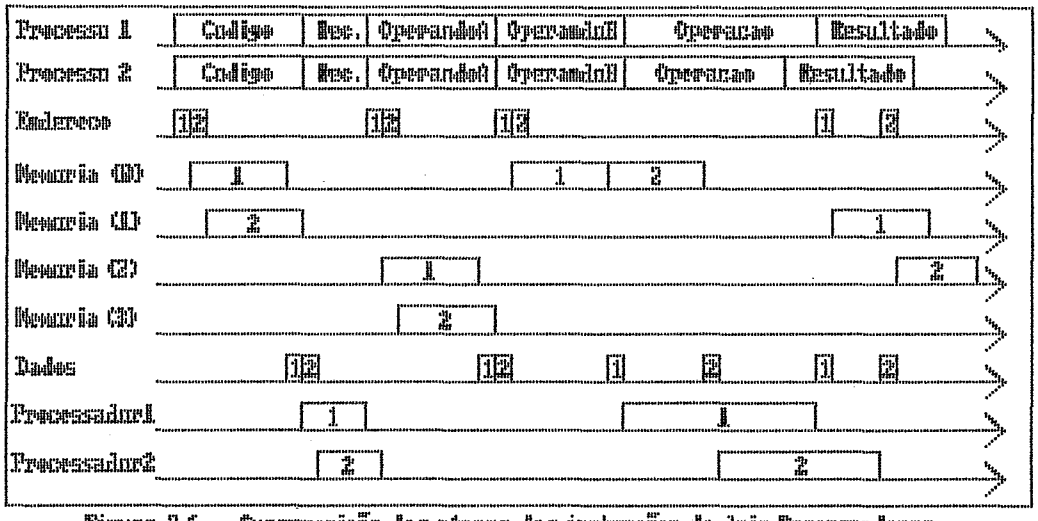


Figura 3.6 - Superposição das etapas das instruções de dois Processadores

Observa-se que ao haver esse conflito de pedidos para o uso de um recurso, um atraso é sempre acrescentado ao tempo de execução do processador de menor prioridade. Pode-se argumentar que, ao se colocar muitos processadores no sistema nesse esquema de prioridades, aqueles menos prioritários terão um tempo de processamento muito maior que os de maior prioridade e que esse acréscimo de tempo será exclusivamente de espera ociosa. Tal coisa deve suceder realmente porém o aumento do número de processadores faz com que os recursos passem a ser usados mais intensa-

mente; nas testagens, optou-se pelo uso da prioridade estática pela simplicidade que esta permite em termos de simulação; numa implementação real certamente um esquema alternativo deveria ser adotado.

Se os recursos mais lentos (por exemplo, os módulos de memória) forem replicados, de forma a compensarem, pela quantidade, seu maior tempo de operação, e se existir uma distribuição razoavelmente uniforme no uso dos diversos módulos, pode-se esperar que a velocidade máxima da máquina seja estabelecida pela velocidade dos barramentos, que são os recursos não replicáveis. Pode-se dizer ainda que a máquina deverá se comportar como se o tempo de acesso à memória fosse igual ao tempo de ciclo do barramento.

Por exemplo, se o tempo de um ciclo de barra, tanto de endereços quanto de dados, for de 50 nanosegundos, a máquina se comportará, para uma carga contínua de processos e um número suficiente de processadores e bancos de memória, como se fosse uma máquina com tempo de acesso à memória de 50 nanosegundos, qualquer que fosse a velocidade real dos bancos de memória. Tal máquina poderia fazer até 20 milhões de leituras ou escritas por segundo.

Ao se contabilizar o número de acessos à memória necessários para um dado problema, resolvido num algoritmo adequado, pode-se prever o tempo total de processamento apenas considerando o tempo de ciclo da barra. Isso faz com que a máquina tenha um desempenho previsível, o que não ocorre com outras arquiteturas de alto desempenho; nessas, o desempenho de pico é, as vezes, dependendo da aplicação, uma ordem de grandeza maior que o desempenho médio.

Pode-se argumentar que essa máquina ficaria com um desempenho muito suscetível ao tipo de problema a resolver ou ao tipo de algoritmo utilizado; considerando-se porém que uma das melhores aplicações dos multiprocessadores está no ambiente multitarefa, pode-se esperar que a distribuição aleatória de endereços e de tipos de operações, inerente a esse tipo de ambiente, permita que os diversos processos ativos "completem as lacunas" deixadas uns pelos outros.

Na análise acima, para efeito de simplificação, colocou-se a execução da operação aritmética sendo feita dentro de cada processador. Observa-se contudo que, assim como a memória pode ser compartilhada, as unidades operadoras também podem sê-lo. Abre-se, assim, a possibilidade de dividir o processador em duas partes: Unidade de Controle e Unidade Operadora; a primeira cuidará de todo o estado do processador e do controle de sequenciamento das etapas da instrução e a segunda ficará reduzida a uma unidade lógica/aritmética.

Essa divisão pode parecer, a princípio, desnecessária, devido à grande diferença de tamanho entre as duas partes. Pode-se alegar que uma unidade lógica/aritmética é muito menor que o resto todo do processador. A situação não é bem essa, se considerar-se como operações, não apenas aquelas fundamentais: adição, subtração, "E" lógico etc, mas também as operações aritméticas de ponto flutuante. Tais operações demandam circuitos relativamente grandes e complexos que chegam, por vezes, a rivalizar em tamanho com o resto do processador (CHEN, 1988). Na figura abaixo pode-se ver como ficariam os tempos de execução dos dois processa-

dores caso se transforme as unidade operadoras em recursos do sistema.

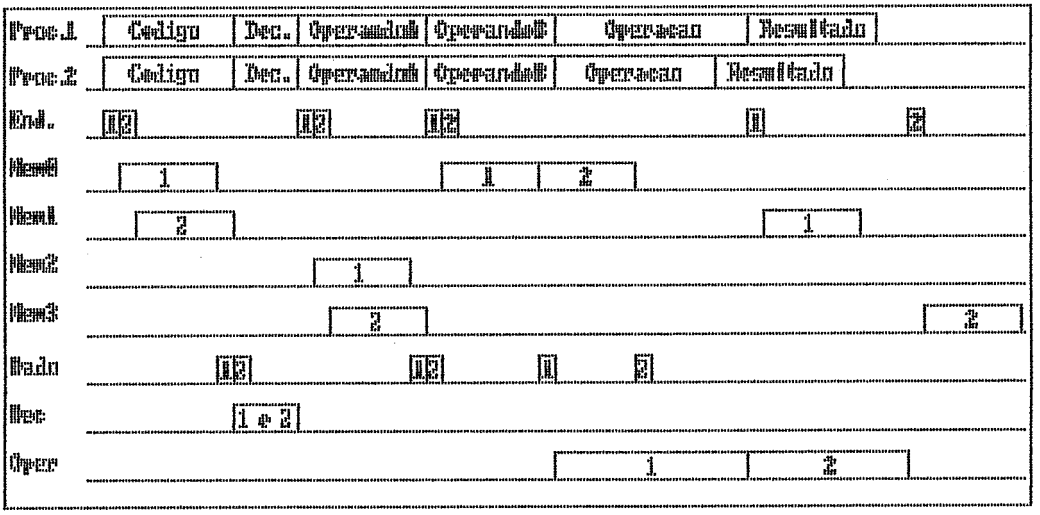


Figura 3.7 - Particionamento da unidade operadora

Os tempos de execução foram ligeiramente aumentados, porém a taxa de utilização do recurso "Operador lógico/aritmético" ficou bem melhor e o tamanho total do sistema ficou um pouco reduzido.

A divisão do processador em duas partes é feita considerando-se o exposto acima. A unidade de controle é o unico módulo que não é compartilhado entre os processadores.

A divisão em módulos funcionais pequenos tem uma característica interessante em termos de implementação. O custo da pastilha de um circuito integrado é função principalmente de dois fatores: custo de desenvolvimento e tamanho da pastilha.

O primeiro fator pode ser reduzido se reduzirmos a complexidade e/ou o tamanho do circuito, ou se o circuito for composto de vários pedaços similares.

O segundo fator deriva do fato de que existe uma quantidade de impurezas diferente de zero na "bolacha" em que são criadas as pastilhas. Essas impurezas distribuem-se de forma mais ou menos aleatórias na "bolacha". Quanto maior for a pastilha, maior será a área total perdida devido à perda de pastilhas contaminadas. A relação entre o tamanho da pastilha e seu custo chega a ser exponencial devido a esse fator.

Como as unidades de controle são pequenas e podem ser agrupadas fisicamente numa mesma pastilha, pode-se, sem perda de generalidade, aumentar a taxa de integração do sistema, caso a tecnologia disponível assim o permita. O mesmo também ocorre, naturalmente, com as unidades operadoras.

Para melhor avaliar a importância da abordagem dada a essa arquitetura, é bom lembrar que num sistema multiprocessador com barramento compartilhado a velocidade desse barramento é um dos limitantes do desempenho do sistema (KINNEY, 1978). Por se "prender" o barramento durante todo o ciclo de acesso à memória, sua velocidade máxima é relativamente pequena em relação à velocidade dos processadores.

Na maior parte dos sistemas, para contornar esse problema, os projetistas colocam uma memória local a cada processador para seu uso exclusivo. Isso faz com que o tempo de acesso à memória possa ser definido pela velocidade dessa memória local; usa-se o barramento comum apenas para a comunicação e sincronização entre os processos.

Essa solução, embora acelere o processamento, au-

menta substancialmente o custo de cada "nó" processador, pelo acréscimo da memória local. Além disso, e principalmente, limita severamente a "transparência" computacional, do ponto de vista do usuário. Por exemplo: um programador ou um programa compilador deve levar em conta a existência de diferentes contextos nas memórias locais ao gerar um código executável. Numa paralelização de problemas, a replicação do código do programa em cada memória local será obrigatória. Em certos tipos de problema, a replicação de toda a base de dados pode ser necessária. Por exemplo, na solução de um sistema de 200 equações a 200 incógnitas pela redução de Gauss, todos os 40 mil termos da matriz de coeficientes deverão ser replicados nas memórias locais de todos os processadores; ou se deve criar uma política relativamente complexa de alocação de tarefas e da base de dados; tal política será possivelmente bastante sensível ao número de processadores disponíveis.

Na arquitetura proposta, por não existir o problema de consistência entre memórias, já que só existe uma, a "transparência para o usuário" é total. Para ele, ao se paralelizar um algoritmo, basta saber que deve inicializar os dados na memória e criar um mecanismo de alocação dos subprocessos gerados entre os P processadores disponíveis. Pode-se aqui recomendar que se crie algoritmos paralelos que funcionem para qualquer número de processadores, permitindo uma total compatibilidade entre máquinas com quantidades diferentes de processadores.

III.2 - ARQUITETURA PROPOSTA

A máquina proposta é do tipo "fortemente conectada". A comunicação entre os vários processadores se dá através de uma memória comum. Os processadores não têm memória local; todas as operações são feitas na memória comum.

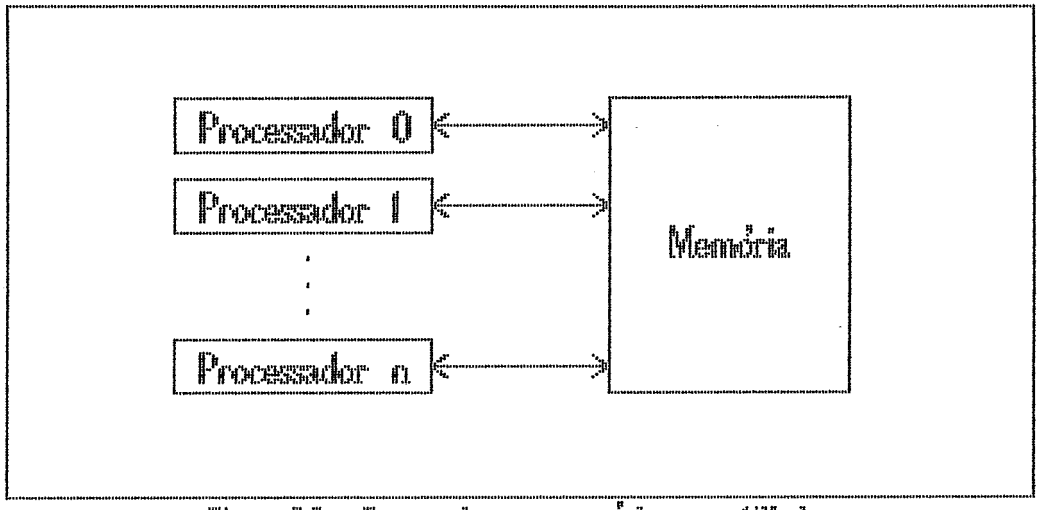


Figura 3.8 - Processadores com memória compartilhada

Os processadores são "partidos" em dois módulos: unidade de controle e unidade operadora.

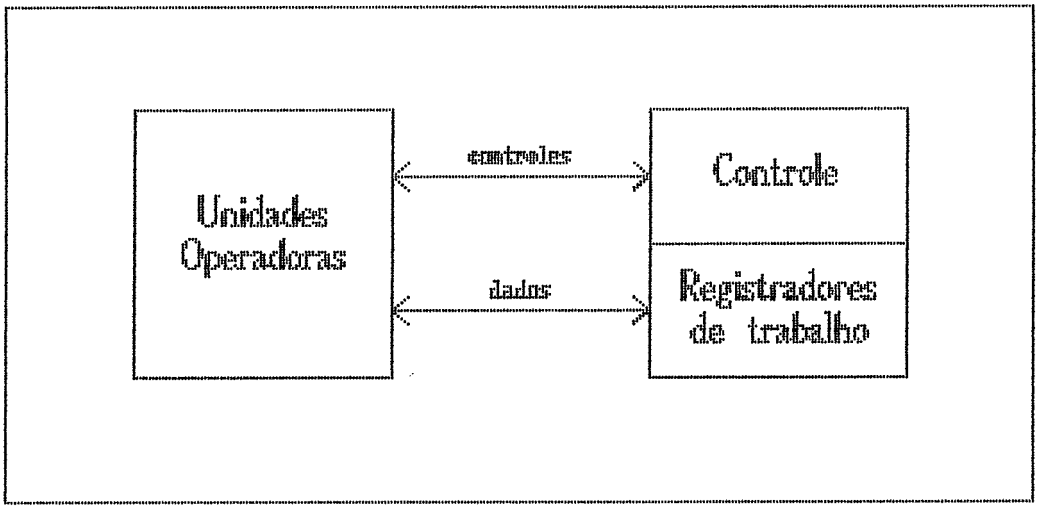


Figura 3.9 - Divisão do processador em duas partes

Na unidade de controle estão os registradores, a unidade de microprograma de controle e uma unidade somadora. O operador é o encarregado de realizar todas as operações aritméticas e lógicas sobre os registradores. Um operador é totalmente subordinado às unidades de controle.

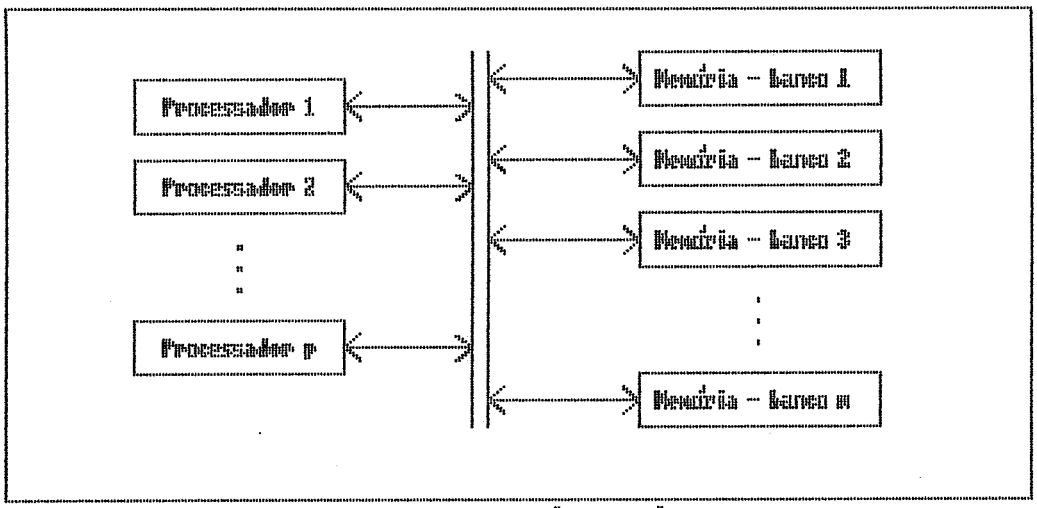


Figura 3.10 - Divisão da memória em vários bancos

A memória comum é composta, na verdade, de vários módulos, que operam independentemente entre si. Isso é necessário pois a velocidade do sistema deve ser limitada apenas pelo tempo de ciclo do barramento de memória e não pela velocidade do sistema de memória. Por exemplo: se o ciclo de barra fosse de 100 nanosegundos e o de memória fosse de 300 nanosegundos, dever-se-ia colocar pelo menos 3 bancos de memória para que o barramento pudesse operar no seu limite.

Na prática, o número de bancos deve ser um múltiplo de 2, pois a melhor forma de dividir o espaço de endereçamento é utilizando-se os bits menos significativos do endereço para selecionar o banco de memória. Fazendo dessa

forma, todos os processos deverão acessar todos (ou quase todos) os bancos, fazendo com que estes possam operar superpondo seus tempos de operação.

Conforme dito anteriormente, existem, na verdade, dois barramentos, um entre as Unidades de Controle e os Módulos de Memória e um outro entre as Unidades de Controle e os Módulos Operadores lógico/aritméticos.

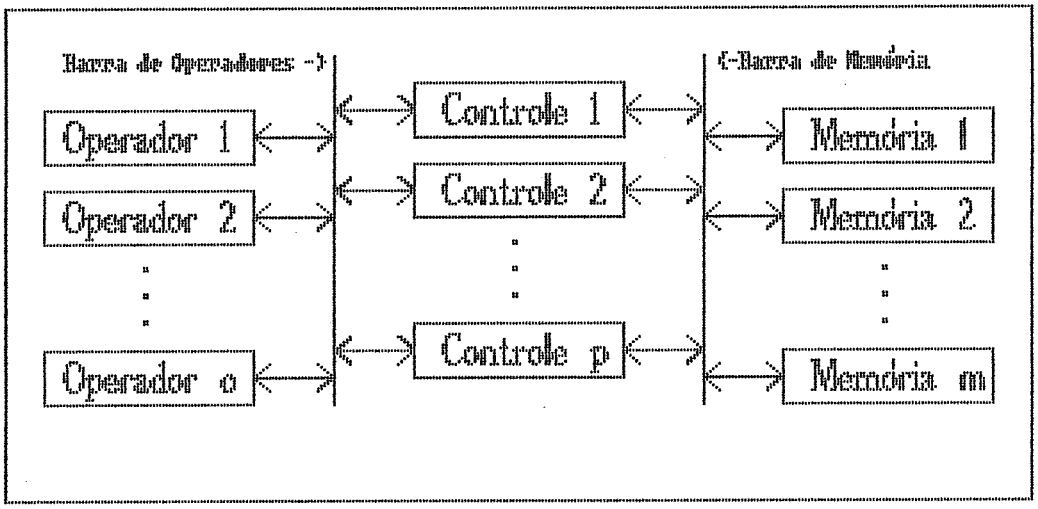


Figura 3.1 - Arquitetura proposta

As Unidades de Controle carregam o estado do processador; os Módulos de Memória e as Unidades Operadoras são "recursos" compartilhados pelas Unidades de Controle.

Quando uma unidade de Controle deseja acessar a memória, coloca seu pedido no barramento de memória. A cada ciclo de barra, todos os pedidos são avaliados por um módulo chamado "Árbitro de Barra", que decide qual deverá ser a Unidade de Controle que poderá, no ínicio do ciclo seguinte, usar o barramento. Essa avaliação será feita considerando a prioridade dos processadores. Essa prioridade pode ser fixa ou variável, estática ou dinâmica. Nas simulações,

apenas para simplificar a criação do simulador, optou-se por usar uma prioridade fixa e pré-estabelecida; esse esquema é o pior possível pois faz com que os processadores de menor prioridade fiquem, quase sempre, esperando que os de maior prioridade não utilizem a barra.

Cada Unidade de Controle tem um número fixo, pré-estabelecido, que lhe é característico, e que é usado para indicar "quem" é que está querendo utilizar a barra.

Após fazer um pedido de uso da barra, uma Unidade de Controle fica esperando pela habilitação. Ao receber uma habilitação (no final de um ciclo de barra), a unidade habilitada colocará na barra de endereços seu número, qual operação deve ser feita (leitura, escrita ou leitura-destrutiva) e o endereço; caso a operação seja de escrita, colocará também, na barra de dados, o dado a ser escrito na memória; caso a operação seja de leitura, esperará que o Módulo de Memória acessado responda, colocando na barra de dados o dado lido e o número da Unidade de Controle que o "pediu".

O barramento de operandos funciona de forma similar; as Unidades de Controle fazem (ou não) seus pedidos de uso de operador, o árbitro dessa barra analisa os pedidos e, decidindo pela prioridade das Unidades de Controle, habilita uma delas. Ao ser habilitada, a Unidade de Controle coloca no barramento um ou dois operandos, seu número no sistema (número da Unidade de Controle), e o código da operação a efetuar; passa então a esperar que a unidade operadora que "pegou" seu pedido responda, colocando no barramento o número da Unidade de Controle juntamente com o re-

sultado da operação e os bits de condição (erro, zero, vai um, divisão por zero etc.).

Os Módulos de Memória e as Unidades Operadoras também têm, no sistema, um número próprio, pré-determinado, que indica "quem" eles são e qual sua prioridade no sistema (no caso do esquema de prioridades ser estático). Da mesma forma que as Unidades de Controle, esses módulos deverão pedir para usar os barramentos; os pedidos serão avaliados pelo Árbitro do barramento, que habilitará aquele módulo de maior prioridade; ao ser habilitado, o módulo colocará na barra o dado pedido (ou o resultado) e o número da Unidade de Controle que fez a solicitação.

Pode-se ver que os três tipos de módulos operam, em relação à barra, de forma similar: pedido de uso, habilitação, uso do barramento.

Cada Módulo de Memória tem, tanto na "entrada" de pedidos, quanto na "saída" de dados, um sistema de fila, permitindo que várias Unidades de Controle façam pedidos de acesso em rápida sequência, sendo atendidos na velocidade do módulo.

No caso de operações de escrita, a Unidade de Controle não espera por resposta, prosseguindo seu processamento tão logo consiga colocar o endereço e o dado no barramento.

O módulo de memória dispõe de um tipo especial de leitura, chamado "leitura destrutiva": após ler o dado na "pastilha", colocando-o na fila de saída, todos os bits naquele endereço são "zerados". Esse tipo de operação serve, basicamente, para permitir um fácil mecanismo de sincroni-

zação de processos (ver mais detalhes no ítem sobre sincronização de processos).

Os Módulos Operadores trabalham de forma um pouco diferente dos Módulos de memória. Eles não são referenciados explicitamente por endereços; por serem todos iguais, qualquer um deles pode fazer a operação que está sendo pedida no barramento. Um Operador não tem filas, nem na entrada nem na saída; pode estar em dois estados: executando uma operação ou esperando inativo; quando está esperando, coloca no barramento uma indicação de que existe um operador disponível. Quando uma Unidade de Controle "pede" um operador o árbitro verifica se existe algum disponível; caso haja, comanda a transferência de dados entre a Unidade de Controle e o Operador disponível de maior prioridade; caso não haja nenhum disponível, aquele ciclo de barra será um ciclo ocioso; no próximo ciclo o árbitro tentará novamente ver se existe algum Operador disponível para habilitar a transferência de dados.

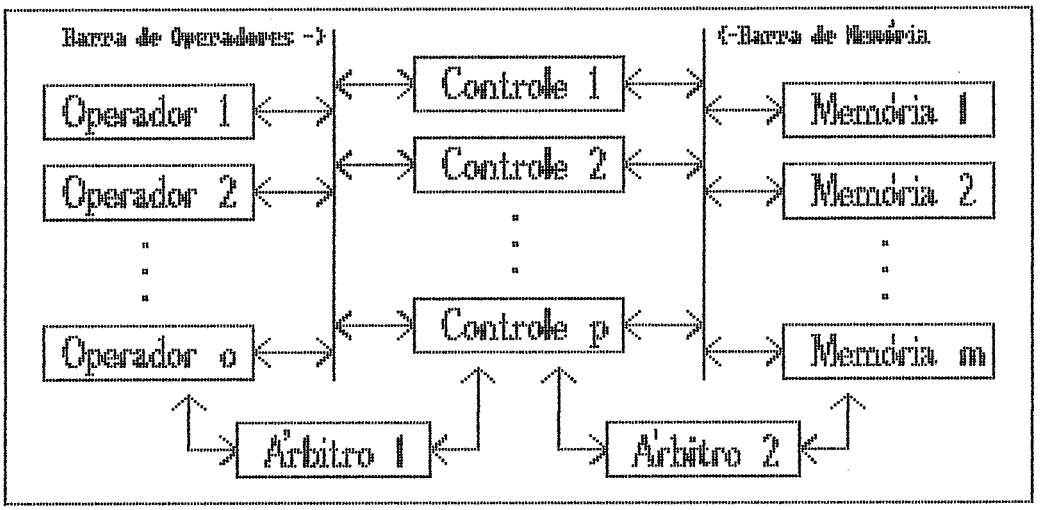


Figura 3.12 - Arquitetura proposta (com os arbitros)

Pode-se ver na figura acima, um diagrama completo da arquitetura, com todos os módulos presentes.

Considerações a respeito de aspectos relativos à proteção de memória, deteção de falhas nos módulos e nos barramentos e em reconfigurabilidade da arquitetura serão feitos mais adiante (item IV.3). O acréscimo desses mecanismos não deve influir no desempenho "normal" da arquitetura, por isso não serão tratados aqui.

A forma como a Unidade de Controle opera internamente (seu número de registradores, seu conjunto de instruções e seus modos de endereçamento) foi definida visando obter uma máquina adequada a processamento numérico. Não é objetivo desse trabalho definir um excelente conjunto de instruções ou uma ótima estrutura interna. Para a definição dessa estrutura, levou-se em consideração os tipos de problemas que seriam usados nas simulações, criando instruções adequadas a uma fácil programação. Alguns processadores de 16 e de 32 bits, de maior expressão encontrados na literatura foram estudados para que a definição dessa estrutura não ficasse longe do tipo de estrutura encontrada na prática (TITUS, 1978 e WALKER,1985). A ortogonalidade das instruções e a simplicidade dos modos de endereçamento foram consideradas e algum esforço foi feito para alcançá-las.

No item IV.6 pode-se ver, em detalhes, a estrutura definida para a máquina.

III.3 - VALIDAÇÃO

Essa arquitetura será validada se certas pretenções, características de sua concepção, forem comprovadas:

1 - Pretende-se criar uma arquitetura para alto desempenho, o que implica numa alta capacidade de processamento;

2 - Para um número adequado de módulos, a máquina deverá poder sempre operar com os barramentos próximos à saturação; isso implica que a sua capacidade, medida em termos de acesso à memória, deverá ser próxima à velocidade da barra de memória;

3 - Dado um algoritmo, avaliando-se a relação entre a quantidade de acessos à memória por operação aritmética, pode-se determinar qual será a capacidade de cálculo alcançável pela máquina (previsibilidade);

4 - Para algoritmos com alto grau de paralelização, pode-se esperar que a aceleração devida ao acréscimo de mais processadores será próxima do máximo teórico, ou seja, que a velocidade será uma função linear do número de processadores no sistema, respeitando-se o limite estabelecido pela velocidade do barramento.

Faz-se, a seguir, uma análise teórica de como se espera que a arquitetura se comporte, baseado nas premissas acima.

Considerando-se que a unidade básica de um programa, a instrução, é composta de três tipos de operação: acessos à memória, decodificação de instrução e operação sobre os dados, pode-se calcular o tempo total de processamento somando-se o tempo total de cada um desses tipos de atividades:

Tempo total = nAcc x tAcc + nOper x tOper + tDec x nInst (1)

onde:

nAcc - número total de acessos à memória

tAcc - tempo gasto em cada acesso

nOper - número de operações aritméticas

tOper - tempo médio para realizar uma operação aritmética

tDec - tempo gasto para se decodificar uma instrução

nInst - número de instruções executadas

Na figura abaixo, uma sequência de 3 instruções ilustra esse somatório.

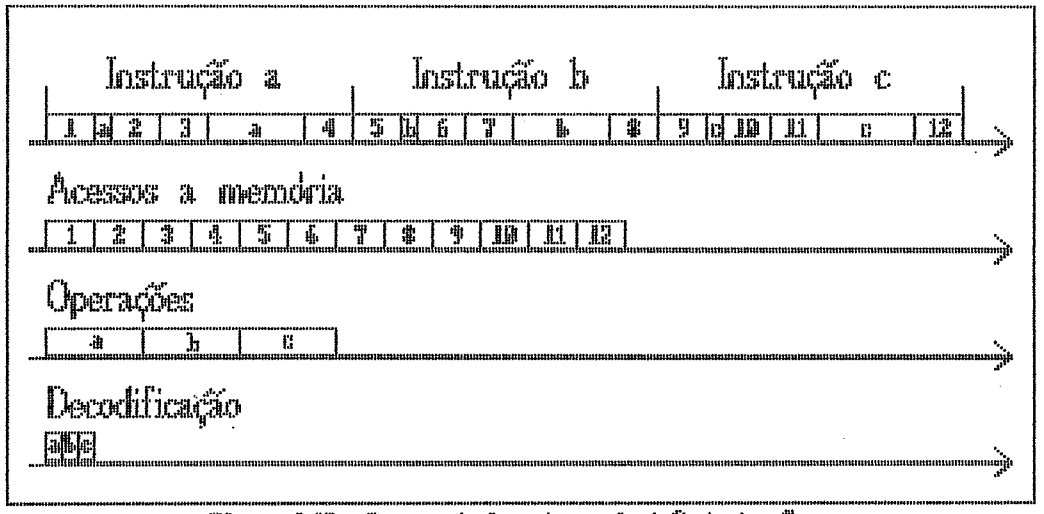


Figura 3.13 - Esquema das etapas de três instruções

O tempos gastos por cada processador para acessar os barramentos, modificam a fórmula (1) acima, acrescentando os termos a, b e c:

$$\text{Tempo total} = nAcc \times (tAcc + a) + nOper \times (tOper + b) + nInstr \times (tDec + c) \qquad (2)$$

Um algoritmo iterativo costuma apresentar uma relação entre o número de acessos à memória, o número de operações efetuadas e o número de instruções a serem decodificadas. Por isso, pode-se simplificar a fórmula acima ao se "embutir" o terceiro termo "nInstr x (tDec + c)" nos dois anteriores; a fórmula fica então:

$$\text{Tempo total} = nAcc \times (TAcc + A) + nOper \times (TOper + B) \quad (3)$$

Considerando-se que tAcc e tOper são parâmetros de entrada, e que nAcc e nOper podem ser obtidos através de simulações, pode-se determinar as constantes "A" e "B", fazendo simulações com valores diferentes de tAcc e tOper.

Com esses dados em mãos, pode-se calcular o tempo de processamento de um processador conhecendo-se o programa a executar e as velocidades dos módulos.

Supondo que exista uma relação entre o número de acessos à memória e o número de operações, pode-se caracterizar a execução de um processo em unidades de tempo da seguinte forma:

$$\text{Tempo unitário} = \frac{nAcc}{nOper} \times TAcc + TOper \quad (4)$$

Nessa arquitetura, esse tempo é de grande importância pois indica quantas vezes um processador acessará a memória para cada operação efetuada. Com a relação acima, pode-se determinar o número de processadores que deverão levar o barramento à saturação:

$$P = \frac{\text{Tempo unitário}}{\text{TAcc}} \qquad (5)$$

O número de operadores suficientes para se obter a saturação é dado por:

$$O = \frac{\text{TOper}}{\text{TAcc}} \times \frac{1}{\text{Tempo unitário}} \qquad (6)$$

O número de bancos de memória para que o barramento de memória possa operar próximo à saturação deve ser tal que:

$$M > \text{TAcc} \qquad (7)$$

III.4 - CUSTO DE CONVERSÃO DE ALGORITMOS

Conforme visto no item II.2, ao se paralelizar um algoritmo, deve-se levar em consideração certas características da máquina que irá executá-lo; assim, para uma máquina pipeline, enfatiza-se o processamento vetorial, para um multiprocessador dá-se especial atenção ao tamanho dos "grãos" em que o problema será dividido.

Existe uma etapa de inicialização e uma etapa de finalização agregadas a cada sub-processo gerado. A sincronização entre os processos norteia a escolha do tamanho dos sub-processos. A relação entre o "tamanho" das inicialização e finalização e o "tamanho" total do sub-processo limita a aplicação da paralelização.

Ao se converter um algoritmo da forma serial para a paralela, acrescenta-se a inicialização e a finalização, fazendo com que o algoritmo paralelo seja sempre maior que

o serial. A essa relação de aumento pode-se chamar "custo de conversão serial/paralelo" ou simplesmente "custo de conversão". Esse custo varia, não somente com a natureza do problema mas também com o seu tamanho; assim, um problema como a "redução de Gauss" tem um custo altíssimo para matrizes de ordem baixa (n < 10), caindo de forma rápida conforme se aumenta a ordem da matriz; a queda é proporcional a "n" ao quadrado. Já a "multiplicação de matrizes" tem o decaimento do custo proporcional apenas a "n". Na tabela 4.1 e no capítulo IV pode-se ver como esse custo se comporta nos dois tipos de problemas citados. Esta avaliação foi feita considerando-se programas feitos na linguagem de máquina do processador definido neste trabalho; uma outra forma de programar poderia apresentar um resultado um pouco diferente.

III.5 - SINCRONIZAÇÃO DE PROCESSOS

Conforme visto no ítem II.2, num sistema com vários processadores, a paralelização de um algoritmo quase sempre leva à necessidade de comunicação entre os processos, o que implica numa necessidade de sincronização entre eles. Pela pequena análise feita anteriormente, pode-se ter uma idéia de como isso influi no tamanho dos processos e como existe uma relação estreita entre o custo de comunicação e o número de processadores; relembrando, deve-se minimizar:

$$p \frac{ts}{tt}$$

onde "p" é o número de processadores, "ts" o tempo de sincronização médio e "tt" o tempo total médio de cada subprocesso paralelo.

Um sistema multiprocessador tem uma aplicação excelente num ambiente multi-tarefa pois, nesse tipo de sistema, a necessidade de interação entre os processos fica quase sempre restrita a comunicações com o Sistema Operacional, fazendo com que ts seja muitíssimo menor que tt.

Devido à dificuldade em se simular um tal sistema, a avaliação foi feita apenas com problemas mais específicos; escolheu-se problemas matemáticos que envolvem muito cálculo pois tais problemas são frequentemente os responsáveis pelo interesse em se criar computadores de alto desempenho.

Os dois tipos de problemas selecionados têm em comum a grande quantidade de cálculo, porém diferem quanto à necessidade de sincronização e ao grau de paralelização. Entende-se por "grau de paralelização" de um algoritmo como um fator ligado à quantidade de sub-processos que este problema pode gerar.

Tanto a multiplicação de matrizes quanto a redução de Gauss geram processos em quantidade proporcional à ordem da matriz ao quadrado; porém, conforme poderá ser visto com mais detalhes adiante, a redução de Gauss gera processos com um grau de dependência muito grande entre eles, limitando, na prática, seu "grau de paralelismo" a um fator proporcional apenas à ordem da matriz. Além disso, o tipo de sincronização também difere entre os dois tipos de problema: enquanto a multiplicação de matrizes necessita de

sincronização apenas para iniciar os processos, a redução de Gauss, além da etapa de inicialização,tem etapas de sincronização ao longo da execução de cada sub-processo.

III.6 - PARALELIZAÇÃO DE PROBLEMAS MATEMÁTICOS

Dois tipos de problemas foram selecionados para servirem como modelo para a avaliação da arquitetura. A escolha baseou-se, não apenas no fato de se normalmente usar tais problemas para testar máquinas de alto desempenho (LEBLANC, 1988 e BARBOSA, 1988) como também pelas suas características de paralelização e necessidade de sincronização. O primeiro deles, a multiplicação de matrizes, apresenta um alto grau de paralelização e uma total independência entre os sub-processos gerados; o segundo tem um baixo grau de paralelização e uma certa interdependência entre os sub-processos.

III.6.1 - MULTIPLICAÇÃO DE MATRIZES

A multiplicação de matrizes é um processo frequentemente usado em processamento gráfico (HAMPTON, 1986). É um problema com um excelente grau de paralelização: o produto de uma matriz N x N gera N ao quadrado processos, totalmente independentes entre si. O produto de duas matrizes quadradas de ordem N gera uma matriz resultado de mesmo formato; cada termo da nova matriz é calculado pelo somatório do produto de uma linha da matriz " A " por uma coluna da matriz " B ":

```
      Matriz A              Matriz B          Matriz resultado

|A11 A12 ... A1n|    |B11 B12 ... B1n|    |R00 R01 ... R0n|
|A21 A22 ... A2n|    |B21 B22 ... B2n|    |R11 R11 ... R1n|
|      .        | x  |      .        | =  |      .        |
|      .        |    |      .        |    |      .        |
|      .        |    |      .        |    |      .        |
|An1 An2 ... Ann|    |Bn1 Bn2 ... Bnn|    |Rn0 Rn1 ... Rnn|
```

Onde:

$$Rij := Ai1 \times B1j + Ai2 \times B2j + \ldots + Ain \times BnJ$$

A resolução por apenas um processo consiste em se calcular os termos das N linhas, cada uma com N colunas, um a um. Divide-se o processo total de cálculo em até N ao quadrado sub-processos.

Uma das soluções por P processadores consiste em se criar N x N sub-processos idênticos, cada um calculando um dos termos da matriz produto.

Como todos os processadores são idênticos, do ponto de vista do Sistema Operacional, para se determinar qual deles calculará qual termo, tem-se duas variáveis de estado, uma para as linhas e outra para as colunas; são inicializadas com o valor 1.

Os processadores, ao começarem a executar o processo, lêem estas variáveis e incrementam o estado do processo. Ao terminarem o cálculo, buscam um novo termo para calcular; caso não haja mais nenhum, voltam para o Sistema Operacional. Naturalmente a coisa não pode ser tão simples assim porque mais de um processador pode ler as variáveis de estado antes que estas possam ser devidamente incrementadas. Para resolver esse problema, os processadores dispõem de uma intrução de "leitura destrutiva", que consiste

em ler o valor de uma posição de memória e instruir a memória para que "zere" esta posição em seguida.

Quando um processador inicia a execução do processo, entra numa "região de sincronização"; ele entra num laço de leitura destrutiva de uma das variáveis, só saindo se esta for diferente de zero; em seguida, testa se já se calculou toda a matriz, caso em que volta ao S.O. Caso ainda haja termos para calcular, lê o valor da outra variável, faz o incremento devido, atualiza o estado na memória, restaura o valor da variável de sincronização e só então calcula o termo da matriz produto.

É fácil ver que o cálculo por N processadores é mais complicado e mais custoso que por apenas um processador; isso implica que, se somarmos o esforço computacional dos P processadores, o total será sempre maior que o custo para apenas um processador. Chama-se a esse aumento de "Custo de conversão de um algoritmo serial para um paralelo". Esse custo não é fixo, variando com a ordem das matrizes a multiplicar. Embora o aumento de custo de conversão cresça quadraticamente com a ordem da matriz, o custo de cálculo dos termos cresce de forma cúbica, fazendo que, para matrizes de ordem alta, o custo de conversão possa ser desprezado.

III.6.2 - REDUÇÃO DE GAUSS

A Redução de Gauss é um processo usado para a resolução de sistemas de n equações a n incógnitas.

Um sistema desse tipo pode ser representado como o produto de um vetor de incógnitas por uma matriz de coefi-

cientes e tem como resultado um vetor de valores.

Esquematicamente: x * A = b

onde :

x : vetor de incógnitas
A : matriz de coeficientes
b : vetor de resultados

O método de redução de Gauss consiste em se fazer operações lineares sobre as linhas da matriz e do vetor de resultados até que se obtenha uma matriz "Diagonal Superior", em que todos os termos abaixo da diagonal principal são zeros e todos os termos da diagonal principal são diferentes de zero.

Por exemplo, num sistema de 5 equações a 5 incógnitas, tem-se:

```
x1 x a11 + x2 x a12 + x3 x a13 + x4 x a14 + x5 x a15 = b1
x1 x a21 + x2 x a22 + x3 x a23 + x4 x a24 + x5 x a25 = b2
x1 x a31 + x2 x a32 + x3 x a33 + x4 x a34 + x5 x a35 = b3
x1 x a41 + x2 x a42 + x3 x a43 + x4 x a44 + x5 x a45 = b4
x1 x a51 + x2 x a52 + x3 x a53 + x4 x a54 + x5 x a55 = b5
```

A representação matricial seria:

```
| x1 |   | a11 a12 a13 a14 a15 |   | b1 |
| x2 |   | a21 a22 a23 a24 a25 |   | b2 |
| x3 | x | a31 a32 a33 a34 a35 | = | b3 |
| x4 |   | a41 a42 a43 a44 a45 |   | b4 |
| x5 |   | a51 a52 a53 a54 a55 |   | b5 |
```

Depois de executada a redução teremos:

```
| x1 |   | c11 c12 c13 c14 c15 |   | d1 |
| x2 |   |   0 c22 c23 c24 c25 |   | d2 |
| x3 | x |   0   0 c33 c34 c35 | = | d3 |
| x4 |   |   0   0   0 c44 c45 |   | d4 |
| x5 |   |   0   0   0   0 c55 |   | d5 |
```

Esquematicamente, foi feita a transformação

x * A = b ==> x * C = d

Para se calcular o valor das incógnitas, utiliza-se um processo chamado retro-substituição.

A retro-substituição consiste em se começar a calcular as incógnitas a partir daquela de maior índice.

Por exemplo, x5 = d5/c55;

x4 = (d4 - x5 x c45) / c44;

e assim sucessivamente.

As operações lineares válidas sobre uma matriz são: multiplicação de uma linha por uma constante e soma de uma linha a outra (termo a termo):

```
ci1 = ci1 * k                ci1 = ci1 + cj1
ci2 = ci2 * k                ci2 = ci2 + cj2
ci3 = ci3 * k        e       ci3 = ci3 + cj3
ci4 = ci4 * k                ci4 = ci4 + cj4
ci5 = ci5 * k                ci5 = ci5 + cj5
di  =  di * k                 di =  di + dj
```

O problema da Redução de Gauss não é muito paralelizável; uma matriz de ordem N gera apenas (N-1) sub-processos e eles não são totalmente independentes entre si. Cada sub-processo se encarregará de zerar uma linha da matriz de coeficientes, abaixo da diagonal principal.

Pode-se usar de uma certa concorrência no cálculo da matriz, respeitando-se a condição de que um termo M(Lin,Col) só pode ser zerado se os termos M(Lin,Col-1) e M(Col,Col-1) já estiverem zerados.

Dizendo de outra forma, um termo só pode ser zerado se o termo da coluna anterior, imediatamente abaixo da diagonal principal já foi zerado. Esse problema de sincro-

nização faz com que o tempo de resolução por P processadores fique prejudicado, pois os processadores poderão ficar em uma espera por sincronização ao final da redução de cada termo da linha que lhe cabe.

Embora a avaliação da arquitetura seja feita principalmente pelo problema de multiplicação de matrizes, acrescentamos o problema da Redução de Gauss para mostrar que, mesmo num caso de baixa paralelização, a arquitetura funciona convenientemente.

No apêndice I pode-se ver, em detalhes, os algoritmos usados, sua programação e codificação na linguagem do processador.

CAPÍTULO IV

MATERIAIS E MÉTODOS

IV.1 - EQUIPAMENTO

Todo o trabalho de computação e de edição de textos e figuras foi realizado em microcomputadores compatíveis com o IBM PC-XT.

A edição de textos foi feita usando-se o Editor FÁCIL, da PC Auxiliar. A criação de figuras e sua inserção no texto foi realizada no sistema INSET, da INSET Inc.

Os programas simuladores, bem como todos os outros programas auxiliares (gerador de gráficos, operador de tabelas, calculador de custos de conversão etc.) foram feitos na linguagem Pascal (TURBO PASCAL versão 3).

Utilizou-se as instalações do NUTES e o laboratório da COPPE/Sistemas.

IV.2 - METODOLOGIA

A arquitetura proposta deverá ser validada, ou seja, deve-se mostrar que as premissas assumidas são válidas e que a máquina, se implementada, funcionaria da forma prevista. A validação será feita através da simulação da máquina.

Devido às características próprias dessa arquitetura, a simulação tem que ser feita tão próxima de uma implementação real quanto possível. O simulador foi projetado para executar cada ciclo de barra sequencialmente; somente dessa forma pode-se ter a certeza que as interferências entre os módulos e as superposições no tempo dos processamen-

tos serão avaliadas corretamente. Pode-se chamar a essa forma de trabalhar de "simulação passo-a-passo".

IV.3 - SIMULAÇÃO PASSO-A-PASSO

O simulador, resumidamente, terá três partes: inicialização, simulação e armazenamento dos resultados.

A simulação será um procedimento iterativo de atualização do estado de toda a máquina, a cada unidade de tempo; assim, a cada ciclo, o simulador tratará: de todos os pedidos de uso de barra, de todos os módulos e de todas as operações habilitadas, em todos os módulos. Podemos ver abaixo, o algoritmo de simulação que, embora bastante resumido, permite avaliar quão se aproximam da realidade as simulações.

```
                    INICIALIZA;
                +-- Fim := False;
                |   TempoDeBarra := 0;
                |   REPEAT
SIMULAÇÃO ---+  |     MOSTRA_STATUS_NA_TELA;
             |  |     TRATA_PEDIDOS_NAS_BARRAS;
             |  |     TRATA_UNIDADES_DE_CONTROLE;
             |  |     TRATA_CONTROLADORES_DE_MEMÓRIA;
             |  |     TRATA_OPERADORES_ARITMÉTICOS;
             |  |     TempoDeBarra := TempoDeBarra + 1;
             |  +-- UNTIL Fim;
                    SALVA_RESULTADOS;
```

IV.4 - ETAPAS DE SIMULAÇÃO

Vários programas simuladores foram construídos para a execução das simulações; os diversos fatores que podem influir no desempenho do sistema foram colocados como parâmetros de entrada para que se pudesse variá-los a vontade:

- Número de processadores presentes
- Número de bancos de memória
- Número de operadores disponíveis

- Tempo de acesso à "pastilha" de memória
- Tempo médio de realização de uma operação nos operadores

Normalizou-se todos os tempos acima em múltiplos do tempo de barra (tempo necessário para que um endereço ou um dado seja transferido em qualquer um dos dois barramentos). Desta forma, caso se deseje fazer uma estimativa do tempo real de processamento, se a máquina for implementada, basta determinar quais são esses tempos na tecnologia escolhida.

Pode-se dividir o processo de simulação em duas etapas:

- Simulação dos programas de teste da arquitetura num simulador simplificado, para a depuração dos algoritmos;
- Simulação dos programas no simulador do multiprocessador.

Para cada programa testaremos os algoritmos em Pascal e depois o partiremos em n processos para que seja executável em um ou mais processadores. Em seguida codificaremos os dois programas na linguagem de máquina do processador e testaremos o simulador até que os programas funcionem de forma idêntica ao programa em Pascal.

Na segunda etapa faremos uma medição de quanto tempo virtual se gasta na resolução de cada programa nos dois tipos de implementação: um único processo e vários sub-processos. Essa medição nos informará qual a perda de desempenho pela simples partição do problema em vários processos e servirá como primeiro dado para a validação da arquitetura.

Em seguida, testaremos os programas no simulador do multiprocessador. Inicialmente apenas com um processador ativo para depurarmos possíveis problemas do simulador. Em seguida, ativando-se mais processadores verificaremos a correção funcional do sistema (avaliando o resultado obtido, sem considerarmos o desempenho alcançado).

Finda essa parte dos testes, faremos a validação propriamente dita, ou seja, avaliaremos o desempenho alcançado pela máquina nas aplicações selecionadas.

A partir de parâmetros iniciais, verificar-se-á qual a influência que a quantidade e velocidade de cada um dos módulos tem sobre o sistema.

IV.5 - SIMULAÇÕES A FAZER

A avaliação da arquitetura, como foi dito, será feita sobre problemas que demandam muito cálculo numérico.

Problemas desse tipo têm sido, em parte, os responsáveis pelo interesse em se construir máquinas com grande capacidade de processamento (HWANG, 1984). Quando se cria uma máquina, seu desempenho é testado sob uma carga de processamento próxima daquela a que ela será submetida em sua aplicação prática. Tal carga, numa medição real, leva de alguns segundos a várias horas. Numa simulação em programa, a "velocidade" do simulador está cerca de quatro ordens de grandeza abaixo da velocidade da máquina simulada. Problemas como a Redução de Gauss, com matrizes de 800 por 800 elementos são usadas em avaliações de máquinas reais como a BBN Butterfly (LEBLANC, 1988). Uma matriz dessa ordem, devido ao grau de granularidade que se pode alcançar,

permite que se use um número grande de processadores.

Teremos que limitar a ordem das matrizes que usaremos nas simulações para que o tempo de simulação seja aceitável. Um compromisso será feito entre o tempo de simulação e os custos de conversão e de comunicação.

O custo de conversão, por depender apenas da ordem da matriz (e não do número de processadores), estabelecerá um limite inferior para essa ordem; considerar-se-á que um custo de conversão abaixo de 10 % seja aceitável.

O custo de comunicação e sincronização é função da ordem da matriz e da velocidade e quantidade dos módulos operando na máquina. Usaremos as relações estabelecidas no item III.3 para determinar um outro patamar inferior para a ordem das matrizes a utilizar.

Tomaremos alguns valores iniciais para os parâmetros de simulação baseando-nos nos tempos possíveis de se obter para os "recursos" da máquina, numa implementação em uma tecnologia típica atual. Assim, supôs-se um tempo de ciclo de barra em torno de 50 nanosegundos, encontrável em boa parte dos circuitos comerciais (INMOS, 1987 e OKI, 1987). O tempo de ciclo de memória, para pastilhas de memória dinâmica de 120 nanosegundos, fica em torno de 250 nanosegundos, o que implica num "tAcc" de 5 ciclos (250/50). O tempo de um Operador Lógico/Aritmético, para uma soma em ponto flutuante, deverá se situar em torno de 500 nanosegundos; uma multiplicação (ou divisão) situa-se perto de 2 microsegundos. Chegamos a valores em torno de 10 e 40 ciclos para as operações acima. Um tempo médio de 25 ciclos será atribuido às operações de ponto flutuante.

Como cada processador só pode comandar, de cada vez, uma única operação, o número de operadores presentes NUNCA precisa ser maior que o número de processadores. Com exceção das simulações que envolvem a medição da influência do número de operadores e da sua velocidade, esse número será sempre igual ao de processadores.

Faremos seis grupos de simulações, cada qual objetivando medir um aspecto do comportamento da arquitetura:

1 - Aceleração devida ao acréscimo de mais processadores - o número de bancos de memória será o máximo (32 bancos) para que apenas a influência do número de processadores seja medida;

Uso do barramento de memória - utilizando-se os dados das simulações anteriores verificamos em que condições a barra de memória operou. Suporemos que o barramento está próximo à saturação quando sua utilização ultrapassar os 90 %;

2 - Influência do número de bancos de memória - variando-se o número de bancos de memória verifica-se como as curvas de aceleração e de saturação se comportam nMem = { 1, 2, 4, 8, 16, 32};

3 - Influência do número de operadores - para um número fixo de processadores que possam levar o barramento à saturação, varia-se o número de Unidades Operadoras, para verificar acima de que quantidade o acréscimo de mais Unidades não implique em ganhos consideráveis;

4 - Influência da velocidade dos operadores - como no item anterior, varia-se o número de Unidades Operadoras, só

que com outros tempos médios de operação; dessa forma, pode-se estabelecer uma relação entre a quantidade necessária e sua velocidade de operação;

5 - Influência da velocidade dos Módulos de Memória - com o número máximo de bancos de memória, varia-se o tempo de acesso à memória e o número de Processadores (e de operadores) no sistema para determinar como as curvas de saturação e aceleração se comportam;

6 - Custo de sincronização - pode ser observado medindo-se o número de acessos à memória efetuados. O custo será qualquer acréscimo ao número de acessos quando no sistema só havia um processador ativo;

IV.6 - MEDIÇÃO DO CUSTO DE CONVERSÃO DE ALGORITMOS

Conforme visto acima, existe um custo na conversão de um algoritmo serial para um paralelo. Esse custo faz com que, na avaliação do desempenho da arquitetura, existam dois fatores influenciando: esse custo e o desempenho intrínseco da máquina.

Para medirmos apenas o segundo fator, calculamos o primeiro para cada ordem de matriz e verificamos acima de que ordem a influência da conversão pode ser desprezada. Esta medição é feita supondo que um algoritmo "custa", ao ser executado, apenas o número de acessos à memória (conhecido como gargalo de von Neumann).

Na tabela e nas figuras a seguir pode-se ver como variam os custos em função da ordem da matriz.

Custo de paralelização de um algoritmo

	Multiplicação de matrizes				Redução de Gauss			
N	nAcc1p	nAccnp	Custo(%)	A/F	nAcc1p	nAccnp	Custo(%)	A/F
10	5610	6930	19.05%	3.46	2176	2415	9.90%	3.04
20	42420	47660	10.99%	2.98	15451	16150	4.33%	2.74
30	140430	152190	7.73%	2.82	49826	51185	2.66%	2.65
40	329640	350520	5.96%	2.74	115301	117520	1.89%	2.61
50	640650	672650	4.76%	2.69	221076	225155	1.46%	2.59
60	1106160	1149580	3.78%	2.66	370551	380409	1.10%	2.57
70	1744470	1806310	3.53%	2.64	590336	600325	0.99%	2.56
80	2598400	2681840	3.11%	2.62	880201	895060	0.85%	2.55
90	3693690	3799170	2.78%	2.61	1259176	1268695	0.75%	2.55
100	5060100	5190300	2.51%	2.60	1721251	1732830	0.67%	2.54

N ..... Ordem da matriz
nAcc1p ... Número de acessos a memória para um processo
nAccnp ... Número de acessos a memória para n processos
Custo .... Custo de conversão para um algoritmo paralelo
A/F ..... Acessos a memória por operações de ponto flutuante

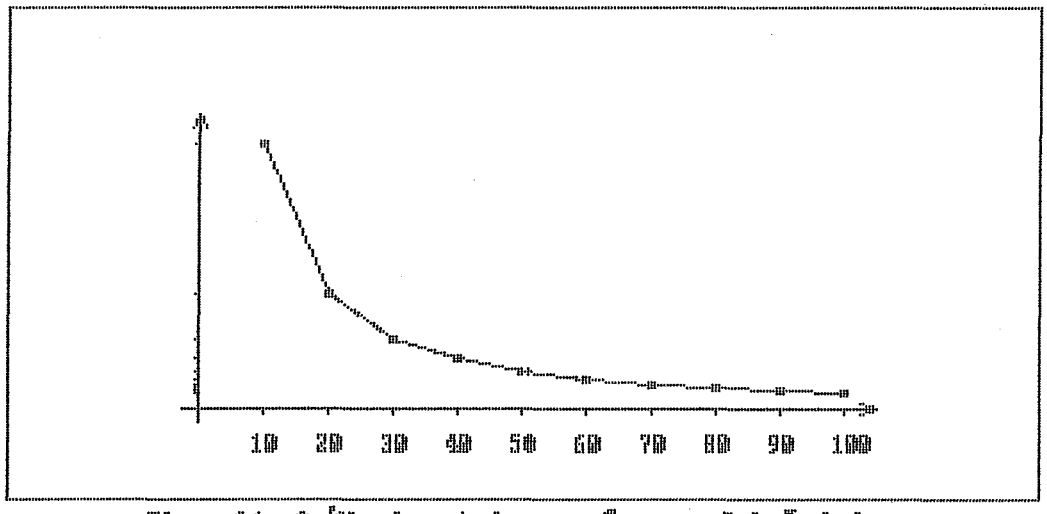


Figura 4.1 - Gráfico de custo de conversão para a Redução de Gauss

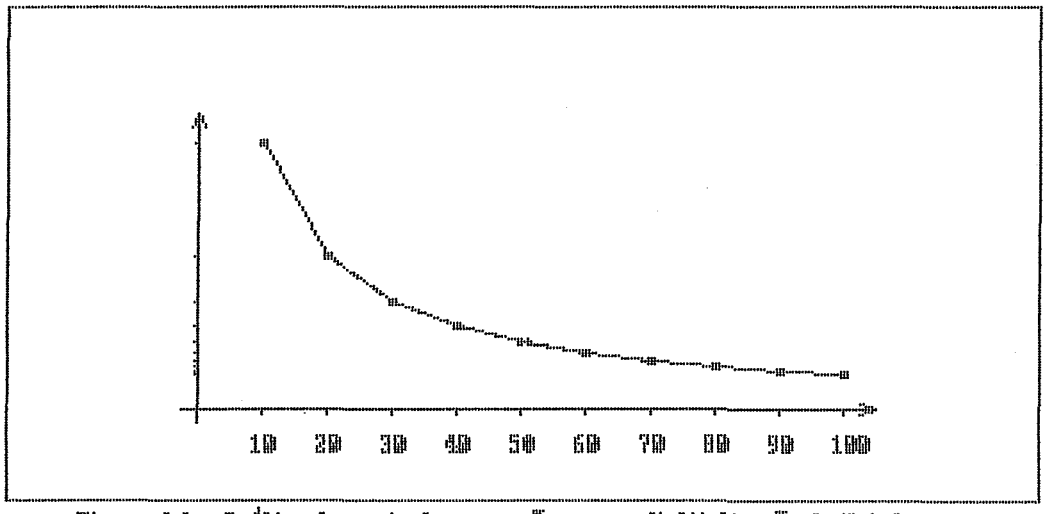


Figura 4.2 - Gráfico do custo de conversão para a Multiplicação de Matrizes

IV.7 - MODELO DO PROCESSADOR

Para fins de simulação, torna-se necessário definir um modelo computacional para os processadores que formarão a máquina. Como se pretende validar a arquitetura apenas em aplicações de processamento numérico, definiu-se a estrutura interna visando simplificar a programação voltada para cálculos.

Definiu-se uma máquina com formato constante dos dados: tanto o código de operação quanto os dados (inteiros e reais) são representados em 32 bits.

Para melhor aproveitar o código de operação, este foi dividido em dois campos de 16 bits, que representarão duas instruções a serem executadas em sequência.

O número de registradores e os modos de endereçamento foram definidos visando a aproveitar da melhor forma possível os bits disponíveis para sua representação.

O conjunto de instruções contém as principais instruções encontradas nos processadores em geral (TITUS, 1981 e INMOS, 1987).

A maior parte das operações são de "dois endereços": o operando 1 recebe o resultado da operação realizada com os operandos 1 e 2.

Pode-se ver abaixo uma descrição resumida das características gerais do processador.

IV.7.1 - REGISTRADORES

O processador se apresenta ao programador como uma máquina com 16 registradores (de 32 bits cada), sendo que dois deles têm funções especiais; os outros 14 são de uso

geral. Os dois registradores especiais são :

- PC : Contador de programa;

- PSW : Bits de estado do processador.

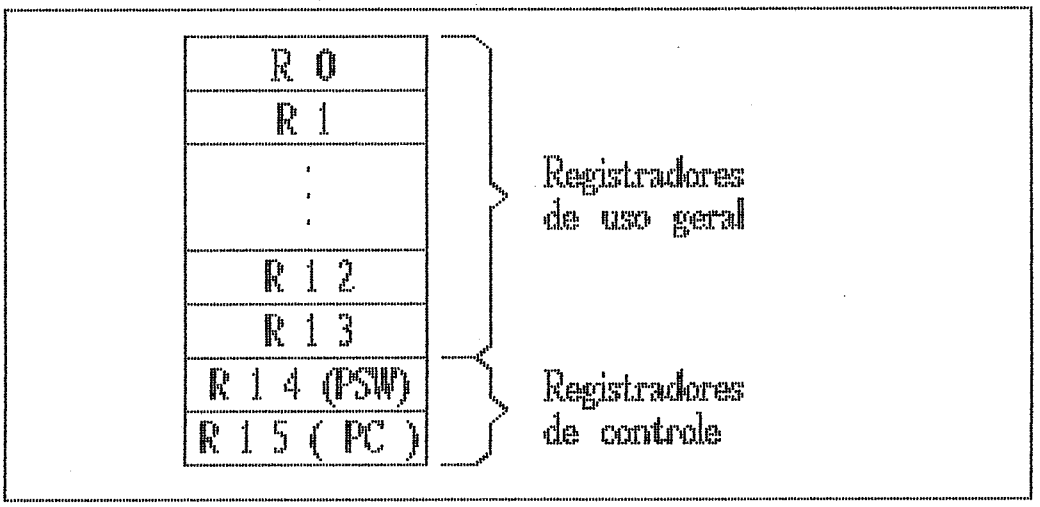


Figura 4.3 - Registradores do processador

O conjunto de instruções opera de forma igual para todos os registradores sendo que, após a busca de um código de operação, e antes de executar a operação, o PC é incrementado de um; o PSW tem seus bits setados de acordo com o resultado das operações efetuadas.

IV.7.2 - ENDEREÇAMENTO

Modo direto

0 - Registrador ..... rXX - O conteúdo do registrador é o operando;

Modos indiretos

1 - Registrador indireto (rXX) - O conteúdo do registrador é o endereço do operando;

2 - Registrador indireto pós-incremento ..... (rXX)+ - O conteúdo do registrador é o endereço do operando; após a operação, o conteúdo do registrador é incrementado;

3 - Registrador indireto
pré-decremento ..... -(rXX) - O conteúdo do registrador é o endereço do operando; antes da operação, o conteúdo do registrador é decrementado;

IV.7.3 - CONJUNTO DE INSTRUÇÕES DO PROCESSADOR

As instruções podem ser de dois tipos:

- Instruções duplas (16 bits cada)

- Instruções únicas (32 bits)

O grupo a que pertence a instrução é definido pelos 4 bits mais significativos da palavra de instrução (Op-Code), da seguinte forma :

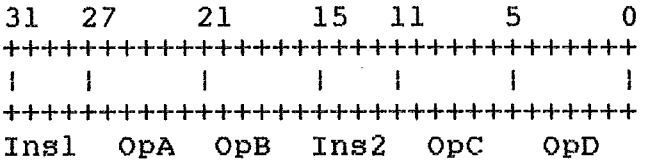
(bits 31, 30, 29 e 28)

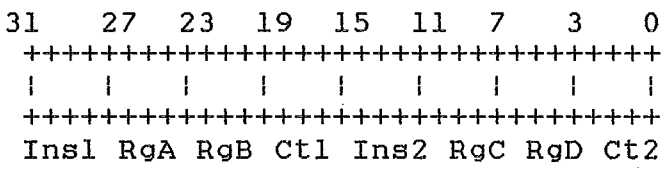
I - 0000 a 1011 - Instruções duplas

II - 1100 a 1111 - Instruções únicas

GRUPO I

São instruções duplas, independentes entre si, cada uma com dois operandos. O tipo de instrução é definido pelos quatro bits do campo 'Instr': Bits 28 a 31 para a instrução 1 e bits 12 a 15 para a instrução 2. A instrução 1 trabalha sobre os operandos A e B, ou sobre os registradores rA e rB e constante 1; a instrução 2 trabalha sobre os operandos C e D ou sobre os registradores rC e rD e constante 2.





Instruções tipo 1 (parte alta do OpCode - bits 16..31)

0 - NOP

1 - OpA := OpB

2 - OpA := k k = { 0..63 }

3 - OpA := INC [OpB]

4 - OpA := DEC [OpB]

5 - rA := (rB + k) k = { 0..15 }

6 - (rA + k) := rB k = { 0..15 }

7 - rA := rA (+ - * / AND OR + -* /) rB

8 - OpA := SqrtInteira [OpB]

9 - OpA := SqrtReal [OpB]

10 - OpA := Rotação sem Carry [OpA]

11 - OpA := Rotação com Carry [OpA]

Instruções tipo 2 (parte baixa do OpCode - bits 0..15)

0 - NOP

1 - OpC := OpD

2 - OpC := k (k = 0..63)

3 - OpC := INC [OpD]

4 - OpC := DEC [OpD]

5 - rC := (rD + k) k= {0..15}

6 - (rC + k) := rD k= {0..15}

7 - rC := rC (+ - * / AND OR + - * /) rD

8 - Compare [rC, rD] - Se igual então PC:=PC+n n= {1..16}

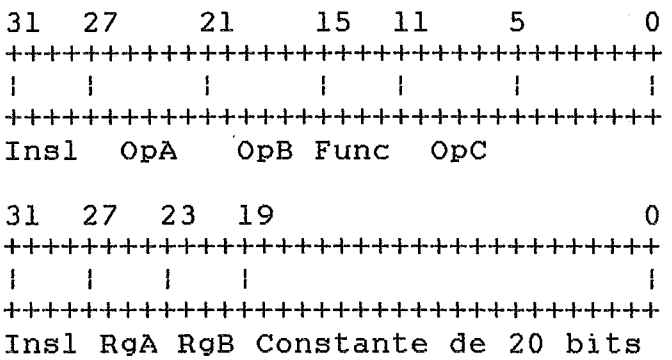
9 - Compare [rC, rD] - Se igual então PC:=PC-n n= {1..16}

10 - Compare [rC, rD] - Se maior então PC:=PC+n n= {1..16}

11 - Compare [rC, rD] - Se maior então PC:=PC-n n= {1..16}

GRUPO II

São instruções sobre três operandos (A, B e C) ou sobre rA e rB; as três primeiras são instruções aritméticas de três operandos; a quarta instrução executa uma iteração com leitura destrutiva, com teste de zero; serve para sincronização de processos.



12 - OpA := OpB (+ - * / AND OR + - * /) OpC

13 - OpA := OpB (+-*/ inteiro) OpC

14 - OpA := OpB (função real) OpC

15 - rA := (rB) destrutivo

Se rA = 0 então PC := PC-1 (espera 0..1M)

CAPÍTULO V

RESULTADOS DAS SIMULAÇÕES

Os gráficos a seguir foram obtidos usando-se os dados coletados nas simulações. No apêndice IV pode-se ver a relação de todas as simulações usadas para gerar os gráficos.

V.1 - ACELERAÇÃO E SATURAÇÃO DO BARRAMENTO

A aceleração obtida pode ser vista no gráfico 5.1 abaixo; pode-se observar que antes de se aproximar da saturação da barra de memória (figura 5.2), a aceleração é quase linear; ao se aproximar dos 90 % de uso, o ganho em desempenho, obtido ao se acrescentar mais processadores, cai rapidamente e, a partir de um certo número, o acréscimo de mais processadores é prejudicial ao desempenho do sistema. Isso se deve, provavelmente, ao fato de que os processadores interferem tanto entre si, que o tempo total de processamento fica aumentado.

O barramento de memória atinge os 90 % de saturação quando o número de processadores está em torno de 17. A partir daí, um joelho na curva indica que os processadores acrescentados competem demais pelo barramento, apresentando cada um deles um desempenho muito baixo. No gráfico 5.2 abaixo pode-se ver a curva de uso do barramento; é interessante relacioná-lo ao gráfico de aceleração acima; pode-se observar que a queda da aceleração ocorre quando o barramento se aproxima da saturação (p=17).

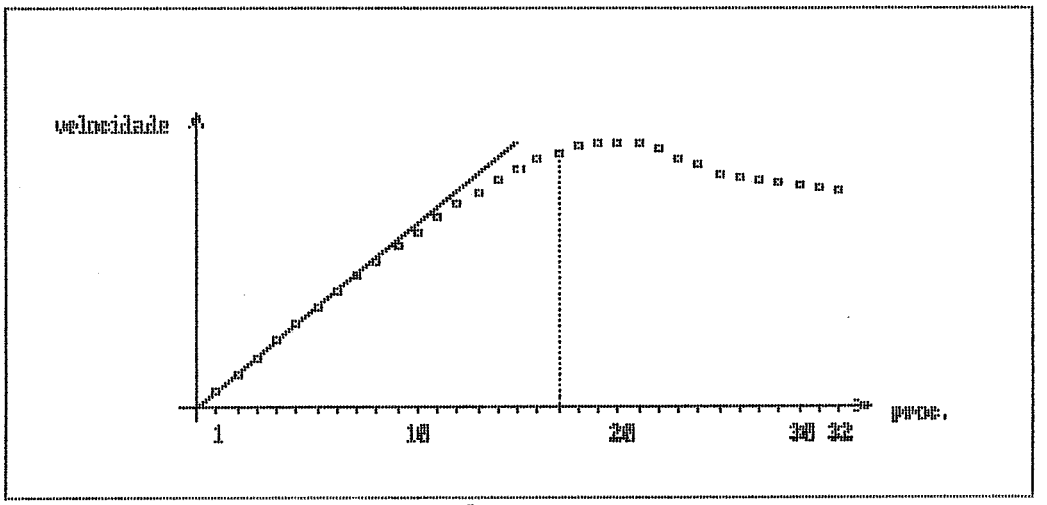


Figura 5.1 - Gráfico da aceleração

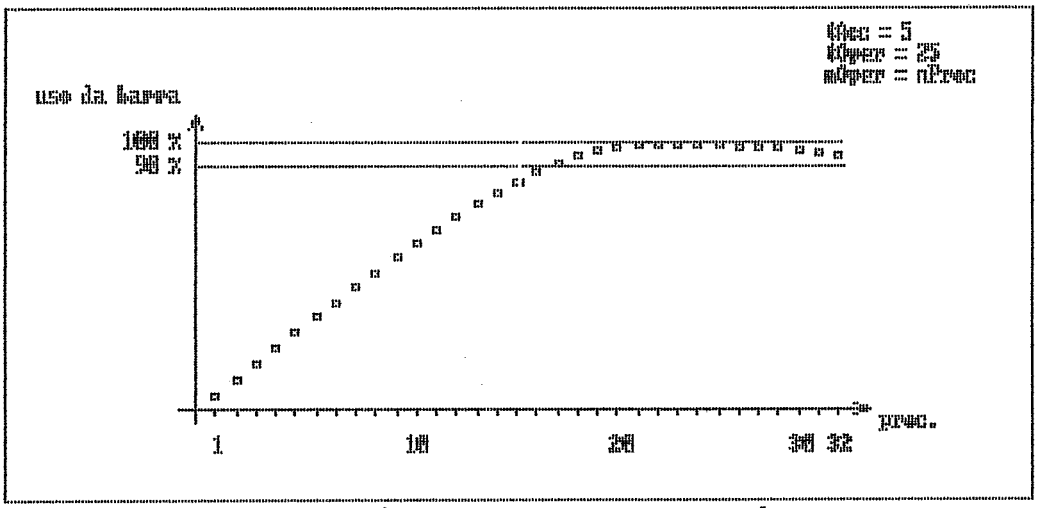


Figura 5.2 - Gráfico de uso do barramento de memória

V.2 - INFLUÊNCIA DO NÚMERO DE BANCOS DE MEMÓRIA

Conforme pode ser observado no gráfico 5.3, o número de bancos de memória presentes no barramento modifica sensivelmente as curvas. Com uma quantidade de bancos muito pequena (1, 2 e 4), o sistema nunca atinge a saturação. Com um número maior (8, 16 e 32) a diferença apresentada é pequena.

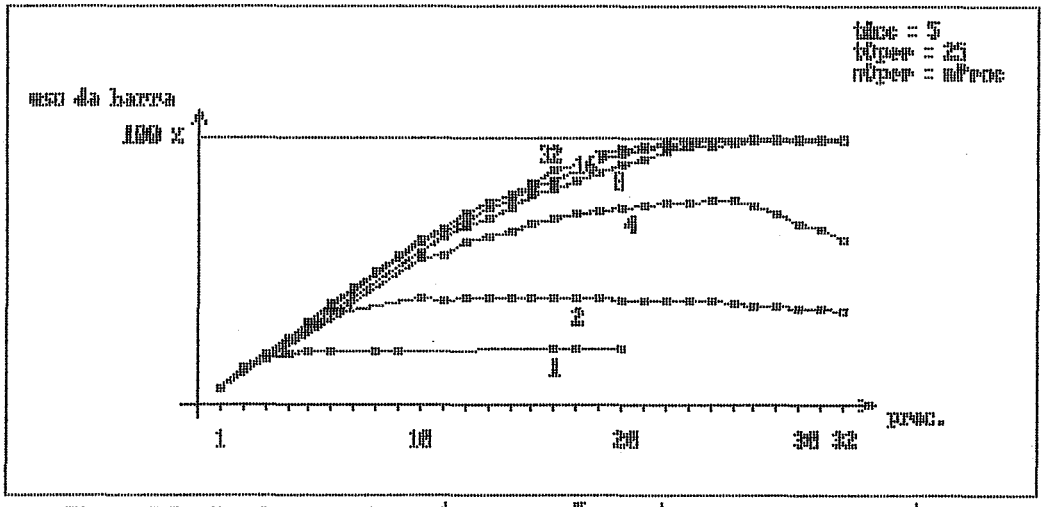


Figura 5.3 - Uso da barra de memória em função do número de barras de memória

V.3 - INFLUÊNCIA DO NÚMERO DE OPERADORES

No gráfico abaixo pode-se ver que o número de operadores influi no sistema até um certo ponto. A partir daí, sua influência fica mínima. Isso ocorre porque os operadores não são endereçados como a memória. A partir de um certo ponto, haverá sempre pelo menos um Operador ocioso no sistema. Pode-se ter a certeza que o número de Operadores nunca precisa ser maior que o de Unidades de Controle, pois estas podem gerar apenas um pedido de operação de cada vez. Pode-se ver porém que, na prática, o número necessário de Operadores é muito menor que o de processadores.

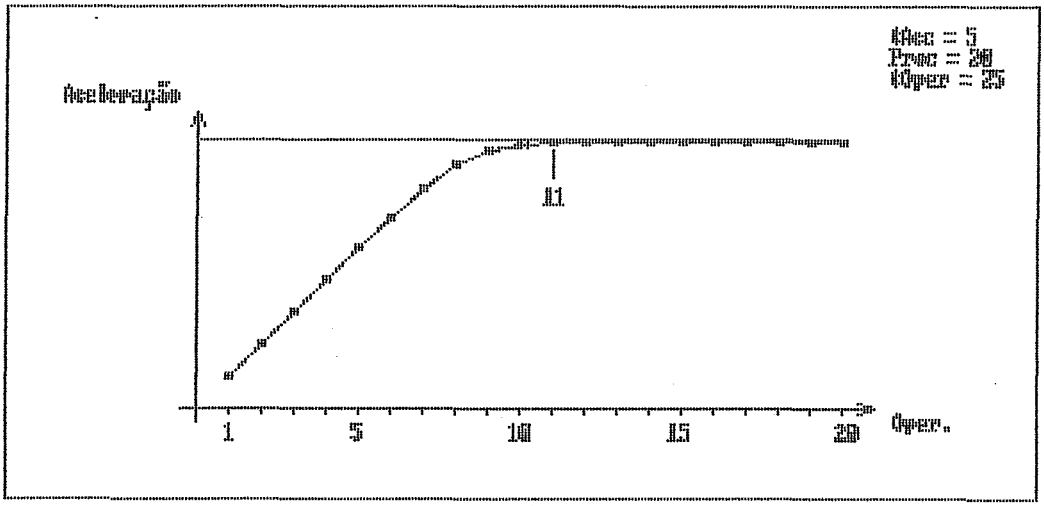


Figura 5.4 - Influência do número de operadores

V.4 - INFLUÊNCIA DA VELOCIDADE DOS OPERADORES

O tempo de operação, assim como o de acesso à memória, modifica apenas a quantidade de processadores necessária para atingir a saturação da barra, e sua influência pode ser contrabalançada modificando o número de Unidades Operadoras no sistema, conforme pode ser visto no gráfico da figura 5.5.

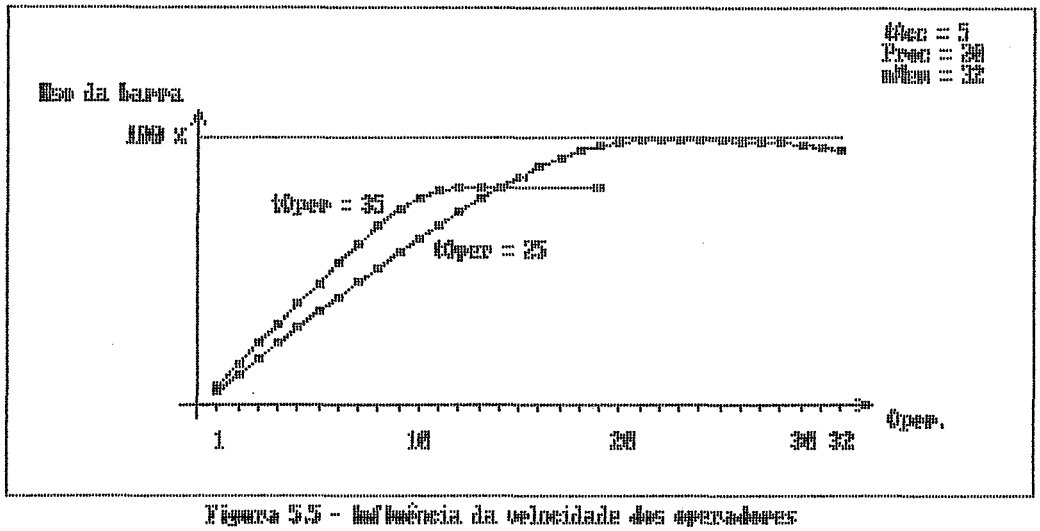


Figura 5.5 - Influência da velocidade dos operadores

V.5- INFLUÊNCIA DA VELOCIDADE DA MEMÓRIA

O tempo de acesso à memória influi no sistema mudando a inclinação da curva de aceleração e modificando o número de processadores que leva a barra à saturação. Não compromete o desempenho global do sistema, apenas modificando a quantidade "adequada" de módulos processadores.

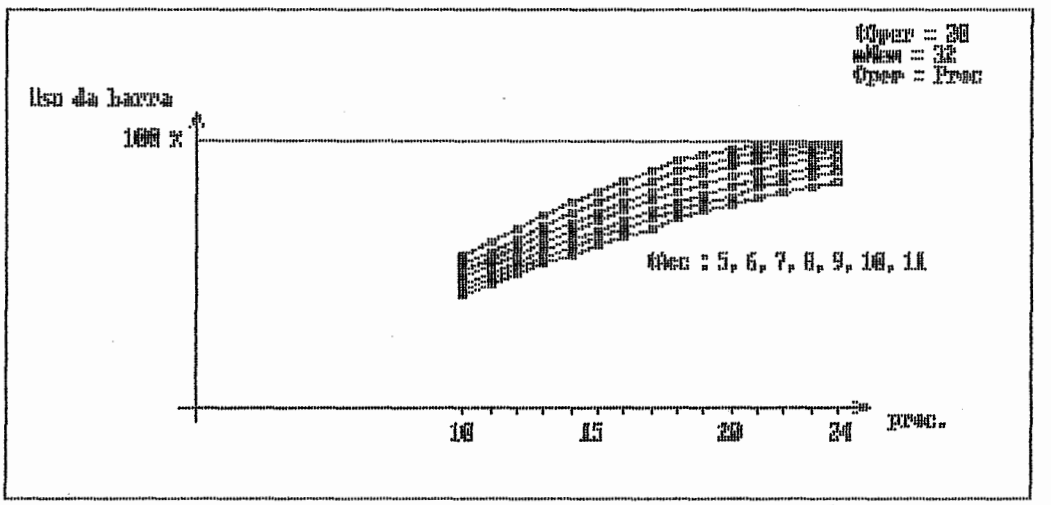


Figura 5.6 - Uso do barramento em função da velocidade da memória

V.6 - CUSTO DE SINCRONIZAÇÃO

A sincronização pouco onera o processamento enquanto o número de processadores é pequeno (menor que 20); ao aumentar-se sua quantidade, como era de se esperar, esse custo aumenta rapidamente. Para matrizes de ordem mais alta aconteceria o mesmo só que numa quantidade maior de processadores.

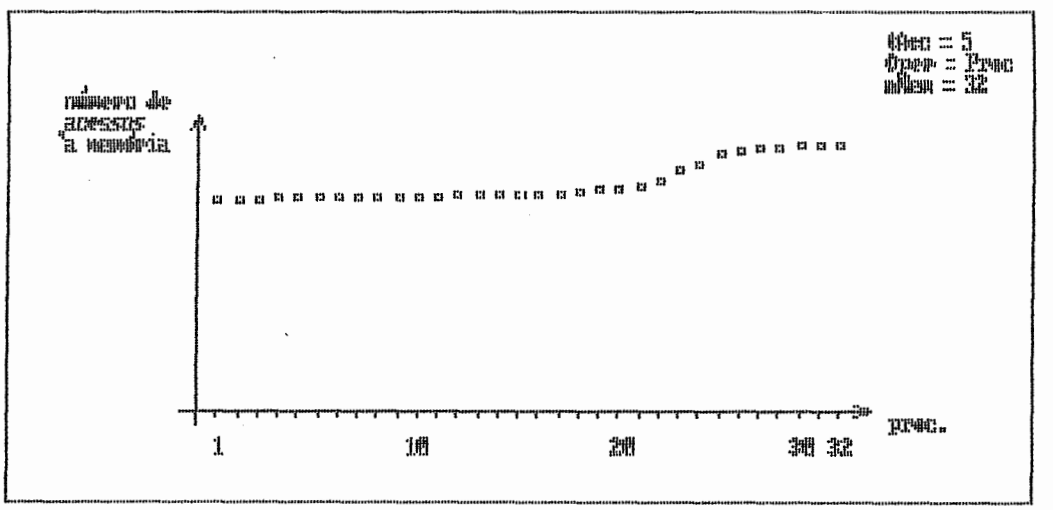


Figura 5.7 - Custo de comunicação-sincronização

CAPÍTULO VI

DISCUSSÃO

Pelos dados mostrados no capítulo anterior, a arquitetura opera muito próximo do esperado. A saturação da barra de memória, que é o fator determinante do desempenho da máquina, pôde sempre ser atingida, bastando mudar-se a quantidade de módulos do sistema. Um estudo das possíveis cargas de trabalho a serem empregadas no sistema pode, facilmente, determinar qual seria uma boa relação entre os números dos módulos presentes. Sugere-se, contudo, uma testagem mais ampla da arquitetura, com vistas a definir uma melhor estrutura interna dos processadores, mais adequada a processamento genérico.

VI.1 - FORMAS DE IMPLEMENTAÇÃO

A implementação da máquina pode ser toda feita com apenas cinco módulos diferentes:

- Unidade de controle
- Unidade Operadora
- Controlador de memória
- Árbitro de barramento
- Memória dinâmica

Essa arquitetura presta-se otimamente para uma implementação em circuitos integrados, devido a sua característica de alta modularidade, baixa complexidade e possibilidade de repetição. Naturalmente a modularização pode englobar mais de um tipo de módulo. Assim, se a tecnologia disponível for de alta densidade, pode-se, por exemplo, ou

colocar várias unidades de controle numa mesma pastilha ou colocar unidades de controle juntas com operadores e árbitros.

O desempenho esperado de uma máquina implementada com essa arquitetura deve se aproximar do desempenho obtido nas simulações. Não é facil prever quais seriam sua velocidade de operação e sua taxa de integração sem que se entre nos detalhes do projeto dos circuitos. Pode-se esperar contudo que a característica de modularidade da arquitetura permita uma grande flexibilidade de implementação. Estima-se, por exemplo, que se possa integrar duas Unidades de Controle por pastilha, numa tecnologia atual de GATE ARRAY CMOS. Criando-se toda a pastilha (método FULL CUSTOM) esse número deverá ultrapassar uma dezena. A velocidade que se poderia atingir nos barramentos deverá situar-se próxima a 30 MegaHertz, para uma implementação em nMOS ou CMOS.

Pode-se verificar, pela descrição da arquitetura, que, para uma máquina de 32 bits, cada Unidade de Controle terá um número muito grande de conexões externas:

Com o barramento de memória

- 32 pinos para o endereço;
- 32 pinos para o dado;
- 8 pinos para o número da Unidade de controle endereçando;
- 8 pinos para o número da Unidade de controle transferindo dados;
- 2 pinos para o tipo de operação (leitura/escrita e destrutiva/não destrutiva);

Sub-total = 82

Com o barramento dos operadores

- 32 pinos para o operando 1;
- 32 pinos para o operando 2;
- 32 pinos para o resultado;
- 8 pinos para a Unidade de controle transferindo operandos;
- 8 pinos para a Unidade de Controle recebendo um resultado;
- 8 pinos para a função a executar;

sub-total = 120

TOTAL = 202 pinos

A implementação de uma pastilha com mais de 200 pinos é comercialmente inviável. Se forem multiplexados todos os barramentos em um só verificar-se-á que existirão 6 tipos de ciclos:

- Endereçamento da memória
- Dado para a memória
- Dado da memória
- Operando 1 para o operador
- Operando 2 para o operador
- Resultado de uma operação

Num problema como a Multiplicação de matrizes (ou a Redução de Gauss), em que cada 5 acessos à memória correspondem 2 operações aritméticas, o total de ciclos de barramento multiplexado para cada conjunto completo é de:

- 5 endereçamentos;
- 5 dados lidos;
- 4 operandos;
- 2 resultados;

Na máquina teórica simulada, o total de ciclos era apenas de 5 (o número de acessos à memória) em dois barramentos. Para fins de implementação, pode-se esperar então que a velocidade efetiva do barramento multiplexado seja 3,2 vezes menor que a simulada. Isso pode parecer terrível porém, se observarmos que tal queda de desempenho é acompanhada por uma queda proporcional no número de bancos de memória, no de processadores e no de operadores, a compensação, na forma de custo, é proporcional.

VI.2 - MÁQUINA BIDIMENSIONAL

Nas simulações verificou-se que o tempo de processamento de um processador varia linearmente com a velocidade dos módulos de memória. Um tempo de acesso muito grande seria compensado pela presença de um número maior de processadores. Uma vez que a velocidade do sistema depende, basicamente da velocidade do barramento, isso não afetaria, em princípio, o desempenho da máquina.

Modificando-se um pouco a estrutura já descrita, ao invés de controladores de memória, coloca-se um outro tipo de módulo, a Interface Interbarras; esse módulo faria a ligação das unidades de controle com um outro tipo de barramento; este teria controladores de memória, com pastilhas de memória dinâmica nele ligadas. O tempo de comunicação entre os dois barramentos seria o responsável pelo aumento no tempo de acesso "visto" pela unidade de controle.

Pode-se ter então uma estrutura cruzada, de barramentos primários "verticais" onde estariam ligadas unidades de controle e módulos Interbarras e barramentos "hori-

zontais", onde ficariam os controladores de memória e também as Interfaces Interbarras (figura abaixo).

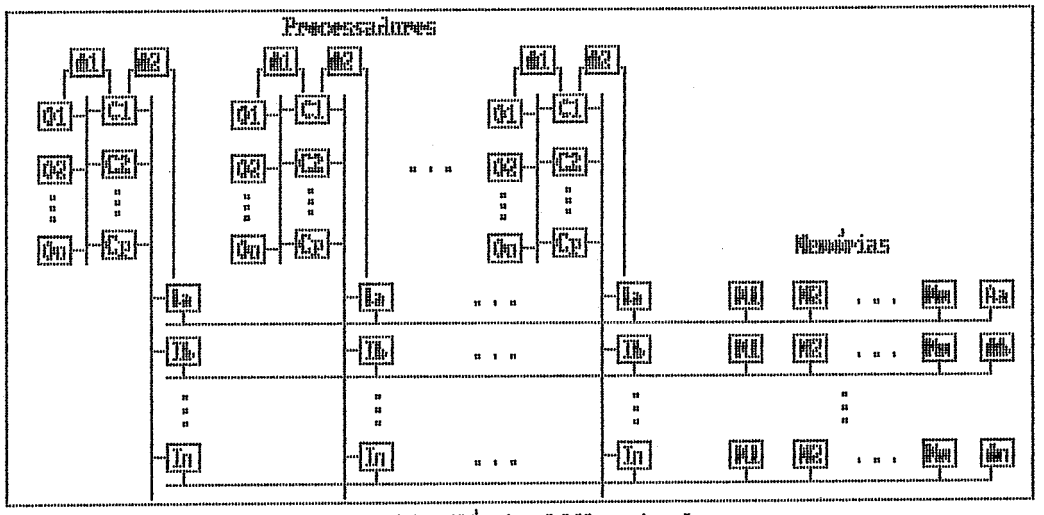


Figura 6.1 – Máquina bidimensional

Se em cada barramento "vertical" forem ligados, por exemplo, 30 unidades de controle e implementados 16 desses barramentos ligados a 16 barramentos "horizontais" com 8 bancos de memória cada, tem-se uma máquina com 480 processadores e capacidade de processamento com 16 vezes a capacidade de um barramento "vertical".

Por exemplo, se as pastilhas fossem construídas em tecnologia CMOS com tempo de barra de 50 nanosegundos (20 MegaHertz), a máquina teria a capacidade de 320 milhões de acessos à memória por segundo. Na resolução de um problema do tipo da multiplicação de matrizes que realiza duas operação de ponto flutuante para cada cinco acessos à memória, a máquina poderia apresentar uma capacidade de cálculo em torno de 128 MegaFlops.

Considerando-se o exposto no item anterior sobre a redução da velocidade por um fator de 3,2, devido à multiplexação dos barramentos, uma re-distribuição dos módulos numa quantidade maior de barramentos "verticais" restauraria essa perda de desempenho; nos barramentos horizontais a redução de velocidade seria por um fator de 2, devido à multiplexação dos dados e endereços, levando à duplicaçào do número de barramentos "horizontais".

VI.3 - TOLERÂNCIA A FALHAS

Conforme foi colocado no capítulo III, na definição da arquitetura não se levou em consideração aspectos como tolerância a falhas e confiabilidade.

Analisando-se apenas a máquina "unidimensional", podemos dizer que uma falha pode ocorrer de duas formas:

- internamente a um módulo
- externamente aos módulos, compromentendo o barramento.

O segundo tipo permite apenas a deteção da falha e não sua correção em tempo de execução. Um "curto" gerado por um módulo em um barramento, por exemplo, pode ser facilmente detectado porém só poderá ser corrigido pela remoção física do módulo defeituoso. Uma ruptura ou um curto nas trilhas de um barramento só poderá ser corrigida com a maquina desligada. A descoberta de uma falha de transmissão através de um barramento pode ser feita por mecanismos simples como a inserção de bits de paridade nos dados transmitidos ou uma verificação, em cada módulo, se o dado que este estiver enviando for igual àquele que estiver lendo do barramento.

Uma falha no interior de um módulo pode ser descoberta pela replicação dos módulos. Para cada unidade ativa existiria uma idêntica a ela, com a mesma numeração no sitema; uma delas, a ativa, aciona os barramentos; a outra, apenas observa se o dado que estiver "saindo" para o barramento é idêntico aquele que ela colocaria, caso fosse a unidade ativa. Se uma diferença for percebida, a segunda unidade pode levar a máquina a um estado de verificação.

Ao se constatar uma falha, a unidade defeituosa é desabilitada. Uma etapa de restauração do contexto é iniciada e, ou se substitui a unidade defeituosa por outra sobressalente ou esta é apenas desabilitada, passando seu processo e seu contexto para uma fila de processos pendentes.

No caso de uma falha num módulo de memória, a redundância funciona da mesma forma, apenas a restauração do contexto é mais demorada pois requer a cópia de cada posição de memória para o módulo que for entrar em funcionamento. Caso não haja módulos sobressalentes, o número de bancos será abaixado a metade e toda a memória deverá ser re-mapeada. Por exemplo: uma máquina com 16 bancos de memória ativos e nenhum sobressalente; ao se detectar a falha, 8 bancos serão reconfigurados, 1 será desabilitado e 7 passarão ao estado de sobressalentes.

Todos os procedimentos acima podem estar programados em cada unidade de controle. Ao se detectar a falha, uma delas executará o micro-programa de reconfiguração da máquina. No caso de implementações com mais de uma unidade de controle por pastilha, apenas um micro-código de recupe-

ração de erro por pastilha bastará para restaurar o funcionamento da máquina.

Na máquina bidimensional, uma falha num barramento pode ser, não apenas detectada, más também recuperada isolando-se o defeituoso. A restauração do estado original dos processadores daquele barramento pode apresentar problemas pois este não pode ser usado para a transferência dos contextos dos processadores. Pode-se pensar então, sem que isso onere demais a implementação, na criação de um barramento serial alternativo, comum a todos os módulos do sistema. A transferência de contexto, no caso de uma falha de barramento, seria feita por esse canal serial; a transferência seria lenta, é verdade, porém, como só seria feita em casos excepcionais, não teria nenhuma importância no desempenho normal da máquina.

Outros mecanismos melhores de proteção, deteção e correção podem ser aventados; o importante é lembrar que, devido às características de modularidade da arquitetura e à baixa complexidade dos seus módulos, qualquer mecanismo pode ser facilmente implantado.

CAPÍTULO VII

CONCLUSÕES

A Arquitetura de Computadores busca a descoberta e o estudo de novas estruturas e novas máquinas processadoras. A diversidade de tipos de máquinas e arquiteturas existentes mostra que, até o momento, nenhuma delas satisfaz plenamente todos os requisitos exigidos pela aplicação dos computadores na nossa sociedade. A busca de uma arquitetura geral que seja de ampla aplicabilidade e que não apresente as deficiências das já existentes é o objetivo principal de um pesquisador desse ramo da Ciência da Computação.

Objetivos intermediários, menos ambiciosos, visam conseguir arquiteturas com uma boa aplicabilidade em áreas mais restritas.

Neste trabalho foi proposta uma nova arquitetura de multiprocessadores. Pretende-se que sua aplicação possa ser de uso geral. Seu desempenho não deverá ser melhor que máquinas otimizadas para apresentarem um alto desempenho em áreas mais específicas. Seu comportamento foi validado apenas em processamento numérico através de simulações que apresentam um razoável grau de realismo. Pode-se argumentar que sua aplicação num espectro mais amplo de tipos de programas não apresentaria resultados tão satisfatórios. Somente uma testagem nas outras áreas de aplicação poderá levar a conclusões a esse respeito. Esse pode ser um dos desdobramentos deste trabalho.

Pretendeu-se aqui dar uma contribuição à computação abrindo o espaço para a criação de uma classe de máquinas generalistas com alto desempenho e com um alto grau de modularidade. A implementação dessas máquinas em circuitos Integrados VLSI e sua validação mais conclusiva é também um dos desdobramentos deste trabalho.

O tipo de posicionamento que levou à proposta dessa arquitetura foi aquele de questionar sempre se as soluções usadas para resolver um problema são as mais adequadas ou se existem outras de melhor aplicabilidade. Espera-se que a análise e a crítica dessa arquitetura e deste trabalho sejam feitas com o mesmo espírito.

REFERÊNCIAS BIBLIOGRÁFICAS

AOUIZERAT (1984) Roger Aouizerat, "Shared Memory Devices Can Link PDP-11s Into Load Balanced Clusters". Computer Technology Review Summer 1984.

BARBOSA (1988) Valmir C. Barbosa e Edil S. T. Fernandes, "Perspectivas em Processamento Paralelo para Computação Científica" - Pesquisa Operacional Vol 8 No.1 Jul 1988.

BRIGGS (1977) Faye A.Briggs e Edward S.Davidson,"Organization of Semiconductor Memories for Parallel Pipelined Processors" - IEEE Transactions on computers Vol.C26 No.2 fev 1977.

BURNETT (1970) G.J. Burnett e E.G. Coffman, " A Study of Interleaved Memory Systems" - Index Systems, Inc. Boston e Princeton University, Princeton New Jersey.

CHEN (1988) Marina C. Chen, "The Generation of a Class of Multipliers : Synthesizing Highly Parallel Algorithms in VLSI" - IEEE Transactions on Computers Vol.37 No.3 mar 1988.

HAMPTON (1986) Patrick Hampton, Richard Denker, "Multiprocessor APs Simplify Calculations for Image Processing". Computer Technology Review Summer 1986.

HWANG (1984) Kai Hwang e Faye A.Briggs, "Computer Architeture and Parallel Processing" - MacGraw Hill 1984.

INMOS (1987) Inmos Corporation, "Preliminary Data IMS T800 Transputer" - Inmos Corporation P.O. Box 16000 Colorado Springs, CO - USA 1987.

JONES (1980) Anita K. Jones e Petes Schwarz, "Experience Using Multiprocessor Systems - A Status Report", Computing Surveys Vol.12, No.2 1980.

KINNEY (1978) L. L. Kinney e R. G. Arnold, "Analysis of a Multiprocessor System with a Shared Bus" , University of Minesota e Honeywell Systems and Research Center - Sigarch Newsletter Vol.6 No.7 Abril 1978.

KRAJEWSKI (1985) Rich Krajewski, "Multiprocessing an Overview". Byte Vol. 10 No. 5 Mai 1985.

LEBLANC (1988) Thomas J. LeBlanc, "Problem Decomposition and Communication Tradeoff in a Shared Multiprocessor". Numerical Algorithms for Modern Parallel Computer Architetures, Vol.13, IMA Volumes in Mathematics and its Applications - Springer Verlag 1988.

LUNDSTROM (1987) Stephen F. Lundstrom, "Applications Considerations in the System Design of Highly Concurrent Multiprocessors" - IEEE Transactions on Computers Vol.36 No.11 nov 1987.

OBERMEIER (1988) Klaus K. Obermeier, "Side by Side". Byte Vol. 13 No. 12 Nov 1988.

OKI (1987) Oki Asic Data Book - Gate array, Standart Cell. Jun 1987.

RAMAMORTLHY (1977) C.V.Ramamortlhy e H.F. Li, "Pipeline Architeture" - Computing Surveys Vol.9 No.1 mar 1977.

SLEWLOREK (1982) Daniel P. Slewlorek, C. Gordon Bell e Allen Newell, "Computer Structures: Principles and Examples" - McGraw-Hill 1985.

STONE (1987) Harold S. Stone, " High Performance Computer Architeture" - Addison Wesley 1987.

TITUS (1981) Christopher A. Titus, Johnathan A. Titus e Leo Scanlon, "16 Bits Microprocessors" - Howard W. Sams 1981.

UBEROI (1985) Anil Uberoi, "Silicon Chip Advances Optimize Integration of 32 Bit Microprocessors". Computer Technology Review Spring 1985.

WACHS (1986) Andrew S. Wachs, "Parallel Processing in Real Time Enhances High End Performance". Computer Technology Review Summer 1986.

WALKER (1985) Paul Walker, "The Transputer". Byte Vol. 10 No. 5 Mai 1985.

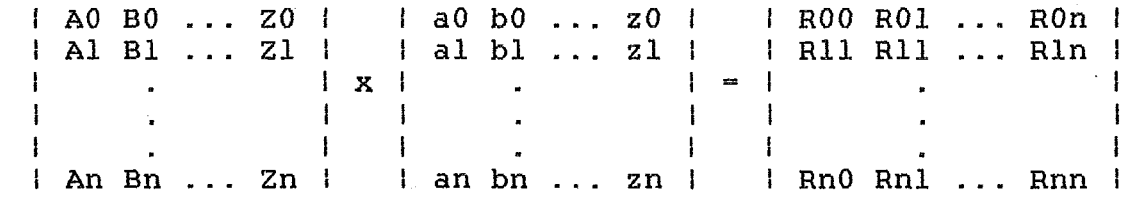
WESTE (1985) Neil H.E. Weste, Kamran Eshraghian, "Principles of CMOS VLSI Design". Addison Wesley 1985.

WILSON (1988) Pete Wilson, " The CPU Wars", Byte Vol. 13 No. 5 Mai 1988.

APÊNDICE I

PARALELIZACAO DE ALGORITMOS

1 - MULTIPLICAÇÃO DE 2 MATRIZES N x N



1.1 - ALGORITMO 1 (Um processo apenas)

```
Const
  N  =  xxx;

Var
  I,
  J,
  K   : Integer;
  V   : Real;
  M1,
  M2,
  MR  : Array [ 1..N, 1..N ] of Real; {Matriz}

BEGIN
  FOR I := 1 TO N DO
    FOR J := 1 TO N DO
      BEGIN
        V := 0;
        FOR K := 1 TO N DO
          V := V + M1 [ j, k ] * M2 [ K, J ];
        MR [ J, I ] := V;
      END;
END.
```

ou

```
BEGIN
  I := 1;
  REPEAT
    I := 1;
    REPEAT
      V := 0;
      K := 1;
      REPEAT;
        P := M1 [ J, K ];
        P := P * M2 [ K, I ];
        V := V + P;
        K := K + 1;
      UNTIL K > N;
      MR [ J, I ] := V;
      I := I + 1;
    UNTIL I > N;
    J := J + 1;
  UNTIL J > N;
END.
```

1.2 - CÓDIGO NA LINGUAGEM DA MÁQUINA

```
   N   := n;
  _B1  := BM1;
   B2  := BM2;
  _BR  := Bres;
   Teste := Nao igual a zero;
  _J   := 0;
J: BJ  := B2;
  _ I  := 0;
I: BI  := B1;
  _BI  := BI + I;
   V   := 0;
  _K   := N;
K: P   := (BJ)+;
  _P   := P * (BI);
   BI  := BI + N;
  _V   := V + P;
   K   := DEC ( K );
  _Se teste entao PC := PC - 3;   ( K )
  _(BR)+ := V;
   BJ  := B2;
  _I := INC ( I );
   Compare I, N;
  _Se teste entao PC := PC - 8;   ( I )
   B2 := B2 + N;
  _J := INC ( J );
   Compare J, N
  _Se teste entao PC := PC - 11; ( J )
   PC := 0;   ( Volta ao sistema )
```

1.3 - ALGORITMO 2 (n processos)

```
BEGIN
  REPEAT
    REPEAT                          |
      J := SincJz;                  |
    UNTIL J > 0;                    |
    I := SincI;                     |
    IF J > N THEN                   |
      BEGIN                         |
        SincI := I + 1;             |--- Região de sincronização
        J := J + 1;                 |
        IF I > N THEN               |
          Fim := True;              |
      END                           |
    ELSE                            |
      SincJ := J + 1;               |
    V := 0;
    K := N
    REPEAT
      P := M1 [ J, K ];
      P := P * M2 [ K, I ];
      V := V + P;
      K := K + 1;
    UNTIL K > N;
    MR [ J, I ] := V;
  UNTIL Fim;
END.
```

1.4 - CÓDIGO NA LINGUAGEM DA MÁQUINA

```
   _N := n;
    SincJ := xxxx;
   _SincI := yyyy;
    SincJz := SincJ + $80000000;
   _SincBRes := zzzz;
    Cont := cccc;
   _B1 := aaaa;
    B2 := bbbb;
   _Teste := Igual a zero;
S:  J := ( SincJz );                          |
   _Se teste entao PC := DEC ( PC ) ( S )     |
S2: I := ( SincI );                           |
   _BR := ( SincBRes );                       |- Sincronização
    Comp J, N;                                |
   _Se teste entao PC := PC + 11;    ( I );   |
S3: ( SincBRes ) := INC ( BR );               |
   _( SincJ ) := INC ( J );                   |
    J := J + N;
   _BI := B1;
```

```
     BI := BI + I;
    _BJ := B2;
     BJ := BJ + J;
    _V := 0;
     K := N;
    _Teste := Nao igual a zero;
K:   P   := (BJ)+;
    _P   := P * (BI);
     BI := BI + N;
    _V   := V + P;
     K   := DEC ( K );
    _Se teste entao PC := PC - 3;     ( K )
     (BR)+ := V;
    _Teste := Igual a zero;
S1: J := ( SincJz );                          |
    _Se teste entao PC := DEC ( PC ); (S1)    |-- Sincronização
     ( Cont ) := INC ( Cont );                |
    _PC := PC - 13                    (S2)    |
I:   ( SincI ) := INC ( I );
    _J := 1;
     Compare I, N;
     Se Teste entao PC := PC - 14;    ( S3 )
```

2 - REDUÇÃO DE GAUSS

Sistema de 5 equações a 5 incógnitas:

```
x1 * a11 + x2 * a12 + x3 * a13 + x4 * a14 + x5 * a15 = b1
x1 * a21 + x2 * a22 + x3 * a23 + x4 * a24 + x5 * a25 = b2
x1 * a31 + x2 * a32 + x3 * a33 + x4 * a34 + x5 * a35 = b3
x1 * a41 + x2 * a42 + x3 * a43 + x4 * a44 + x5 * a45 = b4
x1 * a51 + x2 * a52 + x3 * a53 + x4 * a54 + x5 * a55 = b5
```

Representação matricial:

```
| x1 |   | a11 a12 a13 a14 a15 |   | b1 |
| x2 |   | a21 a22 a23 a24 a25 |   | b2 |
| x3 | x | a31 a32 a33 a34 a35 | = | b3 |
| x4 |   | a41 a42 a43 a44 a45 |   | b4 |
| x5 |   | a51 a52 a53 a54 a55 |   | b5 |
```

Depois de executada a redução:

```
| x1 |   | c11 c12 c13 c14 c15 |   | d1 |
| x2 |   |  0  c22 c23 c24 c25 |   | d2 |
| x3 | x |  0   0  c33 c34 c35 | = | d3 |
| x4 |   |  0   0   0  c44 c45 |   | d4 |
| x5 |   |  0   0   0   0  c55 |   | d5 |
```

Representação da conversão feita:

```
x * A = b   ==>  x * C = d
```

Operações lineares válidas sobre uma matriz:

```
ci1 = ci1 * k                ci1 = ci1 + cj1
ci2 = ci2 * k                ci2 = ci2 + cj2
ci3 = ci3 * k        e       ci3 = ci3 + cj3
ci4 = ci4 * k                ci4 = ci4 + cj4
ci5 = ci5 * k                ci5 = ci5 + cj5
di =  di * k                 di =  di + dj
```

2.1 - ALGORITMO 1 (Um processo apenas)

```
Const
  N  =  xxx;
Var
  X, Y, X2  : Integer;
  F   : Real;
  M   : Array [ 1..N, 1..N ] of Real;{Matriz}
  B   : Array [ 1..N ] of Real;      {Vetor de resultados}
BEGIN
  FOR X := 1 TO N - 1 DO
    FOR Y := 2 TO N DO
      BEGIN
        F := M [ Y, X ] / M [ X, X ];
        FOR X2 := X TO N DO
          M [ Y, X2 ] - M [ X, X2 ] * F;
        B [ Y ] := B [ Y ] - B [ X ] * F;
      END;
END;
```

ou

```
BEGIN
  X := 1;
  REPEAT
    Y := X + 1;
    REPEAT
      F := M [ Y, X ] / M [ X, X ];
      K := X;
      REPEAT
        F1 := M [ X, K ] * F;
        M [ Y, K ] := M [ Y, K ] - F1;
      UNTIL K > N;
      F1 := B [ X ] * F;
      B [ Y ] := B [ Y ] - F1;
      Y := Y + 1;
    UNTIL Y > N;
    X := X + 1;
  UNTIL X = N;
END.
```

2.2 - CODIGO NA LINGUAGEM DE MAQUINA

Supõe-se que os termos da matriz e os valores dos resultados encontram-se previamente carregados na memória, da seguinte forma:

a11, a12, ..., a1n, b1,

a21, a22, ..., a2n, b2,

.

.

.

an1, an2, ..., ann, bn.

```
N,      Dimensao da matriz
N1,     Dimensao da matriz + 1
N2,     Dimensao da matriz + 2
X,      Contador de colunas
Y,      Contador de linhas
K,      Contador auxiliar de colunas
BX,     Base da linha x
BY,     Base da linha y
BL1,    Base da Linha 1 ( linha X )
BL2,    Base da Linha 2 ( linha Y )
F,      Auxiliar de calculos

         N := xxx
        _N1 := INC [ N ]
         N2 := INC [ N ]
        _BX := ( PC )+
        _  # BaseDaMatriz
         BY := BX
        _X := 1
LoopX:   BY := BX + N1
        _Y := INC [ X ]
LoopY:   BL1 := BX
        _BL2 := BY
         F := ( BL2 ) / ( BL1 )
        _K := X
LoopK:  _( BL2 )+ := ( BL2 ) - ( BL1 )+ * F
         K := INC [ K ]
        _Compare ( K, N2 ) Se <> entao PC:=PC-1 (( LoopK ))
         BY := BY + N1
        _Y := INC [ Y ]
```

```
        _Compare( Y, N1 ) Se <> entao PC:=PC-6 (( LoopY ))
         BX := BX + N2
        _X := INC [ X ]
        _Compare ( X, N ) Se <> entao PC:=PC-9  (( LoopX ))
        _(( Volta ao S.O. ))
```

2.3 - ALGORITMO 2 (n processos)

```
BEGIN
  REPEAT
    Y := yAtual;        { Leitura destrutiva de yAtual }
  UNTIL Y > 0;
  IF Y <= N THEN
    BEGIN
      Y := Y + 1;
      yAtual := Y;      { Atualiza o ponteiro de processos }
      X := 1;
      REPEAT            { Calcula ( y-1 ) linhas da matriz }
        K := X;
        F := M [ Y, X ] / M [ X, X ];
        REPEAT
          M [ Y, K ] := M [ Y, K ] - M [ X, K ] * F;
          K := K + 1;
        UNTIL K > ( N + 1 );
        X := X + 1;
        REPEAT          { Espera pela habilitação }
          H := Contador
        UNTIL H = X;
      UNTIL X > Y;
      Contador := X;    { Habilita outro processo }
    END
  ELSE
   yAtual := Y;         { Restaura o ponteiro de processos }
END;
```

2.4 - CÓDIGO NA LINGUAGEM DA MÁQUINA

```
Calcula: BV := ( PC )+
        _BX := ( PC )+
        _ # InicioDaAreaDeVariaveis
        _ # InicioDaAreaDeDados
         N1 := INC [ ( BV )+ ]
        _BY := INC [ BV ]
        _Y := (BV)destr - Se Y=0 entao PC:=PC-1, espera nnn
        _Compare ( Y, N1 ) Se = entao PC:=PC+12  (( VSO1 ))
         BaseY := ( BV )
        _BaseY := BaseY + N1
         ( BY )+ := BaseY
        _( BV ) := INC [ Y ]              ;Libera o semáforo
         X := 1
        _N2 := INC [ Y ]
```

```
LoopX:    BL1 := BX
          _BL2 := BaseY
          _F := ( BL2 ) / ( BL1 )
           K := X
          _X := INC [ X ]
LoopK:  _( BL2 )+ := ( BL2 ) - ( BL1 )+ * F
           K := INC [ K ]
          _Compare ( X, N2 ) Se <> entao PC:=PC-2 (( LoopK ))
          _Compare ( X, Y ) Se = entao PC:=PC+4    (( VSO2 ))
           H := ( BY )
          _Compare ( H, X ) Se <> entao PC := PC - 1
           BX := BX + N1
          _BX := INC [ BX ]
           BaseY := BaseY + N1
          _PC := PC - 9                           (( LoopX ))

VSO1:      ( BV ) := Y
          _PC := INC [ PC ]
VSO2:     _( BY ) := X
          _PC := ( PC )
          _ #  EnderecoDeVoltaAoSO
```

APÊNDICE II

DESCRIÇÃO DO SIMULADOR

Foram criados vários programas simuladores para este trabalho:

- Simulador dos algoritmos sequenciais - para a depuração dos algoritmos sequenciais;
- Simulador dos algoritmos paralelos - para a depuração da paralelização dos algoritmos seriais;
- Simulador do processador - para a depuração do conjunto de instruções e dos programas em linguagem de máquina;
- Simulador do multiprocessador para o problema da Redução de Gauss;
- Simulador do multiprocessador para o problema da Multiplicação de matrizes.

além de programas para a coleta dos dados gerados pelas simulações, para a geração dos gráficos e para a estimativa dos custos de conversão e de sincronização.

É preciso explicar que os dois últimos simuladores não puderam ser integrados em um só devido a problemas de tamanho da área de dados e da diferença de tipo de dados entre eles. A Redução de Gauss, por executar operações de divisão, necessitava de dados no formato REAL, porém usava um pequeno espaço pois precisava apenas de uma matriz N x N. A Multiplicação de Matrizes, por outro lado, por executar apenas operações de soma e multiplicação, pôde usar dados no formato INTEIRO, porém necessitava de um maior espaço por ter que armazenar três matrizes N x N. Uma coisa compensava a outra, em termos de espaço de memória, só que

dentro de um mesmo código não foi possível compatibilizar os dois tipos de dados: REAL e INTEIRO;

A listagem que faz parte deste anexo é do último simulador, para a Multiplicação de Matrizes. O Programa foi feito em Pascal, usando-se o Turbo Pascal versão 3.

OPERAÇÃO

A operação do simulador é bastante simples. No arquivo de parâmetros "PAR." Colocam-se:

1 - Ordem da matriz (de 2 à 49);

2 - Tempo de acesso à memória (de 3 à 253);

3 - Tempo médio de operação (de 3 à 253);

4 - Número de bancos de memória { 1, 2, 4, 8, 16, 32 };

5 - Número de processadores (de 1 à 56);

6 - Número de Unidades Operadoras (de 1 à 64);

O arquivo "PAR." tem duas linhas apenas, cada uma delas com um conjunto dos parâmetros acima. O simulador executará até seis laços iterativos, um para cada parâmetro, com o parâmetro 1 no laço mais externo e o parâmetro 6 no mais interno. Na figura abaixo podemos ver um arquivo "PAR." Típico. A cada iteração, varia-se um dos parâmetros e salva-se o resultado da simulação num arquivo "Dados??.SIM".

Ao se iniciar um conjunto de simulações, uma variável guardada num arquivo chamado "NumAtual" é incrementada. Essa variável é usada para indexar o arquivo "Dados??.SIM", de forma que nunca se escreve dados novos sobre outros já existentes.

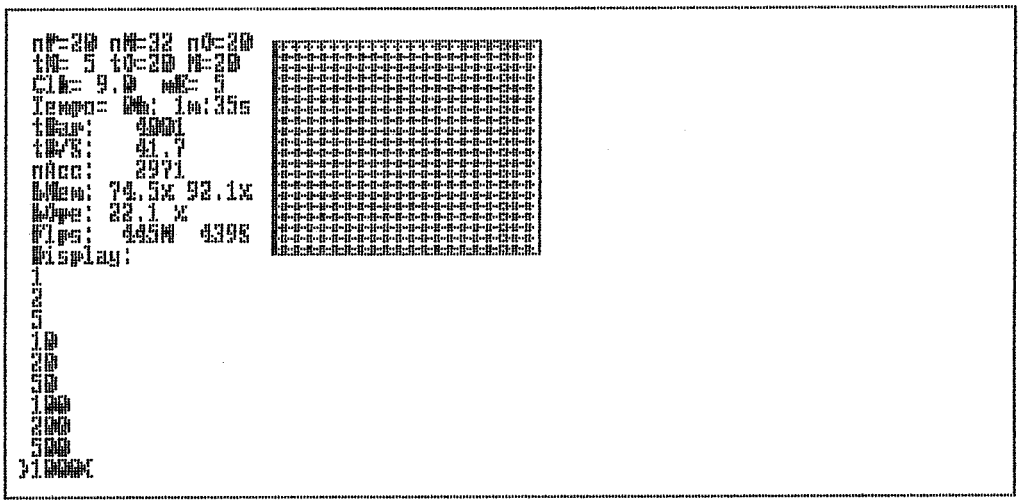
Arquivo de parâmetros do simulador

Ao se iniciar um bloco de simulações, o programa "mede" a velocidade do computador através de uma avaliação do seu tempo médio de cálculo aritmético. Essa velocidade é mostrada na tela do simulador no campo "Clk=...".

Uma tela do simulador pode ser vista na figura abaixo. Nela, vemos, na primeira linha, os números de módulos ativos na máquina (Processadores, Memórias e Operadores); na segunda linha temos o tempo de acesso à memória, o tempo médio de operação e a ordem da matriz. Na quarta linha temos um relógio que é zerado ao início de cada simulação. Nas linhas 5 a 10 temos:

- Ciclos de barra até o momento (tempo virtual da máquina);
- Ciclos de barra por segundo (velocidade de simulação);
- Número total de acessos à memória até o momento;
- Uso do barramento de memória: médio e de pico;
- Uso do barramento de operadores;
- Número de operações feitas por todos os operadores: multiplicações, divisões (m) e adições e subtrações (s).

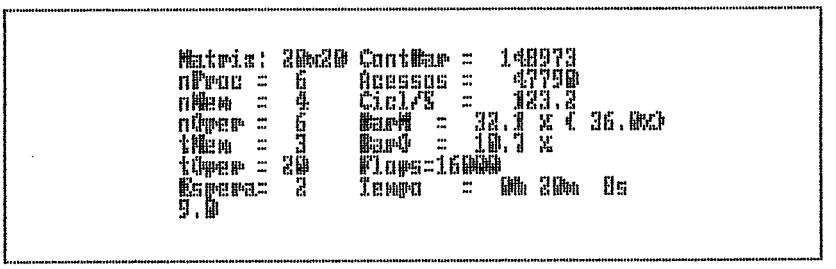
A cada N ciclos de barra, a tela é atualizada. Esse valor N pode ser escolhido usando-se as setas "^" e "v" do teclado; o valor N será mostrado entre os sinais "<" e ">" nas linhas 12 a 21. Na figura abaixo, o valor de N é 1000.



Tela de execução do simulador

Na parte direita da tela temos uma representação da matriz resultado da multiplicação. A matriz, ao início de cada simulação é toda zerada. Conforme um termo é calculado, um sinal "+" é colocado na posição do termo na matriz; ao final de cada simulação, todos os termos deverão estar com o sinal "+"; a observação da matriz permite verificar o andamento da simulação. Podemos ver na figura acima que a simulação encontra-se no seu início, com apenas alguns termos já calculados. Caso se deseje interromper a simulação, basta pressionar as teclas <F1> e <ESC> em sequência. O simulador fechará o arquivo "Dados??.SIM" com o resultado de todas as simulações feitas até o instante da interrupção e pedirá que se pressione a tecla <ESC> novamente. Ao fazê-lo, o programa terminará a execução, voltando o controle ao "DOS".

Na figura abaixo podemos ver o resultado de uma simulação, na forma como é criado nos arquivos "Dados??.SIM".



Arquivo de resultados gerado pelo simulador

```
{
*************************************************************
*         T E S E     D E     M E S T R A D O               *
*         ---------------------------------                 *
*          SIMULADOR DO MULTIPROCESSADOR                    *
*          ----------------------------                     *
*           MULTIPLICACAO  DE MATRIZES     Data : 07/04/90 *
*************************************************************
}

Program Matriz ( Input, Output );

{
     ************************************************
*******   E S T R U T U R A S     G L O B A I S     *******
     ************************************************
}
Const
  DrvRd   : String [ 16 ] = '\sn\';
  DrvWr   : String [ 16 ] = '\sn\;

  nMatriz     = 5;
  LFilaIn     = 63;
  LFilaOut    = 63;
  nProcCt     = 55;   { 260 bytes cada }
  nMemCt      = 31;   { 672 bytes cada ( fila = 63 ) }
  nOperCt     = 63;   { 16 bytes cada }
  tAccCt      = 3;
  tOperCt     = 25;
  ModEsperaCt = 1;

  MD  = 10;    { 1000 }
  VD  : Array [ 1..10 ] of Integer =
          ( 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000 );

  ModPegaCar  = 5;
  CorFundo    = 0;
  CorLetra    = 15;
  CorTitulo   = 15;
  LimMem      = 7600;
  InicioPar   = 90;
  InicioM1    = 101;
  InicioM2    = 2601;
  InicioMP    = 5101;
  nInstr      = 34;

Type
  StringExec  = String [ 70 ];
  String16    = String [ 16 ];
  String32    = String [ 32 ];
  String80    = String [ 70 ];
  String255   = String [ 255 ];
```

```
LProc  = Record
           Reg           : Array [ 0..24 ] of Integer;
           PedeEnd,
           PedeDadoWR,
           PedeDadoRD,
           PedeOperacao,
           PedeResultado,
           LeituraWR,
           ApagaDado,
           Carry,
           Zero          : Boolean;
           ContEspera    : Integer;
           Operacao,
           RegFunc,
           Posicao       : Byte;
           Car1, Car2    : Char;
           StExec        : StringExec;
           nOpCod,
           nRD,
           nWR,
           FlopMult,
           FlopSoma      : Real;
         End;

LProc2 = Record
           Reg           : Array [ 0..15 ] of Integer;
           RegI,
           RegE,
           RegL, RegW,
           RegRes,
           RegX, RegY,
           RegZ,
           RegN          : Integer;
           B             : Array [ 1..9 ] of Boolean;
           ContEspera    : Integer;
           Operacao,
           RegFunc,
           Posicao       : Byte;
           Car1, Car2    : Char;
           StExec        : StringExec;
           nOpCod,
           nRD,
           nWR,
           FlopMult,
           FlopSoma      : Real;
         End;

LProcN = Record
           Reg           : Array [ 0..15 ] of Integer;
           RegI, RegE,
           RegL, RegW,
           RegRes,
           RegX, RegY,
           RegZ, RegN    : Integer;
           PedeEnd,
           PedeDadoWR,
           PedeDadoRD,
```

```
            PedeOperacao,
            PedeResultado,
            LeituraWR,
            ApagaDado,
            Carry,
            Zero          : Boolean;
            ContEspera    : Integer;
            Operacao,
            RegFunc,
            Posicao       : Byte;
            Car1, Car2    : Char;
            StExec        : StringExec;
            nOpCod,
            nRD, nWR,
            FlopMult,
            FlopSoma      : Real;
          End;

RecIn   = Record
            Ender, Dado   : Integer;
            Proc          : Byte;
            ApagaDado,
            LeituraWR     : Boolean;
          End;
RecOut  = Record
             Dado   : Integer;
             Proc   : Byte;
           End;

LMem    = Record
            FilaIn        : Array [ 0..LFilaIn  ] of RecIn;
            FilaOut       : Array [ 0..LFilaOut ] of RecOut;
            PontInRD,
            PontInWR,
            PontOutRD,
            PontOutWR,
            ContTempo,
            TamFilaIn,
            TamFilaOut,
            Estado        : Byte;
            PedeDado      : Boolean;
            Leituras,
            Escritas,
            ContFilaIn,
            ContFilaOut   : Real;
          End;
LOper   = Record
            FlopMult, FlopSoma,
            Oper1, Oper2,
            Resultado     : Integer;
            Proc,
            Operacao,
            ContTempo,
            Estado        : Byte;
            Zero,
            PedeRes       : Boolean;
          End;
```

```
Const
  StProgr  : Array [ 0..nInstr ] of StringExec =

{0} ( 'eFILWeWr01+F  e0ILWeWr11+0',
{1}   '00',
{2}   'e0ILWeWr21+0  e0ILWeWr31+0',
{3}   'e0SaILWeWrRa41x4K0c_Tz  N1t_G_',
{4}   'e0+e+eILWeWr51  x4y2c_TMxFK9o+xzo+Fz+F+F+F+F+F+F+F',
{5}   'e0+eILWeWr61  x3y5c_TM  +F+F',
{6}   'x5ylo-5z  x6ylo+6z',
{7}   'e0+ew6IEWw',
{8}   'e0+e+ew5+wIEWw  e0w4+wIEWw',
{9}   'K08y  K09y',
{10}  'e6+6ILWeWrx1  e5ILWeWry1O*IOWoWR  yxx8O+IOWoWR  8z',
{11}  'x5ylo+5z',
{12}  '+9  x1y9c_Tz  xFK9o+xzK6o+Fz',
{13}  'e6+6ILWeWrx1  e5ILWeWry1O*IOWoWR  yxx8O+IOWoWR  8z',
{14}  'x5ylo+5z',
{15}  '+9  x1y9c_Tz  xFK6o+xzo+Fz',
{16}  'e6+6ILWeWrx1  e5ILWeWry1O*IOWoWR  yxx8O+IOWoWR  8z',
{17}  'x5ylo+5z',
{18}  '+9  x1y9c_Tz  xFK9o+Fz',
{19}  'e6+6ILWeWrx1  e5ILWeWry1O*IOWoWR  yxx8O+IOWoWR  8z',
{20}  'x5ylo+5z',
{21}  '+9  x1y9c_Tz  +F+F+F+F+F+F',
{22}  'e6+6ILWeWrx1  e5ILWeWry1O*IOWoWR  yxx8O+IOWoWR  8z',
{23}  'x5ylo+5z',
{24}  '+9  x1y9c_Tz  +F+F+F',
{25}  'e6+6ILWeWrx1  e5ILWeWry1O*IOWoWR  yxx8O+IOWoWR  8z',
{26}  'x5ylo+5z',
{27}  '+9  x1y9c_TM  xFK9o-xzo-Fz',
{28}  'e4w8IEWw',
{29}  'x0K3o+0z  x2y4c_TM  K0Fy',     {xFKFo-Fz-F-F',}
{30}  'e0w4+wIEWw',
{31}  'd_SaWa', '00', '00', '00' );
```

```
{-------------------------}
{+-------------+   +-------------+     +-------------+
 |PL           |   |PX       x   |FimC |PP           |
 |             |   |         x   |     |             |
 |             | x |         x   |  =  |             |
 |x x x x x x  |   |         x   |     |         P   |
 |             |   |         x   |     |             |
 |             |   |         x   |     |        FimP|
 +-------------+   +-------------+     +-------------+
```

```
Registradores
0 - E
1 - N
2 - FimP     FimMProduto
3 - FimC     FimColuna
4 - PP       PosicaoMP
5 - PX       PosicaoM2
6 - PL       PosicaoM1
8 - P
9 - K
```

```
 0  Inic:E := (PC)+
        _N := (E)+                          ;OrdemDaMatriz
 2       FimP := (E)+                       ;FimProd
        _FimC := (E)+                       ;FimCol
                                  ;InicioMP    0 - InicioM1     1
 3 Sinc: PP := (E) destr          ;InicioM2    2 - FimDeCalculo 3
        _Se PP=0 Entao PC := DEC [PC] ( Espera xx )
 4       PX := (E+2)
        _Comp [PP,FimP] Se > Entao PC := PC + 25    ;((Fim))
 5       PL := (E+1)
        _Comp [FimC,PX] Se > Entao PC:= PC + 2      ;((A))
 6       PX := PX - N
        _PL := PL + N
 7      _(E+1) := PL
 8 A:    (E+2) := INC [PX]
        _(E) := INC [PP]                  ;Libera sincronizador
 9       P := 0
        _K := 0
10 Loop:_P := P + (PL)+ * (PX)
11      _PX := PX + N
12       K := INC [ K ]
        _Comp [N,K] Se = Entao PC := PC + 15        ;((FimL))
13      _P := P + (PL)+ * (PX)
14      _PX := PX + N
15       K := INC [ K ]
        _Comp [N,K] Se = Entao PC := PC + 12        ;((FimL))
16      _P := P + (PL)+ * (PX)
17      _PX := PX + N
18       K := INC [ K ]
        _Comp [N,K] Se = Entao PC := PC + 9         ;((FimL))
19      _P := P + (PL)+ * (PX)
20      _PX := PX + N
21       K := INC [ K ]
        _Comp [N,K] Se = Entao PC := PC + 6         ;((FimL))
22      _P := P + (PL)+ * (PX)
23      _PX := PX + N
24       K := INC [ K ]
        _Comp [N,K] Se = Entao PC := PC + 3         ;((FimL))
25      _P := P + (PL)+ * (PX)
26      _PX := PX + N
27       K := INC [ K ]
        _Comp [N,K] Se > Entao PC := PC - 18        ;((Loop))
28 FimL:_(PP) := P
29       E := E + 3
        _Comp [FimP,PP] Se > Entao PC - 21
30 Fim: _(E) := PP                        ;Libera sincronizador
31 VaSO:_Halt
}

Var
  StatusMem     : Array [ 0..nMemCt ] of LMem;
  StatusOper    : Array [ 0..nOperCt ] of LOper;
  StatusProc    : Array [ 0..nProcCt ] of LProc;
  StatusProc2   : Array [ 0..nProcCt ] of LProc2
                                    Absolute StatusProc;
```

```
StatusProcN   : Array [ 0..nProcCt ] of LProcN
                                        Absolute StatusProc;

DadosM        : Array [ 0..LimMem ] of Integer;

BarMem   : Record
             Endereco,
             Dado           : Integer;
             PedeEnd,
             PedeDadoMem,
             PedeDadoProc   : Array [ 0..63 ] of Boolean;
             ApagaDado,
             LeituraWR      : Boolean;
             ProcEnd,
             ProcDado,
             HabEnd,
             HabDadoMem,
             HabDadoProc    : Byte;
           End;

BarOper  : Record
             Oper1, Oper2,
             Resultado      : Integer;
             PedeOper,
             PedeRes        : Array [0..63] of Boolean;
             ProcOper,
             ProcRes,
             HabOper,
             HabRes,
             Operacao       : Byte;
             OperDisponivel : Integer;
             Zero           : Boolean;
           End;
QuaseFim,
FimForcado,
Fim          : Boolean;

UsoMax,
ContBar,
BarDadoOciosa,
BarEndOciosa,
BarOperOciosa,
BarResOciosa,
FlopMultTotal,
FlopSomaTotal,
ContAcc,
ContRD,
ContWR,
ContOpCod   : Real;

ContBarInt,
BarDadoOciosaInt,
BarEndOciosaInt,
BarOperOciosaInt,
BarResOciosaInt,
FlopMultTotalInt,
```

```
  FlopSomaTotalInt,
  ContAccInt,
  ContRDInt,
  ContWRInt,
  ContOpCodInt   : Integer;

  Depuracao,
  PassoAPasso    : Boolean;

  Ni,  Nf,  OrdemDaMatriz,
  nPi, nPf, nProc,
  nMi, nMf, nMem,
  nOi, nOf, nOper,
  tAi, tAf, tAcc,
  tOi, tOf, tOper, tOperSoma, tOperMult,
  mEi, mEf, ModEspera,

  Numatual,
  nProcAtivos,
  ContDisplay,
  ContPegaCar,
  VelDisplay,
  ModDisplay,

  Horas,
  Minutos,
  Segundos,
  Centesimos    : Integer;
  I             : Byte;
  Clock         : Real;

  NomeArq   : String32;
  Arq       : Text;
  ErroIo    : Boolean;

{
    ************************************************
******* V A R I A V E I S    D O    V I D E O   *******
    ************************************************
}
  TelaGr0 : Array [1..50, 1..160] of Byte Absolute $B800:0;
  TelaGr1 : Array [1..50, 1..160] of Byte Absolute $BA00:0;
  TelaGr2 : Array [1..50, 1..160] of Byte Absolute $B805:0;
  TelaGr3 : Array [1..50, 1..160] of Byte Absolute $BA05:0;

  BGr0, BGr1, BGr2, BGr3,
  BGr4, BGr5, BGr6, BGr7  : Array [ 1..80 ] of Byte;

  Base      : Array [ 1..192, 1..80 ] of Byte;

  VidGr0  : Array [ 1..50, 1..320 ] of Byte Absolute BGr0;
  VidGr1  : Array [ 1..50, 1..320 ] of Byte Absolute BGr1;
  VidGr2  : Array [ 1..50, 1..320 ] of Byte Absolute BGr2;
  VidGr3  : Array [ 1..50, 1..320 ] of Byte Absolute BGr3;
  Vid     : Array [ 1..16000 ] of Byte Absolute VidGr0;
```

```
  VidTx0   : Array [ 1..25, 1..640 ] of Byte Absolute BGr0;
  VidTx1   : Array [ 1..25, 1..640 ] of Byte Absolute BGr1;
  VidTx2   : Array [ 1..25, 1..640 ] of Byte Absolute BGr2;
  VidTx3   : Array [ 1..25, 1..640 ] of Byte Absolute BGr3;
  VidTx4   : Array [ 1..25, 1..640 ] of Byte Absolute BGr4;
  VidTx5   : Array [ 1..25, 1..640 ] of Byte Absolute BGr5;
  VidTx6   : Array [ 1..25, 1..640 ] of Byte Absolute BGr6;
  VidTx7   : Array [ 1..25, 1..640 ] of Byte Absolute BGr7;

  TTx0, TTx1, TTx2, TTx3,
  TTx4, TTx5, TTx6, TTx7   : Array [ 0..127 ] of Byte;

  TTx    : Array [ 0.. 7, 0..127 ] of Byte Absolute TTx0;

  TSin0,
  TSin1,
  TSin2,
  TSin3    : Array [ 1..5 ] of Byte;
  TSin4    : Array [ 0..3, 1..5 ] of Byte Absolute TSin0;

  Car, Car2  : Char;
  Lin, Col   : Byte;
  CorFL,
  CorFT      : Integer;

{
     ************************************************
*******     R O T I N A S     D O     B I O S       *******
     ************************************************
}
Procedure APAGA_VIDEO;

BEGIN
  FillChar ( Vid, 16000, 0 );
END;

{--------------------------}
Procedure TECLE_ESC;

BEGIN
  WRITE ( 'Tecle ESC para voltar ao sistema' );
  REPEAT
    READ ( KBD, Car );
  UNTIL Ord ( Car ) = 27;
  TextMode;
  ClrScr;
  Halt;
END;

{--------------------------}
Procedure ERRO ( St  : String80 );

Var
  I, J  : Integer;
```

```
BEGIN
  GotoXY ( 1, 23 );
  WRITELN ( St );
  IF NOT ( Depuracao ) THEN
    BEGIN
      {$I-} Close ( Arq ) {$I+};
      ErroIO := ( IOresult > 0 );
      IF ErroIO THEN
        WRITELN ('Erro ao fechar o arquivo "'+NomeArq+'"');
    END;
  TECLE_ESC;
END;

{---------------------------}
Procedure MEDE_VELOCIDADE ( Var  R : Real );

Var
  I    : Real;
  J    : Integer;
  Reg  : Record
           AL, AH  : Byte;
           BX, CX, DX, BP, SI, DI, DS, ES, FLAGS : Integer;
         End;
BEGIN
  WRITELN ('Medindo "velocidade" do computador - AGUARDE');
  I := 1;
  Reg.AH := $01;
  Reg.CX := 0;
  Reg.DX := 0;
  INTR ( $1A, Reg );
  REPEAT
    I := I + 1;
    Reg.AH := $00;
    INTR ( $1A, Reg );
  UNTIL Reg.DX >= 60;
  R := I / 885;
END;

{---------------------------}
Procedure ZERA_TEMPO;
Var
  Reg  : Record
           AL, AH  : Byte;
           BX, CX, DX, BP, SI, DI, DS, ES, FLAGS : Integer;
         End;
BEGIN
  Reg.AH := $01;
  Reg.CX := 0;     { Parte alta }
  Reg.DX := 0;     { Parte baixa }
  INTR ( $1A, Reg );
END;

{---------------------------}
Procedure PEGA_CARACTER ( Var  Car, Car2  : Char );
```

```
Var
  Reg   : Record
            AL, AH, BL, BH, CL, CH, DL,DH   : Byte;
            BP, SI, DI, DS, ES, FLAGS       : Integer;
          End;
BEGIN
  Car := Chr ( 255 );
  Reg.AH := 6;
  Reg.DL := $FF;
  INTR ( $21, Reg );
  IF (Reg.FLAGS AND 64) = 0 THEN
    BEGIN
     Car := Chr ( Reg.AL );
      IF Ord ( Car ) = 0 THEN
        BEGIN
          Reg.AH := 6;
          Reg.DL := $FF;
          INTR ( $21, Reg );
          IF ( Reg.FLAGS AND 64 ) = 0 THEN
            Car2 := Chr ( Reg.AL );
        END;
    END;
END;

{---------------------------}
Procedure ESPERA_CARACTER ( Var  Car, Car2  : Char );

BEGIN
  REPEAT
    PEGA_CARACTER ( Car, car2 );
  UNTIL Ord ( Car ) <> 255;
END;

{---------------------------}
Procedure PEGA_TEMPO;

Var
  C, D  : Integer;
  Reg   : Record
            AL, AH  : Byte;
            BX, CX, DX, BP, SI, DI, DS, ES, FLAGS :Integer;
          End;
BEGIN
  Reg.AH := $00;
  INTR ( $1A, Reg );
  C := Reg.CX;     { Parte alta }
  D := Reg.DX;     { Parte baixa }
  Reg.AH := $01;
  Reg.CX := 0;
  Reg.DX := 0;
  INTR ( $1A, Reg );
  Centesimos := D MOD 18 + Centesimos;
  D := D DIV 18;
  Segundos := D MOD 60 + Segundos;
  D := D DIV 60;
  Minutos := D MOD 60 + Minutos;
```

```
  IF Centesimos > 100 THEN
    BEGIN
      Centesimos := Centesimos - 100;
      Segundos := Succ ( segundos );
    END;
  IF Segundos > 60 THEN
    BEGIN
      Segundos := Segundos - 60;
      Minutos := Succ ( Minutos );
    END;
  IF Minutos > 60 THEN
    BEGIN
      Minutos := Minutos - 60;
      Horas := Succ ( Horas );
    END;
END;

{---------------------------}
Procedure INICIALIZA_CARACTERES;

Const
  Tb4x8  : Array [ 1..3, 0..3 ] of Byte =
             ( ( $3C, $66, $3C, 0 ),     {0}
               ( $10, $54, $10, 0 ),     {+}
               ( $00, $54, $00, 0 ) );   {-}
Var
  I, J   : Byte;
  TROM   : Array [0..127, 0..7] of Byte Absolute $F000:$FA6E;

BEGIN
  FOR I := 0 TO 127 DO  { Caracteres }
    FOR J := 0 TO 7 DO
      TTx [ J, I ] := TROM [ I, J ];
  FOR J := 0 TO 3 DO
    FOR I := 1 TO 3 DO  { Sinais 0+-ok }
      TSin4 [ J, I ] := Tb4x8 [ I, J ];
END;

{-----------------------------------}
Procedure ESCREVA ( St  : String255 );

Var
  I, J   : Byte;
  A      : Char;
  C      : Integer;
BEGIN
  FOR I := 1 TO Length ( St ) DO
    BEGIN
      A := St [ I ];
      IF A = '\' THEN
        BEGIN
          Col := 1;
          IF Lin < 25 THEN
            Lin := Succ ( Lin );
        END
      ELSE
```

```
        BEGIN
          C := Ord ( A );
          VidTx0 [ Lin, Col ] := TTx0 [ C ];
          VidTx1 [ Lin, Col ] := TTx1 [ C ];
          VidTx2 [ Lin, Col ] := TTx2 [ C ];
          VidTx3 [ Lin, Col ] := TTx3 [ C ];
          VidTx4 [ Lin, Col ] := TTx4 [ C ];
          VidTx5 [ Lin, Col ] := TTx5 [ C ];
          VidTx6 [ Lin, Col ] := TTx6 [ C ];
          VidTx7 [ Lin, Col ] := TTx7 [ C ];
          IF Col < 80 THEN
            Col := Succ ( Col )
          ELSE
            IF Lin < 25 THEN
              BEGIN
                Lin := Succ ( Lin );
                Col := 1;
              END;
        END;
    END;
END;

{-------------------------------------}
Procedure ESCREVA_SINAIS ( St : String255 );

Const
  StSin  : String [ 5 ] = '0+-k ';

Var
  I, P   : Byte;
  N      : Integer;
BEGIN
  FOR I := 1 TO Length ( St ) DO
    BEGIN
      P := Pos ( St [ I ], StSin );
      IF P > 0 THEN
        BEGIN
          VidGr0 [ Lin, Col ] := TSin0 [ P ];
          VidGr1 [ Lin, Col ] := TSin1 [ P ];
          VidGr2 [ Lin, Col ] := TSin2 [ P ];
         END;
      Col := Succ ( Col );
    END;
END;

{-------------------------------------}
Procedure COLOCA_MOLDURA ( Xe, Yc, Xd, Yb  : Integer );

BEGIN
  Draw ( Xe, Yc, Xd, Yc, CorLetra );
  Draw ( Xd, Yc, Xd, Yb, CorLetra );
  Draw ( Xd + 1, Yc, Xd + 1, Yb, CorLetra );
  Draw ( Xd, Yb, Xe, Yb, CorLetra );
  Draw ( Xe, Yb, Xe, Yc, CorLetra );
  Draw ( Xe - 1, Yb, Xe - 1, Yc, CorLetra );
END;
```

```
{-------------------------------------}
Procedure ATUALIZA_TELA;

Var
  I   : Integer;

BEGIN
  FOR I := 1 TO 50 DO
    BEGIN
      Move ( VidGr0 [ I ], TelaGr0 [ I ], 80 );
      Move ( VidGr1 [ I ], TelaGr1 [ I ], 80 );
      Move ( VidGr2 [ I ], TelaGr2 [ I ], 80 );
      Move ( VidGr3 [ I ], TelaGr3 [ I ], 80 );
    END;
END;

{
     ****************************************************
******* R O T I N A S   D E   A R Q U I V A M E N T O *****
     ****************************************************
}
Procedure TRAZ_NUMATUAL;

Var
  NomeArq,
  St        : String80;
  Arq       : Text;
  Code      : Integer;

BEGIN
  NomeArq := DrvWr + 'NumAtual.';
  ErroIO := ( IOresult > 0 );
  Assign ( Arq, NomeArq );
  {$I-} Reset ( Arq ) {$I+};
  ErroIO := ( IOresult > 0 );
  IF NOT ( ErroIO ) THEN
    BEGIN
      {$I-} ReadLn ( Arq, St ) {$I+};
      ErroIO := ( IOresult > 0 );
      IF NOT ( ErroIO ) THEN
        Val ( St, NumAtual, Code );
    END;
  IF ErroIO OR ( Code <> 0 ) THEN
    ERRO ( 'Erro ao ler "Numatual"' );
END;

{-------------------------------------}
Procedure SALVA_NUMATUAL;

Var
  NomeArq,
  St        : String80;
  Arq       : Text;
  Code      : Integer;
```

```
BEGIN
  NumAtual := Succ ( NumAtual );
  Str ( NumAtual, St );
  NomeArq := DrvWr + 'NumAtual.';
  Assign ( Arq, NomeArq );
  {$I-} ReWrite ( Arq ) {$I+};
  ErroIO := ( IOresult > 0 );
  IF NOT ( ErroIO ) THEN
    BEGIN
      {$I-} WRITELN ( Arq, St );
      Close ( Arq ) {$I+};
      ErroIO := ( IOresult > 0 );
    END;
  IF ErroIO THEN
    ERRO ( 'Erro ao atualizar arquivo "Numatual"' );
END;

{-------------------------------------}
Procedure FINALIZA;

BEGIN
  IF NOT ( Depuracao ) THEN
    BEGIN
      {$I-} Close ( Arq ) {$I+};
      ErroIO := ( IOresult > 0 );
      IF ErroIO THEN
        ERRO ('Erro ao fechar o arquivo "' + NomeArq +'"');
      SALVA_NUMATUAL;
    END;
  TECLE_ESC;
END;

{-------------------------------------}
Procedure INICIALIZA_ARQUIVO;

Var
  St1   : String16;
BEGIN
  Str ( NumAtual, St1 );
  NomeArq := DrvWr + 'Dados' + St1 + '.SIM';
  ErroIO := ( IOresult > 0 );
  Assign ( Arq, NomeArq );
  {$I-} ReWrite ( Arq ) {$I+};
  ErroIO := ( IOresult > 0 );
  IF ErroIO THEN
    ERRO ( 'Erro ao criar arquivo "' + NomeArq + '"' );
END;

{-------------------------------------}
Procedure GRAVA_SIMULACAO;
Var
  Code  : Integer;
  A, B  : Real;
  St1,  St2,  St3,
  St4,  St5,  St6,
  St7,  St8,  St9,
```

```
  St10, St11, St12,
  St13, St14, St15,
  St16, St17, St18,
  St19                : String16;

{--------------------------}
Procedure ESC_DSK ( St : String80 );

BEGIN
  {$I-} WRITELN ( Arq, St ) {$I+};
  ErroIO := ( IOresult > 0 );
  IF ErroIO THEN
    ERRO ('Erro ao salvar dados no arquivo "'+NomeArq+'"');
END;

{--------------------------}
BEGIN
  GotoXY ( 1, 23 );
  WRITELN ( '- Salvando dados em "', NomeArq, '"' );
  Str ( OrdemDaMatriz:2, St1 );
  Str ( ( nProc + 1 ):2, St2 );
  Str ( ( nMem + 1  ):2, St3 );
  Str ( ( nOper + 1 ):2, St4 );
  Str ( tAcc:2, St5 );
  Str ( tOper:2, St6 );
  Str ( Clock:4:1, St7 );
  Str ( ModEspera:2, St18 );
  Str ( ContBar:7:0, St8 );
  Str ( ( ContRD + ContWR ):7:0, St9 );
  B := ContBar;
  A := B / ( Horas * 3600 + Minutos * 60 + Segundos + 1 );
  Str ( A:7:1, St10 );
  A := ( 1 - BarEndOciosa / B ) * 100;
  Str ( A:5:1, St11 );
  A := ( 1 - UsoMax ) * 100;
  Str ( A:5:1, St19 );
  A := ( 1 - BarOperOciosa / B ) * 100;
  Str ( A:5:1, St12 );
  Str ( FlopMultTotal:7:0, St13 );
  Str ( FlopSomaTotal:7:0, St14 );
  Str ( Horas:2, St15 );
  Str ( Minutos:2, St16 );
  Str ( Segundos:2, St17 );
  ESC_DSK ( 'Matriz: ' +St1+'x'+St1+' ContBar = ' +St8  );
  ESC_DSK ( 'nProc = ' +St2+      '    Acessos = ' +St9 );
  ESC_DSK ( 'nMem  = ' +St3+      '    Cicl/S  = ' + St10 );
  ESC_DSK ( 'nOper = ' +St4+    '    BarM  = '+St11+'%  ('+
                                          St19+ '%)');
  ESC_DSK ( 'tMem  = ' +St5+      '    BarO  = '+St12+'% ');
  ESC_DSK ( 'tOper = '+St6+'    Flops='+St13 + ' '+St14 );
  ESC_DSK ('Espera= '+St18+'    Tempo='+St15+'h '+St16+'m '
                                          +St17+'s');
  ESC_DSK ( St7 );
  ESC_DSK ( '--' );
END;
```

```
{--------------------------}
Procedure TRAZ_PARAMETROS;

Var
  NomeArq   : String80;
  Arq       : Text;
  St        : Array [ 1..2 ] of String80;
  ErroIO    : Boolean;
  N         : Byte;

{--------------------------}
Procedure BUSCA ( Stl : Stringl6; Var V : Integer );

Var
  P, I, L, Code   : Integer;

BEGIN
  P := Pos ( Stl, St [ N ] );
  IF P > 0 THEN
    BEGIN
      L := Length ( Stl );
      Val ( Copy ( St [ N ], P + L, 2 ), V, Code );
      IF Code > 0 THEN
        ERRO ( 'Erro ao ler parametro "'+ Stl + '"' );
    END
  ELSE
    ERRO ( 'Erro ao ler parametro ( "' + Stl +
           '" nao encontrado )' );
END;

{--------------------------}
BEGIN
  WRITELN ( '- Trazendo ', DrvRd, 'Par.' );
  NomeArq := DrvWr + 'Par.';
  ErroIO := ( IOresult > 0 );
  Assign ( Arq, NomeArq );
  {$I-} Reset ( Arq ) {$I+};
  ErroIO := ( IOresult > 0 );
  IF NOT ( ErroIO ) THEN
    BEGIN
      {$I-} ReadLn ( Arq, St [ 1 ] ) {$I+};
      ErroIO := ( IOresult > 0 );
      IF NOT ( ErroIO ) THEN
        BEGIN
          {$I-} ReadLn ( Arq, St [ 2 ] ) {$I+};
          ErroIO := ( IOresult > 0 );
        END;
    END;
  IF ErroIO THEN
    ERRO ( ' - Erro ao ler arquivo "' + NomeArq + '"' )
  ELSE
    BEGIN
      N := 1;
      BUSCA (  'N=', Ni );
      BUSCA ( 'nP=', nPi );
      BUSCA ( 'nM=', nMi );
```

```
      BUSCA ( 'nO=', nOi );
      BUSCA ( 'tM=', tAi );
      BUSCA ( 'tO=', tOi );
      BUSCA ( 'mE=', mEi );
      N := 2;
      BUSCA (  'N=', Nf );
      BUSCA ( 'nP=', nPf );
      BUSCA ( 'nM=', nMf );
      BUSCA ( 'nO=', nOf );
      BUSCA ( 'tM=', tAf );
      BUSCA ( 'tO=', tOf );
      BUSCA ( 'mE=', mEf );
      {$I-} Close ( Arq ) {$I+};
      ErroIO := ( IOresult > 0 );
      IF ErroIO THEN
        ERRO ( 'Erro ao atualizar arquivo "Numatual"' );
    END;
END;

{
  ************************************************************
**** P R O C E D I M E N T O S    P R I N C I P A I S *****
  ************************************************************
}
Procedure DISPLAY;

Var
  cAux,
  CE,
  CB,
  A, B   : Real;
  I, Y   : Byte;
  Stt    : String [ 255 ];

  St1,  St2,  St3,
  St4,  St5,  St6,
  St7,  St8,  St9,
  St10, St11, St12,
  St13, St14, St15,
  St16, St17, St18,
  St19, St20, St21,
  St22                : String16;

{--------------------------}
Procedure CONTADORES;

Var
  L   : Byte;

BEGIN
  PEGA_TEMPO;
  Str ( ( nProc + 1 ):2, St11 );
  Str ( ( nMem + 1  ):2, St12 );
  Str ( ( nOper + 1 ):2, St13 );
  Str ( tAcc:2, St14 );
  Str ( tOper:2, St15 );
```

```
  Str ( OrdemDaMatriz:2, St19 );
  Str ( Clock:4:1, St20 );
  Str ( ModEspera:2, St21 );
  B := ContBar;
  Str ( ContBar:7:0, St1 );
  A := ( 1 - BarEndOciosa / B ) * 100;
  Str ( A:5:1, St2 );
  A := ( 1 - UsoMax ) * 100;
  Str ( A:5:1, St22 );
  A := B / ( Horas * 3600 + Minutos * 60 + Segundos + 1 );
  Str ( A:7:1, St3 );
  A := ( 1 - BarOperOciosa / B ) * 100;
  Str ( A:5:1, St4 );
  Str ( FlopMultTotal:5:0, St6);
  Str ( FlopSomaTotal:5:0, St7 );
  Str ( ModDisplay:5, St8 );
  Str ( ( ContRD + ContWR ):7:0, St9 );
  Str ( Horas:2, St16 );
  Str ( Minutos:2, St17 );
  Str ( Segundos:2, St18 );
  Stt :=  ' nP=' + St11 + ' nM=' + St12 + ' nO=' + St13 +
          '\ tM=' + St14 + ' tO=' + St15 + ' N=' + St19 +
          '\ Clk=' + St20 + '  mE=' + St21 +
          '\ Tempo=' + St16 + 'h:' + St17 + 'm:' +St18+'s'+
          '\ tBar:' + St1 +
          '\ tB/S:' + St3 +
          '\ nAcc:' + St9 +
          '\ bMem:' + St2 + '%' + St22 +
        '%\ bOpe:' + St4 +
       ' %\ Flps:' + St6 + 'M' + St7 + 'S' +
          '\ Display:' +
          '\ 1    \ 2    \ 5    \ 10   \ 20   ' +
          '\ 50   \ 100  \ 200  \ 500  \ 1000 ';

  Lin := 1; Col := 1;  ESCREVA ( Stt );
  Lin := Lin - 10 + VelDisplay;
  Col := 6;  ESCREVA ( '<' );
  Col := 1;  ESCREVA ( '>' );
END;

{--------------------------}
Procedure MATRIZ;

Var
  N, CA,
  X, Y    : Byte;
  Pos,
  V       : Integer;
  C       : Char;

BEGIN
  N := OrdemDaMatriz;
  Lin := 2;
  CA := 21;
  Pos := InicioMP;
```

```
  FOR Y := 1 TO N DO
    BEGIN
      Stt := '';
      FOR X := 1 TO N DO
        BEGIN
          V := DadosM [ Pos ];
          IF V > 0 THEN C := '+'
          ELSE
            IF V < 0 THEN C := '-'
            ELSE C := '0';
          Stt := Stt + C;
          Pos := Succ ( Pos );
        END;
      Col := CA;
      ESCREVA_SINAIS ( Stt );
      Lin := Succ ( Lin );
    END;
END;

{---------------------------}
Procedure ATUALIZA_CONTADORES;

BEGIN
  ContBar := ContBar + ContBarInt;
  BarDadoOciosa := BarDadoOciosa + BarDadoOciosaInt;
  BarEndOciosa := BarEndOciosa + BarEndOciosaInt;
  BarOperOciosa := BarOperOciosa + BarOperOciosaInt;
  BarResOciosa := BarResOciosa + BarResOciosaInt;
  FlopMultTotal := FlopMultTotal + FlopMultTotalInt;
  FlopSomaTotal := FlopSomaTotal + FlopSomaTotalInt;
  ContAcc := ContAcc + ContAccInt;
  ContRD := ContRD + ContRDInt;
  ContWR := ContWR + ContWRInt;
  ContOpCod := ContOpCod + ContOpCodInt;

  CE := BarEndOciosaInt;
  CB := ContBarInt;
  cAux := CE / CB;
  IF UsoMax > cAux THEN
    UsoMax := cAux;

  ContBarInt := 0;
  BarDadoOciosaInt := 0;
  BarEndOciosaInt := 0;
  BarOperOciosaInt := 0;
  BarResOciosaInt := 0;
  FlopMultTotalInt := 0;
  FlopSomaTotalInt := 0;
  ContAccInt := 0;
  ContRDInt := 0;
  ContWRInt := 0;
  ContOpCodInt := 0;
END;
```

```
{--------------------------}
BEGIN
  ContDisPlay := Succ ( ContDisplay );
  IF ContDisplay > ModDisplay THEN
    BEGIN
      ATUALIZA_CONTADORES;
      ContDisplay := 1;
      CONTADORES;
      MATRIZ;
      ATUALIZA_TELA;
      COLOCA_MOLDURA ( 158, 3, OrdemDaMatriz * 8 + 162,
                       OrdemDaMatriz * 4 + 3 );
    END;
END;

{**********************************}
Procedure PASSO_A_PASSO;

{--------------------------}
Procedure VEL_DISPLAY ( Inc  : Integer );

BEGIN
  IF ( ( VelDisplay + Inc ) < 11 ) AND
     ( ( VelDisplay + Inc ) > 0 ) THEN
    BEGIN
      VelDisplay := VelDisplay + Inc;
      ModDisplay := VD [ VelDisplay ];
      ContDisplay := 1001;
    END;
END;

{--------------------------}
BEGIN
  Car := Chr ( 255 );
  ContPegaCar := Pred ( ContPegaCar );
  IF PassoAPasso THEN
    ESPERA_CARACTER ( Car, Car2 )
  ELSE
    IF ContPegaCar <= 0 THEN
      BEGIN
        ContPegaCar := ModPegaCar;
        PEGA_CARACTER ( Car, Car2 );
      END;
  IF Ord ( Car ) <> 255 THEN
    BEGIN
      IF Ord ( Car ) <> 27 THEN
        QuaseFim := False;
      CASE Ord ( Car ) OF
        70 : QuaseFim := True;
        27 : IF QuaseFim THEN
               BEGIN
                 Fim := True;
                 FimForcado := True;
               END;
```

```
          0 : BEGIN
                CASE Ord ( Car2 ) OF
                  59 : QuaseFim := True;
                  80 : VEL_DISPLAY (  1 );
                  72 : VEL_DISPLAY ( -1 );
                END;
              END
        ELSE
          BEGIN
            Car := UpCase ( Car );
            CASE Car OF
              'P' : PassoAPasso := NOT ( PassoAPasso );
              '+' : VEL_DISPLAY (  1 );
              '-' : VEL_DISPLAY ( -1 );
            END;
          END;
        END;
      END;
END;

{************************************}
Procedure BARRAS_RECEBEM_MODULOS;

Var
  I   : Byte;

{--------------------------}
Procedure TRATA_ARBITRO;

Var
  HEnd, HDMem, HDProc, HOper, HRes  : Byte;

BEGIN
  HEnd := 100;
  HDMem := 100;
  HDProc := 100;
  HOper := 100;
  HRes := 100;
  I := 0;
  REPEAT
    IF ( BarMem.PedeEnd [ I ] ) AND
       ( BarMem.PedeDadoProc [ I ] ) THEN
      BEGIN
        HEnd := I;
        HDProc := I;
      END;
    I := Succ ( I );
  UNTIL ( I > nProc ) OR ( HEnd < 100 );
  IF HDProc = 100 THEN
    BEGIN
      I := 0;
      REPEAT
        IF BarMem.PedeDadoMem [ I ] THEN
          HDMem := I;
        I := Succ ( I );
      UNTIL ( I > nMem ) OR ( HDMem < 100 );
    END;
```

```
  IF HEnd = 100 THEN
    BEGIN
      I := 0;
      REPEAT
        IF ( BarMem.PedeEnd [ I ] ) AND
           NOT ( BarMem.PedeDadoProc [ I ] ) THEN
          HEnd := I;
        I := Succ ( I );
      UNTIL ( I > nProc ) OR ( HEnd < 100 );
    END;
  IF BarOper.OperDisponivel > 0 THEN
    BEGIN
      I := 0;
      REPEAT
        IF BarOper.PedeOper [ I ] THEN
          HOper := I;
        I := Succ ( I );
      UNTIL ( I > nProc ) OR ( HOper < 100 );
    END;
  I := 0;
  REPEAT
    IF BarOper.PedeRes [ I ] THEN
      HRes := I;
    I := Succ ( I );
  UNTIL ( I > nOper ) OR ( HRes < 100 );
  IF HEnd = 100 THEN
    BarEndOciosaInt := BarEndOciosaInt + 1;
  IF ( HDMem = 100 ) AND ( HDProc = 100 ) THEN
    BarDadoOciosaInt := BarDadoOciosaInt + 1;
  IF HOper = 100 THEN
    BarOperOciosaInt := BarOperOciosaInt + 1;
  IF HRes = 100 THEN
    BarResOciosaInt := BarResOciosaInt + 1;
  BarMem.HabEnd := HEnd;
  BarMem.HabDadoMem := HDMem;
  BarMem.HabDadoProc := HDProc;
  BarOper.HabOper := HOper;
  BarOper.HabRes := HRes;
END;

{--------------------------}
Procedure CONTROLE;

BEGIN
  BarMem.ProcEnd := 100;
  BarMem.ProcDado := 100;

  IF BarMem.HabEnd < 100 THEN
    WITH StatusProcN [ BarMem.HabEnd ] DO
      BEGIN
        BarMem.Endereco := RegE;
        BarMem.ApagaDado := ApagaDado;
        BarMem.LeituraWR := LeituraWR;
        BarMem.ProcEnd := BarMem.HabEnd;
        PedeEnd := False;
      END;
```

```
  IF BarMem.HabDadoProc < 100 THEN
    WITH StatusProcN [ BarMem.HabDadoProc ] DO
      BEGIN
        BarMem.Dado := RegW;
        nWR := nWR + 1;
        BarMem.ProcDado := BarMem.HabDadoProc;
        PedeDadoWR := False;
      END;

  IF BarOper.HabOper < 100 THEN
    WITH StatusProcN [ BarOper.HabOper ] DO
      BEGIN
        BarOper.Oper1 := RegX;
        BarOper.Oper2 := RegY;
        BarOper.Operacao := Operacao;
        BarOper.ProcOper := BarOper.HabOper;
        PedeOperacao := False;
      END;

  FOR I := 0 TO nProc DO
    WITH StatusProcN [ I ] DO
      BEGIN
        RegL := BarMem.Dado;
        BarMem.PedeEnd [ I ] := PedeEnd;
        BarMem.PedeDadoProc [ I ] := PedeDadoWR;
        BarOper.PedeOper [ I ] := PedeOperacao;
      END;
END;

{--------------------------}
Procedure MEMORIA;

Var
  St1    : String [ 10 ];
  J, C   : Integer;

BEGIN
  IF BarMem.HabDadoMem < 100 THEN
    WITH StatusMem [ BarMem.HabDadoMem ] DO
      BEGIN
        BarMem.Dado := FilaOut [ PontOutRD ].Dado;
        BarMem.ProcDado := FilaOut [ PontOutRD ].Proc;
        PontOutRD := ( PontOutRD + 1 ) AND LFilaOut;
        IF TamFilaOut > 0 THEN
          TamFilaOut := Pred ( TamFilaOut );
        IF TamFilaOut = 0 THEN
          PedeDado := False;
      END;
  FOR I := 0 TO nMem DO
    BarMem.PedeDadoMem [ I ] := StatusMem [ I ].PedeDado;
END;
```

```
{--------------------------}
Procedure OPERADORES;

BEGIN
  BarOper.ProcRes := 100;
  IF BarOper.HabRes < 100 THEN
    WITH StatusOper [ BarOper.HabRes ] DO
      BEGIN
        BarOper.Resultado := Resultado;
        BarOper.Zero := Zero;
        BarOper.ProcRes := Proc;
        PedeRes := False;
      END;
  FOR I := 0 TO nOper DO
    WITH StatusOper [ I ] DO
      BarOper.PedeRes [ I ] := PedeRes;
END;

{--------------------------}
BEGIN
  TRATA_ARBITRO;
  OPERADORES;
  CONTROLE;
  MEMORIA;
END;

{**********************************}
Procedure MODULOS_OPERAM;

{--------------------------}
Procedure CONTROLE;

Const
  StRegs    : String [ 25 ] = '0123456789ABCDEFielwrxyzn';
  StOper    : String [ 5 ] = '+-*/r';
  StEspera  : String [ 36 ] =
                  '1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ';

Var
  I        : Byte;
  N,
  Destino,
  Fonte    : Integer;
  Teste    : Boolean;

{----------------}
Procedure PEGA_NOVA_POSICAO;

BEGIN
  WITH StatusProc [ I ] DO
    BEGIN
      IF Posicao > Length ( StExec ) THEN
        BEGIN                              {'eFSlSeSrWeWril+FG_'}
          Reg [ 17 ] := Reg [ 15 ];
          LeituraWR := True;
          PedeEnd := True;
```

```
            PedeDadoRD := True;
            Reg [ 15 ] := Reg [ 15 ] + 1;
            StExec := 'WeWrilG_';
            Posicao := 1;
          END;
        Car1 := StExec [ Posicao ];
        Car2 := StExec [ Posicao + 1 ];
        Posicao := Posicao + 2;
      END;
END;

{----------------}
BEGIN
  FOR I := 0 TO nProc DO
    WITH StatusProc [ I ] DO
      BEGIN
        IF Posicao = 100 THEN
          PEGA_NOVA_POSICAO;
        IF BarMem.ProcDado = I THEN
          BEGIN
            StatusProcN [ I ].RegL := BarMem.Dado;
            PedeDadoRD := False;
          END;
        IF BarOper.ProcRes = I THEN
          BEGIN
            StatusProcN [ I ].RegZ := BarOper.Resultado;
            Zero := BarOper.Zero;
            PedeResultado := False;
          END;
        REPEAT
          CASE Car1 OF
           'W' : BEGIN
                   CASE Car2 of
                    'e' : Teste := PedeEnd;
                    'w' : Teste := PedeDadoWR;
                    'r' : Teste := PedeDadoRD;
                    'o' : Teste := PedeOperacao;
                    'R' : Teste := PedeResultado;
                    'a' : Teste := ApagaDado;
                    'n' : BEGIN
                            ContEspera := Pred(ContEspera);
                            Teste := ( ContEspera > 0 );
                          END
                   ELSE
                     ERRO ( 'FlipFlop a testar inex.' );
                   END;
                   IF NOT ( Teste ) THEN
                     PEGA_NOVA_POSICAO;
                 END;
            'I' : BEGIN
                    CASE Car2 of
                      'L' : BEGIN
                              PedeEnd := True;
                              PedeDadoRD := True;
                              LeituraWR := True;
                            END;
```

```
            'E' : BEGIN
                    PedeEnd := True;
                    PedeDadoWR := True;
                    LeituraWR := False;
                  END;
            'O' : BEGIN
                    PedeOperacao := True;
                    PedeResultado := True;
                  END;
          END;
          PEGA_NOVA_POSICAO;
        END;
'N' : BEGIN
          ContEspera := Pos ( Car2, StEspera ) *
                        ModEspera * 10;
          PEGA_NOVA_POSICAO;
        END;
't' : BEGIN
          ContEspera := ContEspera - 1;
          IF ContEspera <= 0 THEN
            PEGA_NOVA_POSICAO;
        END;
'S' : BEGIN
          CASE Car2 of
           'e' : PedeEnd := True;
           'w' : PedeDadoWR := True;
           'r' : PedeDadoRD := True;
           'o' : PedeOperacao := True;
           'l' : LeituraWR := True;
           'a' : ApagaDado := True;
           'R' : PedeResultado := True;
          ELSE
            ERRO ('FlipFlop a setar inexistente');
          END;
          PEGA_NOVA_POSICAO;
        END;
'R' : BEGIN
          CASE Car2 of
           'e' : PedeEnd := False;
           'w' : PedeDadoWR := False;
           'r' : PedeDadoRD := False;
           'o' : PedeOperacao := False;
           'l' : LeituraWR := False;
           'a' : ApagaDado := False;
           'R' : PedeResultado := False;
          ELSE
           ERRO('FlipFlop a resetar inexistente');
          END;
          PEGA_NOVA_POSICAO;
        END;
'+' : BEGIN
          N := Pos ( Car2, StRegs ) - 1;
          IF N < 0 THEN
            ERRO ( 'Registrador a incr. inex.' )
          ELSE
```

```
            Reg [ N ] := Reg [ N ] + 1;
          PEGA_NOVA_POSICAO;
        END;
'-' : BEGIN
          N := Pos ( Car2, StRegs ) - 1;
          IF N < 0 THEN
            ERRO ( 'Registrador a decr. inex.' )
          ELSE
            Reg [ N ] := Reg [ N ] - 1;
          PEGA_NOVA_POSICAO;
        END;
'O' : BEGIN
          Operacao := Pos ( Car2, StOper ) - 1;
          IF Operacao < 0 THEN
            ERRO ( 'Operacao inex.' );
          PEGA_NOVA_POSICAO;
        END;
'T' : BEGIN
          IF ((Car2 = 'd') AND Zero ) OR
             ((Car2 = 'z') AND NOT ( Zero ) ) OR
             ((Car2 = 'm') AND NOT ( Carry ) ) OR
             ((Car2 = 'M') AND(Carry OR Zero))THEN
             Posicao := 100;
          PEGA_NOVA_POSICAO;
        END;
'c' : BEGIN
          Zero := ( StatusProcN [ I ].RegX =
                    StatusProcN [ I ].RegY );
          Carry := ( StatusProcN [ I ].RegX <
                    StatusProcN [ I ].RegY );
          PEGA_NOVA_POSICAO;
        END;
'K' : BEGIN
          StatusProcN [ I ].RegY := Ord(Car2) -48;
          PEGA_NOVA_POSICAO;
        END;

'o' : BEGIN
          CASE Car2 of
           '+' : StatusProcN [ I ].RegZ :=
                 StatusProcN [ I ].RegX +
                 StatusProcN [ I ].RegY;
           '-' : StatusProcN [ I ].RegZ :=
                 StatusProcN [ I ].RegX -
                 StatusProcN [ I ].RegY;
          END;
          PEGA_NOVA_POSICAO;
        END;

' ' : PEGA_NOVA_POSICAO;

'd' : BEGIN
          nProcAtivos := Pred ( nProcAtivos );
          PEGA_NOVA_POSICAO;
        END;
```

```
            'G' : BEGIN
                    N := StatusProcN [ I ].RegI;
                    IF N <= nInstr THEN
                      BEGIN
                        StExec := StProgr [ N ];
                        IF StExec = '' THEN
                          ERRO ( 'Instrucao vazia' );;
                        Car1 := StExec [ 1 ];
                        Car2 := StExec [ 2 ];
                        Posicao := 3;
                      END
                    ELSE
                      ERRO ( 'RI > nInstr' );;
                  END
          ELSE
            BEGIN
              Destino := Pos ( Car1, StRegs ) - 1;
              Fonte := Pos ( Car2, StRegs ) - 1;
              IF ( Destino < 0 ) OR ( Fonte < 0 ) THEN
                ERRO ( 'Reg ' + Car1 + Car2 +
                       ' a mover inex.' + StExec )
              ELSE
                Reg [ Destino ] := Reg [ Fonte ];
              IF Car1 = 'i' THEN
                nOpCod := nOpCod + 1;
              PEGA_NOVA_POSICAO;
            END;
          END;
        UNTIL ( Car1 = 'W' ) OR ( Car1 = 't' );
      END;
END;

{--------------------------}
Procedure OPERADORES;

Var
  I   : Byte;

BEGIN
  FOR I := 0 TO nOper DO
    WITH StatusOper [ I ] DO
      BEGIN
        CASE Estado OF
          0 : BEGIN
                IF BarOper.HabOper < 100 THEN
                  BEGIN
                    BarOper.HabOper := 100;
                    BarOper.OperDisponivel :=
                                  BarOper.OperDisponivel -1;
                    Oper1 := BarOper.Oper1;
                    Oper2 := BarOper.Oper2;
                    Proc :=  BarOper.ProcOper;
                    Operacao := BarOper.Operacao;
                    CASE Operacao OF
                      0 : Resultado := Oper1 + Oper2;
                      1 : Resultado := Oper1 - Oper2;
```

```
                    2 : Resultado := Oper1 * Oper2;
                    3 : BEGIN
                          IF Oper2 <> 0 THEN
                            Resultado := Oper1 DIV Oper2
                          ELSE
                            ERRO ( 'Divisao por zero' );
                        END;
                    4 : BEGIN
                         IF Oper1 >= 0 THEN
                          Resultado := Trunc(Sqrt(Oper1))
                         ELSE
                          Erro('Raiz de negativo');
                        END;
                  END;
                  IF Operacao > 1 THEN
                    BEGIN
                     FlopMultToTalInt:=FlopMultTotalInt+1
                     ContTempo := tOperMult;
                    END
                  ELSE
                   BEGIN
                    FlopSomaTotalInt:=FlopSomaTotalInt+1;
                    ContTempo := tOperSoma
                   END;
                  Zero := ( Resultado = 0 );
                  Estado := 1;
                END;
            END;
        1 : BEGIN
              ContTempo := Pred ( ContTempo );
              IF ContTempo < 1 THEN
                BEGIN
                  PedeRes := True;
                  Estado := 2;
                END;
            END;
        2 : IF BarOper.HabRes = I THEN
              BEGIN
                Estado := 0;
                BarOper.OperDisponivel :=
                              BarOper.OperDisponivel+1;
              END;
      END;
    END;
END;

{--------------------------}
Procedure MEMORIA;

Var
  I, N, E   : Integer;

BEGIN
  IF BarMem.ProcEnd < 100 THEN
    BEGIN
      N := BarMem.Endereco MOD ( nMem + 1 );
```

```
    WITH StatusMem [ N ] DO
      BEGIN
        FilaIn [ PontInWR ].Ender := BarMem.Endereco;
        FilaIn [ PontInWR ].ApagaDado :=BarMem.ApagaDado;
        FilaIn [ PontInWR ].Dado := BarMem.dado;
        FilaIn [ PontInWR ].LeituraWR :=BarMem.LeituraWR;
        FilaIn [ PontInWR ].Proc := BarMem.ProcEnd;
        PontInWR := Succ ( PontInWR ) AND LFilaIn;
        TamFilaIn := Succ ( TamFilaIn );
      END;
  END;
FOR I := 0 TO nMem DO
  WITH StatusMem [ I ] DO
    BEGIN
      CASE Estado OF
        0 : BEGIN
              IF TamFilaIn > 0 THEN
                BEGIN
                  ContTempo := 0;
                  Estado := 1;
                END;
            END;
        1 : BEGIN
              ContTempo := Succ ( ContTempo );
              IF ContTempo >= tAcc THEN
                Estado := 2;
            END;
        2 : BEGIN
              E := FilaIn [ PontInRD ].Ender;
              IF ( E <= LimMem ) AND ( E >= 0 ) THEN
                BEGIN
                  IF FilaIn [ PontInRD ].LeituraWR THEN
                    BEGIN
                      FilaOut[PontOutWR].Dado:=DadosM[E];
                      FilaOut[PontOutWR].Proc:=
                                  FilaIn[ PontInRD ].Proc;
                      PontOutWR :=  Succ ( PontOutWR )AND
                                              LFilaOut;
                      TamFilaOut := Succ ( TamFilaOut );
                      ContRDInt := ContRDInt + 1;
                      PedeDado := True;
                      IF FilaIn [PontInRD].ApagaDado THEN
                        DadosM [ E ] := 0;
                    END
                  ELSE
                    BEGIN
                      DadosM [E] :=FilaIn[PontInRD].Dado;
                      ContWRInt := ContWRInt + 1;
                    END;
                  PontInRD := Succ(PontInRD) AND LFilaIn;
                  TamFilaIn := Pred ( TamFilaIn );
                  Estado := 0;
                END
              ELSE
                ERRO ( 'Posicao de memoria inex.' );
            END;
```

```
        END;
      END;
END;
{--------------------------}
BEGIN
  CONTROLE;
  OPERADORES;
  MEMORIA;
END;

{**********************************}
Procedure INICIALIZA;

{--------------------------}
Procedure INICIALIZA_VARIAVEIS;

Var
  J   : Byte;
  I   : Integer;

BEGIN
  PassoAPasso := False;
  QuaseFim := False;
  Fim := False;
  FimForcado := False;
  ContPegaCar := 0;
  VelDisplay := MD;
  ModDisplay := VD [ VelDisplay ];
  Horas := 0;
  Minutos := 0;
  Segundos := 0;
  Centesimos := 0;
  UsoMax := 1;
  ContBar := 0;
  BarDadoOciosa := 0;
  BarEndOciosa := 0;
  BarOperOciosa := 0;
  BarResOciosa := 0;
  FlopMultTotal := 0;
  FlopSomaTotal := 0;
  ContAcc := 0;
  ContRD := 0;
  ContWR := 0;
  ContOpCod := 0;

  ContBarInt := 1;
  BarDadoOciosaInt := 0;
  BarEndOciosaInt := 0;
  BarOperOciosaInt := 0;
  BarResOciosaInt := 0;
  FlopMultTotalInt := 0;
  FlopSomaTotalInt := 0;
  ContAccInt := 0;
  ContRDInt := 0;
```

```
  ContWRInt := 0;
  ContOpCodInt := 0;
  nProcAtivos := nProc + 1;
  ContDisplay := ModDisplay + 1;
  tOperSoma := tOper DIV 2;
  tOperMult := tOper * 2;
  BarMem.HabEnd := 100;
  BarMem.HabDadoMem := 100;
  BarMem.HabDadoProc := 100;
  BarOper.HabOper := 100;
  BarOper.HabRes := 100;
  BarOper.ProcRes := 100;
  BarOper.Zero := False;
  BarOper.OperDisponivel := nOper + 1;

  FOR I := 0 TO LimMem DO
    DadosM [ I ] := 0;
  FOR I := 0 TO nInstr DO
    DadosM [ I ] := I;
  DadosM [ 1 ] := InicioPar;

{---------- Processadores ----------}
  FOR I := 0 TO nProc DO
    WITH StatusProc2 [ I ] DO
      BEGIN
        FOR J := 1 TO 9 DO
          B [ J ] := False;
        ContEspera := 0;
        Posicao := 100;
        StExec := '';
        FOR J := 0 TO 15 DO
          Reg [ J ] := 0;
        RegN := I;
        BarMem.PedeEnd [ I ] := False;
        BarMem.PedeDadoProc [ I ] := False;
        nOpCod := 0;
        nRD := 0;
        nWR := 0;
        FlopMult := 0;
        FlopSoma := 0;
      END;

{------------- Memorias -------------}
  FOR I := 0 TO nMem DO
    WITH StatusMem [ I ] DO
      BEGIN
        PontInRD := 0;
        PontInWR := 0;
        PontOutRD := 0;
        PontOutWR := 0;
        TamFilaIn := 0;
        TamFilaOut := 0;
        Estado := 0;
        ContTempo := 0;
        Leituras := 0;
```

```
          Escritas := 0;
          ContFilaIn := 0;
          ContFilaOut := 0;
          PedeDado := False;
          BarMem.PedeDadoMem [ I ] := False;
        END;

{------------ Operadores  ------------}
  FOR I := 0 TO nOper DO
    WITH StatusOper [ I ] DO
      BEGIN
        Estado := 0;
        PedeRes := False;
        BarOper.PedeOper [ I ] := False;
        BarOper.PedeRes [ I ] := False;
        FlopMult := 0;
        FlopSoma := 0;
        Operacao := 0;
        Oper1 := 0;
        Oper2 := 0;
        Resultado := 0;
      END;
END;

{---------------------------}
Procedure INICIALIZA_MATRIZ;

Var
  N, X, Y    : Byte;
  PM1, PM2   : Integer;

BEGIN
  N := OrdemDaMatriz;
  PM1 := InicioM1;
  PM2 := InicioM2;
  FOR Y := 1 TO N DO
    BEGIN
      FOR X := 1 TO N DO
        BEGIN
          DadosM [ PM1 ] := X;
          PM1 := Succ ( PM1 );
          IF X = Y THEN
            DadosM [ PM2 ] := 1;
          PM2 := Succ ( PM2 );
        END;
    END;
  DadosM [ InicioPar ] := OrdemDaMatriz;          {N}
  DadosM [ InicioPar + 1 ] := InicioMP + N*N-1;   {FimProd}
  DadosM [ InicioPar + 2 ] := InicioM2 + N;       {FimCol}
  DadosM [ InicioPar + 3 ] := InicioMP;
  DadosM [ InicioPar + 4 ] := InicioM1;
  DadosM [ InicioPar + 5 ] := InicioM2;
  DadosM [ InicioPar + 6 ] := 0;
END;
```

```
{--------------------------}
BEGIN
  INICIALIZA_CARACTERES;
  INICIALIZA_VARIAVEIS;
  INICIALIZA_MATRIZ;
END;

{**********************************}
Procedure EXECUTA;

BEGIN
  APAGA_VIDEO;
  INICIALIZA;

  GotoXY ( 1, 23 );
  APAGA_VIDEO;
  ZERA_TEMPO;
  PEGA_TEMPO;
  BARRAS_RECEBEM_MODULOS;

  REPEAT
    IF nProcAtivos <= 0 THEN
      BEGIN
        Fim := True;
        ContDisplay := 1001;
      END;
    DISPLAY;
    PASSO_A_PASSO;
    MODULOS_OPERAM;
    BARRAS_RECEBEM_MODULOS;
    ContBarInt := ContBarInt + 1;
  UNTIL Fim;

  IF FimForcado THEN
    BEGIN
      FINALIZA;
      Halt;
    END;
END;

{    ************************************************
******* P R O G R A M A     P R I N C I P A L     ********
     ************************************************
}
BEGIN
  Hires;
  GraphBackGround ( 1 );
  HiresColor ( CorLetra );
  MEDE_VELOCIDADE ( Clock );
  IF Clock < 6 THEN
    BEGIN
      WRITELN ( 'Clock = ', Clock:4:1 );
      WRITELN ( 'Tecle <ESC> para interromper,
                                <ENTER> para prosseguir');
```

```
      REPEAT
        PEGA_CARACTER ( Car, Car2 );
      UNTIL ( Ord ( Car ) = 27 ) OR ( Ord ( Car ) = 13 );
      IF Ord ( car ) = 27 THEN Halt;
    END;
  Depuracao := False;
  Ni  := nMatriz;  Nf := Ni;  nPi := nProcCt; nPf := nPi;
  nMi := nMemCt;  nMf := nMi; nOi := nOperCt; nOf := nOi;
  tAi := tAccCt;  tAf := tAi; tOi := tOperCt; tOf := tOi;
  mEi := ModEsperaCt; mEf := mEi;
  IF NOT ( Depuracao ) THEN
    BEGIN
      TRAZ_PARAMETROS;
      TRAZ_NUMATUAL;
      INICIALIZA_ARQUIVO;
    END;
  FOR OrdemDaMatriz := Ni TO Nf DO
    FOR ModEspera := mEi TO mEf DO
      BEGIN
        nMem := nMi;
        REPEAT
          FOR tAcc := tAi TO tAf DO
            FOR nProc := nPi TO nPf DO
              FOR tOper := tOi TO tOf DO
                BEGIN
                  nOper := nOi;
                  IF nOi > nOperCt THEN
                    BEGIN
                      nOper := nProc;
                      nOf := 0;
                    END;
                  REPEAT
                    EXECUTA;
                    IF NOT ( Depuracao ) THEN
                      GRAVA_SIMULACAO;
                    nOper := nOper + 1;
                  UNTIL nOper > nOf;
                END;
          nMem := ( nMem + 1 ) * 2 - 1;
        UNTIL nMem > nMf;
      END;
  FINALIZA;
END.
```

APÊNDICE III

RELAÇÃO DE FIGURAS

APÊNDICE IV

SIMULAÇÕES REALIZADAS

PARÂMETROS DE ENTRADA

np - Número de processadores

nM - Número de bancos de memória

nO - Número de Unidades Operadoras

tM - Tempo de acesso à memória

tO - Tempo médio de operação

VALORES OBTIDOS

ContBarr - Número total de ciclos de barra

nAcessos - Número total de acessos às memórias

BarM - Uso da barra de memória (percentual médio)

BarMp - Uso da barra de memória (percentual de pico)

BarO - Uso da barra de Operadores (percentual médio)

1 - Variação do número de processadores

np	nM	nO	tM	tO	ContBarr	nAcessos	BarM	BarMp	BarO
1,	32,	1,	3,	20,	838001.,	47660.,	5.7,	6.5,	1.9
2,	32,	2,	3,	20,	419323.,	47695.,	11.4,	14.0,	3.8
3,	32,	3,	3,	20,	281068.,	47696.,	17.0,	19.3,	5.7
4,	32,	4,	3,	20,	210450.,	47736.,	22.7,	25.7,	7.6
5,	32,	5,	3,	20,	168694.,	47770.,	28.3,	31.9,	9.5
6,	32,	6,	3,	20,	141784.,	47837.,	33.7,	38.2,	11.3
7,	32,	7,	3,	20,	122396.,	47912.,	39.1,	44.3,	13.1
8,	32,	8,	3,	20,	107050.,	47878.,	44.7,	50.2,	14.9
9,	32,	9,	3,	20,	96901.,	48073.,	49.6,	56.1,	16.5
10,	32,	10,	3,	20,	87014.,	48201.,	55.4,	62.0,	18.4
11,	32,	11,	3,	20,	80042.,	48071.,	60.1,	66.8,	20.0
12,	32,	12,	3,	20,	73985.,	48098.,	65.0,	72.2,	21.6
13,	32,	13,	3,	20,	69140.,	48377.,	70.0,	76.4,	23.1
14,	32,	14,	3,	20,	65158.,	48306.,	74.1,	79.6,	24.6
15,	32,	15,	3,	20,	61699.,	48426.,	78.5,	83.7,	25.9

np	nM	nO	tM	tO	ContBarr	nAcessos	BarM	BarMp	BarO
16,	32,	16,	3,	20,	59178.,	48578.,	82.1,	88.6,	27.0
17,	32,	17,	3,	20,	56584.,	48646.,	86.0,	90.9,	28.3
18,	32,	18,	3,	20,	54993.,	48847.,	88.8,	94.0,	29.1
19,	32,	19,	3,	20,	53574.,	49142.,	91.7,	96.2,	29.9
20,	32,	20,	3,	20,	53070.,	49596.,	93.5,	97.7,	30.1
21,	32,	21,	3,	20,	52705.,	49862.,	94.6,	98.5,	30.4
22,	32,	22,	3,	20,	52988.,	50668.,	95.6,	99.2,	30.2
23,	32,	23,	3,	20,	53955.,	51750.,	95.9,	99.8,	29.7
24,	32,	24,	3,	20,	56132.,	53918.,	96.1,	99.8,	28.5
25,	32,	25,	3,	20,	57684.,	55406.,	96.1,	99.9,	27.7
26,	32,	26,	3,	20,	60118.,	57482.,	95.6,	99.8,	26.6
27,	32,	27,	3,	20,	61251.,	58213.,	95.0,	99.5,	26.1
28,	32,	28,	3,	20,	61806.,	58648.,	94.9,	99.9,	25.9
29,	32,	29,	3,	20,	62456.,	58953.,	94.4,	99.4,	25.6
30,	32,	30,	3,	20,	62999.,	59131.,	93.9,	98.9,	25.4
31,	32,	31,	3,	20,	63845.,	59410.,	93.1,	99.4,	25.1
32,	32,	32,	3,	20,	64803.,	59622.,	92.0,	98.6,	24.7

2 - Variação do número de bancos de memória

np	nM	nO	tM	tO	ContBarr	nAcessos	BarM	BarMp	BarO
1,	1,	1,	3,	20,	841622.,	47660.,	5.7,	6.7,	1.9
3,	1,	3,	3,	20,	313194.,	47699.,	15.2,	17.2,	5.1
4,	1,	4,	3,	20,	249850.,	47875.,	19.2,	19.7,	6.4
5,	1,	5,	3,	20,	240540.,	48078.,	20.0,	20.3,	6.7
6,	1,	6,	3,	20,	240285.,	48006.,	20.0,	20.4,	6.7
8,	1,	8,	3,	20,	241225.,	48241.,	20.0,	20.5,	6.6
9,	1,	9,	3,	20,	240689.,	48109.,	20.0,	20.7,	6.6
16,	1,	16,	3,	20,	243416.,	48672.,	20.0,	21.2,	6.6
17,	1,	17,	3,	20,	243978.,	48785.,	20.0,	21.5,	6.6
19,	1,	19,	3,	20,	247401.,	49468.,	20.0,	21.5,	6.5
10,	2,	10,	3,	20,	125766.,	48176.,	38.3,	40.1,	12.7
11,	2,	11,	3,	20,	124057.,	47987.,	38.7,	39.8,	12.9
12,	2,	12,	3,	20,	124456.,	48215.,	38.7,	40.4,	12.9
13,	2,	13,	3,	20,	123986.,	48160.,	38.8,	40.1,	12.9
14,	2,	14,	3,	20,	123776.,	48198.,	38.9,	40.2,	12.9
15,	2,	15,	3,	20,	124140.,	48269.,	38.9,	40.1,	12.9
16,	2,	16,	3,	20,	123992.,	48258.,	38.9,	40.2,	12.9
17,	2,	17,	3,	20,	124636.,	48382.,	38.8,	40.1,	12.8
18,	2,	18,	3,	20,	125434.,	48547.,	38.7,	40.0,	12.8
19,	2,	19,	3,	20,	126059.,	48682.,	38.6,	39.9,	12.7
20,	2,	20,	3,	20,	126917.,	48850.,	38.5,	39.9,	12.6
21,	2,	21,	3,	20,	129025.,	49304.,	38.2,	39.6,	12.4
22,	2,	22,	3,	20,	131905.,	49885.,	37.8,	39.3,	12.1
23,	2,	23,	3,	20,	134663.,	50430.,	37.4,	39.0,	11.9
24,	2,	24,	3,	20,	139458.,	51402.,	36.9,	38.5,	11.5
25,	2,	25,	3,	20,	144113.,	52335.,	36.3,	37.9,	11.1

np	nM	nO	tM	tO	ContBarr	nAcessos	BarM	BarMp	BarO
26,	2,	26,	3,	20,	148987.,	53315.,	35.8,	37.6,	10.7
27,	2,	27,	3,	20,	153736.,	54260.,	35.3,	36.7,	10.4
28,	2,	28,	3,	20,	159349.,	55396.,	34.8,	36.3,	10.0
29,	2,	29,	3,	20,	164875.,	56498.,	34.3,	35.8,	9.7
10,	4,	10,	3,	20,	96073.,	47911.,	49.9,	55.5,	16.7
11,	4,	11,	3,	20,	90072.,	47970.,	53.3,	56.8,	17.8
12,	4,	12,	3,	20,	84888.,	48054.,	56.6,	61.1,	18.8
13,	4,	13,	3,	20,	80347.,	48042.,	59.8,	63.9,	19.9
14,	4,	14,	3,	20,	77244.,	48128.,	62.3,	65.8,	20.7
15,	4,	15,	3,	20,	75152.,	48259.,	64.2,	68.4,	21.3
16,	4,	16,	3,	20,	72563.,	48361.,	66.6,	70.5,	22.0
17,	4,	17,	3,	20,	70934.,	48365.,	68.2,	72.1,	22.6
18,	4,	18,	3,	20,	69206.,	48369.,	69.9,	73.1,	23.1
19,	4,	19,	3,	20,	68317.,	48470.,	70.9,	74.6,	23.4
20,	4,	20,	3,	20,	67711.,	48568.,	71.7,	75.9,	23.6
21,	4,	21,	3,	20,	66790.,	48646.,	72.8,	76.7,	24.0
22,	4,	22,	3,	20,	66177.,	48691.,	73.6,	76.8,	24.2
23,	4,	23,	3,	20,	66730.,	49011.,	73.4,	77.5,	24.0
24,	4,	24,	3,	20,	68423.,	49636.,	72.5,	77.3,	23.4
25,	4,	25,	3,	20,	72290.,	50482.,	69.8,	75.7,	22.1
26,	4,	26,	3,	20,	77447.,	51537.,	66.5,	72.2,	20.7
27,	4,	27,	3,	20,	81377.,	52347.,	64.3,	68.7,	19.7
28,	4,	28,	3,	20,	86571.,	53410.,	61.7,	66.0,	18.5
29,	4,	29,	3,	20,	91864.,	54483.,	59.3,	62.9,	17.4
10,	8,	10,	3,	20,	90712.,	47886.,	52.8,	57.9,	17.6
11,	8,	11,	3,	20,	83192.,	47946.,	57.6,	63.5,	19.2
12,	8,	12,	3,	20,	77432.,	48006.,	62.0 ,	67.1,	20.7
13,	8,	13,	3,	20,	72407.,	48027.,	66.3,	70.7,	22.1
14,	8,	14,	3,	20,	68706.,	48133.,	70.1,	74.3,	23.3
15,	8,	15,	3,	20,	65397.,	48187.,	73.7,	79.0,	24.5
16,	8,	16,	3,	20,	62222.,	48226.,	77.5,	81.5,	25.7
17,	8,	17,	3,	20,	60160.,	48303.,	80.3,	84.9,	26.6
18,	8,	18,	3,	20,	58039.,	48347.,	83.3,	87.8,	27.6
19,	8,	19,	3,	20,	56117.,	48418.,	86.3,	90.3,	28.5
20,	8,	20,	3,	20,	55166.,	48555.,	88.0,	92.6,	29.0
21,	8,	21,	3,	20,	53699.,	48590.,	90.5,	95.1,	29.8
22,	8,	22,	3,	20,	53116.,	48720.,	91.7,	97.4,	30.1
23,	8,	23,	3,	20,	52852.,	48971.,	92.7,	97.5,	30.3
24,	8,	24,	3,	20,	52205.,	48964.,	93.8,	98.5,	30.6
25,	8,	25,	3,	20,	52497.,	49400.,	94.1,	99.0,	30.5
26,	8,	26,	3,	20,	52633.,	49605.,	94.2,	99.6,	30.4
27,	8,	27,	3,	20,	52746.,	49655.,	94.1,	99.6,	30.3
28,	8,	28,	3,	20,	54232.,	50608.,	93.3,	100.0,	29.5
29,	8,	29,	3,	20,	57429.,	52229.,	90.9,	100.0,	27.9
10,	16,	10,	3,	20,	88602.,	47872.,	54.0,	61.2,	18.1
11,	16,	11,	3,	20,	81054.,	47934.,	59.1,	65.3,	19.7
12,	16,	12,	3,	20,	74869.,	47947.,	64.0,	69.1,	21.4
13,	16,	13,	3,	20,	70070.,	48046.,	68.6,	74.4,	22.8
14,	16,	14,	3,	20,	66149.,	48085.,	72.7,	77.5,	24.2
15,	16,	15,	3,	20,	62641.,	48151.,	76.9,	82.0,	25.5
16,	16,	16,	3,	20,	59626.,	48280.,	81.0,	84.9,	26.8

np	nM	nO	tM	tO	ContBarr	nAcessos	BarM	BarMp	BarO
17,	16,	17,	3,	20,	57063.,	48270.,	84.6,	88.5,	28.0
18,	16,	18,	3,	20,	55458.,	48417.,	87.3,	92.7,	28.9
19,	16,	19,	3,	20,	53936.,	48457.,	89.8,	94.1,	29.7
20,	16,	20,	3,	20,	52745.,	48522.,	92.0,	96.3,	30.3
21,	16,	21,	3,	20,	51751.,	48666.,	94.0,	97.5,	30.9
22,	16,	22,	3,	20,	51643.,	48894.,	94.7,	98.5,	31.0
23,	16,	23,	3,	20,	51225.,	48955.,	95.6,	99.4,	31.2
24,	16,	24,	3,	20,	50977.,	49116.,	96.3,	99.6,	31.4
25,	16,	25,	3,	20,	51414.,	49265.,	95.8,	100.0,	31.1
26,	16,	26,	3,	20,	51641.,	49812.,	96.5,	100.0,	31.0
27,	16,	27,	3,	20,	51703.,	49902.,	96.5,	100.0,	30.9
28,	16,	28,	3,	20,	52052.,	50289.,	96.6,	100.0,	30.7
29,	16,	29,	3,	20,	52384.,	50634.,	96.7,	100.0,	30.5

3 - Variação do número de operadores

np	nM	nO	tM	tO	ContBarr	nAcessos	BarM	BarMp	BarO
20,	8,	1,	3,	20,	449030.,	48289.,	10.8,	42.4,	3.6
20,	8,	2,	3,	20,	225405.,	48324.,	21.4,	44.7,	7.1
20,	8,	3,	3,	20,	150606.,	48388.,	32.1,	45.3,	10.6
20,	8,	4,	3,	20,	113795.,	48387.,	42.5,	49.2,	14.1
20,	8,	5,	3,	20,	91369.,	48459.,	53.0,	62.5,	17.5
20,	8,	6,	3,	20,	76988.,	48606.,	63.1,	73.8,	20.8
20,	8,	7,	3,	20,	66634.,	48663.,	73.0,	83.9,	24.0
20,	8,	8,	3,	20,	60493.,	48829.,	80.7,	91.1,	26.4
20,	8,	9,	3,	20,	57213.,	48885.,	85.4,	91.5,	28.0
20,	8,	10,	3,	20,	56317.,	48933.,	86.9,	93.2,	28.4
20,	8,	11,	3,	20,	55775.,	48922.,	87.7,	92.7,	28.7
20,	8,	12,	3,	20,	56050.,	49002.,	87.4,	94.0,	28.5
20,	8,	13,	3,	20,	55689.,	48848.,	87.7,	93.5,	28.7
20,	8,	14,	3,	20,	56099.,	49022.,	87.4,	93.5,	28.5
20,	8,	15,	3,	20,	55488.,	48930.,	88.2,	93.5,	28.8
20,	8,	16,	3,	20,	55488.,	48930.,	88.2,	93.5,	28.8
20,	8,	17,	3,	20,	55488.,	48930.,	88.2,	93.5,	28.8
20,	8,	18,	3,	20,	55488.,	48930.,	88.2,	93.5,	28.8
20,	8,	19,	3,	20,	55488.,	48930.,	88.2,	93.5,	28.8
20,	8,	20,	3,	20,	55488.,	48930.,	88.2,	93.5,	28.8

4 - Variação da velocidade da memória

np	nM	nO	tM	tO	ContBarr	nAcessos	BarM	BarMp	BarO
10,	32,	10,	3,	20,	87348.,	47870.,	54.8,	61.6,	18.3
11,	32,	11,	3,	20,	79918.,	47903.,	59.9,	67.6,	20.0
12,	32,	12,	3,	20,	73897.,	47964.,	64.9,	71.5,	21.7
13,	32,	13,	3,	20,	69206.,	48018.,	69.4,	76.0,	23.1
14,	32,	14,	3,	20,	64447.,	48044.,	74.5,	80.2,	24.8
15,	32,	15,	3,	20,	61140.,	48176.,	78.8,	83.9,	26.2
16,	32,	16,	3,	20,	58518.,	48209.,	82.4,	86.9,	27.3
17,	32,	17,	3,	20,	55970.,	48244.,	86.2,	90.5,	28.6
18,	32,	18,	3,	20,	54035.,	48336.,	89.5,	93.5,	29.6
19,	32,	19,	3,	20,	52952.,	48469.,	91.5,	95.3,	30.2
20,	32,	20,	3,	20,	51980.,	48565.,	93.4,	97.2,	30.8
21,	32,	21,	3,	20,	51496.,	48788.,	94.7,	99.2,	31.1
22,	32,	22,	3,	20,	51409.,	49181.,	95.7,	99.0,	31.1
23,	32,	23,	3,	20,	51388.,	49358.,	96.0,	99.4,	31.1
24,	32,	24,	3,	20,	52047.,	49998.,	96.1,	100.0,	30.7
10,	32,	10,	4,	20,	92114.,	47874.,	52.0,	58.2,	17.4
11,	32,	11,	4,	20,	85254.,	48044.,	56.4,	63.1,	18.8
12,	32,	12,	4,	20,	78859.,	48097.,	61.0,	68.2,	20.3
13,	32,	13,	4,	20,	72608.,	48034.,	66.2,	72.4,	22.0
14,	32,	14,	4,	20,	68495.,	48123.,	70.3,	75.0,	23.4
15,	32,	15,	4,	20,	64821.,	48168.,	74.3,	80.8,	24.7
16,	32,	16,	4,	20,	61425.,	48206.,	78.5,	83.3,	26.0
17,	32,	17,	4,	20,	58755.,	48291.,	82.2,	88.2,	27.2
18,	32,	18,	4,	20,	56282.,	48312.,	85.8,	90.2,	28.4
19,	32,	19,	4,	20,	54759.,	48379.,	88.3,	93.4,	29.2
20,	32,	20,	4,	20,	53561.,	48545.,	90.6,	95.2,	29.9
21,	32,	21,	4,	20,	52768.,	48705.,	92.3,	97.2,	30.3
22,	32,	22,	4,	20,	52109.,	48858.,	93.8,	98.2,	30.7
23,	32,	23,	4,	20,	51981.,	49245.,	94.7,	99.0,	30.8
24,	32,	24,	4,	20,	52398.,	49779.,	95.0,	99.3,	30.5
10,	32,	10,	5,	20,	96637.,	47930.,	49.6,	54.9,	16.6
11,	32,	11,	5,	20,	89558.,	48019.,	53.6,	59.6,	17.9
12,	32,	12,	5,	20,	82058.,	47957.,	58.4,	64.0,	19.5
10,	32,	10,	5,	20,	96637.,	47930.,	49.6,	54.9,	16.6
11,	32,	11,	5,	20,	89558.,	48019.,	53.6,	59.6,	17.9
12,	32,	12,	5,	20,	82058.,	47957.,	58.4,	64.0,	19.5
13,	32,	13,	5,	20,	76205.,	48043.,	63.0,	68.7,	21.0
14,	32,	14,	5,	20,	71743.,	48067.,	67.0,	72.5,	22.3
15,	32,	15,	5,	20,	67931.,	48139.,	70.9,	76.1,	23.6
16,	32,	16,	5,	20,	64593.,	48236.,	74.7,	79.9,	24.8
17,	32,	17,	5,	20,	61555.,	48238.,	78.4,	83.3,	26.0
18,	32,	18,	5,	20,	58751.,	48271.,	82.2,	87.2,	27.2
19,	32,	19,	5,	20,	57262.,	48443.,	84.6,	89.8,	27.9
20,	32,	20,	5,	20,	55385.,	48515.,	87.6,	92.4,	28.9
21,	32,	21,	5,	20,	54228.,	48641.,	89.7,	94.7,	29.5
22,	32,	22,	5,	20,	53414.,	48826.,	91.4,	96.6,	30.0
23,	32,	23,	5,	20,	53081.,	49110.,	92.5,	97.9,	30.1
24,	32,	24,	5,	20,	53060.,	49634.,	93.5,	98.4,	30.2

np	nM	nO	tM	tO	ContBarr	nAcessos	BarM	BarMp	BarO
10,	32,	10,	6,	20,	101802.,	47945.,	47.1,	52.7,	15.7
11,	32,	11,	6,	20,	93691.,	47947.,	51.2,	56.5,	17.1
12,	32,	12,	6,	20,	86560.,	48046.,	55.5,	60.9,	18.5
13,	32,	13,	6,	20,	80667.,	48098.,	59.6,	65.0,	19.8
14,	32,	14,	6,	20,	75482.,	48130.,	63.8,	68.6,	21.2
15,	32,	15,	6,	20,	71353.,	48223.,	67.6,	71.5,	22.4
16,	32,	16,	6,	20,	67612.,	48262.,	71.4,	76.0,	23.7
17,	32,	17,	6,	20,	64407.,	48315.,	75.0,	79.1,	24.8
18,	32,	18,	6,	20,	62038.,	48428.,	78.1,	82.8,	25.8
19,	32,	19,	6,	20,	59500.,	48439.,	81.4,	86.0,	26.9
20,	32,	20,	6,	20,	57591.,	48489.,	84.2,	89.3,	27.8
21,	32,	21,	6,	20,	56450.,	48676.,	86.2,	92.3,	28.3
22,	32,	22,	6,	20,	55072.,	48829.,	88.7,	94.1,	29.1
23,	32,	23,	6,	20,	54524.,	49083.,	90.0,	96.2,	29.3
24,	32,	24,	6,	20,	53952.,	49329.,	91.4,	97.0,	29.7
10,	32,	10,	7,	20,	106580.,	47909.,	45.0,	50.1,	15.0
11,	32,	11,	7,	20,	97766.,	47911.,	49.0,	54.1,	16.4
12,	32,	12,	7,	20,	91106.,	48014.,	52.7,	57.5,	17.6
13,	32,	13,	7,	20,	84631.,	48124.,	56.9,	61.8,	18.9
14,	32,	14,	7,	20,	78558.,	48067.,	61.2,	65.4,	20.4
15,	32,	15,	7,	20,	75244.,	48253.,	64.1,	68.9,	21.3
16,	32,	16,	7,	20,	71310.,	48353.,	67.8,	73.1,	22.4
17,	32,	17,	7,	20,	67719.,	48384.,	71.4,	76.2,	23.6
18,	32,	18,	7,	20,	65015.,	48438.,	74.5,	80.3,	24.6
19,	32,	19,	7,	20,	62640.,	48562.,	77.5,	82.4,	25.5
20,	32,	20,	7,	20,	60393.,	48604.,	80.5,	85.1,	26.5
21,	32,	21,	7,	20,	58700.,	48710.,	83.0,	88.3,	27.3
22,	32,	22,	7,	20,	57127.,	48837.,	85.5,	90.8,	28.0
23,	32,	23,	7,	20,	56162.,	49043.,	87.3,	92.4,	28.5
24,	32,	24,	7,	20,	55605.,	49267.,	88.6,	95.3,	28.8
10,	32,	10,	8,	20,	112022.,	47988.,	42.8,	47.6,	14.3
11,	32,	11,	8,	20,	103027.,	47976.,	46.6,	51.4,	15.5
12,	32,	12,	8,	20,	95251.,	48071.,	50.5,	55.4,	16.8
13,	32,	13,	8,	20,	88642.,	48099.,	54.3,	59.1,	18.1
14,	32,	14,	8,	20,	82067.,	48035.,	58.5,	62.6,	19.5
15,	32,	15,	8,	20,	77742.,	48174.,	62.0,	66.7,	20.6
16,	32,	16,	8,	20,	74297.,	48247.,	64.9,	69.4,	21.5
17,	32,	17,	8,	20,	70922.,	48422.,	68.3,	72.4,	22.6
18,	32,	18,	8,	20,	67654.,	48405.,	71.5,	75.7,	23.6
19,	32,	19,	8,	20,	65547.,	48599.,	74.1,	78.7,	24.4
20,	32,	20,	8,	20,	63556.,	48781.,	76.8,	82.6,	25.2
21,	32,	21,	8,	20,	60901.,	48712.,	80.0,	85.0,	26.3
22,	32,	22,	8,	20,	59249.,	48757.,	82.3,	88.1,	27.0
23,	32,	23,	8,	20,	58174.,	49019.,	84.3,	90.0,	27.5
24,	32,	24,	8,	20,	57329.,	49203.,	85.8,	92.4,	27.9
10,	32,	10,	9,	20,	116758.,	47932.,	41.1,	45.8,	13.7
11,	32,	11,	9,	20,	107970.,	47980.,	44.4,	48.8,	14.8
12,	32,	12,	9,	20,	98671.,	47936.,	48.6,	53.3,	16.2
13,	32,	13,	9,	20,	92535.,	48044.,	51.9,	56.9,	17.3
14,	32,	14,	9,	20,	86623.,	48146.,	55.6,	59.7,	18.5
15,	32,	15,	9,	20,	81846.,	48196.,	58.9,	63.7,	19.5
16,	32,	16,	9,	20,	77675.,	48243.,	62.1,	66.1,	20.6

np	nM	nO	tM	tO	ContBarr	nAcessos	BarM	BarMp	BarO
17,	32,	17,	9,	20,	74204.,	48355.,	65.2,	70.6,	21.6
18,	32,	18,	9,	20,	70848.,	48409.,	68.3,	72.2,	22.6
19,	32,	19,	9,	20,	68068.,	48589.,	71.4,	76.1,	23.5
20,	32,	20,	9,	20,	66217.,	48692.,	73.5,	78.7,	24.2
21,	32,	21,	9,	20,	63824.,	48731.,	76.4,	81.9,	25.1
22,	32,	22,	9,	20,	62290.,	48934.,	78.6,	84.2,	25.7
23,	32,	23,	9,	20,	60719.,	49020.,	80.7,	87.6,	26.4
24,	32,	24,	9,	20,	59996.,	49333.,	82.2,	89.4,	26.7

5 - Variação da velocidade dos operadores

np	nM	nO	tM	tO	ContBarr	nAcessos	BarM	BarMp	BarO
1,	32,	1,	3,	20,	838001.,	47660.,	5.7,	6.5,	1.9
2,	32,	2,	3,	20,	419323.,	47695.,	11.4,	14.0,	3.8
3,	32,	3,	3,	20,	281068.,	47696.,	17.0,	19.3,	5.7
4,	32,	4,	3,	20,	210450.,	47736.,	22.7,	25.7,	7.6
5,	32,	5,	3,	20,	168694.,	47770.,	28.3,	31.9,	9.5
6,	32,	6,	3,	20,	141784.,	47837.,	33.7,	38.2,	11.3
7,	32,	7,	3,	20,	122396.,	47912.,	39.1,	44.3,	13.1
8,	32,	8,	3,	20,	107050.,	47878.,	44.7,	50.2,	14.9
9,	32,	9,	3,	20,	96901.,	48073.,	49.6,	56.1,	16.5
10,	32,	10,	3,	20,	87014.,	48201.,	55.4,	62.0,	18.4
11,	32,	11,	3,	20,	80042.,	48071.,	60.1,	66.8,	20.0
12,	32,	12,	3,	20,	73985.,	48098.,	65.0,	72.2,	21.6
13,	32,	13,	3,	20,	69140.,	48377.,	70.0,	76.4,	23.1
14,	32,	14,	3,	20,	65158.,	48306.,	74.1,	79.6,	24.6
15,	32,	15,	3,	20,	61699.,	48426.,	78.5,	83.7,	25.9
16,	32,	16,	3,	20,	59178.,	48578.,	82.1,	88.6,	27.0
17,	32,	17,	3,	20,	56584.,	48646.,	86.0,	90.9,	28.3
18,	32,	18,	3,	20,	54993.,	48847.,	88.8,	94.0,	29.1
19,	32,	19,	3,	20,	53574.,	49142.,	91.7,	96.2,	29.9
20,	32,	20,	3,	20,	53070.,	49596.,	93.5,	97.7,	30.1
21,	32,	21,	3,	20,	52705.,	49862.,	94.6,	98.5,	30.4
22,	32,	22,	3,	20,	52988.,	50668.,	95.6,	99.2,	30.2
23,	32,	23,	3,	20,	53955.,	51750.,	95.9,	99.8,	29.7
24,	32,	24,	3,	20,	56132.,	53918.,	96.1,	99.8,	28.5
25,	32,	25,	3,	20,	57684.,	55406.,	96.1,	99.9,	27.7
26,	32,	26,	3,	20,	60118.,	57482.,	95.6,	99.8,	26.6
27,	32,	27,	3,	20,	61251.,	58213.,	95.0,	99.5,	26.1
28,	32,	28,	3,	20,	61806.,	58648.,	94.9,	99.9,	25.9
29,	32,	29,	3,	20,	62456.,	58953.,	94.4,	99.4,	25.6
30,	32,	30,	3,	20,	62999.,	59131.,	93.9,	98.9,	25.4
31,	32,	31,	3,	20,	63845.,	59410.,	93.1,	99.4,	25.1
32,	32,	32,	3,	20,	64803.,	59622.,	92.0,	98.6,	24.7

np	nM	nO	tM	tO	ContBarr	nAcessos	BarM	BarMp	BarO
20,	8,	1,	3,	30,	648947.,	48296.,	7.4,	41.9,	2.5
20,	8,	2,	3,	30,	325283.,	48303.,	14.8,	43.2,	4.9
20,	8,	3,	3,	30,	217428.,	48324.,	22.2,	43.6,	7.4
20,	8,	4,	3,	30,	163533.,	48358.,	29.6,	44.0,	9.8
20,	8,	5,	3,	30,	131526.,	48412.,	36.8,	46.0,	12.2
20,	8,	6,	3,	30,	109782.,	48428.,	44.1,	54.7,	14.6
20,	8,	7,	3,	30,	94498.,	48489.,	51.3,	61.5,	16.9
20,	8,	8,	3,	30,	83749.,	48564.,	58.0,	72.3,	19.1
20,	8,	9,	3,	30,	74809.,	48632.,	65.0,	76.2,	21.4
20,	8,	10,	3,	30,	69372.,	48687.,	70.2,	82.2,	23.1
20,	8,	11,	3,	30,	65347.,	48722.,	74.6,	86.8,	24.5
20,	8,	12,	3,	30,	63272.,	48781.,	77.1,	83.2,	25.3
20,	8,	13,	3,	30,	62200.,	48711.,	78.3,	84.3,	25.7
20,	8,	14,	3,	30,	62100.,	48637.,	78.3,	83.9,	25.8
20,	8,	15,	3,	30,	62376.,	48775.,	78.2,	84.2,	25.7