ANÁLISE DOS MICROPROCESSADORES CISC DE 32 BITS NA IMPLEMENTAÇÃO DE SISTEMAS PARA PROCESSAMENTO PARALELO

José Luiz Ribeiro Filho

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. Newton Faller, Ph.D.
(Presidente)

Prof. Edil Severiano Tavares Fernandes, Ph.D.

Prof. Júlio Salek Aude, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 1989

RIBEIRO FILHO, JOSÉ LUIZ

Análise dos Microprocessadores CISC de 32 bits na Implementação de Sistemas para Processamento Paralelo [Rio de Janeiro] 1989

IX, 169 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1989)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1.Processamento Paralelo 2.Micro - processadores I. COPPE/UFRJ

II. Título (série).

À minha esposa Maria Teresa

AGRADECIMENTOS

Ao Doutor Newton Faller pelo incentivo e pelas valiosas sugestões apresentadas no decorrer deste trabalho.

Aos demais membros da banca, que muito me honraram com sua participação e aos amigos da NCE pelo apoio e colaboração prestados.

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

ANÁLISE DOS MICROPROCESSADORES CISC DE 32 BITS NA IMPLEMENTAÇÃO DE SISTEMAS PARA PROCESSAMENTO PARALELO

José Luiz Ribeiro Filho

Março, 1989

Orientador: Prof. Newton Faller, Ph.D.

Programa: Engenharia de Sistemas e Computação

Este trabalho apresenta uma análise do suporte ao processamento paralelo oferecido por quatro dos microprocessadores de 32 *bits* mais populares.

Apresentamos inicialmente uma visão geral das arquiteturas e topologias utilizadas no processamento paralelo onde verificamos que os microprocessadores estão melhor adaptados às arquiteturas de multi-processadores. Em seguida analisamos as facilidades e dificuldades que cada microprocessador apresenta em relação às diversas características de *software* e *hardware* necessárias aos sistemas para processamento paralelo.

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

ANALYSIS OF 32 BIT CISC MICROPROCESSORS FOR THE IMPLEMENTATION OF PARALLEL PROCESSING SYSTEMS

José Luiz Ribeiro Filho

March, 1989

Thesis Supervisor: Newton Faller, Ph.D.

Department: Engenharia de Sistemas e Computação

This work presents an analysis of the support offered to the parallel processing by four of the most popular 32 bit microprocessors.

We first show a general overview of the architectures and topologies used in parallel processing. We could verify that the 32 bit microprocessors are better suited to the multi-processors architectures. Then we make an analysis of the facilities and dificulties shown by each microprocessor related to the software and hardware characteristics needed by the parallel processing.

Índice

CAPÍTULO I

INTRODUÇÃO

I.1. Apresentação do Trabalho

Este trabalho descreve e analisa os principais elementos relacionados com o projeto de sistemas para processamento paralelo baseados em microprocessadores com arquitetura *CISC*. A restrição do nosso universo às arquiteturas *CISC* pode ser explicada pelo sucesso comercial apresentado por esses microprocessadores, determinando uma preferência para sua utilização por parte dos fabricantes de sistemas para processamento paralelo. A grande quantidade de *software* disponível para esses microprocessadores foi outro fator determinante. Estes *softwares* podem ser utilizados nas arquiteturas paralelas baseadas nesses microprocessadores de forma transparente ao usuário.

Reconhecemos a existência de inúmeras possibilidades para a aplicação dos conceitos e técnicas de paralelismo ao nível de integração de circuitos. Entretanto, este tipo de abordagem foge ao escopo deste trabalho. Focalizamos nosso objetivo no nível da Arquitetura de Sistemas.

Escolhemos como ponto de partida quatro dos principais microprocessadores comerciais de 32 *bits* (Intel, Motorola, National e Zilog) que representam, neste momento, o estado da arte no campo dos microprocessadores *CISC*.

Os diversos elementos que compõem a arquitetura de um processador (modelo de programação, co-processadores, memória, entrada e saída, etc.) são apresentados e analisados sob a perspectiva do processamento paralelo.

Os microprocessadores foram escolhidos em função do histórico de seus fabricantes (principalmente na área dos circuitos de alta integração), sua popularidade, (caracterizada pela sua utilização em diversos sistemas comerciais no Brasil e no exterior) além de sua disponibilidade no mercado nacional.

I.2. Motivação

Este trabalho foi motivado pela experiência que adquirimos através de projetos de sistemas computacionais baseados em microprocessadores de 32 *bits* da Motorola. Os primeiros conceitos e idéias relacionadas com a operação de vários processadores em paralelo nos chegaram por intermédio do trabalho desenvolvido no projeto de um **supermicrocomputador** de 32 *bits* (PEGASUS 32X®) em que tomamos parte no Núcleo de Computação Eletrônica da U.F.R.J.

O surgimento de sistemas computacionais no exterior, com múltiplos microprocessadores operando paralelamente, também foi um fator de grande influência para a elaboração deste trabalho. Assim, a conjunção destes fatores nos encorajaram a desenvolver um trabalho que nos permitisse analisar a aplicação dos microprocessadores de 32 *bits* às arquiteturas paralelas.

I.3. Objetivos Pretendidos

Apesar das idéias de execução paralela de tarefas não serem recentes (ENSLOW[1], HWANG[23]), só há pouco tempo vêm sendo aplicadas em sistemas computacionais comerciais de médio custo. Os avanços tecnológicos obtidos com o advento dos circuitos VLSI têm sido apontados (THURBER[5], WALLICH[8]) como fator determinante para o desenvolvimento dos sistemas para processamento paralelo. Neste contexto, os microprocessadores de 32 *bits* lançados nos últimos 3 anos representam o elemento chave para o projeto desses sistemas. A utilização desses microprocessadores em sistemas convencionais de alto desempenho é fato consagrado. Entretanto, os sistemas paralelos baseados em microprocessadores encontram-se ainda nos estágios iniciais de sua evolução. Carecem de pesquisas mais aprofundadas e ferramentas de *software* mais adequadas à sua operação.

O principal objetivo deste trabalho é identificar as características dos microprocessadores *CISC* que possam facilitar, ou dificultar, a construção de sistemas de processamento paralelo. Esperamos oferecer aos engenheiros projetistas de sistemas digitais que desejam ingressar na área de sistemas paralelos, os subsídios e referências necessárias ao projeto desses sistemas e assim, incentivar o desenvolvimento das arquiteturas paralelas no país.

I.4. Trabalhos na Área

Os principais trabalhos que serviram de base para a

elaboração desta tese encontram-se relacionados a seguir.

Os trabalhos de ENSLOW[1,3] e HAYNES[4] que tratam da organização dos sistemas para processamento paralelo, foram de fundamental importância para a elaboração do capítulo II. Destacamos ainda THURBER & WALD[5] com seu trabalho sobre processamento paralelo e associativo, KUNG[9] com a arquitetura sistólica além de GURD, WATSON [6] & KIRKHAM[27], MENDELSON & SILBERMAN[28] e GAJSKY[7] com trabalhos sobre as arquiteturas *Data-Flow*.

Nos capítulos III e VI, especial atenção é dada ao trabalho de NG[11,29], versando sobre o suporte dos microprocessadores às linguagens de alto nível e sistemas operacionais. Também são importantes os trabalhos de JACKSON[30], HEERING[31] e PATEL[17].

O capítulo IV apresenta-se fundamentado nos trabalhos de SMITH[13,14] principalmente com relação às memórias *cache*. Igualmente importantes são os trabalhos de WILSON[32], YEN *et alli* [33] e CHEUNG *et alli* [34].

As principais referências para a elaboração do capítulo V foram os trabalhos de FURHT & MILUTINOV[18], HYDE[35] e WILKES[36].

Finalmente no capítulo VII, tomamos por base os trabalhos de HWANG[20] e GEHRINGER[24] em que são analisados vários sistemas comerciais para processamento paralelo.

I.5. Organização do Trabalho

Este trabalho está dividido em oito capítulos. O capítulo II apresenta uma revisão das idéias e conceitos relativos ao processamento paralelo. São discutidas as diversas arquiteturas e topologias existentes bem como sua aplicabilidade às diversas classes de problemas.

Do capítulo III ao V são apresentadas as características de *software* e módulos de *hardware* relevantes ao processamento paralelo. Para cada tópico é feita inicialmente uma revisão dos conceitos envolvidos e das características importantes. Em seguida, cada um dos quatro microprocessadores é apresentado de acordo com o suporte oferecido ao tópico em questão e finalmente são apresentadas as conclusões relativas a cada tópico ou ao capítulo.

O capítulo III faz uma apresentação geral dos microprocessadores destacando o suporte oferecido às linguagens de alto nível, seus modelos de programação e interligação com co-processadores.

O capítulo IV trata do suporte ao módulo de memória, descrevendo sua forma de organização e hierarquia onde é dado um destaque especial ao suporte às memórias *cache*.

No capítulo V é discutido o suporte à execução multitarefa, compartilhamento e proteção de memória através do módulo de gerenciamento de memória. Também são analisadas as características necessárias à implementação de **Memória Virtual**.

O sexto capítulo trata do suporte oferecido aos

sistemas operacionais. As facilidades para tratamento de exceções, manipulação de dispositivos de entrada e saída, troca de contexto e comunicação entre processadores são discutidas neste capítulo.

Apresentamos no sétimo capítulo alguns exemplos de sistemas para processamento paralelo. Procuramos estabelecer uma relação entre os tópicos analisados nos capítulos anteriores com as soluções apresentadas pelos diversos fabricantes de sistemas a fim de melhorar o desempenho de seus sistemas.

Finalmente o capítulo VIII relata as conclusões do trabalho de acordo com o objetivo pretendido. Deixamos ainda uma sugestão para a elaboração de trabalhos na mesma área com relação às arquiteturas do tipo *RISC*.

CAPÍTULO II

ARQUITETURAS PARA PROCESSAMENTO PARALELO

Desde o aparecimento dos computadores digitais, no início dos anos 50, um grande esforço tem sido feito no sentido de se conseguir aumentar, cada vez mais, o poder computacional dos sistemas. Cada nova geração de computadores que surgia trazia o objetivo de atingir e resolver um conjunto de necessidades que havia emergido nas gerações anteriores. Ao mesmo tempo, traziam soluções a uma série de outros problemas, aumentando o conjunto das aplicações atingidas pelos sistemas computacionais e ainda determinavam o surgimento de novas necessidades. Esta busca de soluções às novas necessidades e a ampliação do campo das aplicações dos sistemas computacionais existe até hoje, e continua a alimentar a evolução desses sistemas. A execução de tarefas em paralelo tem sido uma das opções seguidas quando se esgotam as possibilidades para o aumento de desempenho através do aumento da velocidade nominal dos processadores.

As idéias relacionadas com a execução paralela de tarefas não são recentes. Há muito tempo, de acordo com ENSLOW[1], o paralelismo foi introduzido, em diversos graus, nas máquinas sequenciais do tipo Von Neumman. A disponibilidade de recursos computacionais e a necessidade de se resolver problemas complexos em intervalos de tempo razoáveis, determinaram a divisão das tarefas

computacionais complexas em várias componentes individuais e sua alocação aos recursos computacionais disponíveis, separadamente.

O aperfeiçoamento e a evolução dessas idéias, aliado ao surgimento dos microprocessadores, proporcionaram um grande aumento no interesse para a construção de grandes estruturas de multi-computadores. Duas grandes áreas de pesquisa (PAKER[2]) podem ser identificadas:

- Construção de sistemas computacionais de uso geral
- Construção de máquinas especiais de propósito específico

Em ambos os casos, o estudo e a avaliação das formas de interconexão das diversas partes que compõem os sistemas constituem uma área de grande importância para melhoria da eficiência dos sistemas computacionais.

II.1. Fatores que Motivam o Desenvolvimento de Sistemas Paralelos

Vários são os fatores que motivam a construção de sistemas paralelos. ENSLOW[3] destaca os seguintes:

- Desempenho
- Flexibilidade
- Disponibilidade
- Confiabilidade

Poderia supor-se que para aumentar o **desempenho** de um sistema paralelo bastaria duplicar-se o *hardware* desse sistema. Tal suposição admite que as tarefas computacionais podem ser subdivididas e distribuídas entre os processadores. Isto, porém, só pode ser feito após a perfeita identificação do tipo de tarefa, do estabelecimento de uma política de alocação e da coordenação entre as partes de forma a operarem eficientemente. O tipo mais elementar de paralelismo consiste na execução de n tarefas independentes e não relacionadas, alocadas a n processadores.

Os sistemas para processamento paralelo apresentam, geralmente, opções para operação conjunta ou individual dos diversos elementos de processamento sobre cada tipo de aplicação garantindo uma grande **flexibilidade** para a configuração do sistema a cada instante.

A **disponibilidade** mede a capacidade de um sistema para responder a solicitações. O grau de disponibilidade de um sistema está diretamente relacionado com seu desempenho. Um sistema que opere de forma degradada apresentará uma capacidade menor para atendimento a solicitações, proporcional ao grau de degradação. Três fatores combinados determinam o grau de disponibilidade de um sistema:

- A probabilidade de ocorrência de uma falha que faça com que o sistema funcione de forma degradada,

. A extensão em que o sistema é degradado pela falha,

. O espaço de tempo em que o sistema permanece degradado.

Neste contexto, a degradação refere-se aos efeitos no desempenho e na funcionalidade do sistema. Nos casos em que é difícil diminuir a probabilidade de ocorrência de defeitos, a redução do grau de degradação resultante do defeito pode se apresentar como uma estratégia alternativa para manter elevado o grau de disponibilidade.

A **confiabilidade** é definida por PAKER[2] como a capacidade que um sistema apresenta para funcionar satisfatoriamente face a falhas de *hardware* ou *software*.

Sistemas de múltiplos processadores apresentam alto potencial de confiabilidade. Tal potencial, pode ser explorado de acordo com a natureza dos níveis e limites de confiabilidade exigidos, através da replicação de unidades de processamento e módulos acessórios. A replicação de componentes nos sistemas de múltiplos processadores abre uma nova perspectiva no campo da detecção e recuperação de erros. Introduz por outro lado, um maior grau de complexidade aos sistemas e determina a existência de mecanismos para a sincronização de eventos.

II.2. Características dos Sistemas Paralelos

As principais características das estruturas para processamento paralelo de alto desempenho apresentadas por HAYNES[4] são:

- Compostas por um grande número, possivelmente heterogêneo, de elementos computacionais,

- O número de elementos é conceitualmente possível de ser expandido a um custo de *hardware* não muito maior do que linear, obtendo-se um aumento de velocidade não muito menor do que linear,

- Usadas para a solução de um único problema por vez.

Esta última característica está relacionada com a utilização de técnicas de paralelismo ao nível de aplicação, ou seja, aproveitamento de todos os recursos computacionais disponíveis simultaneamente para a execução de uma única tarefa.

II.3. Limites Atuais Relativos aos Sistemas Paralelos

O campo de aplicações para os sistemas de processamento paralelo tem se mostrado bastante vasto. Sua utilização porém, não é irrestrita. Podemos destacar (HAYNES[4]) quatro grandes questões neste campo:

. Qual a melhor arquitetura para uma classe específica de problemas ?

. Quais os benefícios obtidos decorrentes do uso de arquiteturas dedicadas em relação às arquiteturas de propósito geral, para a solução de problemas específicos ?

. Qual a forma mais eficiente de interconexão, mantendo o custo satisfatório ?

. Quais os métodos para projeto de algoritmos e programação mais eficientes para sistemas paralelos ?

Muitos pesquisadores vêm tentando responder a estas questões, porém não se apresentaram ainda respostas definitivas.

II.4. Níveis de Paralelismo

ENSLOW[3] destaca quatro níveis principais onde o paralelismo pode ser utilizado para melhorar o desempenho dos sistemas computacionais:

Nível 1 - Dispositivos e Circuitos:

Velocidade do *hardware*

Nível 2 - Arquitetura do Sistema:
Algoritmos implementados nas unidades funcionais

Nível 3 - Organização do Sistema:
Topologia de interconexão das unidades funcionais

Nível 4 - *Software* do Sistema:
Velocidade e eficiência do sistema operacional e do *software* para suporte

O aumento de desempenho no primeiro nível é obtido através da utilização de novas tecnologias no projeto dos dispositivos e circuitos eletrônicos que permitam o aumento da sua velocidade de operação. Isto pode ser conseguido através da utilização do paralelismo no nível de integração de circuitos. A aplicação de técnicas como processamento em *pipeline*, leituras de instruções em avanço (*pre-fetching*) e controle de alocação de unidades funcionais de forma eficiente (*scoreboarding*, por exemplo) permitem obter um aumento significativo no desempenho dos processadores.

No segundo nível, pode-se aumentar o conjunto de unidades funcionais operando em paralelo sobre um conjunto de dados. A utilização de múltiplas unidades funcionais tais como unidades de processamento lógico e aritmético e unidades para ponto flutuante permitem que, sob certas circunstâncias, diversas operações possam ocorrer

em cada uma dessas unidades paralelamente.

No terceiro nível está a execução de processos, ou partes de um mesmo processo, em diversas unidades computacionais. Neste caso temos um sistema computacional composto pela interligação de sub-sistemas com capacidade autônoma de processamento. Esses sub-sistemas operando de forma cooperativa conferem flexibidade para tratamento de diversos tipos de problemas, confiabiladade e um aumento quase linear no desempenho global do sistema.

Finalmente, no quarto nível opera-se a coordenação dos três níveis anteriores através de sistemas operacionais e do *software* de suporte explorando-se ao máximo as características de cada nível. Os sistemas operacionais e linguagens de programação devem oferecer facilidades e ferramentas adequadas à utilização eficiente dos recursos disponíveis nos níveis anteriores. O suporte ao processamento paralelo neste nível parece-nos ainda bastante incipiente. Os sistemas operacionais e linguagens atualmente disponíveis para a implementação de algoritmos para processamento paralelo não permitem ainda a obtenção de um grau de eficiência satisfatório com relação aos programadores.

II.5. Classificação dos Sistemas Paralelos

Existem várias tentativas no sentido de classificar os sistemas paralelos. Devido à grande flexibilidade e à enorme multiplicidade de objetivos com que os sistemas são desenvolvidos, as classificações propostas atingem, na

maioria dos casos, apenas parcialmente os sistemas existentes.

THURBER[5] apresenta a proposta de M.J.Flynn que classifica os sistemas segundo sua forma de operação sobre dados e instruções. São apresentadas quatro classes:

.SISD - Máquinas que operam uma instrução por vez, sobre um dado de cada vez (máquinas Von Neumann tradicionais),

.SIMD - Máquinas que operam a mesma, e única instrução sobre um conjunto de múltiplos dados (processadores matriciais),

.MIMD - Máquinas que executam, simultaneamente, múltiplas instruções sobre um conjunto de múltiplos dados (multi-processadores),

.MISD - Máquinas que executam múltiplas instruções sobre uma mesma palavra de dados (processadores *pipeline*).

Outra proposta de classificação dos sistemas paralelos é apresentada por ENSLOW[3] e HAYNES[4], que utiliza como parâmetros a **Arquitetura** e a **Organização** (topologia de interconexão) do sistema.

Arquiteturas Paralelas:

- .Múltiplas Unidades Funcionais de Propósito Específico
- .Processadores Associativos
- .Processadores Matriciais
- .Processadores de Fluxo de Dados (*Data-Flow*)
- .Múltiplos Processadores

Organização dos Sistemas Paralelos:

- . Redes Comutadas
- . Redes em Anel
- . Sistemas Sistólicos
- . Barramento Compartilhado
- . Hipercubos

II.6. Arquiteturas Paralelas

As estruturas compostas por Múltiplas Unidades Funcionais de Propósito Específico são construídas com o objetivo de solucionar problemas bastante particulares. Apresentam velocidades extremamente altas e são muito eficientes já que uma grande parte do *hardware* estará sempre em uso a maior parte do tempo. Por outro lado, são capazes de executar apenas um conjunto restrito de operações e necessitam estar conectadas a sistemas convencionais. O fato de estarem restritas à solução de problemas específicos representa, sob alguns aspectos, uma

desvantagem. Além disso, o *software* para controle de sua operação apresenta-se bastante complexo. Os Processadores Associativos são usados para o processamento de problemas computacionais de grande complexidade tais como: previsão de tempo, processamento da dados de sistemas nucleares e sistemas bélicos para defesa. Sua operação baseia-se na obtenção de palavras de memória através de seu conteúdo ou algum tipo de atributo de um sub-campo de seu conteúdo, ao invés de seu endereço. Em seu trabalho, THURBER[5] afirma que um considerável esforço tem sido dispendido no sentido de se obter soluções com uma boa relação entre o custo e o desempenho para este tipo de arquitetura.

Uma grande expectativa tem sido verificada em relação à eliminação de estrangulamentos nos atuais sistemas de uso geral através do uso da associatividade em pequena escala (mecanismos de proteção, alocação de recursos, gerenciamento de memória, etc.). THURBER[5] afirma também que devido aos fatores ligados ao custo, as aplicações da associatividade estarão limitadas a problemas tais como a alocação de recursos e mecanismos para memória virtual ao invés do gerenciamento de bancos de dados ou pesquisa de dados em grandes arquivos.

A manipulação de estruturas de dados em forma matricial apresenta-se de forma bastante eficiente. através dos Processadores Matriciais (*Array Processors*). Estes processadores são formados por um grande conjunto de unidades aritméticas operando de forma cadenciada e executando a mesma operação em dados distintos (HAYNES[4]). Nestes sistemas, a unidade de controle emite uma instrução

para que todas as unidades de execução operem sobre seus dados locais. Cada unidade de execução pode efetuar pequenas modificações na instrução global (geralmente nos endereços dos operandos), ou ainda ser programada para ignorar a instrução.

Para que todo o paralelismo potencial de um Processador Matricial possa ser atingido, uma série de elementos são necessários. Deverá haver um *software* de suporte que permita identificar as operações que podem ser paralelizadas, assim como efetuar a distribuição das tarefas e a alocação dos recursos de forma eficiente. Atualmente, todos esses fatores são de preocupação do próprio programador que deve escolher criteriosamente o algoritmo que será usado e a forma de alocação dos dados.

Os sistemas do tipo Fluxo de Dados (*Data-Flow*), representam uma ruptura radical com o modelo Von Neumann. Num modelo simples descrito por WATSON[6], os dados movem-se ao longo dos arcos do grafo que representa o algoritmo do programa até os **nós** de execução. Em cada nó uma instrução é executada tão logo todos os operandos estejam presentes em suas entradas. Somente um elemento pode existir em um arco num dado instante.

Nesta arquitetura, as instruções não apresentam referências à memória uma vez que os arcos do grafo permitem que os operandos sejam transferidos diretamente de um nó de execução para outro.

Embora já existam algumas implementações (protótipos de laboratório) de máquinas de Fluxo de Dados, existe ainda uma série de problemas relacionados com as diversas formas

de dependência (dados, controle, etc.). Esses problemas são extensivamente abordados no trabalho de GAJSKY[7], que procura demonstrar que as alterações necessárias às arquiteturas das máquinas de Fluxo de Dados para sua solução acabam por transformá-las em algo semelhante aos sistemas de múltiplas unidades funcionais.

Questões relativas à aceitação das linguagens de programação destas máquinas, nas áreas não científicas, merecem ainda muitas discussões. Acredita-se que tais linguagens não permitam o aumento da produtividade dos programadores, podendo mesmo vir a diminuí-la.

A disponibilidade de processadores completos, tratados como unidades autônomas e de baixo custo, tem propiciado o crescimento do interesse nas estruturas baseadas em múltiplos processadores. Nestes sistemas, cada processador é inteiramente programável e pode executar seu próprio programa.

Concordamos com HAYNES[4] quando afirma que os sistemas baseados nesse tipo de arquitetura são bem mais flexíveis do que os sistemas já apresentados. Entretanto, seu controle também é complexo, necessitando de estruturas de interconexão com características mais gerais como barramentos padronizados.

Há várias classes importantes de aplicações que podem utilizar o paralelismo assíncrono, a nível de processos, obtendo aumentos lineares na velocidade de processamento com o aumento do número de processadores. A taxa de aumento de velocidade poderá depender vários fatores. Relacionamos a seguir alguns fatores que consideramos importantes:

. Sincronização

Poderá haver perda no desempenho global se for exigida a coordenação entre os processadores periodicamente.

. Algoritmos

Um algoritmo elaborado para um sistema não paralelo pode não conseguir obter todas as vantagens das arquiteturas paralelas.

. *Overhead*

Um algoritmo paralelo pode necessitar de mais *passos* do que seu equivalente serial para resolver o mesmo problema. Este gasto adicional corresponde ao custo do gerenciamento do paralelismo.

. Contenções

Múltiplos processadores disputando os mesmos recursos podem causar uma redução da velocidade média de execução de cada processador, se fossem considerados individualmente.

. Entrada e Saída

Tarefas que necessitem de um número elevado de operações de entrada e saída poderão não se beneficiar das vantagens oferecidas pelo processamento paralelo.

WALLICH[8] divide os sistemas de múltiplos processadores em duas classes principais: **sistemas baseados em barramentos** e **sistemas não baseados em barramentos.** Os sistemas baseados em barramentos dividem-se ainda em duas sub-classes: sistemas fortemente acoplados e sistemas fracamente acoplados.

Os sistemas fortemente acoplados, também chamados **multiprocessadores**, apresentam como principal característica uma área de memória comum (memória global) cujo acesso é compartilhado entre os diversos processadores. Nessa arquitetura, é feito o uso intensivo de estruturas de barramento de alta velocidade, responsáveis pela interligação dos módulos de memória com os módulos de processamento e E/S.

Os sistemas fracamente acoplados, chamados **multicomputadores**, apresentam memória individual (local a cada processador), embora possam apresentar uma pequena área de memória global para compartilhamento de dados.

Apesar das arquiteturas baseadas em barramentos apresentarem um alto grau de eficiência através da utilização de barramentos de alta velocidade, estão limitadas a um certo número de processadores pela *banda-passante* de cada barramento. A partir deste número, a interferência decorrente da introdução de mais unidades de processamento torna-se prejudicial ao desempenho global do sistema.

Os sistemas de múltiplos processadores encontram aplicações nas seguintes áreas:

. Controle de processos em tempo real

Uma aplicação pode ser composta de tarefas autônomas e bem definidas que interagem entre si.

. Solução de algoritmos estruturados de forma paralela (ex: multiplicação de matrizes)

Neste trabalho iremos focalizar nossa atenção nas arquiteturas de múltiplos processadores pois os microprocessadores apresentam características que os tornam mais adequados a essas arquiteturas. A concepção dos microprocessadores *CISC* é voltada para a solução de uma classe de problemas bastante ampla. Oferecem suporte para a utilização de barramentos além de poderem ser tratados como unidades autônomas de processamento.

II.7. Formas de Interconexão

Tem havido uma preocupação crescente no sentido de que a comunicação entre os elementos que compõem as estruturas paralelas represente um dos principais fatores para que o paralelismo de fato seja obtido (HAYNES[4]). Fazer com que o dado ou a instrução certa esteja disponível para o processador certo no tempo certo, é essencial para manter todos os processadores ocupados a maior parte do tempo. Tal situação torna-se impossível, sem que haja caminhos apropriados para a circulação das informações.

Dentre as várias formas de interconexão existentes, as **Redes Comutadas** e as **Redes em Anel** representam pontos

extremos relativos à complexidade de implementação e eficiência dos sistemas. As Redes Comutadas apresentam alta eficiência e alta complexidade enquanto as Redes em Anel podem ser implementadas com facilidade mas com eficiência relativamente baixa.

As Redes Comutadas representam a forma mais flexível e geral de interligação. Nessa topologia de interconexão cada processador está logicamente conectado a todos os outros módulos. Para sua implementação, são necessários n^2 comutadores para interligar n processadores a n módulos de memória. Para que seja estabelecida uma rota de comunicação, basta efetuar-se a comutação dos elementos apropriados de acordo com os endereços dos módulos de origem e destino. Este esquema torna-se impraticável para grandes matrizes devido ao número de comutadores necessários, aumentando sobremaneira o custo do *hardware*.

Na interconexão por Rede em Anel, cada processador está conectado aos outros processadores através de um único meio de comunicação disputado por todos. Sua estrutura é bastante simples tanto para o *hardware* quanto do ponto de vista lógico. Porém, somente 1/n da *banda-passante* está disponível para cada um dos n processadores. Este esquema torna-se interessante apenas nos casos em que sejam reduzidas as necessidades de comunicação, ou nos casos em que os processadores possam ser mantidos ocupados com tarefas que não necessitem de cooperação enquanto esperam que um dado seja transmitido ou recebido.

Devido às caracteríscas apresentadas, tanto as Redes Comutadas quanto as Redes em Anel parecem não se adequarem

às estruturas paralelas de alto desempenho.

Entre estas duas formas de interconexão encontram-se formas intermediárias dentre as quais podemos destacar os **Sistemas Sistólicos**, estruturas **baseadas em barramentos** e os **Sistemas Hipercubo.**

O conceito de Sistema Sistólico foi originariamente proposto para implementação em VLSI (KUNG[9]) para a solução de problemas envolvendo manipulação de matrizes de dados.

Consiste, basicamente, de um conjunto de elementos de processamento, também chamados **células**, capazes de realizar algumas operações simples. Podem ser interligados em forma de matriz ou árvore. A comunicação com o mundo exterior ocorre apenas nas **células de fronteira.**

O princípio básico da matriz sistólica consiste em substituir um único elemento complexo de processamento, por uma matriz de elementos de processamento com estruturas mais simples. Desta forma, pode-se conseguir um aumento substancial no desempenho sem que seja necessário o aumento da *banda-passante* da memória.

Neste sistema é feita uma analogia com um coração que pulsa, enviando os dados através da matriz de células. Neste caso, uma vez que um dado é trazido da memória, pode ser efetivamente utilizado em cada célula que atravesse enquanto estiver sendo *bombeado* de célula para célula através da matriz. Isto é possível para um grande número de aplicações ditas *compute-bound* em que múltiplas operações são realizadas em cada dado de forma repetitiva.

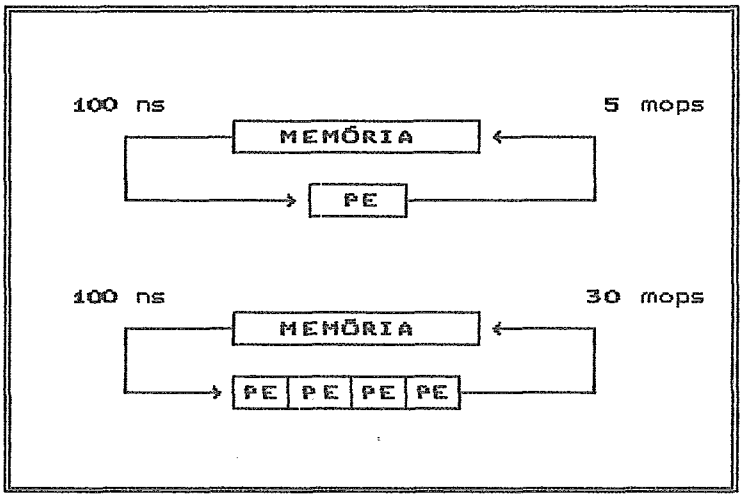


FIGURA II.1

Segundo KUNG[9], as tarefas computacionais podem ser divididas em duas classes, relativamente à operação do processador: *compute-bound* e *i/o-bound*. Uma tarefa é dita *compute-bound* quando o número total de operações realizadas sobre um único operando é maior que o número total de acessos à memória. Para que uma tarefa do tipo *i/o-bound* tenha sua velocidade de processamento aumentada, deverá haver necessariamente, um aumento da *banda-passante* da memória. Isto só poderá ser conseguido com a utilização de componentes mais rápidos (o que pode se tornar muito caro), ou ainda através de técnicas de entrelaçamento (que pode introduzir problemas complexos de gerenciamento de memória). O aumento da velocidade de processamento das tarefas do tipo *compute-bound* pode ser efetuado de forma simples e barata, através da filosofia sistólica.

Embora as arquiteturas baseadas em **Barramento Compartilhado** sejam extremamente poderosas, é certo também, que estão sempre limitadas pela velocidade na transferência

de dados (*banda-passante*) que um barramento pode suportar. Este limite pode ser o principal fator determinante para se estabelecer o número máximo de processadores que um sistema pode apresentar.

Nos sistemas **não baseados em barramentos**, cada processador apresenta memória local e comunica-se com outros processadores através de **mensagens**. Nestes sistemas um processador está diretamente conectado a um sub-conjunto de processadores que chamaremos de seus *vizinhos*. Mensagens entre processadores que não sejam vizinhos têm que passar por processadores intermediários. A eficiência do processamento neste caso, está diretamente ligada ao número de mensagens trocadas e processadores intermediários que deverão atravessar. A topologia **Hipercubo** é caracterizada pela flexibilidade que apresenta para interconexão dos processadores de forma a minimizar o número de processadores intermediários.

Um dos parâmetros de grande importância para os sistemas Hipercubo é sua **dimensão**. Um sistema Hipercubo possui dimensão n quando cada unidade de processamento (nó) encontra-se interligada a n-1 unidades vizinhas. A escolha da dimensão mais adequada à solução de um determinado problema é o principal fator para a eficiência destes sistemas.

O número máximo de nós intermediários que uma mensagem precisa atravessar para que dois nós não contíguos possam se comunicar é igual à dimensão do sistema. Na média porém, são utilizados apenas a metade do valor que representa a dimensão do sistema, para que a maior parte das

comunicações sejam efetuadas (assume-se que não existem contenções entre os diversos processadores em um mesmo canal de comunicações).

As organizações baseadas em barramento compartilhado apresentam-se adequadas a sistemas com até algumas dezenas de processadores. Já as organizações do tipo Hipercubo têm apresentado excelente desempenho para sistemas com centenas de processadores Supõe-se que poderão suportar até milhares de processadores operando paralelamente. Em geral, as organizações Hipercubo são projetadas para suportar uma proporção de 10:1 entre a taxa computacional e a taxa de comunicação.

Normalmente, essas topologias adaptam-se muito bem às aplicações científicas e de engenharia que envolvam simulações de fenômenos inerentemente concorrentes. Tais aplicações geram tipicamente, vários processos independentes e simultâneos que apresentam como principal característica a interação maior entre processos adjacentes.

Podem existir também formas híbridas de organização baseadas em barramentos, empregando memória local ou *cache* em cada processador, com a utilização de **Troca de Mensagens** para referências não locais. Esta pode ser a melhor opção para um número intermediário de processadores.

Segundo WALLICH[8], pouco *software* tem sido escrito para os sistemas Hipercubo. A maioria apresenta-se compatível com os sistemas operacionais tipo **UNIX®** e linguagens de alto nível como Fortran e C. Alguns dispõem de ferramentas para simulação que permitem o acompanhamento

da execução dos programas em sistemas mono-processados. Devido à complexidade envolvida no projeto de algoritmos paralelos e sua adequação a uma topologia, não se pode considerar a programação de sistemas Hipercubo como uma tarefa trivial.

A análise que pretendemos desenvolver em relação aos microprocessadores de 32 *bits* não se prende a nenhuma topologia de interconexão específica embora os microprocessadores apresentem, em sua totalidade, facilidades para o seu emprego em topologias baseadas em barramentos compartilhados. Isto não impede que sejam utilizados em sistemas do tipo Hipercubo, por exemplo. Deverão apenas, receber suporte externo (controladores e lógica discreta adicional) para adequarem-se a essas formas de interconexão.

II.8. Sistemas Multi-microprocessados

O baixo custo e o aumento significativo de desempenho dos microprocessadores tornaram as estruturas **multi-microprocessadas** uma opção bastante atraente para a construção de computadores de uso geral com uma expectativa de custo razoável.

Até bem pouco tempo atrás os microprocessadores eram utilizados, basicamente, nos sistemas de uso pessoal e na realização de tarefas bastante específicas como controladores de dispositivos, interfaces em geral e terminais. Com o surgimento dos microprocessadores de 32 *bits*, este quadro já começa a apresentar mudanças bastante

significativas. Esses processadores apresentam conjuntos de instruções bastante poderosos, maior capacidade aritmética, e até mesmo unidades para gerenciamento de memória. Tais recursos propiciaram a construção de sistemas em que é possivel a execução de *software* que apresente características como **multitarefa** e mais recentemente **multiprocessamento.**

Alguns dos principais sistemas multi-microprocessados, comercialmente disponíveis são apresentados por WALLICH[8] e MOKHOFF[10]. Dentre eles podemos destacar a série FX, produzida pela Alliant, que permite ao usuário configurar até 8 processadores para trabalharem na mesma aplicação, simultaneamente. Neste sistema, cada elemento de processamento possui um conjunto poderoso de instruções para **ponto flutuante** e processamento vetorial. A comunicação entre os elementos de processamento é realizada por meio de barramentos específicos.

A linha Balance, da Sequent, permite a interligação de até 30 elementos de processamento com capacidade de operação conjunta ou individual, dependendo da carga do sistema. A comunicação entre os processadores é feita através de memória compartilhada. Todos os módulos que compõem o sistema são interligados por um barramento central.

Na classe das Redes Comutadas, destacam-se os sistemas Butterfly, produzidos pela BBN Advanced Computers. Nestes sistemas podem estar conectados até 256 processadores através de uma rede comutada.

A topologia Hipercubo é representada pelas máquinas

Hypercube (Intel Scientific Computers), Ncube e Connection Machine (Thinking Machines).

Os sistemas multi-microprocessados, deste modo, vêm consolidando sua posição diante dos sistemas de médio e grande portes como uma alternativa que apresenta maior flexibilidade, confiabilidade e menor custo.

CAPÍTULO III

OS MICROPROCESSADORES DE 32 BITS

As dramáticas mudanças ocorridas a partir dos anos 80 na estrutura de custos das arquiteturas de alto desempenho através da introdução dos circuitos com tecnologia VLSI, vêm propiciando e mesmo encorajando o desenvolvimento de sistemas baseados em múltiplos microprocessadores. Além disso, a própria evolução dos microprocessadores tem apresentado taxas de crescimento nos índices de desempenho extremamente elevadas. Assim, a associação desses elementos em arquiteturas de múltiplos processadores apresentam, em muitos casos, desempenho superior aos sistemas de médio porte que utilizam circuitos de tecnologia LSI.

Este capítulo apresenta uma visão geral das principais características dos microprocessadores de 32 *bits* mais populares, destacando os pontos que permitam ou venham a facilitar sua utilização para a construção de sistemas para processamento paralelo. Os microprocessadores **80386** da INTEL, **MC68020** e **30** da MOTOROLA, **NS32032** da NATIONAL e **Z80000** da ZILOG foram escolhidos por representarem no momento, o estado da arte para a construção de sistemas multiprocessadores convencionais, baseados em microprocessadores.

São apresentadas e analisadas as características do conjunto de instruções, os modelos de programação de cada um dos microprocessadores e seus modos de endereçamento e

as interfaces a unidades co-processadoras.

III.1. Características Importantes para Suporte à Programação

Os projetos de microprocessadores eram dominados até recentemente, por preocupações com programas escritos em linguagem de máquina (*assembly*) e com a manutenção da compatibilidade com as características contidas nas arquiteturas de gerações anteriores. Os microprocessadores atuais de 32 *bits* têm demonstrado entretanto, uma preocupação maior principalmente relacionada com aspectos e características necessárias ao suporte às linguagens de alto nível e sistemas operacionais para multiprocessamento.

A filosofia de construção desses microprocessadores deve ser analisada à luz de alguns conceitos importantes de *software*, para que melhor possa ser compreendida.

Ortogonalidade

Se todos os tipos de dados reconhecidos por uma arquitetura puderem ser utilizados por todo o conjunto de instruções este é dito Ortogonal.

A tabela apresenta os tipos básicos de dados que os microprocessadores reconhecem e as funções suportadas. Nem todos os microprocessadores considerados reconhecem todos os tipos de dados apresentados na tabela . Os microprocessadores MC68020/30 por exemplo, não apresentam instruções para manipulação de cadeias de caracteres

(*strings*) como unidade básica; já o NS32032 não dispõe de instruções para operações sobre números BCD não compactados.

TABELA III.1

TIPO	MC68020/30	NS32032	80386	Z80000
BYTE	ADD,SUB, NEG,LOAD, STORE, RANGE CHK, COMPARE	ADD,SUB, MUL,DIV, MOD,ABS, NEG,LOAD, STORE, RANGE CHK, COMPARE	ADD,SUB, MUL,DIV, MOD, NEG,LOAD, STORE, COMPARE	ADD,SUB, NEG,LOAD, STORE, RANGE CHK, COMPARE
WORD, LONG	ADD,SUB, MUL,DIV, MOD, NEG,LOAD, STORE, RANGE CHK, COMPARE	ADD,SUB, MUL,DIV, MOD,ABS, NEG,LOAD, STORE, RANGE CHK, INDEXING, COMPARE	ADD,SUB, MUL,DIV, MOD, NEG,LOAD, STORE, RANGE CHK, COMPARE	ADD,SUB, MUL,DIV, MOD, NEG,LOAD, STORE, RANGE CHK, INDEXING, COMPARE
PACKED BCD	ADD,SUB, NEG,UNPK	ADD,SUB	DAA,DAS	DAB, ROTATE L/R
UNPACKED BCD	PACK		AAA,AAS, AAM,AAD	
BIT	CLR,SET, TEST, TOGGLE	CLR,SET, TEST, TOGGLE, FINDFIRST1	CLR,SET, TEST, TOGGLE, FINDFIRST	CLR,SET, TEST
BIT FIELD	CLR,SET, TEST, TOGGLE, FINDFIRST1 INSERT, EXTRACT	INSERT, EXTRACT	INSERT, EXTRACT	EXTRACT
LOGICAL	SHIFT L/R, ROTATE L/R AND,OR,NOT EOR, SAVE Cond.	SHIFT L/R, ROTATE L/R AND,OR,NOT EOR, BIT CLEAR, Save Cond.	SHIFT L/R, ROTATE L/R AND,OR,NOT EOR	SHIFT L/R, ROTATE L/R AND,OR,NOT EOR, Save Cond.
STRING		MOVE, COMPARE, SEARCH	MOVE, COMPARE, SEARCH	MOVE, COMPARE, SEARCH

Os microprocessadores Z80000 e MC68020/30 não apresentam um suporte tão bom para operações com inteiros de *bytes* quanto o oferecido para *words* e *double-words* de inteiros. Algumas instruções como a multiplicação não admitem o uso de operandos do tipo *byte*. Isto não implica porém, que tal operação não possa ser efetuada.

Os quatro microprocessadores apresentam de alguma forma, suporte para manipulação de números no formato BCD. Os microprocessadores da MOTOROLA dispõem de instruções especiais para adição, subtração e negação de dígitos BCD compactados (*packed BCD*) bem como instruções para compactação e descompactação de dígitos neste formato. O microprocessador NS32032 suporta apenas a adição e a subtração de números BCD compactados, e o 80386 permite que instruções normais de adição e subtração sejam aplicadas em números BCD compactados. Posteriormente torna-se necessária a utilização de instruções especiais de ajuste decimal a fim de corrigir o resultado final. O Z80000 opera de modo semelhante ao 80386 sobre os números no formato BCD. Apresenta também instruções especiais para deslocamento de dígitos BCD. Estas instruções podem ser utilizadas para deslocamento de números *multi-byte* BCD equivalendo a multiplicações ou divisões inteiras por dez.

são muito importantes para o suporte às linguagens de alto nível. De acordo com NGI11, suas aplicações mais comuns estão ligadas à alocação de memória, gráficos por *bit-map*, etc. O MC68020/30 apresenta instruções mais eficientes do que o NS32032 e o Z80000. Além da extração e inserção de

bits, comuns aos três processadores, o MC68020/30 pode desligar, complementar ou ligar todo o campo de *bits*.

Instruções booleanas são igualmente admitidas pelos microprocessadores. As instruções para manipulação de cadeias de caracteres (*strings*) usadas para cópia e comparação eficiente de tipos de dados estruturados, normalmente encontrados em linguagens de alto nível, são oferecidas apenas pelos microprocessadores da National, Intel e Zilog. Os microprocessadores MC68020 e 30 não apresentam suporte direto para operações em cadeias de caracteres. Admitem que tais operações possam ser realizadas por unidades co-processadoras que utilizem sua

torna a operação através de co-processadores transparente para o programador.

Simetria

De acordo com NG[11], se for possível a aplicação a todo o conjunto de instruções de um mesmo conjunto de modos de endereçamento para acesso a operandos, este é dito

Os microprocessadores MC68020 e 30 apresentam em seu conjunto de instruções 18 modos básicos de endereçamento. A maior parte dos modos de endereçamento pode ser aplicada a todas as instruções. Existem porém, algumas limitações: a) os registradores de endereçamento não podem ser usados como operandos para a maioria das instruções que efetuam cálculos, b) somente um operando, em instruções de 2

operandos, pode ser referenciado pelos modos comuns de endereçamento (o outro operando deverá, necessariamente, estar em algum dos registradores da máquina).

A criação de códigos de instrução de tamanho reduzido e uma utilização mais racional do conjunto de registradores pela redução do grau de simetria do conjunto de instruções.

Os 14 modos básicos de endereçamento do NS32032 são permitidos a todas as instruções mas, não são tão poderosos quanto os apresentados pelos microprocessadores da Motorola. Possuem alto grau de simetria, trazendo consigo casos, tornar menor a velocidade de execução.

O microprocessador Intel 80386 possui apenas 8 modos básicos de endereçamento. Não permite por exemplo, o endereçamento de operandos por conteúdo de memória (endereçamento indireto pela memória). Alguns modos de processamento (16/32 *bits*, protegido/virtual), dificultando sobremaneira a geração do código das instruções. Possui ainda um modo específico para o registrador **EAX** (*short-form*) que, juntamente com as outras características já citadas, determina uma quebra com o conceito de

A maior parte dos 9 modos básicos de endereçamento do Z80000 pode ser utilizada por todo o conjunto de instruções. Da mesma forma que os processadores da Motorola, permite que apenas um dos operandos, em instruções de mais de um operando, possa ser especificado

por qualquer dos 9 modos. O outro operando estará em um de seus registradores ou será especificado pelo modo imediato. Concordamos com NG[11] quando afirma que o Z80000 pode ser considerado um microprocessador com baixo grau de simetria.

A tabela III.2 apresenta uma comparação direta entre os diversos modos de endereçamento apresentados por cada um dos microprocessadores.

Embora a Ortogonalidade e a Simetria sejam características importantes para a geração de código e manipulação eficiente de estruturas de dados, não nos parece que ofereçam contribuições diretas ao processamento paralelo. Indiretamente, um código eficiente poderá facilitar o gerenciamento de espaços de memória distribuídos entre os diversos processadores. Como veremos mais adiante, no capítulo IV, a existência de instruções para a realização atômica de operações de leitura e escrita na memória são de maior importância para o processamento paralelo.

TABELA III.2

MODO	MC68020/30	NS32032	80386	Z80000
REGISTRADOR DIRETO	An,Dn	Rn	Rx	RRm
REGISTRADOR INDIRETO	(An)	d(Rn)	Rx	(RRm)
AUTO INCR./DECR.	(An)+, -(An)	(SP)+, -(SP)		(RRm)+, -(RRm)
REG. INDIRETO C/ DESLOC.	(d,An)	d(Rn), d(Pi)	d(Rx)	d(RRm)
REG. IND. C/ ÍNDICE E DESLOCAMENTO	(d,An,Xm)	d(Rn)[Xn]	d(Rx)[Ri]	d(RRm)[Xm]
MEM. IND, REG. IND. COM DESLOCAMENTO	([d1,An], d2)	d2(d1(Pi))		d(RRm)[Xm]
MEM. IND. PÓS INDEXADO	([d1,An], Xm,d2)	d2(d1(Pi))[Xn]		
MEM. IND. PRÉ INDEXADO	([d1,An], Xm,d2)			
PC+DESLOC.	(d,PC)	d(PC)		d(PC)
PC+ÍNDICE+ DESLOCAMENTO	(d,PC,Xm)	d(PC)[Xn]		d(PC)[Xm]
MEM. IND, PC+ DESLOCAMENTO	([d1,PC], d2)			
PC MEM. IND, PÓS-INDEX.	([d1,PC], Xm,d2)			
PC MEM. IND, PRÉ-INDEX.	([d1,PC, Xm],d2)			
ABSOLUTO	xxxx	xxxx	xxxx	xxxx
IMEDIATO	#xxxx	#xxxx	#xxxx	#xxxx

MODOS DE ENDEREÇAMENTO

III.2. Modelos de Programação

A correta utilização do conjunto de registradores de um processador bem como o número de registradores disponíveis pode apresentar importância fundamental para o desempenho global de um sistema computacional.

O número de referências à memória do sistema pode ser reduzido durante a execução de um programa com o aumento do número de registradores. Por outro lado, um número muito grande de registradores pode determinar uma queda no desempenho do sistema no momento em que são realizadas trocas de contextos. Neste instante, pode-se gastar um intervalo de tempo relativamente grande com as operações de cópia do conteúdo de registradores para a memória e a carga de um novo conjunto de dados durante a preparação para a execução de um novo processo.

As figuras **III.1** e **III.2** apresentam os modelos de programação do MC68030. Este microprocessador possui 16 registradores de 32 *bits* para uso geral. Apresenta ainda um apontador de instruções (PC) de 32 *bits*, três apontadores de pilha (um para o supervisor, um para usuário e outro para interrupções) e um registrador de *status* de 16 *bits* (SR). A tabela de vetores para tratamento de exceções é apontada por um regitrador de 32 *bits* (VBR). Existem dois registradores para controle e endereçamento da cache (CACR e CAAR) além de cinco registradores de 32 *bits* e um de 16, responsáveis pela operação da unidade de gerência de memória interna.

Dos 16 registradores de uso geral, 8 são utilizados

para armazenamento de dados (D0-D7) e operações aritméticas e lógicas sobre *bits* e *campos de bits*, *bytes*, *words*, *long words* e *quad-words*(64 *bits*). Os outros 8 registradores de uso geral (A0-A7) são utilizados para endereçamento de operandos. Todo o conjunto de registradores de uso geral é homogêneo. O registrador A7 porém, é usado ainda como apontador de pilha para o **modo usuário**. Existem dois outros registradores que são usados como apontadores das pilhas de interrupções e **modo supervisor** respectivamente (A7' e A7").

O registrador de *status* possui três *bits* para o controle da lógica interna de interrupções, permitindo estabelecer os diversos níveis que o sistema pode aceitar. Neste registrador estão também os *bits* de condição para a operação do processador: *extend*, *negate*, *zero*, *overflow* e *carry* (X,N,Z,V e C) .

A família de microprocessadores da Motorola apresenta oito **espaços de endereçamento** distintos mas apenas cinco estão definidos: Supervisor, Usuário, Código, Dado e Processador (*cpu space*). Os espaços de endereçamento distintos para modo supervisor e modo usuário determinam uma proteção intrínseca ao *hardware* do processador em relação aos acessos de programas de usuários a regiões exclusivas do sistema operacional.

O espaço de endereçamento do processador é usado para mapear dispositivos de *hardware* que auxiliam a operação do processador e operações específicas tais como reconhecimento de interrupções, tratamento de pontos de parada e acesso a co-processadores.

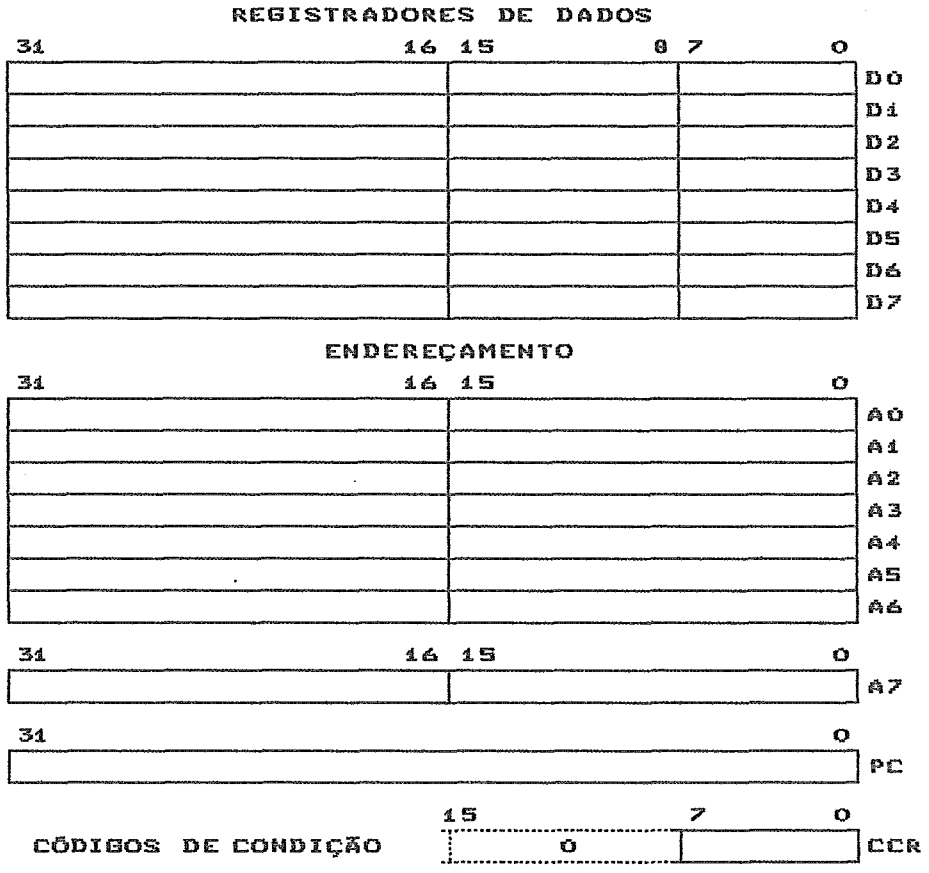


FIGURA III.1 - MODELO DO USUÁRIO (MC68030)

Um espaço de endereçamento é especificado pelo processador através de 3 sinais elétricos (FC0-FC2) durante os ciclos de acesso ao barramento externo.

Para que o modo supervisor possa ter acesso aos outros espaços de endereçamento, os registradores de código de função (SDF/DFC) são utilizados através de instruções especiais. Estas instruções permitem estabelecer o código desejado nos sinais FC0-FC2.

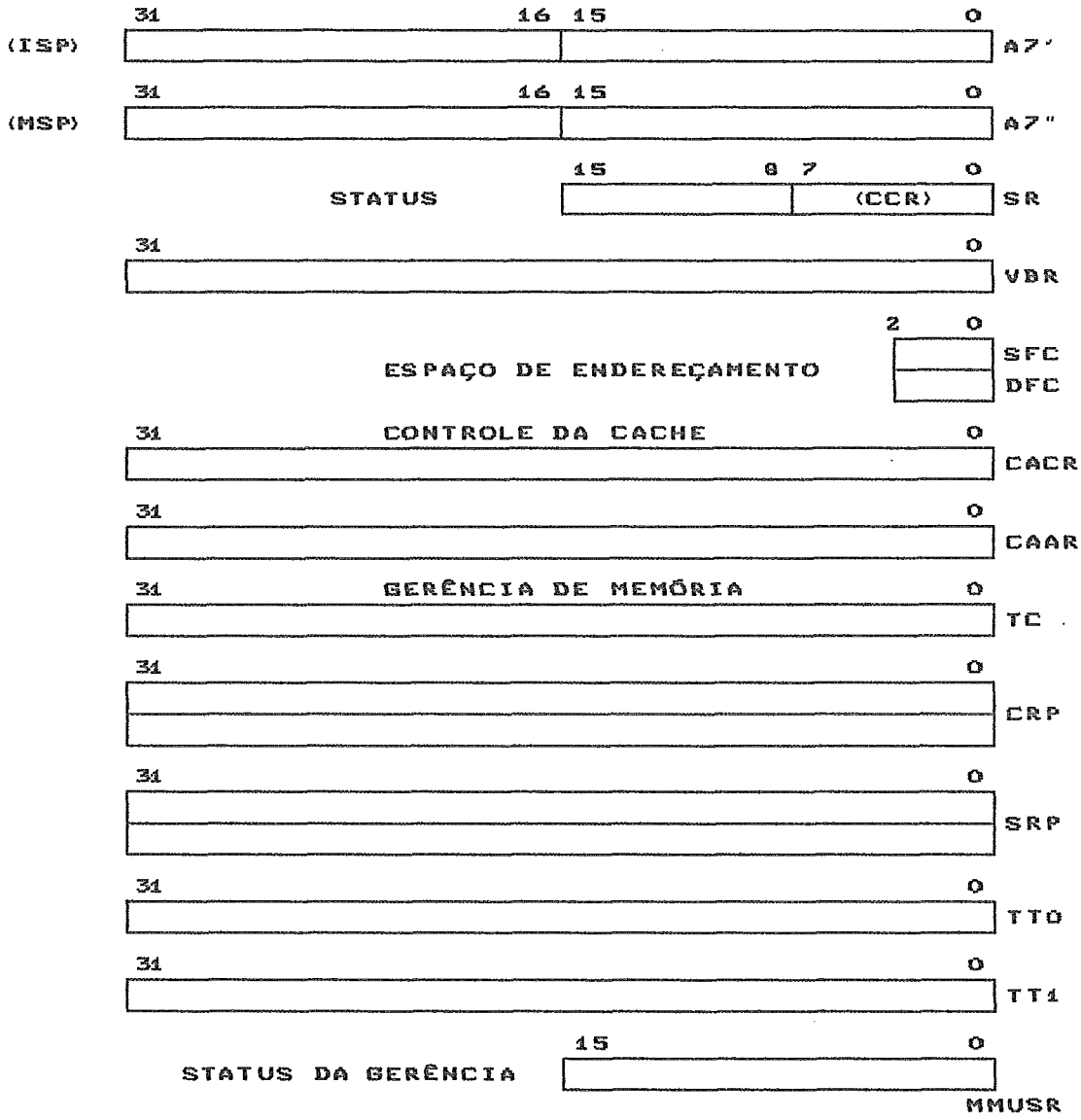


FIGURA III.2 - MODELO DO SUPERVISOR (MC68030)

O microprocessador NS32032 possui 8 registradores de uso geral (RR0-RR7), apresentados na figura III.3. Estes registradores podem ser usados para armazenamento temporário de dados e para realização de operações aritméticas e lógicas sobre seu conteúdo pelas unidades de controle e execução do processador, além do endereçamento de operandos. Estes registradores são homogêneos para a maioria das instruções. As instruções para manipulação de

strings constituem exceções pois utilizam os registradores RR0-RR4 com propósitos específicos. Existem ainda sete outros registradores que são utilizados para endereçamento de operandos e controle. Os registradores de endereçamento são usados também para implementação de algumas funções de suporte às linguagens de alto nível. O registrador FP (*frame pointer*) por exemplo, é usado por sub-rotinas para o acesso aos parâmetros e variáveis locais na pilha. Este registrador mantém o endereço da posição de memória que contém o último *frame pointer*. A instrução **ENTER** é responsável pela sua atualização quando uma sub-rotina é iniciada. Seu conteúdo anterior é restabelecido pela instrução **EXIT**. O registrador SB (*static base*) aponta para o início da região, na memória global, que contém as variáveis globais dos módulos de *software*. O endereço do início da tabela de vetores de interrupções e exceções está contido no registrador INTBASE. O módulo de *software* em execução tem o endereço de seu descritor armazenado no registrador MOD. O processador possui ainda, o registrador apontador de instruções (PC) e dois registradores apontadores de pilha (SP0 e SP1). O registrador SP0 é usado como apontador para a pilha do sistema operacional e rotinas de tratamento de interrupções. O SP1 aponta para o início da pilha do usuário. Todos estes registradores de endereçamento possuem apenas 24 *bits*, permitindo que sejam efetivamente endereçados 16 *Mbytes*.

Os códigos de *status* do processador são mantidos por um registrador de 16 *bits* (PSR). Da mesma forma que a linha da Motorola, o NS32032 apresenta dois modos de operação: **modo**

supervisor e **modo usuário.** Estes dois modos estão relacionados com o nível de proteção em relação a algumas instruções e uso de alguns registradores do processador.

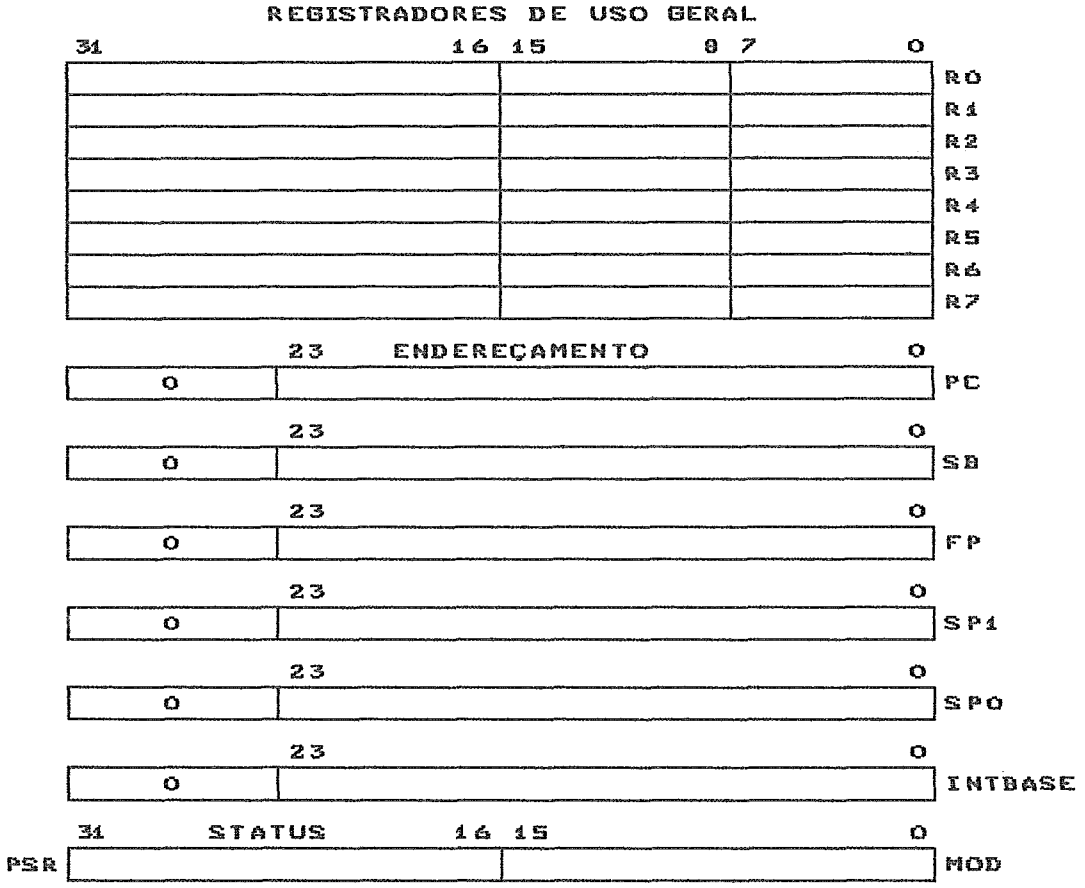


FIGURA III.3 - MODELO GERAL DO NS32032

O microprocessador 80386 possui ao todo 32 registradores distribuídos nas seguintes categorias:

- Registradores de uso Geral
- Registradores Descritores de Segmentos
- Apontador de Instruções e *flags*
- Registradores de Controle

.Registradores apontadores de Tabelas do Sistema

.Registradores de Depuração

.Registradores para Testes

A figura III.4 apresenta o conjunto completo dos registradores do 80386. Os 8 registradores de 32 *bits* para uso geral normalmente mantêm dados e endereços. Permitem a manipulação de operandos de dados de 1, 8, 16, 32 e 64 *bits*, além de campos de *bits* de 1 a 32 *bits*. Os operandos de endereços podem ser de 16 ou 32 *bits*.

O apontador de instruções (EIP) contém o deslocamento em relação ao endereço indicado pelo registrador descritor do segmento de código (CS). O registrador de *flags* (EFLAGS) é responsável pelo controle de algumas operações do processador e indica seu *status* após a execução de cada instrução.

Existem seis registradores descritores de segmentos: CS, SS, DS, ES, FS e GS; associados aos segmentos de código, pilha, dados, extra, *frame* e global respectivamente.

Os três registradores de controle (CR0, CR2 e CR3) são responsáveis, juntamente com registradores apontadores de tabelas do sistema, pelas condições gerais de funcionamento do processador. Através de CR0 por exemplo, pode-se estabelecer a forma de operação da unidade de gerenciamento de memória (segmentação, paginação ou mista), saber o tipo de co-processador aritmético presente (ou se não existe nenhum co-processador associado) e estabelecer o modo de

operação do processador (**protegido** ou **real**). CR2* contém o endereço que ocasionou uma exceção por falta de página para o processador. O endereço base do diretório da tabela de páginas é armazenado em CR3. Este valor é modificado por uma operação automática de troca de contexto ou manualmente, quando for necessária uma invalidação na memória **cache** interna que mantém as últimas referências à tabela de páginas.

Quatro são os registradores responsáveis pelas referências às tabelas de segmentos do sistema: GDT (*Global Descriptor Table*), IDT (*Interrupt Descriptor Table*), LDT (*Local Descriptor Table*) e TSS (*Task State Segment*). Estas tabelas apresentam tamanhos que podem variar entre 0 *bytes* e 64 *Kbytes*. Cada tabela pode conter até 8192 descritores de 8 *bytes*.

A GDT contém descritores dos segmentos que podem ser compartilhados por todos os processos do sistema. Pode conter qualquer tipo de descritor de segmento (código, dado, pilha, global, etc.) à exceção dos descritores usados para tratamento de interrupções e *traps*. Geralmente contém segmentos de dados e código utilizados pelo sistema operacional, processos de usuários e os descritores das LDT's do sistema.

As LDT's contêm os descritores associados a cada processo. Atuam como mecanismo de isolamento entre os

* **CR1** é reservado pela Intel para uso futuro.

segmentos de dados e códigos entre processos e os segmentos do sistema operacional.

A terceira tabela de descritores utilizada pelo 80386, IDT, contém os descritores que indicam as 256 possíveis rotinas para tratamento de exceções. Finalmente, a tabela de descritores para a TSS contém todas as informações relativas à localização, tamanho e nível de privilégio de um segmento descritor de processo.

Para o auxílio à depuração de programas existem seis registradores (*Debug Registers*). Os registradores DR0-DR3 são usados para especificar quatro endereços para pontos de parada enquanto o registrador DR7 é usado para controlar a ativação dos pontos de parada. As indicações relativas ao ponto de parada atingido são apresentadas em DR6.

Os registradores de teste (TR6 e TR7) são usados para controle do teste da memória de dados do TLB (*Translation Lookaside Buffer*) do 80386. TR6 é usado para comandar os testes enquanto TR7 contém os dados necessários para sua realização.

O 80386 é o único dos quatro microprocessadores analisados neste trabalho que não apresenta nenhum tipo de indicação relativa ao seu modo de operação (nível corrente de privilégio) através de sinais elétricos.

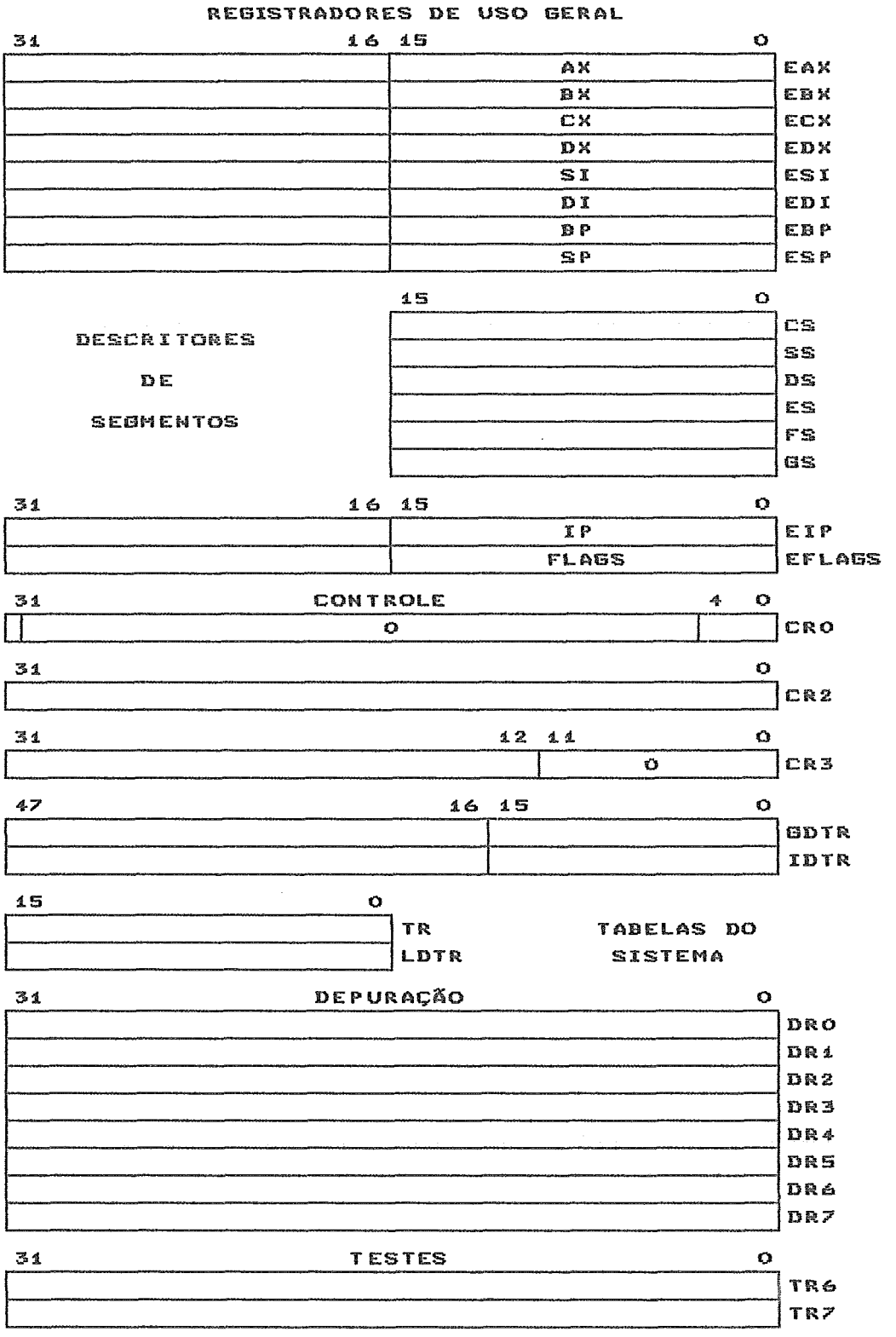


FIGURA III.4 - MODELO GERAL DO 80386

O conjunto de registradores do Z80000 (fig. III.5) é composto por 16 registradores de 32 *bits* para uso geral, 3 registradores para controle de programas, 2 para especificação e configuração da forma de operação do processador e 4 indicadores de localização das tabelas utilizadas pela unidade de gerenciamento de memória.

Os registradores de uso geral podem conter dados ou endereços. Os dados podem ser acessados como *byte*, *word* ou *double-word* (equivalente à *long-word*). Os quatro primeiros registradores de 32 *bits* podem ser acessados como *bytes*, determinando até 16 registradores de 8 *bits*. Da mesma forma, os primeiros 8 registradores de 32 *bits* podem ser tratados como 16 registradores de 16 *bits*.

O Z80000 apresenta dois modos de operação, relativos ao nível de proteção a alguns tipos de dados e instruções: **modo Normal e Sistema**. De modo semelhante aos microprocessadores da Motorola e National, existe uma indicação para o *hardware* sobre o modo corrente de operação através de um dos sinais elétricos do processador.

O controle dos programas é realizado pelo registrador apontador de instruções, pelo registrador de *status* e pelo apontador da pilha do **sistema** (RR14). Um outro registrador RR14 é usado como apontador da pilha para o modo **normal**.

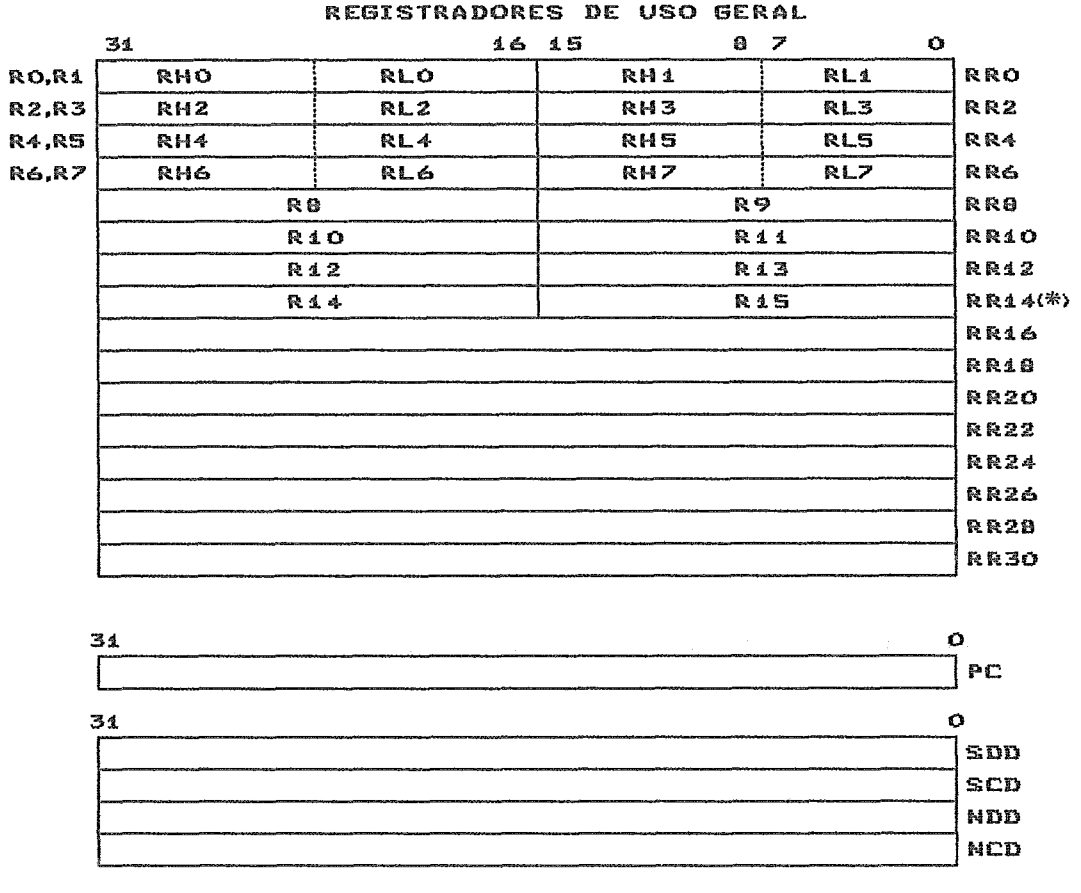
Há três formas diferentes para representação de endereços. Dois *bits* pertencentes aos registradores de controle determinam a seleção do modo de endereçamento **compacto**, **segmentado** ou **linear**.

O modo **compacto** utiliza endereços de apenas 16 *bits*. Este modo pode ser usado por programas cujo espaço total de

endereçamento seja inferior a 64Kbytes, propiciando a geração de um código mais eficiente.

O modo **segmentado** é indicado para programas extensos e que apresentem características especiais como código reentrante e áreas de dados globais e locais separadas. Este modo permite a separação das diversas partes que compõem um programa em segmentos de memória distintos.

O terceiro modo, **linear**, permite que o Z80000 utilize seus 32 *bits* de endereço em um único espaço de endereçamento linear.



(*) RR14 e RR14' são usados como apontadores de pilha.

FIGURA III.5 - MODELO GERAL DO Z80000

O tamanho do conjunto de registradores de uso geral constitui um fator importante para o processamento paralelo. Pode-se imaginar que um número elevado de registradores proporcionará ao processador a maior parte das informações necessárias ao processamento, sem que seja necessária a busca de novas informações na memória. Podemos encontrar este tipo de implementação em todos os sistemas monoprocessados de alto desempenho (supercomputadores). Para microprocessadores porém, tal característica implica em gasto de espaço no *chip* do processador para a implementação dos registradores assim como para sua estrutura de controle e alocação. Além disso, como os microprocessadores foram originariamente concebidos para monoprocessamento de aplicações de uso geral, apresentam instruções específicas que visam tornar as trocas de contextos mais eficientes e simples para o programador. Assim, o aumento do número de registradores nestes microprocessadores pode introduzir problemas e gasto excessivo de tempo nas operações de troca de contexto.

A utilização de memórias *cache* parece-nos adaptar-se melhor aos microprocessadores em sistemas paralelos, assegurando a redução do número de acessos à memória primária sem entretanto, introduzir complicações nos processos de troca de contexto. Neste aspecto, os microprocessadores da Motorola e Zilog apresentam vantagens em relação aos microprocessadores da National e Intel que necessitam de *hardware* externo.

A separação dos acessos à memória em vários tipos de espaços de endereçamento (dados/ código, normal/privilegia-

do) não contribui diretamente para facilitar a implementação de arquiteturas paralelas com estes microprocessadores. Consideramos que criação de barramentos de dados e endereços independentes para os espaços de endereçamento de dados e código poderia apresentar uma contribuição mais efetiva para as arquiteturas paralelas.

III.3. Interface com Co-Processadores

A tecnologia VLSI determinou um aumento substancial na capacidade funcional dos componentes eletrônicos responsáveis pelo processamento computacional propriamente dito. Atualmente, é tecnologicamente possível colocar num único *chip* toda a lógica para o controle e execução de istruções, um número razoável de registradores, unidades aritméticas para números inteiros e ponto flutuante e ainda memórias *cache* e unidades para gerenciamento e proteção de memória. Os fabricantes entretanto, têm procurado separar algumas destas unidades funcionais em componentes individuais. A principal justificativa é a de oferecer maior flexibilidade ao projeto dos sistemas computacionais. Mas, concordamos com BEINS[12] que esta divisão também está intimamente relacionada com o custo final dos microprocessadores. Um *chip* com todas as unidades funcionais mencionadas possui uma complexidade muito maior para desenvolvimento e confecção, acarretando um custo final bem mais elevado. Esta afirmação pode ser comprovada através da própria evolução das *famílias* de microprocessadores. A cada nova geração são incluídas novas

unidades funcionais permanecendo o preço final do componente nos mesmos patamares.

As unidades funcionais, enquanto separadas fisicamente do *chip* do processador, recebem o nome de **co-processadores**. A forma como os co-processadores interagem com o processador principal é definida por cada fabricante de modo a facilitar sua futura integração ao *chip* do microprocessador. Além disso, do ponto de vista do *software*, esta integração deve ocorrer da forma mais transparente possível. Assim, na ausência de um co-processador específico (ponto flutuante, por exemplo), o processador deverá ser capaz de substituí-lo de forma transparente para o programador.

Geralmente são encontradas três formas de interação entre o processadores e co-processadores que são efetuadas através de:

- Monitoramento Inteligente do barramento do processador
- Sinais (e instruções) Especiais
- Instruções Especiais

No primeiro caso (fig. III.6), cada processador monitora todas as instruções que transitam no barramento. Isto obriga que todos os co-processadores e o processador principal operem à mesma frequência. A principal desvantagem é que a velocidade de operação de todo o conjunto é determinada pelo elemento mais lento. Por outro lado, não é necessário nenhum ciclo de barramento para que

o processador comande o início das atividades de um co-processador. O número de co-processadores presentes deverá ser préviamente conhecido para que possa ser feita a emulação de suas funções pelo *software* no caso de sua ausência.

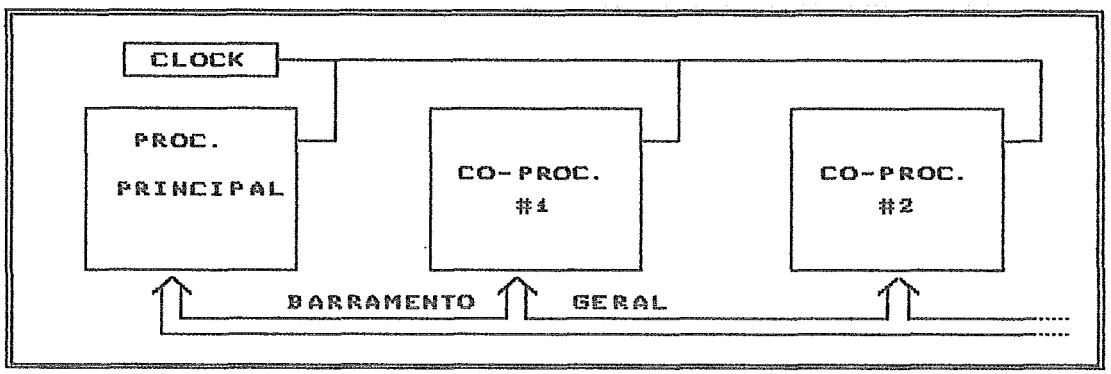


FIGURA III.6 - MONITORAMENTO DE INSTRUÇÕES

No segundo caso, os co-processadores podem ter suas frequências de operação independentes (fig III.7). O processador principal comunica-se com cada co-processador através de um barramento e uma interface de *hardware* específica.

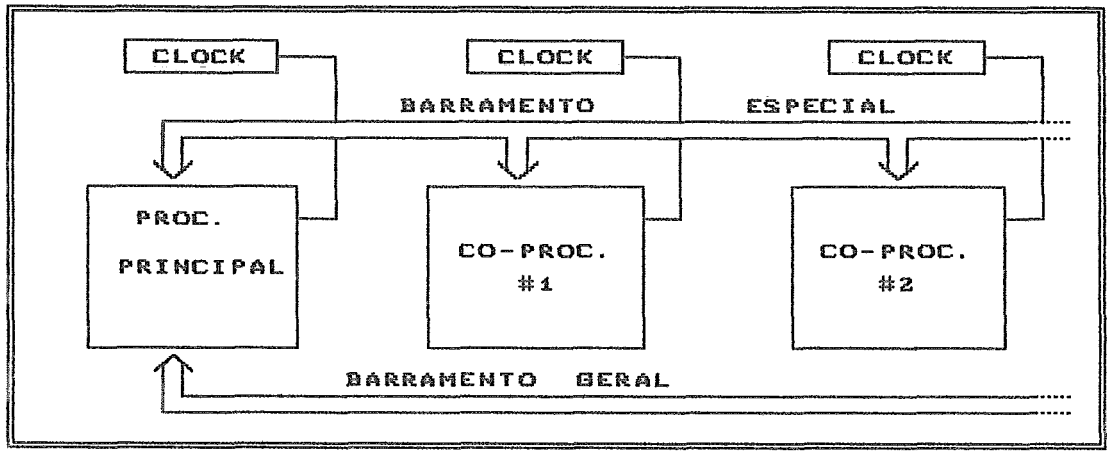


FIGURA III.7 - INTERFACE ESPECIAL

A terceira forma (figura III.8) é a mais transparente do ponto de vista do *software*. Um co-processador é visto como uma extensão do modelo do programador do processador principal. Isto é, o programador utiliza os registradores do co-processador como se estes fizessem parte do conjunto de registradores do próprio processador principal. Ao receber uma instrução de co-processador, o processador principal sinaliza o acesso a um co-processador através do barramento de controle. O não recebimento de uma resposta do co-processador endereçado (*time-out*) determina a emulação de suas funções pelo *software*. Esta forma de interação com co-processadores é adotada pela Motorola. Sua principal vantagem reside na facilidade que é oferecida ao programador para a utilização ou emulação dos co-processadores.

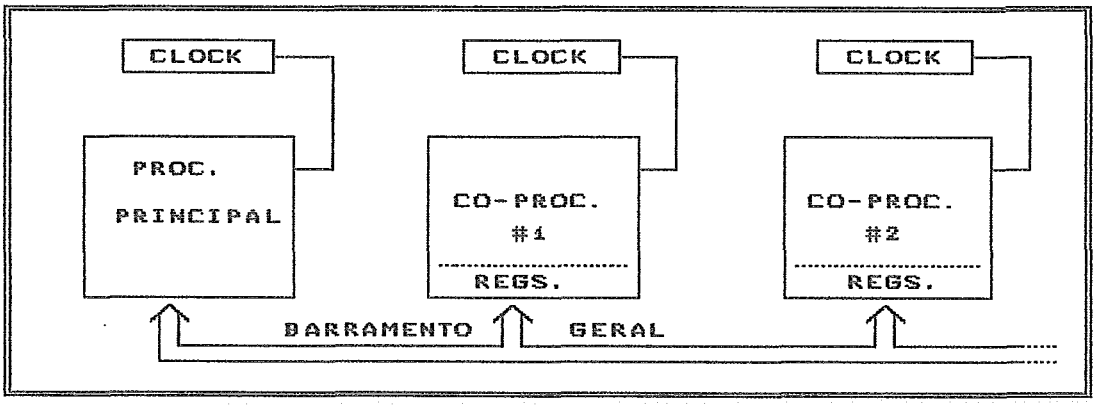


FIGURA III.8 - INSTRUÇÕES ESPECIAIS

Os microprocessadores da Motorola, por apresentarem diversos espaços de endereçamento, utilizam o espaço de endereçamento do processador (*cpu space*) e instruções com prefixo F_H para efetuar a interação com co-processadores. O

conjunto de registradores de cada co-processador é visto como uma extensão do conjunto de registradores do processador. Normalmente, os co-processadores apresentam registradores para o recebimento de comandos, registradores para leitura e escrita de operandos e registradores para leitura de *status*. A sincronização entre o processador e os co-processadores é feita através da leitura do registro de *status* que sinaliza a condição do co-processador. Esta operação mantém o barramento externo do sistema permanentemente em atividade, caracterizando uma desvantagem para os sistemas de processamento paralelo.

A Intel estabeleceu uma interface específica para comunicação com seu co-processador aritmético. A falta de generalidade da interface é a principal desvantagem desta arquitetura. Os co-processadores da Intel (80287 e 80387) só podem ser usados pelos seus processadores (80286 e 80386). Além disso apenas um co-processador pode ser ligado ao 80386. A comunicação entre os dois dispositivos é efetuada através de instruções com prefixo "ESC" (11011_B). Ao identificar este tipo de instrução o 80386 inicia un ciclo especial de entrada e saída em que endereça os registradores de seu co-processador. O co-processador pode operar a uma frequência diferente do processador porém, sua interface de barramento opera obrigatoriamente em sincronismo com o 80386, determinando que o co-processador receba o mesmo *clock* do microprocessador. Sua principal vantagem em relação à interface para co-processadores da Motorola é que não necessita efetuar acessos seguidos ao barramento enquanto aguarda a liberação do co-processador

para enviar um novo comando. Existem sinais especiais de *hardware* (*BUSY#*, *PEREQ* e *ERROR#*) que indicam o estado do co-processador. Por outro lado, a utilização de sinais de *hardware* para a solução deste problema determina uma limitação para o número de unidades co-processadoras. No caso do 80386, apenas uma unidade pode ser conectada caracterizando uma grande desvantagem em relação às arquiteturas dos outros microprocessadores.

A comunicação com co-processadores da National utiliza também uma interface especial em que os sinais de *status* do NS32032 (ST0-ST3) são utilizados para indicar um ciclo de co-processador. Ao receber uma instrução de co-processador em que está previamente codificada sua identificação, o NS32032 sinaliza em suas linhas de *status* e coloca a identificação do co-processador nas linhas de dados (AD0-AD7). A partir deste instante, somente o co-processador identificado deverá responder ao microprocessador que originou a chamada.

A leitura e escrita de operandos, *status* e comandos é efetuada através da codificação do tipo de operação nos sinais ST0-ST3. A interface entre os dois dispositivos não precisa ser síncrona. O sincronismo ocorre apenas durante os acessos do NS32032 e é controlada pelo sinal $\overline{SPC}$ que indica quando o co-processador está pronto para a comunicação. Quando o NS32032 deseja se comunicar com qualquer de seus co-processadores, envia sua identificação e aguarda a resposta através do sinal $\overline{SPC}$.

Do ponto de vista do programador, os co-processadores do NS32032 são vistos como periféricos que são acessados

por instruções especiais do microprocessador. A não existência de um co-processador determina a ocorrência de uma exceção interna quando este for solicitado pelo *software*. Esta exceção pode ser usada para a emulação das funções do co-processador através de rotinas de *software*.

As unidades para co-processamento desempenham um papel de grande importância em relação ao processamento paralelo. Através dos co-processadores aritméticos por exemplo, tem-se conseguido operar números com notação em ponto flutuante sem ocupar o processador principal. O tempo gasto pelo co-processador para efetuar uma operação deste tipo é sempre bem menor do que o tempo dispendido pelo processador principal para executar a mesma operação. Além disso, enquanto o co-processador trabalha, o processador principal pode executar outras instruções. Deste modo, os co-processadores permitem aumentar o desempenho dos sistemas de duas formas: a)por serem especializados, executam suas operações em tempo menor que o processador principal e b)permitem o paralelismo, liberando o processador principal para executar outras instruções.

Das formas de interface processador/co-processador apresentadas, a interface através de **Instruções Especiais** parece ser a que oferece mais vantagens ao processamento paralelo. Como neste tipo de interface o co-processador é visto como uma extensão do modelo geral de programação, sua utilização é transparente ao programador. Além disso, a definição das instruções responsáveis pela comunicação e comando dos co-processadores pode ser livremente estabelecida pelo projetista do sistema (dentro do conjunto

de possibilidades permitido pelo fabricante do microprocessador). Isto torna possível que, através de uma pequena mudança nos compiladores, todos os programas de um sistema possam utilizar as funções de um co-processador sem que seja necessária qualquer alteração nos programas. Este tipo de interface permite ainda que co-processadores de diferentes tipos e em número variável possam ser ligados ao processador principal, bastando apenas a definição do modelo de programação de cada um assim como suas instruções de comando. A velocidade de operação de cada co-processador pode ser independente da velocidade do processador principal.

A utilização de co-processadores, principalmente aqueles que apresentam interfaces através de Instruções Especiais parece a melhor forma para incrementar a capacidade para processamento paralelo nos sistemas que utilizam microprocessadores de 32 *bits*.
Através das interfaces de co-processadores podem ser incorporados ao processador principal, módulos para processamento vetorial e matricial além de módulos para comunicação entre unidades de processamento (comutadores ou canais de alta velocidade) ou qualquer outro elemento que possa colaborar para a realização de operações em paralelo.

Das interfaces para co-processadores apresentadas neste capítulo, a interface dos microprocessadores da Motorola é a que possui características mais adequadas à interligação com unidades de co-processamento que possam colaborar para o aumento do paralelismo como exposto acima. Estes microprocessadores suportam, no momento, até 8 unidades

co-processadoras de diferentes tipos. A definição do modelo de programação e do conjunto de instruções (inicidas por F_H) é responsabilidade do projetista da unidade co-processadora. Além disso, na ausência de uma unidade co-processadora solicitada por um aplicativo, suas funções podem ser emuladas pelo *software* através de uma exceção gerada pelo microprocessador. Deste modo, sua operação é completamente transparente ao usuário do sistema.

O microprocessador da Intel por outro lado é o que apresenta maiores restrições à utilização de unidades co-processadoras. Sua interface permite única e exclusivamente a interligação com seu co-processador aritmético (80287/387).

CAPÍTULO IV

MEMÓRIA

A organização do sub-sistema de memória apresenta importância fundamental para os sistemas de múltiplos processadores baseados em barramentos na medida em que constitui um dos principais meios de interligação das diversas partes de que é composto. Concordamos com FAKER[2] que considera os sub-sistemas de memória o meio mais eficiente para transferência de dados entre os processadores em sistemas com pequeno número de unidades de processamento.

Uma organização típica para sistemas de múltiplos processadores apresenta uma estrutura hierárquica com uma parte da memória local a cada processador e uma parte comum e compartilhada por todos os processadores. A memória local pode estar mapeada no espaço de endereçamento de todo o sistema ou apenas servir como *cache*. Em ambos os casos, seu objetivo principal consiste em permitir que o processador opere a sua máxima velocidade. Na medida em que os dados e instruções que o processador necessitar estiverem contidos na memória local, não será necessário o acesso à memória compartilhada. Evita-se assim a inserção de ciclos de espera (*wait*) para o processador. Por outro lado, a existência de memória local contribui para a diminuição dos acessos à memória compartilhada, deixando o barramento do sistema com menor tráfego. Consequentemente, será menor o

número médio de ciclos de espera para cada processador que necessitar acessar a memória compartilhada.

O uso de memória *cache* está fundamentado nas características de localidade espacial e temporal apresentadas pelos programas. A localidade espacial é uma característica dos programas sequenciais pela qual é altamente provável que a próxima instrução (ou dado) que o processador irá necessitar estará localizada junto às informações utilizadas num determinado instante. A localidade temporal refere-se a probabilidade de reutilização, num futuro próximo, das informações utilizadas num determinado instante. A utilização das memórias *cache* visa principalmente, acelerar as operações de busca de dados e instruções realizadas pelos processadores.

O suporte à **Memória Virtual** é outra característica importante e comum a maior parte dos microprocessadores de 32 *bits*. É através dos mecanismos de suporte à memória virtual que torna-se possível ao *software* utilizar todo o espaço de endereçamento oferecido pelo processador, sem se preocupar com o tamanho da memória física disponível.

A utilização de memória compartilhada nos sistemas de múltiplos processadores exige alguns mecanismos especiais para seu suporte. Esses mecanismos estão ligados principalmente, às estruturas de *software* relacionadas com a sincronização de processos e o compartilhamento de dados e código.

Neste capítulo apresentamos as características de cada um dos quatro microprocessadores relacionadas com a

organização de memória, destacando aquelas importantes ao processamento paralelo.

IV.1. Organização da Memória

A organização de memória nos microprocessadores da linha Motorola é feita através do endereçamento de *bytes*. Além disso, o posicionamento dos *bytes* na memória é feito de forma que os endereços mais baixos contenham os *bytes* de mais alta ordem, como pode ser observado pela figura **IV.1**. Estes microprocessadores não necessitam que os dados estejam alinhados em endereços pares. Porém, as transferências de dados mais eficientes ocorrem quando os dados estão alinhados no limite de endereçamento determinado pelo tamanho do operando (*byte*, *word* ou *long-word*).

	PAR		ÍMPAR	
$00	A	B		A
$02	C	D	B	C
$04			D	

VALOR ARMAZENADO : $ABCD onde **A** é o byte mais significativo e **D** o menos significativo.

FIGURA IV.1 - ALINHAMENTO DAS PALAVRAS

Os tipos de dados possíveis de operação na memória são: *bit*, campos de *bits*, inteiros de 8, 16 ou 32 *bits*, BCD e BCD compactado* . Todos estes tipos de dados podem ser

* A manipulação de **QUAD-WORD** é possível apenas nos registradores dos co-processadores.

acessados em qualquer endereço de *byte*.

Os microprocessadores MC68020 e 30 apresentam 32 *bits* em seus barramentos de dados e endereços, admitindo um espaço de endereçamento linear de até **4 Gigabytes**. Além do barramento de endereços, possui 3 *bits* codificadores de função (FC0,FC1 e FC2). Estes codificadores identificam o espaço de endereçamento e o tipo de referência (código ou dado) que o processador comanda durante cada ciclo de barramento. A tabela **IV.1** apresenta as codificações possíveis, de acordo com o espaço de endereçamento que o microprocessador pretende acessar.

TABELA IV.1

CODIFICAÇÃO	ESPAÇO DE ENDEREÇAMENTO
000	RESERVADO
001	DADOS DE USUÁRIO
010	TEXTO DE USUÁRIO
011	LIVRE P/ DEFINIÇÃO
100	RESERVADO
101	DADOS DO SUPERVISOR
110	TEXTO DO SUPERVISOR
111	PROCESSADOR

CODIFICAÇÃO DOS ESPAÇOS DE ENDEREÇAMENTO (MC68020 e 30)

Das oito possibilidades da tabela, cinco são pré-definidas pela Motorola e uma (011) pode ser definida pelos projetistas. O espaço de endereçamento do processador (*cpu space*) é sub-dividido em 16 tipos diferentes de acesso, identificados pelas linhas de endereço A16-A19. A tabela **IV.2** apresenta os tipos de acesso definidos pelo fabricante.

TABELA IV.2

A19-A16	TIPO DE ACESSO
0000	PONTO DE PARADA (ACK)
0001	M M U
0010	CO-PROCESSADOR
0011	RESERVADO
01xx	RESERVADO
10xx	RESERVADO
110x	RESERVADO
1110	RESERVADO
1111	INTERRUPÇÃO (ACK)

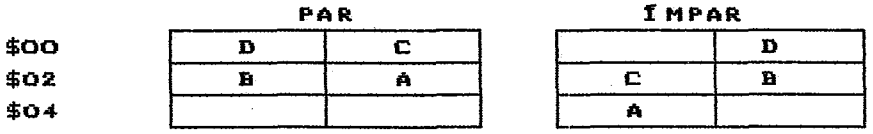
TIPOS DE ACESSO DO ESPAÇO DE ENDEREÇAMENTO DO PROCESSADOR

Este espaço de endereçamento foi criado para permitir a identificação de operações especiais como reconhecimento de interrupções, tratamento de pontos de parada assim como o interfaceamento com co-processadores e outros dispositivos especiais.

O conceito de espaço de endereçamento permite que lhe seja aplicado todo o conjunto de instruções do processador, assegurando um completo isolamento em relação aos outros espaços de endereçamento.

O microprocessador NS32032 apresenta 24 *bits* para endereçamento. Parte de seu barramento de dados (32 *bits*) é multiplexado com o barramento de endereços nas linhas AD0-AD23. Possui ainda um sinal para identificar seu modo de operação: **privilegiado** (ou **supervisor**) e **usuário**. Desta forma, é capaz de mapear dois espaços de endereçamento de até 16*Mbytes* cada um. Não estabelece nenhuma restrição quanto ao alinhamento dos dados. Qualquer dado, independentemente de seu tamanho, pode estar localizado em qualquer endereço de *byte*. Quatro sinais (BE0-BE3) podem ser usados para identificar a localização de um *byte* em

relação a uma *long-word*. Os dados são armazenados na memória de forma inversa aos processadores da Motorola. Aqui, os endereços mais baixos contêm os *bytes* de mais baixa ordem (figura **IV.2**).



VALOR ARMAZENADO : $ABCD onde **A** é o byte mais significativo e**D** o menos significativo.

FIGURA IV.2 - ALINHAMENTO DAS PALVRAS

As operações especiais como reconhecimento de interrupções e comunicação com co-processadores são identificadas através dos sinais ST0-ST3. Estes sinais codificam também informações a respeito do estado em que o microprocessador se encontra a cada ciclo.

O microprocessador Z80000 dispõe de 32 linhas de endereços (A0-A31) e é capaz de endereçar até 4 *Gigabytes* em cada um de seus 4 espaços de endereçamento: modo **privilegiado** para dados, modo **privilegiado** para texto, modo **normal** para dados e modo **normal** para texto. Existe ainda um quinto espaço de endereçamento dedicado às operações de **entrada** e **saída** em que são mapeados apenas 64 *Kbytes*.

A sua organização de memória é semelhante à do NS32032, permitindo acessos a *bytes* em qualquer endereço. As palavras de dados são armazenadas na memória com o *byte* mais significativo no endereço mais alto, como na figura **IV.2**.

O Z80000 apresenta três formas distintas de

endereçamento: modo **compacto**, **segmentado** ou **linear**. O modo compacto utiliza apenas endereços de 16 *bits* (compatível com o Z8000). Para aplicações que necessitem de uma área maior de endereçamento e apresentem seus elementos organizados de forma estruturada, o modo segmentado apresenta-se como o mais indicado. O terceiro modo permite um endereçamento linear compreendendo todo o espaço de endereçamento mapeado pelos seus 32 *bits*.

O 80386 não apresenta nenhum tipo de indicação para o *hardware* a respeito do espaço de endereçamento em uso pelo microprocessador. Sua única indicação refere-se às operações normais de acesso à memória ou operações de entrada e saída. Da mesma forma que os dois microprocessadores anteriormente descritos sua memória é organizada em *bytes* permitindo acesso a operandos do tipo *byte*, *word* e *double-word*. O endereço mais baixo contém o *byte* menos significativo e os sinais BE0-BE3 indicam o *byte* solicitado pelo microprocessador.

Os barramentos de dados e endereços são independentes e apresentam 32 linhas (D0-D31 e A0-A31). Assim, sua capacidade de endereçamento pode atingir até 4 *Gigabytes* de memória física. Seu espaço de endereçamento de entrada e saída pode mapear até 64 *Kbytes* (0000_H a $0FFFF_H$) para interfaceamento com periféricos e mais 16 *bytes* ($800000F8_H$ a $800000FF_H$) para comunicação com seu co-processador aritmético.

IV.2. Memória Cache

A *memória cache* é uma pequena memória, de alta velocidade, localizada entre o processador e a memória primária (normalmente várias vezes mais lenta). A memória *cache* mantém temporariamente, cópias do conteúdo de algumas posições da memória primária que estão sendo acessadas pelo processador em um determinado instante. Isto permite que o desempenho global do sistema seja aumentado, já que o processador pode ler as informações contidas na *cache* num período de tempo várias vezes menor do que gastaria para ler a mesma informação na memória primária. Sua utilização é em geral, transparente para o *software*, a menos do aumento de sua velocidade de execução. Os principais fatores determinantes para o custo de sua implementação são seu tamanho e sua organização.

As memórias *cache* há algum tempo vêm sendo empregadas em *main-frames*. Porém, o advento dos microprocessadores de 32 *bits* tem criado uma demanda crescente para sua utilização em máquinas de pequeno e médio portes baseadas nesses microprocessadores. Os motivos para sua utilização podem ser encontrados através da comparação entre as velocidades de processamento dos microprocessadores disponíveis e os tempos mínimos para acesso às memórias dinâmicas (DRAMs) atuais. Enquanto esses microprocessadores podem executar ciclos de acesso à memória em intervalos de tempo da ordem de 100ns, os tempos típicos para acesso às DRAMs encontra-se na faixa dos 200ns. Assim, a memória *cache* surge como uma solução para que se possa obter o

máximo desempenho desses microprocessadores, conservando-se o custo total dos sistemas em níveis razoáveis.

IV.2.1. Considerações sobre o Projeto de Memórias Cache

De acordo com SMITH[13], o princípio básico de operação da *memória cache* reside no fato de que o processador deverá precisar, no futuro próximo, de informações localizadas em regiões vizinhas, na memória primária (localidade temporal e espacial). O objetivo da utilização da memória *cache* consiste em assegurar que tais informações estejam disponíveis ao processador no menor intervalo de tempo possível.

O sucesso de um projeto de memória *cache* pode ser avaliado através de dois fatores:

- A probabilidade de se encontrar a informação desejada na memória *cache* ou taxa de acerto (*Hit rate*) e

- O tempo médio de acesso à *cache* no caso de acerto.

Além de tentar obter uma taxa de acerto o mais próximo possível do valor 1 (100%) e um tempo médio de acesso o menor possível, o projetista deve otimizar alguns outros fatores de importância secundária:

. Redução do tempo médio de acesso à memória principal

. Redução da taxa média de acesso à memória em sistemas de múltiplos processadores

. Eliminação de qualquer tipo de ciclo de espera do processador por interferência da *cache*, como nos casos de manutenção de consistência em sistemas de múltiplos processadores

IV.2.2. Dimensionamento

A correta especificação e dimensionamento de alguns dos parâmetros das memórias *cache* são essenciais para que seu melhor desempenho seja obtido. O tamanho da *cache*, comprimento de linha e o grau de associatividade são parâmetros importantes que serão discutidos adiante, juntamente com questões como: forma de endereçamento, atualização da memória primária, hierarquia e manutenção da consistência.

Uma **linha** (também chamada **bloco**) é a unidade básica para transferência de informação entre a *cache* e a memória principal. Seu comprimento está intimamente relacionado com a taxa de acerto. A medida que o tamanho da linha aumenta diminui a taxa de erro (*Miss rate*)*, uma vez que a cada

* (1-taxa de acerto)

miss uma grande quantidade de informações é trazida para a *cache*. Por outro lado, uma linha muito grande poderá tornar a probabilidade de utilização de uma informação recém trazida da memória primária, menor do que a probabilidade de re-utilização de uma informação que estava presente na *cache* e tenha sido substituida. Neste caso, poderá ocorrer uma redução na taxa de acerto.

De acordo com SMITH[13], o tamanho ótimo para a linha de uma *cache* depende do tamanho da *cache* e dos parâmetros de acesso (tempo de acesso, *overhead*, etc.). Normalmente, o tamanho de uma linha de *cache* situa-se entre 4 *bytes* para *caches* pequenas (32 *bytes*) e 128 *bytes* para grandes *caches* (da ordem de 128 *Kbytes*).

Assim como o tamanho da linha, o tamanho total da *cache* é muito importante para a redução da taxa da *miss*. Pode-se imaginar, em princípio, que quanto maior a *cache*, menor a taxa de erro. Porém, *caches* muitos grandes tendem a apresentar tempos de acesso maiores. Isto ocorre devido à capacitância associada a um número muito grande de células de memória que aumenta significativamente o tempo necessário para identificação dos níveis lógicos dos sinais elétricos. Outros fatores que determinam o tamanho da *cache* são: área disponível na placa de circuito impresso, custo (os *chips* usados para a memória *cache* chegam a custar **dez** vezes o valor dos *chips* da memória primária, apresentando capacidade de armazenamento bastante inferior), consumo elétrico e dissipação de calor.

IV.2.3. Hierarquia de Cache

Até bem pouco tempo atrás, uma memória *cache* ocupava apenas um nível em um sistema hierárquico de memória, como mostra a figura IV.3.

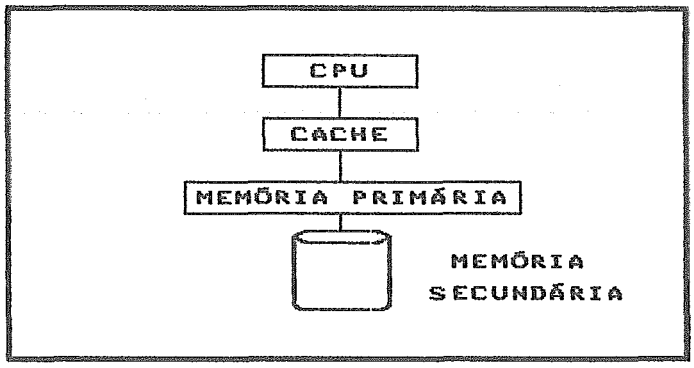


FIGURA IV.3 - HIERARQUIA DE MEMÓRIA

Porém, para *caches* muito grandes (e caras), já se considerava a possibilidade de se introduzir múltiplos níveis, como é apresentado por SMITH[14]. Com a utilização da tecnologia VLSI tais idéias foram concretizadas. Os microprocessadores de 32 *bits* apresentam, à sua maneira, pequenas *caches* associadas ao processador, no mesmo *chip*. Desta forma, os sistemas baseados nestes microprocessadores e que utilizem também *cache* externa (figura IV.4), conseguem obter excelente desempenho. Veremos mais adiante que alguns cuidados especiais devem ser tomados com relação à manutenção da consistência nos diversos níveis de *cache*, em sistemas com múltiplos processadores.

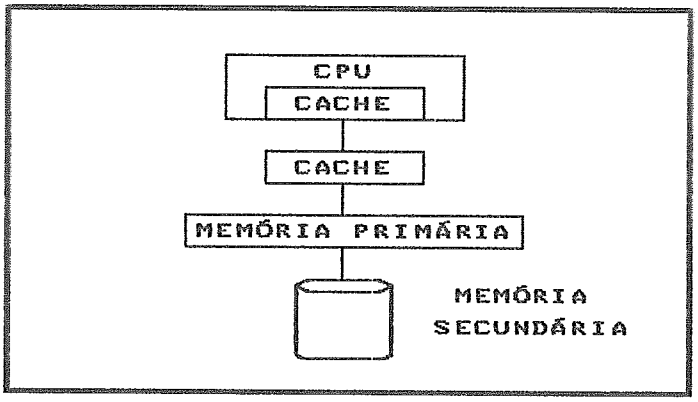


FIGURA IV.4 - HIERARQUIA DE MEMÓRIA

IV.2.4. Políticas de Busca e Substituição

Normalmente, a busca de blocos para a *cache* é realizada por demanda, isto é, um bloco só é trazido da memória primária para a memória *cache* quando o processador solicita uma informação e esta não é encontrada na *cache*. Existe uma forma alternativa para a busca de blocos em que estes são lidos da memória primária para a *cache* quando é possível prever seu uso futuro pelo processador. Esta busca antecipada (*pre-fetching*) pode ocorrer de várias formas. A busca antecipada **sequencial**, por exemplo, lê a linha seguinte à linha solicitada pelo processador na expectativa de que aquela informação possa ser solicitada em seguida.

Em funcionamento normal, a *cache* encontra-se completamente ocupada por linhas válidas. Quando uma informação solicitada pelo processador não é encontrada na *cache* deverá ocorrer não só uma busca da informação necessária na memória primária bem como a substituição de uma das linhas da *cache*.

Os algoritmos de substituição apresentados por SMITH[14] são divididos em dois grupos: algoritmos baseados na taxa de utilização dos blocos e os algoritmos **não** baseados na taxa de utilização dos blocos. Os algoritmos do primeiro grupo baseiam-se no registro dos acessos realizados a cada bloco para determinar o bloco que deverá ser substituído. Dois exemplos destes algoritmos, citados por SMITH[14] são: LRU e *Working Set*. Os algoritmos do segundo grupo baseiam-se em elementos relacionados com a forma de utilização dos blocos. FIFO e RAND (randômico e pseudo-randômico) estão neste grupo.

Se um algoritmo permite que o espaço de memória alocado para um processo específico seja variável, então este algoritmo é dito de espaço variável. Neste caso, uma busca poderá não implicar em substituição. Exemplos de algoritmos de espaço variável são: *Working Set* e *Page-Fault Frequency*.

Memórias *cache* entretanto apresentam tamanho fixo, e normalmente, são pequenas o bastante para não acomodarem mais do que um único processo. Desta forma, os algoritmos de espaço variável não são adequados a estas memórias.

O algoritmo que apresenta implementação mais simples (com relação ao *hardware*) e desempenho bastante aceitável, é baseado no mapeamento direto como apresentado na figura **IV.5.**

Neste algoritmo, uma *cache* com n linhas está mapeada m/n vezes na memória primária de m linhas. Existem então, m/n linhas da memória primária associadas a uma única linha da memória *cache*. Quando o processador solicita uma informação que está contida na linha k da memória primária

e esta informação não é encontrada na *cache*, a linha de índice MOD(k/n) da *cache* será substituida. Este algoritmo pode determinar a substituição consecutiva da mesma linha da *cache* se forem feitos, por exemplo, acessos às posições k e n*k na memória primária.

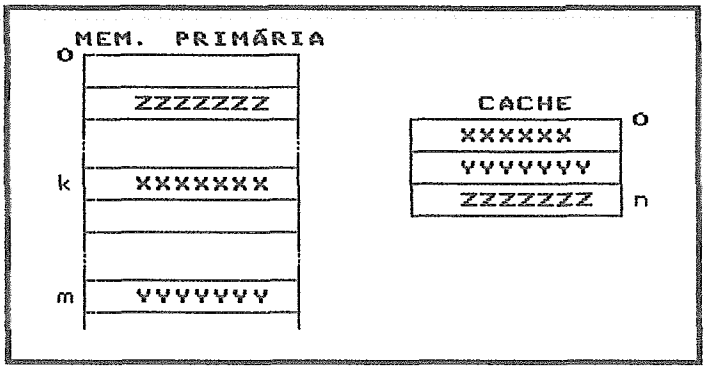


FIGURA IV.5 - MAPEAMENTO DIRETO

IV.2.5. Organização da Memória Cache

As memórias *cache* são normalmente organizadas através de mapeamento direto ou conjuntos associativos (*set associative*). A organização por conjuntos associativos consiste em dividir a memória associativa de comparação de endereços (*Tag*) em vários conjuntos de tamanho fixo. Parte do endereço fornecido pelo processador é utilizado para escolher o conjunto em que será feita a busca e comparação do resto do endereço. O grau de associatividade é um fator importante para o desempenho de uma *cache*. Quando existe apenas um conjunto associativo a *cache* é dita de **mapeamento direto**.

O aumento do grau de associatividade proporciona uma redução da taxa de *miss*. Mas, de acordo com SMITH[14], isto só ocorre até um determinado ponto. À medida que o grau de associatividade aumenta, observa-se um aumento linear de custos e circuitos mas com um ganho cada vez menor na taxa de acerto. Uma *cache* com grau de associatividade dois é relativamente melhor do que uma *cache* com mapeamento direto. Segundo SMITH[13], *caches* com grau de associatividade quatro ou mais apresentam desempenho ligeiramente superior (ou igual) à de grau dois. Tal fato pode ser observado pela tabela IV.3 apresentada por GROCHOWSKI[15] que contem também o tamanho da *cache* com parâmetro de alta importância para o desempenho. A observação cuidadosa desta tabela demonstra que o tamanho da linha é um fator mais importante para o desempenho da *cache* do que seu grau de associatividade.

TABELA IV.3

CONFIGURAÇÃO DA CACHE			PERFORMANCE	
TAMANHO	ASSOCIATIVIDADE	LINHA	ACERTOS	CACHE/DRAM
1K	DIRETA	4BYTES	41%	0.91
8K	DIRETA	4BYTES	73%	1.25
16K	DIRETA	4BYTES	81%	1.35
32K	DIRETA	4BYTES	86%	1.38
32K	2-WAY	4BYTES	87%	1.39
32K	DIRETA	8BYTES	91%	1.41
64K	DIRETA	4BYTES	88%	1.39
64K	2-WAY	4BYTES	89%	1.40
64K	4-WAY	4BYTES	89%	1.40
64K	DIRETA	8BYTES	92%	1.42
64K	2-WAY	8BYTES	93%	1.42
128K	DIRETA	4BYTES	89%	1.39
128K	2-WAY	4BYTES	89%	1.40
128K	DIRETA	8BYTES	93%	1.42

PERFORMANCE DE SISTEMAS DE CACHE
GROCHOWSKI[15]

IV.2.6. Forma de Endereçamento

Na maioria das memórias *cache* o endereço lógico (ou virtual) gerado pelo processador é traduzido para um endereço físico (ou real) antes da procura da informação na *cache*. Porém, pode-se usar diretamente o endereço lógico gerado pelo processador se alguns cuidados forem observados.

O uso do endereçamento virtual é vantajoso principalmente para os microprocessadores de alta velocidade que não dispõem de *cache* interna. Neste caso, o tempo necessário para a tradução do endereço é utilizado diretamente pela *cache* para a busca da informação, permitindo a operação do processador sem a inserção de ciclos de espera. Somente quando houver uma ausência (*miss*) na *cache*, é que será feita pela unidade de gerenciamento de memória a tradução do endereço e a busca da informação na memória primária.

O principal cuidado que deve ser tomado com memórias *cache* com endereçamento virtual refere-se às posições sinônimas. É possível, por exemplo, que dois endereços virtuais distintos sejam traduzidos para um mesmo endereço real. Neste caso, uma escrita em qualquer dos endereços poderá gerar um problema de inconsistência entre as duas posições que deveriam apresentar sempre o mesmo conteúdo. Esta situação pode ocorrer nos seguintes casos:

. Compartilhamento de páginas entre programas

. Atualização de dados na memória primária por dispositivos de entrada e saída

Este problema pode ser resolvido através do *hardware* com a utilização de uma **tabela de tradução reversa** (RTB) ou pela operação conjunta entre *hardware* e *software*. Neste caso, define-se uma região da memória primária em que a *cache* não é mapeada. Nesta região deverão ser alocadas todas as informações compartilháveis. Quando uma leitura ou escrita for realizada por qualquer processador ou dispositivo de entrada e saída nesta região, nenhuma *cache* responderá, garantindo o acesso à informação correta.

IV.2.7. Particionamento e Especialização da Memória Cache

O método mais comum de implementação de *cache* apresenta uma única memória para armazenar dados e instruções indistintamente. Compartilhamos com SMITH[13] a opinião de que pode-se conseguir algumas vantagens com a sub-divisão e especialização da memória *cache*. A separação de dados e instruções em *caches* diferentes tem como finalidade a redução da interferência entre esses dois tipos de informação. Assim, um *loop* de instruções que manipule um conjunto muito grande de dados poderá ser mantido intacto na sua *cache* enquanto os dados são substituidos na *cache* correspondente. A situação oposta, em que um conjunto reduzido de dados é manipulado por um código extenso, também alcança melhor desempenho com a especialização da *cache*.

A especialização pode ir além da separação da *cache* em dados e instruções. Os processadores que apresentam indicações relativas ao modo de operação (ou espaço de endereçamento) podem conseguir melhor desempenho com a divisão da *cache* em dados de supervisor, instruções de supervisor, dados de usuário e instruções de usuário.

É importante observar que a utilização desta filosofia de operação da *cache* bem como o dimensionamento de cada uma das divisões está intimamente relacionado com o propósito do sistema operacional em que será empregada.

IV.2.8. Problemas de Consistência

Os sistemas de múltiplos processadores que apresentam memórias *cache* têm que dispor de mecanismos para que uma informação, que poderá estar presente em várias *caches* num dado instante possa ser mantida consistente com suas diversas cópias.

Consideremos, por exemplo, um sistema com dois processadores com memória *cache* e a memória primária (fig.IV.6). O processador lê uma informação para a *cache* e a modifica. Um outro processador, igual ao primeiro, também lê a mesma informação para sua *cache* e a modifica. Mesmo que ambos os processadores estejam usando *write-through*, a modificação efetuada pelo segundo processador não será percebida pelo primeiro a menos que alguns cuidados sejam observados.

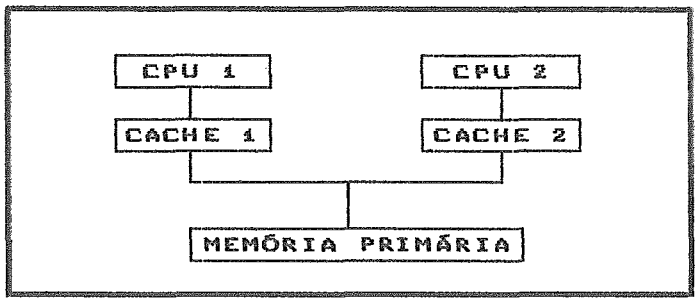


FIGURA IV.6 - CONSISTÊNCIA DAS CACHES

Possíveis soluções (SMITH[14]) que consideramos satisfatórias para a solução deste problema são:

. Cache compartilhada

Todos os processadores do sistema podem usar a mesma memória *cache*. Tal solução torna-se impraticável devido à *banda-passante* de uma única *cache* ser insuficiente para suportar a operação simultânea de vários processadores.

. Escritas em Broadcast

Cada vez que um processador executa uma escrita em sua *cache*, todas as outras *caches* efetuam uma operação de busca. Se for encontrada a mesma informação em qualquer das outras *caches*, a informação é atualizada ou simplesmente invalidada. De acordo com SMITH[14], a invalidação apresenta melhores resultados já que impede o **acesso cruzado** que pode ocorrer na atualização. A atualização tem como desvantagem ainda o aumento na circulação de informações no barramento do sistema.

O principal problema da escrita em *broadcast* é que cada memória *cache* do sistema é obrigada a ceder um ciclo para verificação e possível invalidação a cada escrita de qualquer processador. Segundo SMITH[13], a interferência provocada é, geralmente, aceitável para poucos processadores. Porém, pode-se atingir níveis inaceitáveis de degradação no desempenho, se um número elevado de processadores operarem desta forma.

. Controle pelo software

O *software* pode ser utilizado para garantir a consistência das informações se algumas condições forem obedecidas pelo *hardware*. Alguns elementos como **semáforos**, estruturas de dados compartilhados e filas podem ser alocados pelo *software* em regiões de memória ditas *non-cacheable*, isto é, não mapeadas pela *cache*. A modificação de informações compartilhadas é realizada através de **Regiões Críticas**, controladas por semáforos *non-cacheables*.

. Métodos de Diretórios

É possível manter-se um diretório (distribuído ou centralizado) de todas as linhas da memória primária e usá-lo para assegurar que nenhuma linha seja compartilhada para operações de escrita. Uma das formas para sua implementação é apresentada por SMITH[14]. Nela, considera-se que a memória primária dispõe de k+1 *bits* para

cada uma de suas linhas, onde k representa o número de *caches* do sistema. O *bit* i (i=1,...,k), é colocado em "1" se a *cache* correspondente à sua ordem contiver a linha. O *bit* k+1 é posto em "1" se a linha for modificada em qualquer das *caches*; do contrário será "0". Cada processador possui associada a cada linha de sua *cache* um único *bit* (chamado *bit* de propriedade). Se este *bit* estiver ativo ("1"), estará indicando que o processador correspondente possui a única cópia válida daquela linha. Se o *bit* estiver desligado ("0"), outras *caches* poderão conter cópias da mesma linha.

Um processador pode realizar várias operações neste sistema de diretórios. Se um processador tentar ler uma linha que não está em sua *cache*, o diretório da memória primária é consultado. Há duas respostas possíveis: todos os *bits* estão em "0"; neste caso a linha é transferida para a *cache* solicitante e o *bit* correspondente e posto em "1". Na outra alternativa, o *bit* k+1 estará em "1" indicando que a linha a ser transferida para a *cache* solicitante deverá ser obtida na *cache* cujo *bit* de propriedade estiver ativo. Durante esta operação a memória primária é atualizada e o *bit* k+1 volta ao estado inativo.

Uma tentativa de escrita por qualquer processador poderá resultar em uma das três alternativas a seguir. Se a linha já estiver presente na *cache* e já houver sido modificada anteriormente, o *bit* de propriedade já estará ativo e a escrita ocorrerá imediatamente. Se a linha estiver em qualquer outra *cache*, deverá ser invalidada (em todas as *caches* que estiver presente) e a memória primária

atualizada, se necessário. O diretório da memória primária é atualizado para refletir a nova condição em que apenas uma *cache* possui uma cópia válida da informação. Na terceira possibilidade, a *cache* poderá conter a linha mas o *bit* de propriedade poderá estar desligado. Nesta situação, deverá ser pedida uma permissão ao diretório da memória primária para que a escrita seja realizada. Todas as outras cópias da mesma linha no sistema serão invalidadas (a memória primária poderá não ser atualizada nesta situação).

Pode-se observar que este esquema apresenta alguns problemas de desempenho. A cada acesso de leitura com *miss* ou escrita em qualquer das *caches*, deverá ser efetuada uma consulta ao diretório da memória primária. Tal fato determina o gasto de tempo extra para realização da operação que pode implicar na ocorrência de *overrun* nos acessos de dispositivos de entrada de dados.

A tendência segundo SMITH[13], para a solução dos problemas de consistência segue dois caminhos de acordo com o porte do sistema. Sistemas com pequeno número de processadores operam, normalmente, com barramento compartilhado e o uso dos mecanismos de *hardware* para manutenção da consistência das informações é perfeitamente viável, mesmo levando-se em conta o *overhead* que geralmente introduz. Para os sistemas com número elevado de processadores, o controle através do *software* apresenta-se mais efetivo.

A manutenção da consistência da *cache* em sistemas com barramento compartilhado apresenta limitações, dificuldades e um alto grau de complexidade. Além disso, seu uso

generalizado só tem sido possível através da utilização de componentes com tecnologia VLSI. Sua principal limitação está relacionada com o número máximo de processadores permitidos no sistema pois, mesmo o melhor mecanismo de consistência utiliza o barramento comum para o tráfego entre a memória e as diversas *caches* determinando um limite para sua ocupação.

O controle através do *software* permite que o sistema operacional conheça as áreas da memória primária que são compartilhadas e saiba quando cada evento deverá ocorrer. O sistema operacional pode então, enviar comandos às *caches* dos diversos processadores para assegurar a consistência das informações, fazendo com que uma *cache* seja invalidada toda vez que uma outra *cache* que contiver a mesma informação a houver modificado. O problema que este mecanismo apresenta está ligado à sincronização de eventos e à queda no desempenho se a frequência das invalidações nas *caches* for muito elevada.

IV.2.9. Memórias Cache e os Microprocessadores de 32 bits

A maioria dos microprocessadores de 32 *bits* apresenta algum tipo de mecanismo para melhorar seu desempenho com relação ao tempo de acesso à memória primária. Geralmente são realizadas leituras em avanço (*pre-fetch*) para uma fila interna enquanto outras instruções são processadas. Alguns microprocessadores apresentam ainda pequenas memórias *cache* dentro do mesmo *chip* do processador.

Dos microprocessadores da Motorola, o MC68030

apresenta-se como o mais completo em relação ao suporte às memórias *cache*. Este microprocessador dispõe de duas *caches* internas, uma para código e outra para dados. O excelente desempenho obtido por este microprocessador é consequência de três elementos: ambas as *caches* apresentam tempos de acesso inferiores aos tempos gastos para os acessos externos (mesmo à *caches* externas), preenchimento das linhas através de leituras em bloco (*burst*); e a natureza independente de cada *cache* permitindo que a busca de instruções, dados e acessos externos possam ocorrer simultâneamente com a execução de instruções.

A *cache* de instruções do MC68030 possui 256 *bytes* com mapeamento direto. É organizada em 16 linhas contendo 4 *long-words* (16 *bytes*) por linha. Cada *long-word* é identificada independentemente das outras em cada linha, possibilitando 64 entradas em seu *Tag*. Uma entrada no *Tag* da *cache* de instruções é formada assim pelos 24 *bits* de endereço mais altos, pelo *bit* que identifica o espaço de endereçamento (supervisor/usuário) e quatro *bits* de validade que estão associados a cada uma das *long-words*.

A organização da *cache* de dados é semelhante à *cache* de instruções. O *Tag* porém, é composto pelos 24 *bits* de endereço mais altos, quatro *bits* de validade e todos os *bits* codificadores de função(FC0,FC1 e FC2), indicando explicitamente o espaço de endereçamento associado a cada linha. A política de *write-through* é usada para a escrita de dados, mas sem alocação de espaço no caso de *miss*. A manutenção da consistência dos dados em sistemas de múltiplos processadores pode ser efetuada de duas formas:

áreas *non-cacheables* e controle por *software*. A utilização de áreas *non-cacheables* é um modo simples e transparente ao sistema operacional para assegurar a consistência dos dados de cada *cache*. Quando um processador qualquer no sistema realiza um acesso a qualquer destas áreas, o *hardware* aciona o sinal **CIIN** (*Cache Inhibit*) que impede o funcionamento da *cache* durante aquele ciclo.

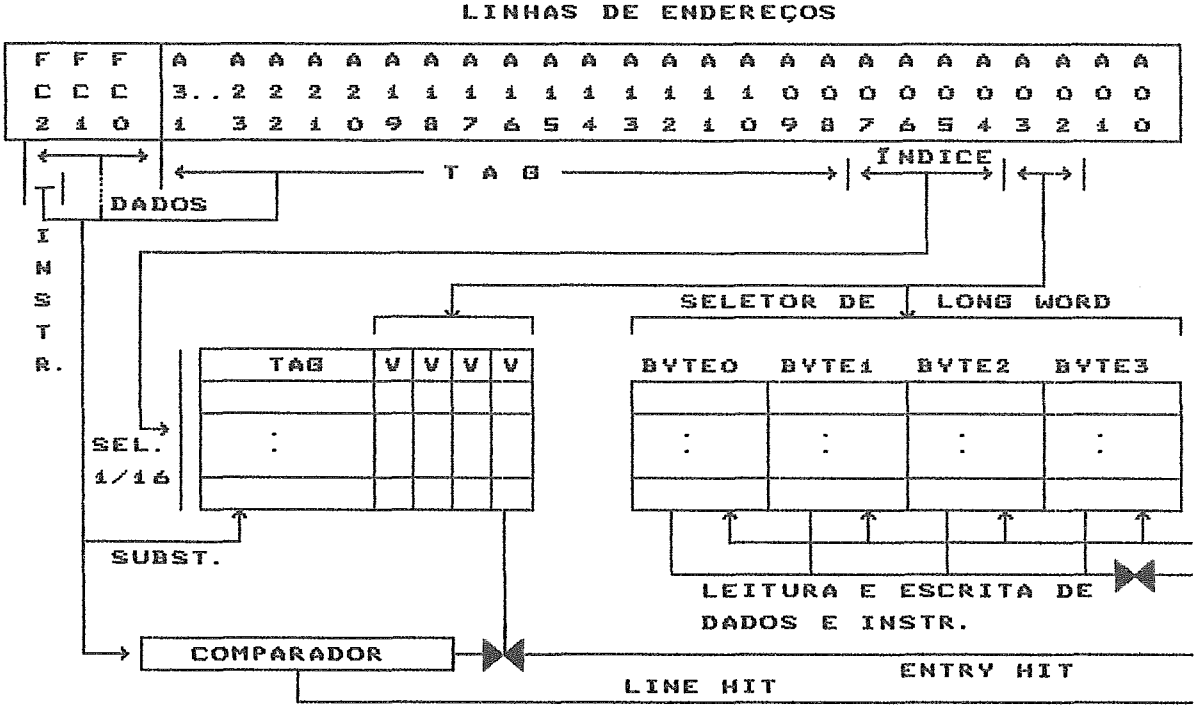


FIGURA IV.7 - ARQUITETURA GERAL DAS CACHES DE DADOS E INSTRUÇÕES DO MC68030

O controle através do *software* apresenta algumas dificuldades na medida em que os acessos às informações compartilhadas devem ser registrados. A cada atualização que venha a ocorrer no conteúdo de qualquer das *caches* implicará na invalidação da palavra modificada, em todas as outras *caches*. Este tipo de operação acarreta um *overhead*

indesejável para o sistema.

Na linha de microprocessadores da National, o NS32032 não apresenta suporte interno às memórias *cache*. Entretanto, foi lançado recentemente o microprocessador NS32532 que apresenta uma *cache* interna de 512 *bytes* para instruções e outra com 1*Kbytes* para dados. Possui ainda circuito para manutenção de coerência dos dados em tempo real, dispensando intervenções de *software*.

O microprocessador Z80000 possui uma *cache* de 256 *bytes* inteiramente associativa e organizada em 16 blocos (ou linhas) com 8 *words*. As transferências da memória primária para a *cache* são feitas em blocos inteiros (*burst*). Pode ser configurada para manter somente instruções, dados ou ambos. A cada bloco está associado um *Tag* (fig.IV.8) que contém os 28 *bits* mais altos de endereço e seu *bit* de validade. Além disso, existe um campo de 8 *bits* em que cada *bit* indica a validade de cada uma das *words* do bloco.

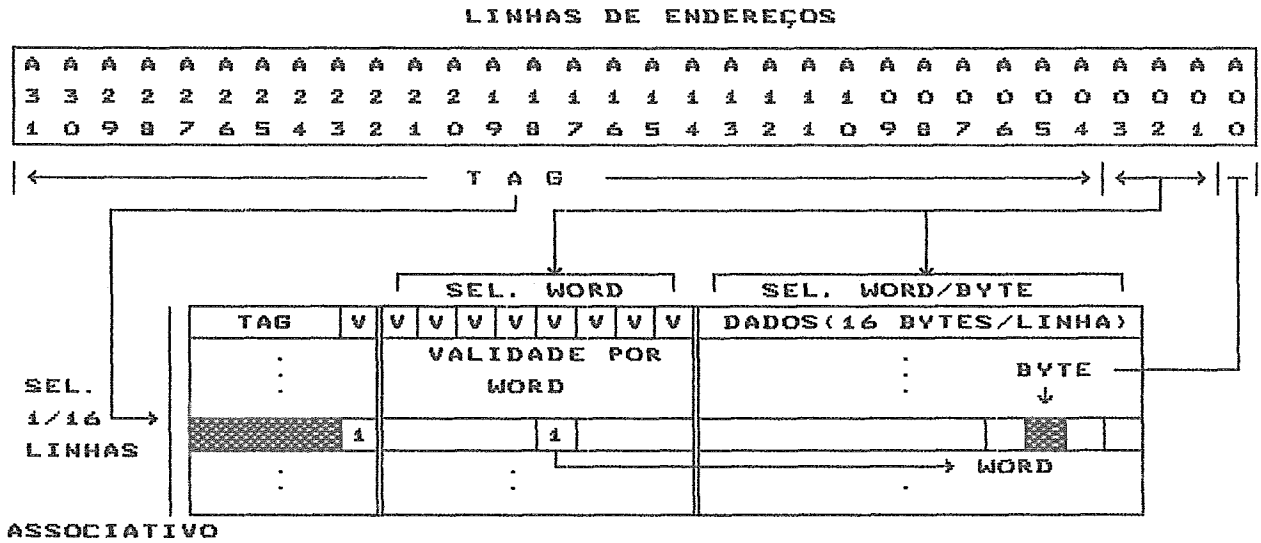


FIGURA IV.8 - ARQUITETURA DA CACHE DO Z80000

A substituição de blocos na *cache* é feita através do algoritmo LRU em que o bloco menos usado é substituído pelo novo bloco recebido da memória primária. A escrita utiliza a política de *write-through*, atualizando também a informação na *cache* se esta lá estiver presente.

O Z80000 dispõe também de um *pipeline* de seis estágios (fig.IV.9) que permite, sob certas circunstâncias, aumentar o desempenho do processador em um fator de aproximadamente seis, de acordo com PHILIPS[16].

Para instruções simples, todos os estágios do *pipeline* (exceto armazenamento de operandos) são completados em apenas um ciclo. Assim, segundo PATEL[17], operando a 25MHz, a taxa máxima de execução de instruções pode atingir 12,5Mips. Na prática porém, a taxa real de execução é da ordem de 1/3 da taxa máxima; dependendo da configuração do sistema e da composição das instruções.

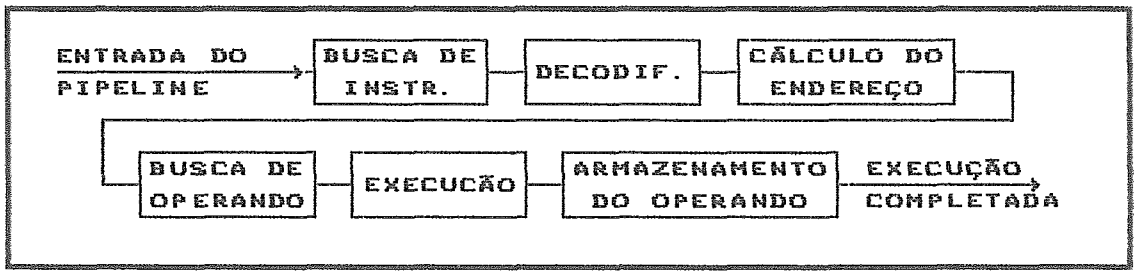


FIGURA IV.9 - PIPELINE DE INSTRUÇÕES DO Z80000

O 80386 da Intel, não apresenta nenhum tipo de suporte interno para memórias *cache*. Este fabricante optou por desenvolver uma unidade controladora externa. Esta unidade, 80385, é capaz de interligar-se diretamente com o microprocessador. Sua principal função é controlar

completamente os acessos da *cache*, acessos do tipo *non-cacheables* e acessos a dispositivos locais em que a *cache* não deva interferir. Realiza ainda o monitoramento do barramento do sistema a fim de identificar ciclos de escrita na memória primária em que são modificadas as informações presentes na *cache*. Neste caso procede à invalidação da linha correspondente no diretório da *cache*. Para a escrita é utilizada a técnica conhecida como *posted write-through*. Esta técnica assegura a escrita na memória primária e na *cache* (somente no caso de *hit*), liberando o processador para operação local, uma vez que tanto dados como endereços são armazenados enquanto o acesso à memória primária estiver em andamento.

O 80385 pode operar por mapeamento direto ou com conjunto associativo de grau dois. Neste último modo, apresenta dois diretórios com 512 entradas. Cada entrada está associada a uma linha que contém 8 *Double-words*. Possui ainda um *Tag* que contém os dezoito *bits* mais altos de endereços, um *bit* de validade para esta linha no *Tag* e oito *bits* de validade associados a cada uma das 8 *Double-words* que compõem um bloco.

A política de substituição de linhas adotada é a LRU, utilizando também um *bit* extra no diretório que é usado para indicar qual o último conjunto (*set*) atualizado.

Da mesma forma que o Z80000, o 80386 apresenta um *pipeline* de instruções de seis estágios que permite a execução paralela de instruções.

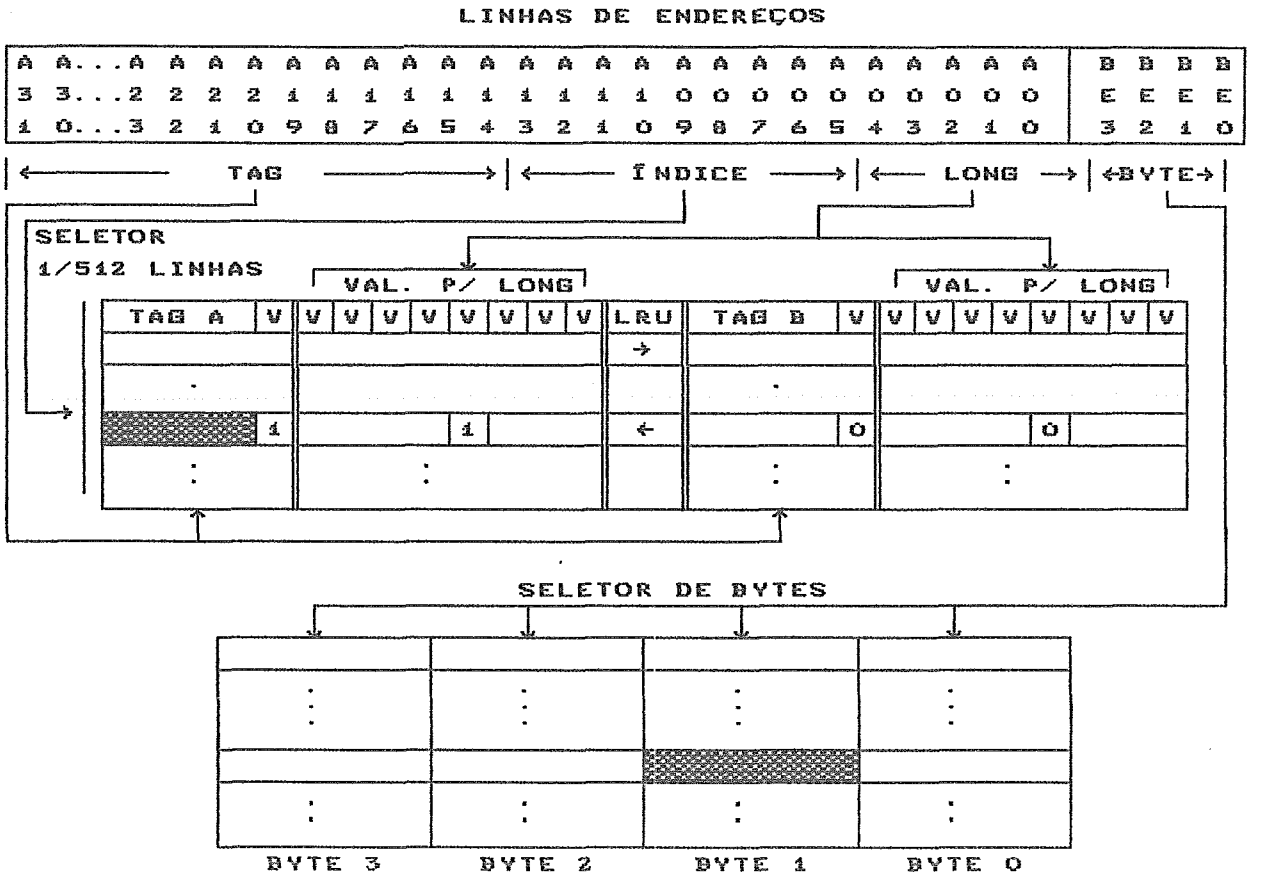


FIGURA IV.10 - DIRETÓRIO DO CONTROLADOR DE CACHE DO 80385

IV.3. Conclusões

Como vimos, o sub-sistema de memória apresenta importância fundamental para os sistemas de processamento paralelo baseados em barramentos. Nos sistemas não baseados em barramentos, o sub-sistema de memória contribui indiretamente para a eficiência do processador a nível local, na medida que as operações entre processador e memória ocorrem sem nenhuma interferência.

Os sistemas de processamento paralelo baseados em

barramentos conseguem melhorar sua eficiência tirando partido da organização hierárquica da memória. Neste aspecto, as memórias *cache* desempenham um papel importantíssimo assegurando uma redução significativa no tráfego de informações entre a memória primária e os processadores. O tamanho da *cache* e o tamanho do bloco devem ser criteriosamente dimensionados em função de características como: velocidade do processador, *banda-passante* do barramento e número máximo de unidades de processamento que o sistema deverá suportar. Em casos extremos, o tempo para transferência de informações entre a memória primária e a *cache* pode atingir um nível de interferência indesejável, tornando o desempenho global do sistema menor do que o obtido com a operação dos processadores sem a *cache*.

Os microprocessadores da Motorola e Zilog demonstram preocupação com processamento paralelo no que diz respeito à organização de memória. Ainda que pequenas, a existência de *caches* nestes microprocessadores permite que possam ser obtidos ganhos reais em desempenho através da operação paralela de algumas unidades de processamento. Podemos constatar por outro lado, a existência de uma grande deficiência com relação à manutenção da consistência de informações nas *caches*, introduzindo dificuldades adicionais para sua utilização em sistemas paralelos. É possível utilizar o controle por *software* para resolver os problemas de consistência, uma vez que podem ser estabelecidas regiões *non-cacheables* na memória primária. Sabemos entretanto, que tal solução é eficiente apenas para

um pequeno número de processadores.

De fato, a utilização de barramento compartilhado não permite um número maior do que algumas dezenas de processadores operando em paralelo. Mas, para atingir este número, o suporte atualmente oferecido não é suficiente. Estruturas de *hardware* para monitoramento do barramento e invalidação das palavras de *cache* (como a oferecida pelo controlador externo 80385 da Intel) permitem aumentar o número de processadores no sistema sem no entanto atingir o limite determinado pelo barramento. Parece-nos que este limite só pode ser atingido com a utilização de métodos de diretórios. Alguns fabricantes de sistemas como a Sequent implementam este método (linha Balance - NS32032) com sucesso em sistemas com até 30 unidades de processamento (MOKHOFF[10]).

CAPÍTULO V

GERENCIAMENTO DE MEMÓRIA

O aumento do poder computacional dos microprocessadores tem propiciado a aplicação de técnicas e soluções que foram desenvolvidas inicialmente apenas para sistemas de maior porte. A capacidade de operação em multitarefa dos sistemas baseados nos microprocessadores de 32 *bits*, determina a existência de algum tipo de mecanismo para gerenciamento e proteção da memória primária do sistema. Esses mecanismos vêm tornando-se mais e mais complexos com a evolução dos sistemas; sendo capazes de implementar hoje técnicas como **Memória Virtual** e vários níveis de proteção e segurança para o *software*, dentro do próprio *chip* da unidade central de processamento.

Neste capítulo procuramos apresentar as principais características das unidades de gerenciamento de memória para sistemas multi-microprocessados, destacando o suporte oferecido por cada um dos microprocessadores analisados neste trabalho.

V.1. Principais Características Necessárias às Unidades de Gerenciamento de Memória

As arquiteturas dos microprocessadores avançados capazes de suportar Memória Virtual utilizam sistemas de memória organizados hierarquicamente (figura **V.1**).

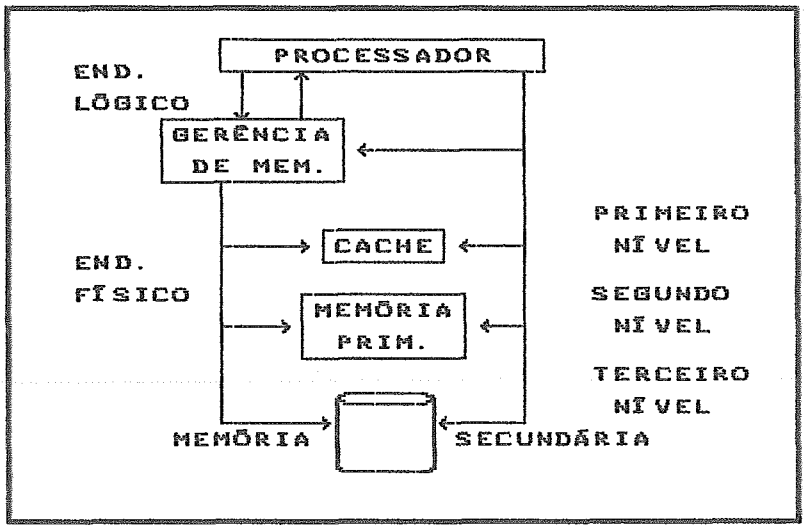


FIGURA V.1 - HIERARQUIA DE MEMÓRIA

Normalmente, um sistema de memória pode apresentar até 3 níveis, envolvendo a manutenção e controle de um grande espaço de endereçamento. Os dispositivos de memória em cada um dos níveis diferem entre si de acordo com o custo, capacidade de armazenamento e velocidade.

No primeiro nível estão as memórias *cache* de alta velocidade, custo elevado e pequena capacidade. No segundo nível está a memória primária, mais lenta porém mais barata e com capacidade de armazenamento de cerca de 2 ordens de grandeza superior às memórias *cache*. O terceiro e último nível contém os dispositivos de armazenamento de alta capacidade, que mantêm os programas e dados que não podem estar completa ou simultâneamente nos níveis anteriores devido a restrições de espaço. Quando um processo precisa ser executado, uma parte do conteúdo da memória do terceiro nível é trazido para o segundo nível que, por sua vez, passa parte de seu conteúdo para o primeiro nível. A

distribuição segundo a especialização dos dispositivos de memória determina uma hieraquia no sistema.

Para suportar uma estrutura hierárquica de memória seu sistema de gerenciamento deve possuir as seguintes características:

. Capacidade para traduzir endereços e suportar alocação dinâmica de memória,

. Oferecer suporte a Memória Virtual e

. Oferecer mecanismos para Segurança e Proteção da memória

As estratégias básicas (FURHTC18]) para a criação de arquiteturas para gerenciamento de memória em sistemas baseados em microprocessadores são as seguintes:

. A unidade de gerenciamento de memória está integrada ao *chip* do processador

. A unidade de gerenciamento de memória está fora do *chip* do processador

A melhoria no desempenho obtida como consequência do reduzido tempo de tradução dos endereços é uma das principais vantagens da unidade de gerência integrada ao *chip* do processador.

A forma de organização da memória é um aspecto de

grande importância para a construção de uma unidade de gerenciamento de memória. Existem dois tipos básicos de organização de memória: **linear** e **segmentado**. No esquema de endereçamento linear, os endereços começam a partir da posição zero e prosseguem linearmente. Nos esquemas de endereçamento segmentados, os programas não são escritos como uma sequência linear de instruções e dados mas em módulos. Geralmente, existem módulos para código, dados e pilhas (*stacks*); podendo existir diversos tipos de módulos de dados (Global, Local, etc.). O espaço lógico de endereçamento é subdividido em vários espaços de endereçamento lineares, cada um com tamanho variável definido pelo programador ou compilador.

A determinação de um endereço lógico efetivo é realizada através da combinação do número do segmento (que é um apontador para um bloco de memória) e um deslocamento dentro deste segmento.

O esquema de endereçamento linear é, em geral, mais adequado para a manipulação de extensas estruturas de dados. Por outro lado, a estruturação do *software* em módulos é conseguida através do sistema segmentado, facilitando a programação.

V.2. Formas de Tradução de Endereços

O mapeamento de endereços lógicos em endereços físicos é realizado pelos circuitos de tradução de endereços. Normalmente, a memória é dividida em blocos de tamanho fixo ou variável, dependendo do esquema de tradução escolhido

pelo projetista. O endereço de uma palavra de memória é obtido através da soma do endereço do início de um bloco (endereço base) com um valor que representa a posição da palavra de memória desejada em relação ao início do bloco (deslocamento relativo).

A tradução de endereços é realizada por meio da associação de um endereço base de um bloco lógico a um endereço base de um bloco físico. A parte do endereço lógico que representa o deslocamento não sofre tradução uma vez que é um valor relativo. A organização da memória em blocos permite que os mecanismos de tradução de endereços possam operar com um número de *bits* significativamente menor do que o fariam se precisassem associar cada endereço lógico completo a um endereço físico.

As três formas básicas para tradução de endereços são:

1- Paginação

2- Segmentação

3- Combinação de Segmentação e Paginação

Nos sistemas paginados os blocos de memória são chamados de **páginas** e apresentam tamanho fixo (*page frames*). Nos sistemas segmentados são chamados **segmentos** e possuem tamanho variável.

A subdivisão do espaço lógico de endereçamento do programa em páginas é feita pelo sistema, tornando-se totalmente transparente ao programador. As páginas

apresentam geralmente, um tamanho pequeno quando comparadas a área total de memória de um sistema. O tamanho típico de uma página situa-se entre 256 *bytes* e 4*Kbytes* de memória enquanto segmentos podem ocupar qualquer área.

O esquema misto combina as características dos esquemas de paginação e segmentação. A parte segmentada do esquema gerencia o espaço virtual dividindo os programas em segmentos enquanto a parte de paginação é responsável pelo gerenciamento da memória física. Neste esquema, cada segmento é formado por um número variável de páginas como pode ser observado na figura **V.2.**

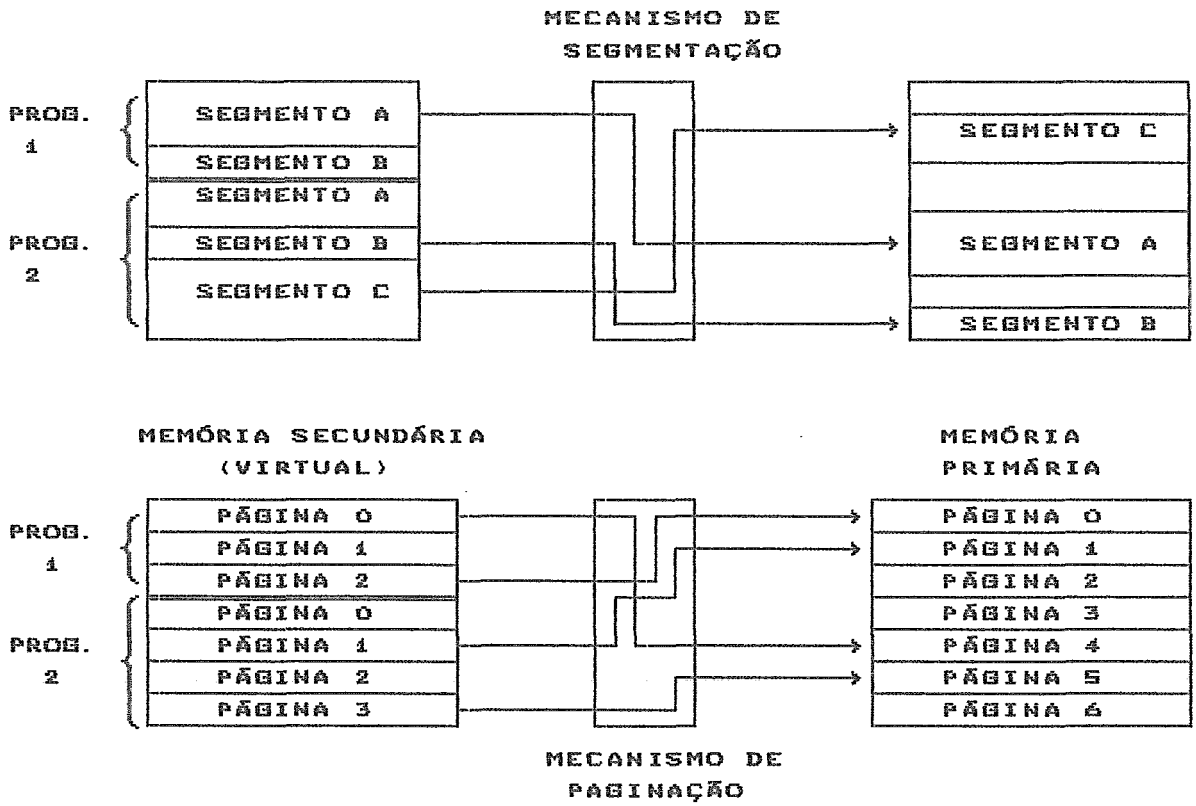


FIGURA V.2 - MECANISMOS DE MAPEAMENTO DE MEMÓRIA

O mecanismo de tradução de endereços escolhido para um sistema operacional apresenta uma grande influência no

desempenho global do sistema. Essa influência pode ser verificada principalmente através das técnicas de gerenciamento de memória responsáveis pela busca, alocação e substituição de páginas. Os sistemas segmentados por exemplo, utilizam algoritmos mais complexos do que os sistemas paginados para a alocação e substituição de segmentos a fim de manter reduzido o grau de fragmentação da memória primária.

Devido aos problemas de fragmentação da memória primária apresentados pelos sistemas segmentados, os sistemas paginados apresentam-se mais eficientes. Todas as páginas possuem o mesmo tamanho e podem assim ser substituidas sem deixar fragmentos de memória inutilizáveis.

Concordamos com FURHT[18] que o problema da fragmentação não parece tão crítico para os computadores atuais em que o espaço físico da memória primária tem atingido facilmente os 16 *Mbytes*. Além disso, a aplicação da técnica de substituição de parte de um segmento por demanda proporciona uma redução significativa dos problemas de fragmentação.

Dois aspectos devem ser observados ainda para a escolha do esquema de tradução de endereços:

- Forma de implementação do mecanismo de tradução

- Seleção do número de níveis de mapeamento

Existem duas técnicas para implementação dos mecanismos de tradução de endereços:

1- Tradução por Tabelas

2- Tradução por Registrador

. Tradução por Tabelas

Os sistemas paginados reservam algumas páginas na memória primária onde é colocado o mapa de tabelas de tradução. Nos sistemas segmentados existe o segmento do mapa de tabelas. As entradas dessas tabelas possuem informações que permitem a tradução dos endereços lógicos em físicos e informações adicionais para proteção e suporte aos algoritmos de alocação e substituição de páginas ou segmentos. No esquema paginado, o método de **Mapeamento Direto** consiste na divisão do endereço gerado pelo processador em dois campos. O campo que contém os *bits* de endereçamento mais altos é usado como índice para a tabela de tradução de endereços. A entrada na tabela referenciada pelo campo de índice contém parte do endereço físico que indica o endereço base de uma página da memória. O outro campo contém o deslocamento relativo.

No esquema segmentado, são utilizados dois apontadores para se obter o endereço virtual e afetuar sua tradução para o endereço físico. Estes dois apontadores são o Seletor de Segmentos e o seu Descritor. O Seletor de Segmentos aponta para uma entrada no segmento da tabela de

tradução que contém o Descritor do segmento referenciado. Este descritor contém o endereço base na memória primária do segmento referenciado. O endereço físico é obtido através da soma deste endereço com o valor fornecido pelo processador que representa o deslocamento relativo.

Os mecanismos de tradução de endereços precisam buscar informações em tabelas que geralmente se encontram na memória primária. Deste modo, para que a tradução de um endereço lógico seja efetuada torna-se necessário um acesso extra à memória para que o endereço base da página ou segmento referenciado seja obtido. Tal fato determina uma redução na eficiência do processador uma vez que para cada acesso efetivo feito à memória são realizados dois acessos. Existem várias soluções para se contornar este problema. Normalmente, uma pequena memória *cache* associativa é incorporada à unidade de gerenciamento de memória. Nesta *cache* permanecem os n últimos descritores acessados. Assim, após um descritor ter sido trazido para a *cache* não será mais necessário o acesso extra à memória para que o endereço físico deste segmento possa ser obtido.

Esta *cache* associativa é geralmente chamada de **TLB** (*Translation Lookaside Buffer*). Quando o mecanismo de tradução de endereços recebe um endereço lógico, cada entrada da TLB é comparada simultaneamente e um acesso de busca de descritor é realizado somente se o descritor necessário não estiver presente. Para substituição de entradas na TLB utiliza-se geralmente o algoritmo LRU (Least Recently Used).

. **Tradução por Registrador**

Nesta técnica a tradução de endereços é feita através de alguns registradores endereçados pelo *hardware*. Os registradores contêm parte do endereço físico que é associado a um campo do endereço lógico do processador da mesma forma descrita no esquema de endereçamento por tabelas. Sua vantagem em relação ao esquema de tabelas reside no fato de que não são necessários acessos à memória primária para que a tradução de endereços ocorra. Os registradores de tradução são *carregados* pelo sistema operacional a cada troca de processo. Normalmente, a tradução por registradores é utilizada nos sistemas segmentados. Nestes sistemas, o número de segmentos geralmente é pequeno (não chega a atingir uma dezena), permitindo que os registradores sejam incorporados ao processador sem grandes problemas. Já nos sistemas paginados, o elevado número de registradores necessários inviabiliza sua incorporação ao processador.

Os esquemas de tradução por tabelas apresentam por outro lado, vantagens com relação ao tempo gasto para trocas de processos, principalmente nos sistemas paginados. Enquanto o esquema de tradução por registradores determina a atualização de todos os registradores a cada troca de processo, no esquema de tradução por tabelas torna-se necessária apenas a substituição do apontador da tabela de tradução (previamente preparada pelo sistema operacional) usada para cada processo.

V.3. Níveis de Mapeamento

O mapeamento de endereços é realizado geralmente em apenas um único nível. Os microprocessadores de 32 *bits* e os *chips* de suporte ao gerenciamento de memória entretanto, têm apresentado possibilidades de utilização de múltiplos níveis. Nestes esquemas são empregadas várias tabelas. A tabela do último nível é a única que contém o endereço físico associado ao endereço lógico. Os níveis mais altos possuem apenas apontadores para entradas nas tabelas de níveis posteriores. A organização das informações por meio de tabelas de múltiplos níveis permite que todo o esquema de mapeamento de memória possa ser modificado com a mudança de uma única entrada nas tabelas de níveis superiores, facilitando o trabalho dos sistemas operacionais.

As principais vantagens dos esquemas de mapeamento em múltiplos níveis sobre o esquema tradicional são:

- Permitem mecanismos de proteção mais sofisticados,
- São capazes de acomodar grandes espaços de endereçamento e
- Permitem o compartilhamento de páginas ou segmentos.

Dos esquemas de tradução de endereços apresentados, a segmentação nos parece a menos adequada aos sistemas de

múltiplos processadores baseados em barramento e memória compartilhados. Nestes sistemas, a fragmentação de memória causada pela segmentação torna-se particularmente crítica. A movimentação de dados necessária à organização dos espaços de memória pode provocar um redução substancial no desempenho desses sistemas se comparado aos sistemas paginados. Por outro lado, as facilidades que a organização do *software* em segmentos proporciona não podem ser deixadas de lado. Assim, podemos concluir que o sistema misto é mais adequado aos sistemas paralelos uma vez que conserva a modularidade dos esquemas segmentados, mantendo as características do esquema paginado para a organização dos espaços de memória.

Com relação a forma de implementação, a tradução por tabelas de múltiplos níveis apresenta inúmeras vantagens para os sistemas de múltiplos processadores. Uma vez criada a árvore de tabelas, o sistema operacional e todos os processadores podem ter uma visão completa da organização da memória do sistema. Isto facilita o compartilhamento de informações entre processadores que podem usar as mesmas tabelas a partir de um nível qualquer. Outro aspecto importante está relacionado com a proteção e segurança das informações. Uma única árvore de tabelas que pode ser acessada por todos os processadores permite manter um grau de coerência maior no gerenciamento das informações.

V.4. Tradução de Endereços nos Microprocessadores de 32 Bits

Dos microprocessadores fabricados pela Motorola, apenas o MC68030 possui unidade interna para gerenciamento de memória. Este microprocessador utiliza o esquema de paginação em seu mecanismo de tradução de endereços. A memória é dividida inicialmente em 8 espaços de endereçamento que são subdivididos em páginas de igual tamanho. O tamanho da página pode ser escolhido pelo programador do sistema operacional, permitindo valores entre 256 *bytes* e 32 *Kbytes*. A unidade é capaz de gerenciar até 4 *Gigabytes* de memória.

A tradução do endereço lógico para físico é realizada através de uma memória *cache* associativa que contém 22 entradas e armazena os últimos descritores de página referenciados. Esta *cache* é composta por uma memória endereçável por conteúdo (CAM) que mantém o endereço lógico e os codificadores de função para comparação. Possui ainda uma memória para os endereços físicos em que cada posição está associada a uma entrada da CAM, além de uma seção de controle que contém o algoritmo de substituição de entradas que implementa uma variação do LRU.

Os descritores de páginas armazenados na *cache* são obtidos nas tabelas de tradução de endereços. Estas tabelas são montadas em forma de árvore na memória primária. A *raiz* da árvore é apontada por um dos dois registradores internos do processador (*CPU Root Pointer* e *Supervisor Root Pointer*). As entradas da tabela nos níveis mais altos da

árvore (figura v.3) podem conter apontadores para outras tabelas. As entradas do último nível contêm os descritores de páginas.

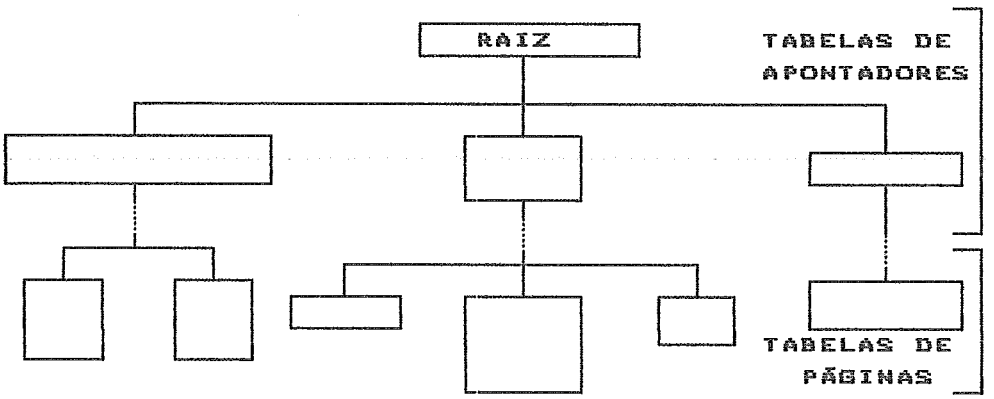
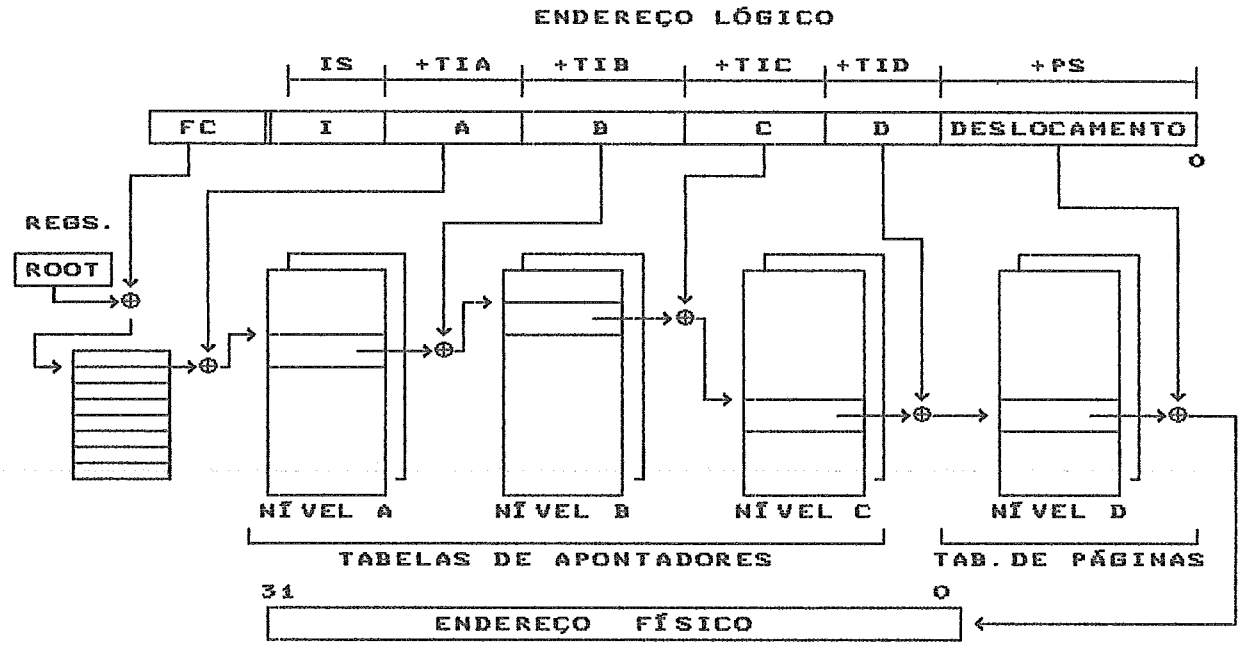


FIGURA V.3 - ÁRVORE DE TRADUÇÃO DE ENDEREÇOS (MC68030)

O número total de níveis da tabela pode ser escolhido entre um e quatro. O número de níveis necessários à tradução de um endereço pode ser variável. Isto é, um endereço x qualquer poderá por exemplo, usar um número de tabelas maior do que um endereço y. Podem ser utilizados até 15 *bits* do endereço lógico como **índice** em cada nível. A maior vantagem deste tipo de estrutura para a tabela de páginas está na capacidade de desalocar grandes espaços de memória do espaço de endereçamento lógico, modificando-se apenas uma entrada nos níveis mais altos da tabela. Além disso, parte desta tabela poderá estar armazenada também na memória secundária, recebendo tratamento semelhante à uma página de dados quando sua falta for detectada.



OBS: OS CAMPOS I,A,B,C e D POSSUEM TAMANHO VARIÁVEL.

FIGURA V.4 - MECANISMO DE TRADUÇÃO DE ENDEREÇOS (MC68030)

As entradas desta tabela contêm informações relativas aos apontadores para os próximos níveis da árvore ou das páginas propriamente ditas.

A unidade dispõe ainda de dois registradores que permitem a tradução transparente ou seja, o endereço físico é igual ao endereço lógico. Cada registrador pode ser usado para definir uma região do espaço de endereçamento em que não haverá tradução de endereços.

O processador NS32032 não apresenta uma unidade interna de gerenciamento de memória. Neste caso, tanto a tradução dos endereços lógicos quanto a proteção da memória têm que ser efetuados por unidades externas tais como o NS32082. Esta unidade opera de forma semelhante a unidade de gerenciamento de memória do MC68030. As páginas entretanto, possuem tamanho fixo de 512 *bytes*. Permite o gerenciamento

de apenas 16 *Mbytes*.

O acesso ao descritor de uma página é feito por meio de uma tabela de páginas de 2 níveis, apontada por um dos dois registradores internos da unidade (um para cada um dos modos de operação do processador). O endereço lógico gerado pelo processador é dividido em três campos: **índice1**, **índice2** e **deslocamento**. O índice1 é usado juntamente com o conteúdo do apontador da tabela para determinar a entrada do seu primeiro nível. O conteúdo do primeiro nível é então concatenado com o campo de índice2, indicando uma entrada no segundo nível da tabela. Esta contém o endereço base da página referenciada que é somado ao campo de deslocamento do endereço lógico para que o endereço físico seja obtido (figura **V.4**).

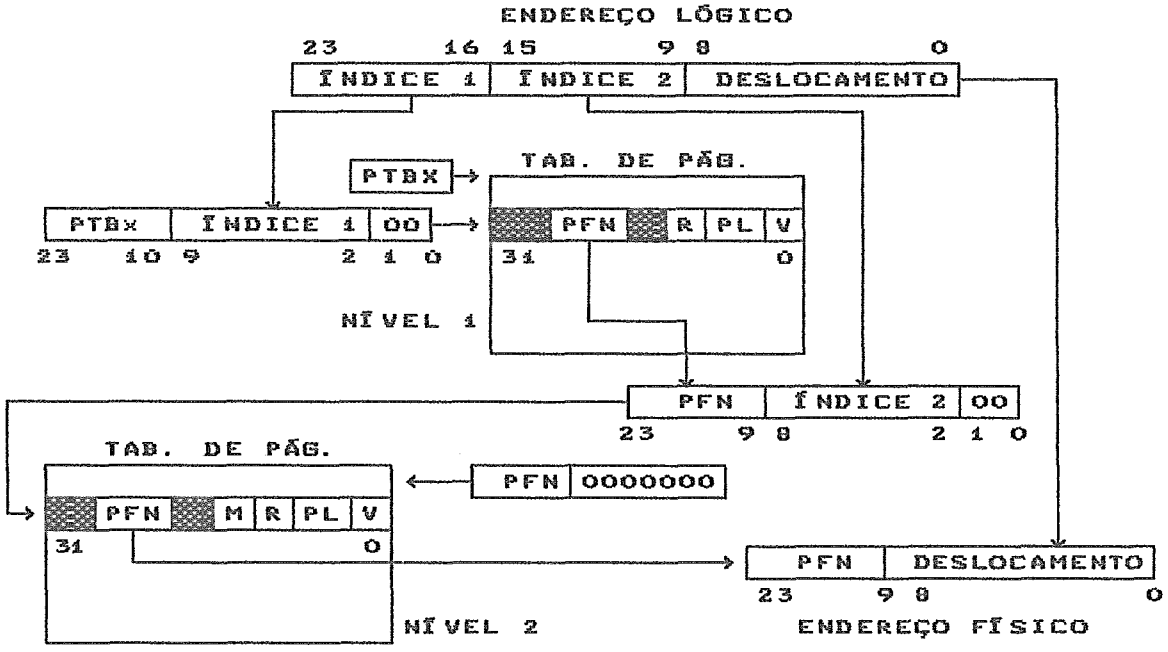


FIGURA V.4 - MECANISMO DE TRADUÇÃO DE ENDEREÇOS (NS32082)

O microprocessador Z80000 permite que a memória seja

organizada de forma linear ou segmentada. Em qualquer das duas formas existem os modos **compactado** e **completo**. O modo compactado existe para manter compatibilidade com Z8000 e permite que sejam endereçados apenas 64 *Kbytes* no endereçamento linear ou 4 segmentos de 16 *Kbytes*. No modo completo podem ser endereçados 4 *Gigabytes* diretamente ou através de segmentos de 16 *Mbytes*.

A unidade de gerência de memória do Z80000 divide qualquer destes espaços de endereçamento em páginas de 1024 *bytes*. Os 22 *bits* mais significativos do endereço lógico são usados pelo mecanismo de tradução para identificar (juntamente com as informações contidas nas tabelas) a página física de memória que deverá ser acessada. A tradução de um endereço lógico para um endereço físico é feita, como nos casos anteriores, por intermédio de tabelas de tradução de endereços. Podem existir até 3 níveis de tabela que são usados como mostra a figura **v.s.** Do mesmo modo que o MC68030, o Z80000 permite que o *hardware* externo identifique seus 4 espaços de endereçamento (supervisor/normal e código/dado). Tal fato obriga a existência de quatro estruturas de tabelas para tradução, associadas a cada um dos espaços de endereçamento. O acesso ao endereço base de cada uma destas tabelas é realizado através de 4 registradores internos à unidade de gerenciamento de memória.

O número de níveis da tabela de tradução pode ser estabelecido em 1, 2 ou 3 níveis, de acordo com a forma de representação de endereços escolhida.

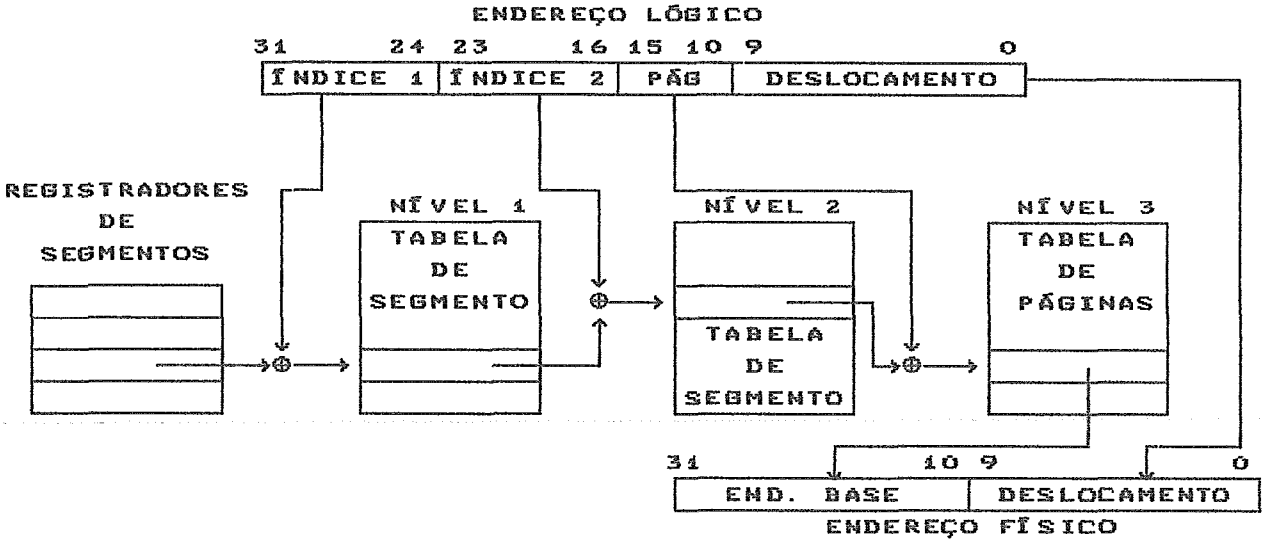


FIGURA V.5 - MECANISMO DE TRADUÇÃO DE ENDEREÇOS (Z80000)

A unidade de gerenciamento de memória dispõe também de uma *cache* associativa de 16 entradas em que o conteúdo das últimas 16 referências às tabelas de tradução permanecem armazenados.

O microprocessador 80386 oferece as três opções de operação em sua unidade interna de gerência de memória como mostra a figura **V.6.**

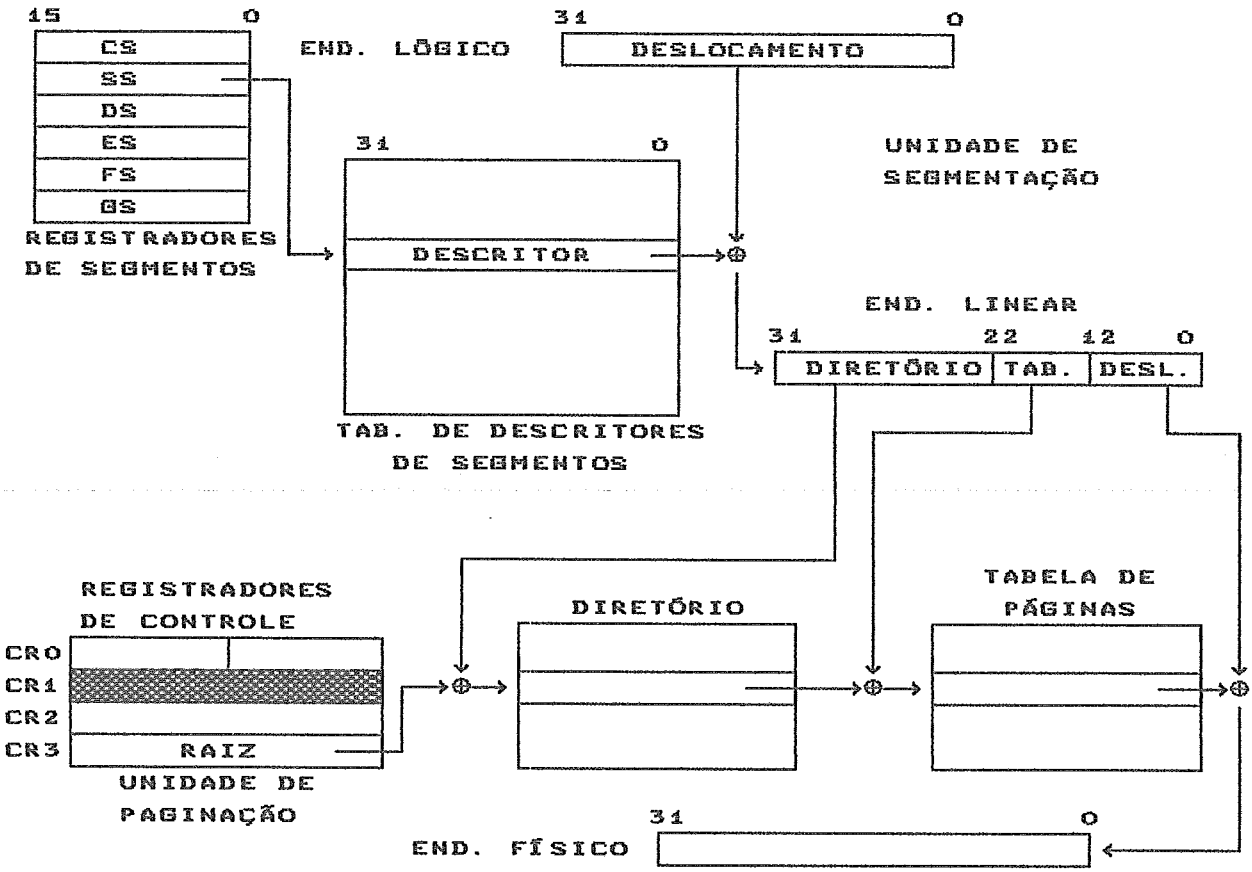


FIGURA V.6 - MECANISMO DE TRADUÇÃO DE ENDEREÇOS (80386)

No modo segmentado, o processador apresenta capacidade para endereçar até seis tipos de segmentos diferentes. A tradução do endereço lógico é realizada através das tabelas de descritores de segmentos residentes na memória do sistema. O acesso a estas tabelas é feito através de registradores internos que contêm o endereço base e o tamanho de cada uma das tabelas de descritores.

Existem três tipos de tabelas: Tabela de Descritores Globais (GDT), Tabela de Descritores Locais (LDT) e Tabela de Descritores de Interrupção (IDT). Cada entrada em qualquer das tabelas ocupa 8 *bytes* podendo existir até 8192 entradas, totalizando 64 *Kbytes* para o tamanho máximo de cada tabela.

Cada entrada na tabela contém um descritor de segmento. Uma entrada associada a um segmento específico é apontada pelo registrador do processador associado ao mesmo segmento. Assim, o registrador CS poderá estar apontando para uma entrada na tabela de Descritores Globais, o registrador DS para uma outra entrada da mesma tabela e o registrador SS poderá estar apontando para uma entrada na tabela de descritores locais por exemplo.

As informações contidas em um descritor podem variar de acordo com o tipo do descritor e com o segmento que descreve. Normalmente, apresentam campos contendo o tipo de segmento associado, o endereço linear que indica a base do segmento referenciado, seu tamanho e sua situação (presente/ausente); entre outras informações.

A unidade de segmentação apresenta ainda uma pequena *cache* em que são carregados cada um dos descritores associados aos seis tipos de segmentos disponíveis. Uma nova informação é trazida para esta *cache* toda vez que o conteúdo de qualquer dos registradores apontadores de segmentos é modificado.

A partir das informações contidas nos descritores e do endereço gerado pelo processador é obtido o endereço **linear** que possui 2^{48} *bits*. Ou seja, os 16 *bits* obtidos no descritor indicam,através da tabela de descritores, o endereço base do segmento; e os 32 *bits* do processador, a posição da informação desejada em relação à base deste segmento.

A tradução de endereços no modo paginado do 80386 é efetuada através de uma tabela de páginas de dois níveis

(fig **v.6**). O endereço **linear** obtido pelo mecanismo de segmentação é dividido em 3 campos: **diretório** (*bits* 22 a 31), **tabela** (*bits* 12 a 21) e **deslocamento** (*bits* 0 a 11). Todas as páginas possuem o mesmo tamanho (4 *Kbytes*) determinado pelo campo de deslocamento.

A unidade possui um conjunto de registradores (CR0-CR3) que contêm informações relativas ao endereço base da tabela de diretório (CR3), o último endereço em que ocorreu uma falta de página (CR2), etc.

A tabela de Diretório de Páginas pode ocupar o espaço de uma página de memória. Uma entrada desta tabela possui informações referentes ao endereço base da segunda tabela de tradução (Tabela de páginas) e os *bits* para controle e estatísticas de acesso.

Um diretório de páginas pode possuir até 1024 entradas, isto é, pode conter apontadores para 1024 tabelas de páginas distintas. A tabela de páginas ocupa o segundo nível do mecanismo de tradução de endereços. As entradas desta tabela contêm informações relativas ao endereço base de uma página da memória física. Este endereço somado ao campo de deslocamento do endereço linear determina uma posição específica na memória física. Cada tabela de páginas pode apresentar até 1024 entradas. Assim, a capacidade de mapeamento do sistema é de 1024 entradas no diretório de páginas multiplicado por 1024 entradas por tabela de páginas totalizando 1 Mega páginas de 4 *Kbytes*.

O modo de tradução de endereços misto é composto pelo mecanismo de segmentação (que gera o endereço linear a partir do endereço lógico) e pelo mecanismo de paginação

que obtém o endereço físico a partir do endereço linear.

Na operação em que apenas o mecanismo de segmentação esteja ativado, o endereço físico é o próprio endereço linear. Já no caso em que apenas o mecanismo de paginação esteja ativado, o endereço linear é o próprio endereço lógico gerado pelo processador.

Uma outra opção disponível ao programador é a operação com toda a unidade de gerenciamento de memória desativada. Neste caso os endereços lógico, linear e físico representam a mesma posição na memória física. A capacidade de endereçamento fica limitada pelo tamanho da memória física.

As unidades de gerenciamento de memória dos microprocessadores implementam o mapeamento por Tabelas de Tradução, facilitando sua utilização em sistemas paralelos. De acordo com o que foi apresentado nas seções **v.2** e **v.3** o microprocessador 80386 oferece o melhor esquema de tradução de endereços. O esquema paginado em 4 níveis do MC68030 pode ser reproduzido de forma bastante satisfatória pelos dois níveis de peginação do 80386. O Z80000, apesar de oferecer um esquema misto de tradução semelhante ao 80386, não apresenta a mesma flexibilidade. O 80386 pode operar no esquema segmentado puro, paginado puro ou no esquema misto. Além disso, a capacidade total de endereçamento lógico do 80386 (2^{48}) é bem maior do que qualquer dos outros microprocessadores. Todas estas características demonstram uma grande preocupação do fabricante do 80386 em relação ao gerenciamento do sub-sistema de memória.

Apesar do 80386 oferecer o melhor suporte para o gerenciamento de memória, os sistemas operacionais parecem

ainda pouco a vontade com as ferramentas oferecidas pelo *hardware*. Os sistemas operacionais com filosofia Unix portados para o 80386 (HENSLER[19]) têm utilizado apenas o mecanismo de paginação deste microprocessador.

V.5. Suporte a Memória Virtual

A técnica de Memória Virtual é empregada para permitir que um programa maior que a memória física possa ser executado. Para que esta técnica possa ser empregada é necessário que o processador apresente capacidade de detectar acessos a páginas de memória (ou segmentos) que não estejam presentes na memória física.

Para suportar a memória virtual o processador deve ser capaz ainda de:

- Cancelar a execução da instrução corrente,
- Preservar para uso posterior o estado da máquina quando ocorrer uma falta de página,
- Executar uma rotina de tratamento para falta de páginas,
- Fornecer as informações necessárias ao sistema operacional para que este possa proceder a alocação e ou substituição adequada das páginas e

.Restabelecer o estado anterior da máquina e prosseguir com o processamento normal.

Todos os microprocessadores de 32 *bits* atuais apresentam capacidade para cancelamento e re-execução de instruções.

As principais informações necessárias ao sistema operacional que a unidade de gerenciamento de memória deve fornecer para que este possa efetuar a substituição de páginas são:

.Bit de Validade

Especifica se um bloco está presente na memória primária.

.Bit de Referência

Indica se algum acesso foi efetuada no bloco de memória associado.

.Bit de Modificação

Indica a ocorrência de acessos de escrita.

A principal função do bit de modificação é a de indicar ao sistema operacional a necessidade de se copiar para o disco o bloco na memória primária associado a entrada na tabela de tradução referenciada.

No caso específico de suporte à memória virtual em microprocessadores, a implementação do mecanismo de

execução de uma instrução em que tenha ocorrido uma exceção por falta de página é feita de duas formas:

. Método de Continuação

. Método de Reinício

Pelo Método de Continuação, a instrução prossegue a partir do ponto (ou microinstrução) em que a exceção ocorreu; enquanto que pelo Método de Reinício, a instrução é inteiramente repetida. Para que o processador que utilize o método de continuação possa prosseguir com sua execução, é necessário que todo o estado da máquina no momento da exceção tenha sido guardado. Este estado refere-se à condição do processador no nível de microinstrução, incluindo um número relativamente grande de registradores temporários, registradores de estado de sequenciamento, etc. Os processadores que empregam este método apresentam geralmente, uma *stack* de erro por falta de página relativamente grande. Instruções que não podem ser interrompidas como as usadas para operações com semáforos (*Test & Set*) que determinam a repetição de toda a instrução (FURHT[18]). Neste caso, o processador deverá apresentar um mecanismo para re-execução deste tipo de instrução.

O Método de Continuação é implementado pelos microprocessadores da Motorola. Sua principal vantagem reside no ganho de tempo de processamento que pode ser obtido principalmente em instruções que efetuam

tranferências de blocos de dados.

O segundo método é usado pelos outros microprocessadores e determina a completa re-execução da instrução corrente após uma exceção por falta de página. Sua principal vantagem está em sua simplicidade de implementação. A cada nova instrução o estado interno do processador é armazenado em registradores temporários de modo que após a ocorrência de uma exceção, a instrução possa ser repetida; restaurando-se as condições iniciais.

Microprocessadores como o Z80000 e o 80386 que dispõem de alguns estágios de *pipeline* antes da efetiva execução da instrução, implementam este método de forma mais eficiente. A exceção é detectada dentro do *pipeline* antes mesmo de chegar à unidade de execução. Desta forma, só se torna necessário colocar na pilha o endereço da instrução que deverá ser executada.

Apesar do Método de Continuação apresentar-se mais eficaz com relação ao tempo de execução das instruções interrompidas por falta de páginas acarreta, por outro lado, alguns problemas para os sistemas de processamento paralelo. Por este método é necessário o armazenamento de uma grande quantidade de informações em memória para que o processador possa prosseguir do ponto onde foi interrompido pela exceção. A movimentação de dados envolvida neste método pode prejudicar o desempenho dos sistemas paralelos baseados em barramento comum e memória compartilhada. O método de reinício não necessita nenhum acesso à memória, aliviando o tráfego de informações no barramento. Além disso, o ganho de tempo proporcionado pelo método de

continuação em relação ao método de reinício não nos parece significativo se considerarmos este diferencial dividido pelo número de instruções executadas entre duas faltas de páginas. Deste modo, o MC68030 apresenta-se em desvantagem em relação aos outros microprocessadores.

V.6. Proteção e Segurança

Nos ambientes de múltiplos processadores, multiprogramação e multitarefa os mecanismos de proteção desempenham funções de extrema importância. Existem basicamente três formas de proteção (FURHT[18]):

. Proteção de Memória

. Proteção de Programa

. Proteção de Usuário

O mecanismo de Proteção de Memória deve ser responsável pela detecção de qualquer tipo de erro de endereçamento antes que este possa causar algum dano ao sistema. A Proteção de Programa deve prevenir a modificação de partes importantes do *software* (como o código do próprio sistema operacional) por programas aplicativos. O mecanismo de Proteção de Usuários evita que um usuário intefira com programas ou informações de outro usuário.

Todos estes mecanismos são bastante semelhantes e sua implementação recai em dois tipos básicos:

. Sistemas de proteção por Hierarquia ou Anéis

. Sistemas de proteção **não** Hierárquicos

Um sistema de proteção Hierárquico apresenta níveis diferenciados de proteção . Seus princípios básicos são:

1- Um programa pode acessar somente dados que pertençam a níveis (ou anéis) de igual ou menor proteção que o seu.

2- Um programa pode solicitar serviços de outros módulos de *software* em níveis (ou anéis) de proteção iguais ou maiores que o seu.

Nos sistemas de proteção não Hierarárquica, é definida uma tabela de operações permitidas para cada processo. Este sistema é mais flexível mas apresenta um grau de complexidade maior para sua implementação.

Os mecanismos de proteção encontrados nos microprocessadores de 32 *bits* são do tipo Hierárquico. Existem entretanto algumas variações na sua implementação.

O MC68020/30 apresenta dois níveis básicos: modo **Supervisor e** modo **Usuário.** Estes dois níveis implementam a proteção de Programa impedindo que um processo que esteja sendo executado em modo usuário tenha acesso a recursos exclusivos do modo supervisor (responsável pela execução do

sistema operacional). Estes processadores apresentam ainda o conceito de múltiplos níveis de acesso através de algumas de suas instruções. Isto permite ao processador estabelecer através do *software* até 256 níveis hierárquicos representando um superconjunto da estrutura de anéis encontrada no 80386 por exemplo.

Os microprocessadores da National e Zilog apresentam apenas os dois níveis básicos: modo **Privilegiado** e modo **Normal**. Já o microprocessador da Intel apresenta um mecanismo de proteção baseado em quatro anéis com níveis de privilégio decrescente com a ordem do anel. O anel de ordem zero possui o maior grau de privilégio enquanto o anel de ordem 3 o menor. Normalmente, o sistema operacional ocupa o anel de maior privilégio, protegendo-se de outros programas do sistema e de usuários. Os dois anéis seguintes são usados geralmente para serviços oferecidos pelo sistema. O terceiro nível é usado por programas aplicativos de usuários.

Este tipo de implementação permite que os níveis de menor privilégio possam solicitar serviços aos níveis de maior privilégio através de instruções simples para chamadas de subrotinas facilitando as operações de troca de contexto. O conceito de segurança refere-se ao acesso limitado às informações. Seu princípio básico reside em permitir que um processo tenha acesso apenas às informações que são necessárias para desempenhar suas funções.

A segurança é estabelecida através de direitos de acesso atribuídos às páginas ou segmentos do sistema. Os direitos de acesso mais comuns são:

. Acesso de Leitura

Um processo pode obter qualquer informação da página ou segmento refenciado.

. Acesso de Escrita

Um processo pode alterar o conteúdo da página e/ou inserir informações adicionais.

. Acesso de Execução

Um processo pode executar o conteúdo de uma página ou segmento.

Os processadores normalmente implementam o controle de segurança através de *bits* relacionados com os direitos de acesso. Estes *bits* são associados aos descritores de páginas (ou segmentos). Quando o processador envia um endereço de uma página, os direitos de acesso àquela página são verificados durante o processo de tradução de endereços.

Os três tipos de direitos de acesso apresentados acima são comuns a todos os mecanismos de gerenciamento de memória dos quatro microprocessadores.

Os mecanismos de proteção e segurança são de grande importância para os sistemas de múltiplos processadores na medida em que este controle se realizado somente através do sistema operacional, torna-se bastante complexo e com grau de confiabilidade inferior aos dos sistemas

monoprocessados. Para assegurar um nível satisfatório de confiabilidade torna-se necessário o monitoramanto dos acessos de cada processador, a cada ciclo. Isto só pode ser conseguido através de mecanismos de *hardware* como os apresentados.

Os microprocessadores da Motorola, National e Zilog apresentam mecanismos semelhantes para proteção e segurança de informações. São implementados com filosofia hierárquica, em dois níveis. A existência de sinais elétricos que indicam o nível corrente em que o microprocessador opera permite a construção de mecanismos de proteção mais elaborados. O mesmo não pode ser afirmado em relação aos microprocessadores da Intel. Seu mecanismo de proteção em anéis é totalmente dependente de definições de programação (tabelas e mapas de *bits*), sem nenhum tipo de sinalização para o *hardware* externo. Este tipo de implementação, apesar de mais elaborado, apresenta um grau de confiabilidade inferior aos demais microprocessadores. A existência de uma falha qualquer nas definições de *software* deixa os mecanismos de segurança completamente vulneráveis.

CAPÍTULO VI

SUPORTE AOS SISTEMAS OPERACIONAIS

O suporte aos sistemas operacionais é um dos principais aspectos a serem considerados na análise de qualquer processador no contexto dos sistemas computacionais atuais. Este aspecto reveste-se de maior importância se considerarmos o papel dos sistemas operacionais em sistemas para processamento paralelo.

As funções desempenhadas pelos sistemas operacionais que necessitam de um suporte especial de *hardware* estão relacionadas com as operações de tratamento de interrupções e exceções, manuseio de entrada e saída e troca de contexto. Além disso, o suporte ao gerenciamento e proteção de memória, compartilhamento de informações, gerenciamento de recursos (processadores, controladores de periféricos e estruturas de *software*) e comunicação entre processadores, é de fundamental importância para os sistemas operacionais com ambientes multitarefa e multiprocessamento.

Neste capítulo iremos analisar o suporte oferecido pelos microprocessadores de 32 *bits* a algumas das funções citadas, mormente aquelas relacionadas com o manuseio de entrada e saída, tratamento de exceções, troca de contexto e comunicação entre processos e processadores.

VI.1. Manuseio de Entrada e Saída

Nos ambientes multitarefa, as operações de entrada e saída são de responsabilidade única e exclusiva do sistema operacional. Acessos não autorizados aos dispositivos de entrada e saída por programas aplicativos devem ser bloqueados pelo *hardware*, assegurando total controle desses dispositivos ao sistema operacional. O tratamento das operações de entrada e saída é de grande importância para as aplicações do tipo *i/o-bound*. Uma estrutura eficiente para atendimento e tratamento de interrupções pode ser um fator preponderante para melhorar o desempenho dessa classe de aplicações.

Os microprocessadores da Motorola utilizam o conceito de entrada e saída mapeadas em memória. Isto significa que as operações de leitura e escrita na memória são realizadas da mesma forma nos dispositivos de entrada e saída. A principal vantagem desta filosofia de operação reside no tratamento uniforme que é dado a todos os componentes de um sistema computacional (memória e dispositivos de entrada e saída). Implica porém, na existência de algum tipo de mecanismo de bloqueio para o acesso de aplicativos e programas não autorizados pelo sistema operacional. Os microprocessadores da Motorola implementam essa proteção através da separação dos espaços de endereçamento. Mapeando-se os dispositivos de entrada e saída no espaço de endereçamento do **supervisor**, por exemplo, evita-se que um programa de usuário possa ter acesso a qualquer dispositivo de entrada e saída sem autorização do sistema operacional.

Na verdade, as operações de entrada e saída, são realizadas pelo sistema operacional (que é executado no espaço de endereçamento do supervisor) através de solicitações do aplicativo, executado em modo usuário. Esta separação permite ainda que todo o espaço de endereçamento de usuário possa ser integralmente aproveitado pelos aplicativos (figura VI.1).

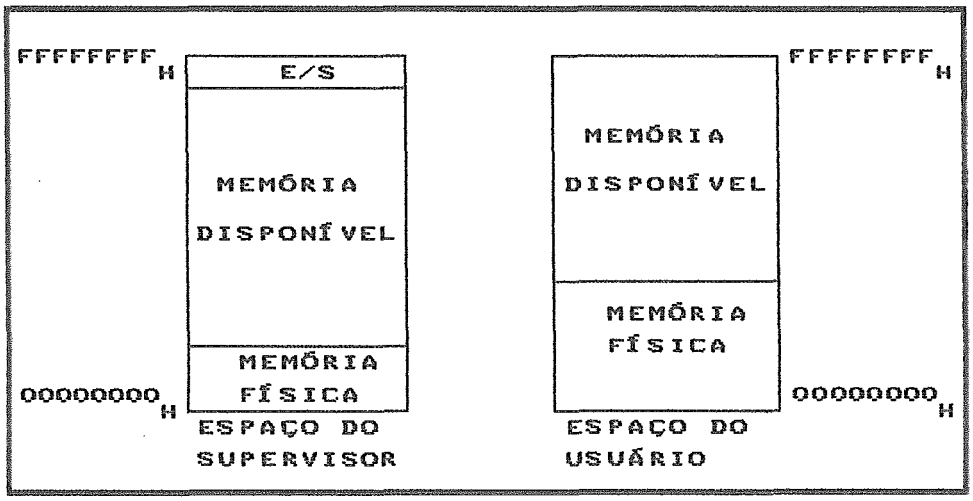


FIGURA VI.1 - E/S MAPEADA EM MEMÓRIA

O processador NS32032 da National opera de modo semelhante ao MC68020/30 através do mapeamento dos dispositivos de entrada e saída no espaço de endeçamento comum à memória do sistema. Sua principal diferença em relação aos microprocessadores da Motorola reside na identificação dos espaços de endereçamento. O NS32032 apresenta apenas dois espaços de endereçamento: normal e privilegiado enquanto o MC68030 dispõe de 8 espaços em que um deles pode ter seu uso definido pelo projetista (que poderá alocá-lo inteiramente para os dispositivos de entrada e saída).

Os microprocessadores da Intel e Zilog não utilizam a filosofia de entrada e saída mapeada em memória. O acesso aos dispositivos de entrada e saída é realizado por meio de instruções especiais (*IN* e *OUT*). Esta separação configura um espaço de endereçamento próprio para operações de E/S em que só podem ser executadas instruções de leitura, escrita e algumas operações lógicas. O acesso a este espaço de endereçamento só pode ser efetuado no Z80000 através do **modo privilegiado.** O 80386 por outro lado, permite estabelecer (através de configuração interna) quais dos seus quatro níveis de proteção e que portas podem ser acessadas pelas instruções de entrada e saída, em cada nível . A configuração é estabelecida através de um mapa de *bits* que é associado às portas de E/S. Outra característica destes dois microprocessadores é que o endereço gerado para uma instrução de E/S não sofre tradução pela unidade de gerenciamento de memória. O 80386 é capaz de endereçar até 64 *Kbytes* em seu espaço de endereçamento de E/S.

Os mecanismos de tratamento e acesso aos dispositivos de entrada e saída dos microprocessadores de 32 *bits* apresentam uma arquitetura bastante tradicional, mantendo as mesmas características dos microprocessadores de 16 e 8 *bits*. Nos microprocessadores que não utilizam E/S mapeada em memória (Intel e Zilog) foram incluídas apenas algumas instruções para permitir a movimentação de blocos de dados nas operações de E/S.

As operações de E/S nos sistemas paralelos representam um dos *gargalos* dessas arquiteturas. Nos sistemas baseados em barramentos compartilhados por exemplo, torna-se

necessário um tratamento especial às operações de E/S a fim de reduzir o tempo de espera dos processadores. Normalmente, recorre-se a barramentos especiais dedicados à E/S e Unidades Inteligentes. Estas unidades implementam os protocolos necessários para acessar os disposistivos de E/S, liberando os processadores para tarefas mais nobres.

Nos sistemas paralelos com topologia hipecúbica são necessários canais de E/S de alta velocidade e em número relativamente elevado, a fim de garantir a eficiência do processamento.

Nenhum dos quatro microprocessadores analisados neste trabalho apresenta mecanismos de E/S eficientes para o processamento paralelo. Neste aspecto, os **Transputers** (INMOS[37]) apresentam-se mais adequados às arquiteturas para processamento paralelo.

VI.2. Tratamento de Exceções

As exceções podem ser originadas interna ou externamente ao processador. Normalmente, as exceções externas são provocadas por:

. Interrupções,

. *Reset*,

. *Bus Errors* e

. *Time-out*

As exceções internas podem ser ocasionadas por:

- Instruções Ilegais,
- Erros de Endereçamento,
- Instruções Especiais (chamadas ao supervisor),
- Condições internas de erro (divisão por zero) e
- Pontos de Parada para o *software*

Geralmente, o suporte ao tratamento de exceções apresentado pelos microprocessadores é baseado no esquema vetorizado. Ao ocorrer uma exceção, o fluxo do processamento é desviado para uma rotina específica para aquele tipo de exceção cujo endereço é obtido por meio de uma tabela (*Tabela de Vetores de Exceções*). Nesta tabela são encontradas algumas entradas com funções pré-definidas pelo fabricante do microprocessador e outras livres para definição pelo projetista do *hardware* do sistema. O projetista do sistema operacional precisará apenas preencher as entradas da tabela com os endereços das rotinas apropriadas para o tratamento de cada exceção.

O mecanismo vetorizado permite ao processador desviar o fluxo do processamento para a rotina de tratamento adequada, imediatamente. Todos os microprocessadores considerados neste trabalho apresentam registradores internos que indicam o endereço do início da tabela de exceções na memória física do sistema.

O mecanismo de *hardware* responsável pelo gerenciamento

das interrupções externas também é de grande importância para a eficiência do sistema. Os microprocessadores da Motorola por exemplo, apresentam sete níveis priorizados para atendimento desse tipo de interrupções. O nível sete possui a maior prioridade. Pode-se estabelecer, através de instruções especiais, o nível a partir do qual serão aceitas as interrupções. Deste modo, se a **máscara** de interrupções for estabelecida para o nível 3, somente interrupções de níveis maiores que 3 serão aceitas. Apenas o nível sete não pode ser mascarado. Em cada um destes níveis pode ser estabelecido um esquema de *daisy-chain* criando-se um sistema hierárquico para as interrupções. Por estar implementado internamente ao microprocessador, este mecanismo de controle de interrupções proporciona uma grande flexibilidade para o *software*.

Os microprocessadores da Intel, National e Zilog apresentam apenas dois níveis para interrupções externas: **interrupções mascaráveis** e **não mascaráveis**. O mecanismo para priorização das interrupções **não** mascaráveis precisa ser estabelecido externamente pelo projetista do sistema que poderá utilizar controladores específicos para este fim (ex: 8259 da Intel). A desvantagem deste tipo de implementação é que o gerenciamento das prioridades torna-se mais elaborado para o *software*, que passa a ter que acessar dispositivos externos ao processador (interfaces de *hardware*). Normalmente estes dispositivos necessitam ser programados para entrar em operação, introduzindo *overhead* extra nas rotinas de gerenciamento das interrupções.

A confiabilidade do sistema é outro fator a ser considerado. Dispositivos adicionais tendem a diminuir o grau de confiabilidade do conjunto na medida em que a probabilidade de ocorrência de falhas torna-se maior.

Todos os microprocessadores apresentam mecanismos semelhantes para tratamento de exceções. Entretanto, os microprocessadores da Motorola destacam-se pelo esquema interno de priorização que nos parece ser mais eficiente. Consequentemente, os sistemas de múltiplos processadores baseados nestes microprocessadores deverão apresentar melhor desempenho no tratamento de exceções.

VI.3. Troca de Contexto

O tempo gasto por um processador para efetuar as trocas de contextos relativos aos diversos processos que executa é um fator de grande relevância para a avaliação de seu desempenho. Uma troca de contexto deve ser realizada no menor intervalo de tempo possível. As operações de troca de contexto assim como as outras atividades desempenhadas por um sistema operacional (escalonamento de processos, manipulação de interrupções, etc.) que não são solicitadas diretamente por aplicativos de usuários, não devem penalizá-los em seu tempo total de execução.

Características do *hardware* dos processadores que normalmente agilizam as operações de troca de contexto incluem geralmente a cópia automática dos registradores de estado do processador para a memória primária do sistema. Em alguns microprocessadores estas operações são realizadas

por instruções especiais através das quais o conteúdo de todo o conjunto de registradores pode ser transferido para a memória primária (e vice-versa). Neste caso, todo controle da operação de troca de contexto é realizado pelo *software*. Esta filosofia é utilizada pelos microprocessadores da Motorola, National e Zilog. O microprocessador da Intel implementa uma forma mais elaborada e poderosa para a troca de contexto. Todas as informações relativas a um processo (*task*) são encontradas nos TSS (*Task State Segments*). Este tipo especial de segmento possui formato fixo e é responsável pelo armazenamento de todas as informações que permitam ao microprocessador iniciar (ou reiniciar) a execução de um processo após uma troca de contexto. A cada um destes segmentos está associado um descritor (*TSS Descriptor*) encontrado na Tabela de Descritores Globais (*GDT*) ou Locais (*LDT*). Este descritor contém informações relativas à localização, tamanho, nível de privilégio, tipo (286 ou 386) e estado (disponível ou ocupado) de um TSS. O registrador TR (*Task Register*) contém o seletor que indica ao processador o descritor do processo corrente.

A partir das estruturas descritas acima, uma troca de contexto no 80386 pode ser explicada, de forma resumida, como uma mudança do conteúdo do registrador TR. Na verdade, para que esta mudança seja efetuada torna-se necessária a utilização uma outra estrutura chamada *GATE*. Os *gates* são usados para controlar a transferência do fluxo do processamento entre os quatro níveis de privilégio do 80386. Existem quatro tipos de *gates*: **call gates, task**

gates, interrupt gates e **trap gates**. Os *call gates* são usados para *chamar* rotinas de níveis de privilégio mais elevados, *interrupt* e *trap gates* para especificar rotinas para tratamento de interrupções e finalmente *task gates*, para determinar trocas de contexto. Os *gates* são descritores encontrados na GDT (ou LDTs) que permitem ao processador verificar a validade e efetuar a transferência do fluxo do processamento de acordo com os critérios de proteção de informações estabelecidos pelo projetista do sistema operacional, responsável pela definição dos *gate descriptors*. No 80386, uma troca de contexto é realizada através de instruções do tipo **JUMP** ou **CALL** que referenciem um *gate descriptor* ou **INT** (implícitamente associado a um *gate descriptor*), provocando uma mudança do nível de privilégio corrente para um nível mais alto. A execução destas instruções determina a transferência de uma cópia de todo o estado interno da unidade de processamento assim como registradores auxiliares para o TSS. Em seguida é feita a leitura do novo TSS com a verificação dos critérios de proteção para o uso das informações do processo anterior (que determinou a troca de contexto) pelo novo processo, antes de dar início à sua execução. A operação de troca de contexto ocorre também com a execução da instrução **IRET** que determina uma operação semelhante a descrita acima porém, restaurando ao processador o estado do processo interrompido por interrupções externas, exceções ou pela execução da instrução **INT**.

Segundo NGI111, um 80386* pode *salvar* o contexto (todos os registradores), carregar um novo contexto (todos os registradores gerais mais seis descritores de segmentos) e voltar à execução normal em menos de 20 microsegundos. Nos outros microprocessadores, toda a manipulação das tabelas de processos é efetuada por instruções comuns determinando um tempo maior para a troca de contexto. Ainda de acordo com NGI111, o NS32032 é o que dispende maior tempo para este tipo de operação.

VI.4. Comunicação e Sincronização de Processos

A comunicação e sincronização de processos são os elementos chave para a operação dos sistemas de processamento paralelo. Somente através dos mecanismos de comunicação é possível permitir que a execução de um processo possa interferir, de forma cooperativa, com a execução de outros processos. Nos sistemas de múltiplos processadores, esta interferência poderá ocorrer simultâneamente com a execução dos outros processos. Os mecanismos de sincronização tornam-se ainda mais importantes para estes sistemas na medida em que os processos, sendo executados a velocidades imprevisíveis, precisam sincronizar-se para que a comunicação seja estabelecida.

* Operando a 16MHz

A comunicação entre processos é implementada através de **Regiões Compartilhadas** de memória ou **Troca de Mensagens** entre processos. A técnica de Troca de Mensagens pode ser aplicada a qualquer tipo de arquitetura ou topologia (STANKOVIC[21]). Seu desempenho poderá ser inferior ao obtido através da técnica de Regiões Compartilhadas quando a topologia do sistema for baseada em barramentos compartilhados.

Dois mecanismos de sincronização são usados na implementação das Regiões Compartilhadas:

.Exclusão Mútua

> Apenas um único processo tem acesso a um conjunto de dados ou instruções compartilhados num dado instante.

.Sincronização Condicional

> Um processo terá acesso a um conjunto de dados somente após estes dados estarem em condições adequadas para sua utilização. Até que tal fato ocorra, os processos que tentarem acessar aquele conjunto de dados deverão permanecer em estado de espera.

Geralmente, as implementações destes mecanismos são efetuadas através de **semáforos** e **monitores**. Para que estes mecanismos possam ser efetivamente implementados é

necessário que o *hardware* apresente algumas características específicas. A capacidade de não aceitar interrupções durante a operação dentro de um **monitor** e a execução de operações de leitura e escrita através de um único ciclo de barramento (atômico) são características imprescindíveis. Esta última característica apresenta importância fundamental nos sistemas de múltiplos processadores. Este tipo de instrução, geralmente chamado *Test & Set*, permite que um processador possa ler o conteúdo de uma variável de memória e atualizar seu valor em apenas um ciclo de barramento. Esta variável é a chave para que o processador possa usar uma região compartilhada. Com a instrução de *Test & Set* o processador lê esta variável, verifica a possibilidade de usar a região associada (*Test*), alocando-a se estiver livre (*Set*). Caso já tenha sido alocada por outro processador, seu conteúdo não será alterado. É importante observar que este tipo de instrução elimina qualquer possibilidade de dois ou mais processadores alocarem simultaneamente a mesma região compartilhada.

A técnica de Troca de Mensagens é baseada no envio e recebimento de conjuntos de dados (*mensagens*). Neste caso, o mecanismo de sincronização está implícito uma vez que para que um processo receba uma mensagem, é necessário que outro processo a tenha enviado. Estes fatos determinam a ordem em que os eventos deverão ocorrer. Para sua implementação são necessárias algumas primitivas básicas que devem ser suportadas pelo sistema operacional ou linguagens de alto nível voltadas para processamento concorrente ou paralelo. Algumas destas primitivas mais

comuns são: **SEND (MSG, DESTINO), RECEIVE (MSG)** e **WAIT.**

Para facilitar a implementação da técnica de Troca de Mensagens, o *hardware* deve oferecer facilidades para a manipulação de blocos de dados, sinais de controle específicos relacionados com a transferência de dados entre processadores a altas velocidades e, eventualmente, barramentos dedicados.

Todos os microprocessadores de 32 *bits* aqui considerados apresentam suporte de *hardware* para a implementação da técnica de Regiões Compartilhadas através de instruções com ciclos indivisíveis (atômicos) de barramento. Alguns, como os microprocessadores da Motorola e Intel, permitem a execução de múltiplos ciclos indivisíveis, facilitando a atualização das tabelas de controle associadas aos recursos compartilhados, mantidas pelos sistemas operacionais. As instruções TAS, CAS e CAS2 nos microprocessadores da Motorola e o prefixo LOCK no 80386 são responsáveis pela execução de múltiplos ciclos. A instrução TAS é usada para o controle de semáforos. As instruções CAS e CAS2 permitem a atualização das listas manipuladas pelos sistemas operacionais com total segurança contra a intervenção de outros processadores. A instrução CAS permite que seja efetuada a retirada ou a inclusão de um elemento (geralmente apontador) numa lista encadeada, em um único ciclo de barramento. A instrução CAS2 executa a mesma função da instrução CAS porém numa lista duplamente encadeada. Deste modo, nenhum outro processador consegue acesso ao barramento compartilhado enquanto estas instruções estiverem sendo executadas. É interessante notar

que o tempo em que o barramento ficará preso dependerá do tamanho da lista. No 80386, o prefixo LOCK antes de qualquer instrução permite que o barramento fique *preso* durante todo o tempo gasto pelo processador para sua execução. A utilização do prefixo LOCK em conjunto com outros prefixos (REP - repetição múltipla - por exemplo) permite a atualização de um conjunto grande de dados (tabelas, listas, mapas de *bits*) em um ciclo único de barramento.

O suporte à Troca de Mensagens é bastante inexpressivo e, em alguns casos, inexistente. Somente os microprocessadores 80386 e Z80000 apresentam instruções específicas para a manipulação de blocos de dados para operações de entrada e saída. Todos os outros dispositivos de *hardware* que facilitam a implementação dos mecanismos usados pela Troca de Mensagens (FIFOS, Controladores e barramentos dedicados, dispositivos para transferências de dados a altas velocidades, etc.) têm que ser construídos a parte.

CAPÍTULO VII

SISTEMAS REAIS PARA PROCESSAMENTO PARALELO

Apresentamos a seguir alguns exemplos de sistemas para processamento paralelo baseados nos microprocessadores de 32 *bits* que analisamos neste trabalho. A maior parte desses sistemas implementa o paralelismo ao nível de múltiplos processadores operando como unidades autônomas. Isto é, todos os processadores executam processos diferentes paralelamente. Alguns poucos permitem o paralelismo de média granularidade através de artifícios e mecanismos especiais incorporados ao *hardware* das unidades de processamento.

Iniciamos este capítulo apresentando o sistema PEGASUS-32X® desenvolvido a partir de 1982 pelo Núcleo de Computação Eletrônica da U.F.R.J., projeto em que tomamos parte. Em seguida são apresentados alguns exemplos de sistemas desenvolvidos e comercializados no exterior além de propostas de empresas e centros de pesquisas nacionais. A maioria dos sistemas são baseados nos microprocessadores da Motorola (MC68020) e National (NS32032). O microprocessador 80386, lançado mais recentemente, só é encontrado na linha Simmetry da Sequent enquanto o Z80000 e o MC68030 ainda não chegaram aos sistemas comerciais.

VII.1. Sistema PEGASUS-32X

O sistema PEGASUS-32X desenvolvido pelo Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro a partir de 1982 foi o primeiro sistema nacional para multiprocessamento baseado em microprocessadores de 32 *bits* de que temos conhecimento. O sistema PEGASUS foi desenvolvido em conjunto com o sistema operacional PLURIX®. O PLURIX é um sistema operacional com filosofia UNIX® que apresenta suporte ao multiprocessamento. Apesar de ser um protótipo de laboratório, o PEGASUS pode ser considerado um marco da capacitação nacional na área de arquitetura de sistemas. Sua arquitetura básica (fig **VII.1**) é a mesma encontrada nos principais sistemas para multiprocessamento baseados em barramentos que são lançados atualmente nos mercados internacionais.

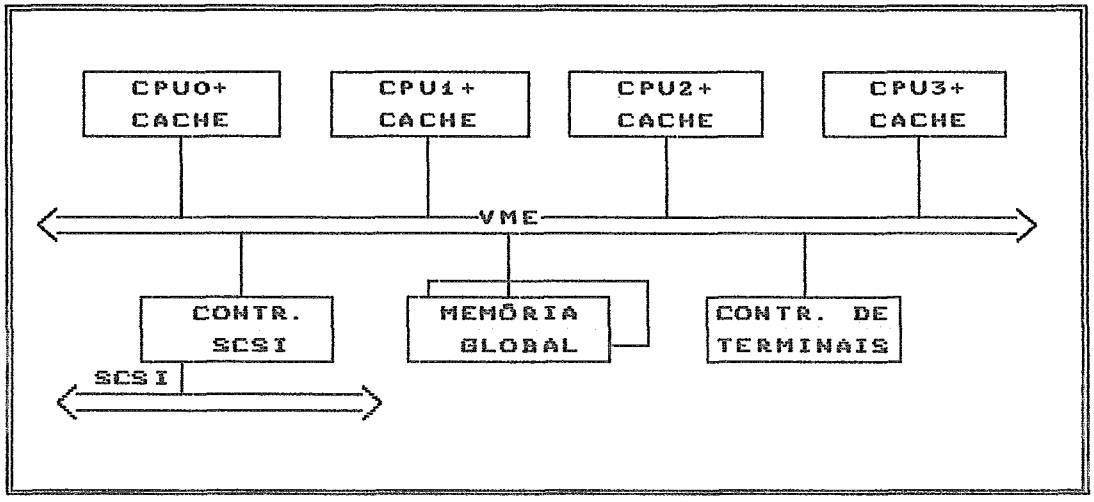


FIGURA VII.1 - ARQUITETURA DO SISTEMA PEGASUS-32X

As unidades de processamento do sistema PEGASUS são compostas por microprocessadores MC68020 (operando a uma

frequência de 12.5 MHz), memória *cache* privada para dados e instruções com 4 *Kbytes* e linhas de 4 *bytes* com mapeamento direto além de uma unidade de gerência de memória proprietária. O circuito de controle da *cache* também é proprietário e possui o controle de consistência baseado na técnica de monitoramento do barramento. Utiliza a política de *write-through* para as operações de escrita. A unidade de gerênciamento de memória utiliza o esquema segmentado implementado através da tradução por registradores. Cada segmento pode ter até 2 *Mbytes* e podem existir até 8 segmentos para o modo supervisor e 8 para o modo usuário.

O barramento utilizado para interligação com os outros módulos dos sistema segue o padrão VME. Este barramento é assíncrono, não multiplexado e possui largura de 32 *bits*. Conectam-se ao VME os módulos de Memória Global e as Unidades de Processamento Periférico. A Memória Global pode atingir até 16 *Mbytes* em módulos de 4 *Mbytes*. Possui circuitos para detecção de erros paridade por *byte* e escrita buferizada (*write buffer*). As Unidades de Processamento Periférico são responsáveis pelo controle das operações de entrada e saída. Apresentam microprocessadores e uma região de memória *dual-port* que permitem a interação com os processadores principais através de comandos de alto nível.

O *software* para suporte ao processamento paralelo (a nível de processos) oferecido pelo sistema operacional é todo baseado na técnica de **Regiões Compartilhadas**. A instrução TAS do MC68020 é o elemento fundamental para a implementação do mecanismo de sincronização de processos.

As variáveis de memória usadas como semáforos são lidas para as *caches* de cada unidade de processamento. Deste forma, enquanto o semáforo estiver indicando a ocupação de uma região crítica, as leituras serão realizadas nas *caches* de cada unidade de processamento. Ao ser liberado o semáforo, as *caches* que o possuem têm sua posição correspondente invalidada obrigando que o próximo acesso seja realizado na memória global. Este esquema permite que as unidades de processamento possam aguardar a liberação dos semáforos sem aumentar o tráfego no barramento do sistema.

Todas as caracterísctica apresentadas visam deixar os processadores do sistema o maior tempo possível dedicado as tarefas de usuários. O sistema pode suportar até 4 unidades de processamento operando em paralelo no nível de processo. Seu desempenho de pico não pôde ser medido em virtude de o sistema não possuir ainda suas 4 unidades de processamento.

VII.2. Sistemas Comerciais

A maior parte dos sistemas comerciais apresenta sistemas operacionais com filofia UNIX, adaptados ao processamento paralelo. Com relação às linguagens de alto nível, apenas a Alliant oferece um compilador FORTRAN capaz de implementar o paralelismo dentro de um único processo, executando suas partes independentes (detectadas e divididas pelo compilador) em várias unidades de processamento simultaneamente. O compilador identifica os

loops do tipo DO e FOR que realizam operações sobre dados que não apresentam nenhuma espécie de interdependência, separando-os para serem distribuídos pelo sistema operacional entre as diversas unidades de processamento.

A grande maioria dos sistemas comerciais para processamento paralelo baseados em microprocessadores utiliza a topologias de barramentos compartilhados. É fato que essas topologias apresentam o pior desempenho para as arquiteturas paralelas, principalmente pela limitação que impõe para o número máximo de unidades de processamento. Porém, sua preferência pelos fabricantes pode ser explicada em função dos problemas de compatibilidade e migração de *software* dos sistemas monoprocessados. Neste primeiro estágio em que as arquiteturas paralelas estão se tornando mais populares, é importante manter um alto grau de compatibilidade, a nível de *software*, com os sistemas monoprocessados, a fim de tornar o menos traumática possível a migração dos aplicativos entre as duas arquiteturas. O mesmo não pode ser dito das topologias baseadas em Redes Comutadas (BBN Advanced Computers) e Hipercúbicas (INTEL, Ncube, Connection Machine) que exigem adaptações nos aplicativos, muitas vezes profundas.

A movimentação de dados no barramento é a principal preocupação dos fabricantes de sistemas paralelos com topologia baseada em barramentos compartilhados. A arquitetura desses sistemas é sempre voltada à minimização do tráfego no barramento. Assim, a implementação de sub-sistemas com organização hierárquica visa reduzir a movimentação de dados entre a memória primária e as

unidades de processamento através do uso intensivo das memórias *cache*. A forma de implementação da *cache* é outro fator de grande importância. Os sistemas que implementam controles através de métodos de diretórios com controladores proprietários conseguem melhores resultados, permitindo a ampliação do limite máximo no número de processadores. Invariavelmente o barramento compartilhado entre as unidades de processamento e a memória é proprietário, com características especiais que visam o aumento da *banda-passante*. Podem existir também barramentos secundários destinados à comunicação entre processadores e interligação a controladores de periféricos. Estes últimos geralmente são barramentos padronizados (MultiBus, SCSI, VME, etc.), facilitando a interligação com diferentes tipos de periféricos.

Entre os sistemas comerciais de processamento paralelo com topologia não baseada em barramentos compartilhados, destacamos os sistemas da BBN (Butterfly) com topologia baseada em rede comutada e Intel Scientific Computers (Hypercube) com topologia hipercúbica.

VII.2.1. Sequent

Como pode ser observado pela figura VII.2, os sistemas produzidos pela Sequent utilizam barramento compartilhado (proprietário), unidades de processamento com *cache* individual, memória primária global e adaptador para barramento padronizado (MultiBus). Há duas famílias: Balance, baseada nos microprocessadores da National e

Simmetry, baseada no 80386 da Intel.

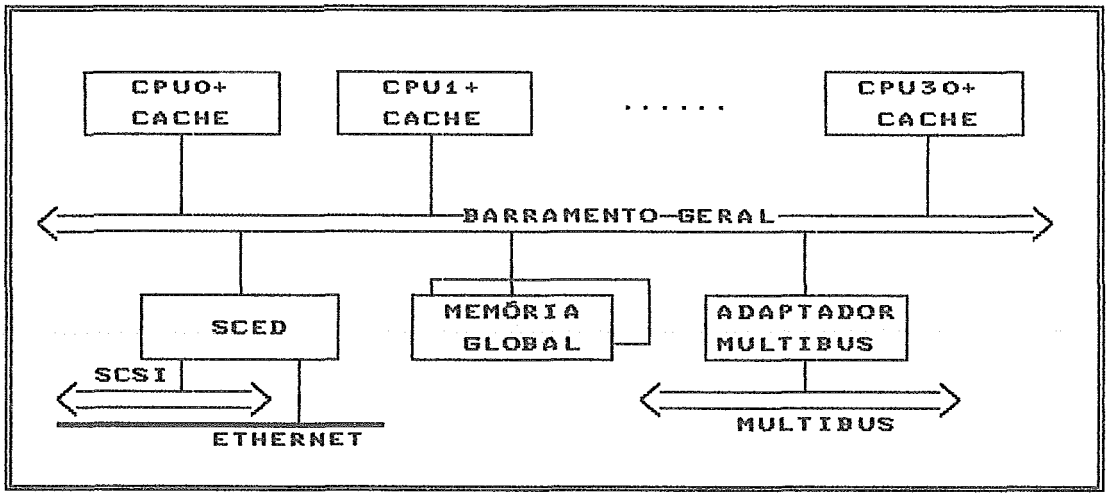


FIGURA VII.2 - ARQUITETURA DOS SISTEMAS SEQUENT

O número de unidades de processamento presentes é variável. Em sua configuração máxima o sistema pode suportar até 30 processadores. Dentre os sistemas comerciais para processamento paralelo baseados em barramentos compartilhados, as linhas Balance e Simmetry alcançam o maior número de processadores. Isto é conseguido através da conjugação de uma série de técnicas e soluções que permitem minimizar o tráfego de informações no barramento. Apresentamos a seguir as principais características destes sistemas, responsáveis pelo excelente desempenho que demonstram.

As unidades de processamento são compostas pelos microprocessadores NS32032 (linha Balance) e 80386 (linha Simmetry), ligados a seus co-processadores aritméticos. Possuem ainda uma memória *cache* de 64 *Kbytes*, dividida em dois conjuntos associativos com linhas de 16 *bytes*. O controle de leitura, escrita e consistência da *cache* é

realizado por circuitos proprietários. Implementam o método de diretório com *copy-back* nas escritas. Este mecanismo reduz substancialmente a movimentação de informações no barramento principal.

O sub-sistema de memória possui capacidade de endereçamento de até 240 *Mbytes* com controle inteligente. Os pedidos de acesso à memória (leituras e escritas) são armazenados em *FIFOS*, liberando o barramento para que outros acessos possam ocorrer. Quando uma informação solicitada à memória torna-se disponível (leituras), este sub-sistema reconecta-se ao processador que havia solicitado a informação.

As funções de gerenciamento e proteção de memória desempenhadas pelo *hardware* interno no caso do 80386, e por controladores externos (NS32082) ligados ao NS32032 não apresentam nenhuma característica especial.

O barramento compartilhado opera de forma **síncrona**. Suporta uma taxa de transferência média em torno de 53.2 *Mbytes* (máxima de 80 *Mbytes/s* - MANUEL[22]) com largura de 64 *bits* para dados e 32 *bits* para endereços multiplexados com a parte baixa da palavra de dados.

As operações de E/S são divididas em dois tipos. Os discos, que geralmente representam um *gargalo* nos sistemas, são acessados através de controladores especiais ligados diretamente ao barramento principal. Os acessos aos outros dispositivos de entrada e saída são realizados por meio de uma interface para um barramento secundário (MultiBus) onde os periféricos estão ligados.

A sincronização e comunicação entre os processos é

realizada através dos recursos oferecidos pelos conjuntos de instruções dos microprocessadores. Entretanto, para reduzir a taxa de ocupação do barramento por processadores que estejam em *loop* aguardando a liberação de um semáforo, os *loops* são realizados nas *caches*. Quando o semáforo é liberado, as posições associadas nas *caches* são invalidadas pelo circuito de controle obrigando a realização de um acesso à memória primária. O processador que primeiro conseguir acesso ao barramento poderá alocar o semáforo.

Todas as características apresentadas acima permitem que as máquinas da Sequent possam atingir um índice de desempenho da ordem de 21 MIPS (HWANG[23], GEHRINGER *et alii*[24]) na linha Balance 21000.

VII.2.2. Encore

O sistema Multimax produzido pela Encore apresenta arquitetura bastante semelhante aos sistemas de Sequent, como pode ser verificado pela figura **VII.3.**

Suas unidades de processamento são baseadas no microprocessador NS32332 da National. Cada placa possui 2 microprocessadores além de memória *cache* (64 *Kbytes* com *write-through*), gerência e mecanismos de proteção de memória (NS32382) . O mecanismo para manutenção de consistência da *cache* é baseado no monitoramento de barramento.

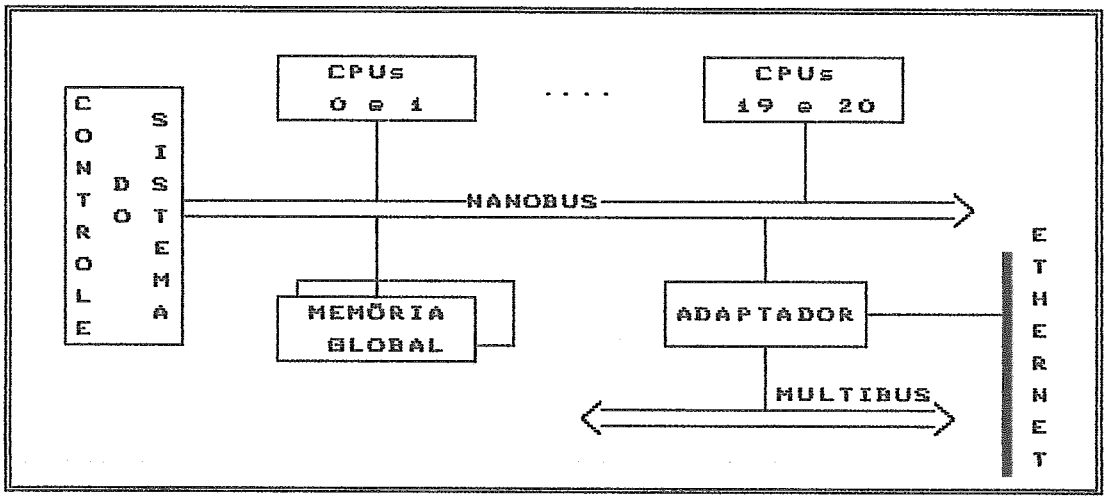


FIGURA VII.3 - ENCORE / MULTIMAX

A memória primária pode atingir até 128 *Mbytes* divididos em módulos de 16 *Mbytes* com bancos entrelaçados e sistema de detecção e correção de erros. Todo o espaço pode ser integralmente compartilhado por todas as unidades de processamento através do seu barramento. O barramento é proprietário (NanoBus) e suporta uma taxa de transferência máxima de 100 *Mbytes/s* com protocolo **síncrono** e largura de 64 *bits* para dados e 32 *bits* para endereços não multiplexados.

As operações de E/S são realizadas através de uma interface especial capaz de controlar barramentos secundários como SCSI e ETHERNET. A sincronização e comunicação entre processos emprega a técnica de Regiões Compartilhadas.

O sistema admite um número máximo de 10 unidades de processamento operando em paralelo e apresenta um nível de desempenho da ordem de 40 MIPS (GEHRINGER[24]).

VII.2.3. Alliant

A linha FX produzida pela Alliant possui a arquitetura mais complexa dentre os sistemas comerciais com topologia baseada em barramento compartilhado (fig. VII.4).

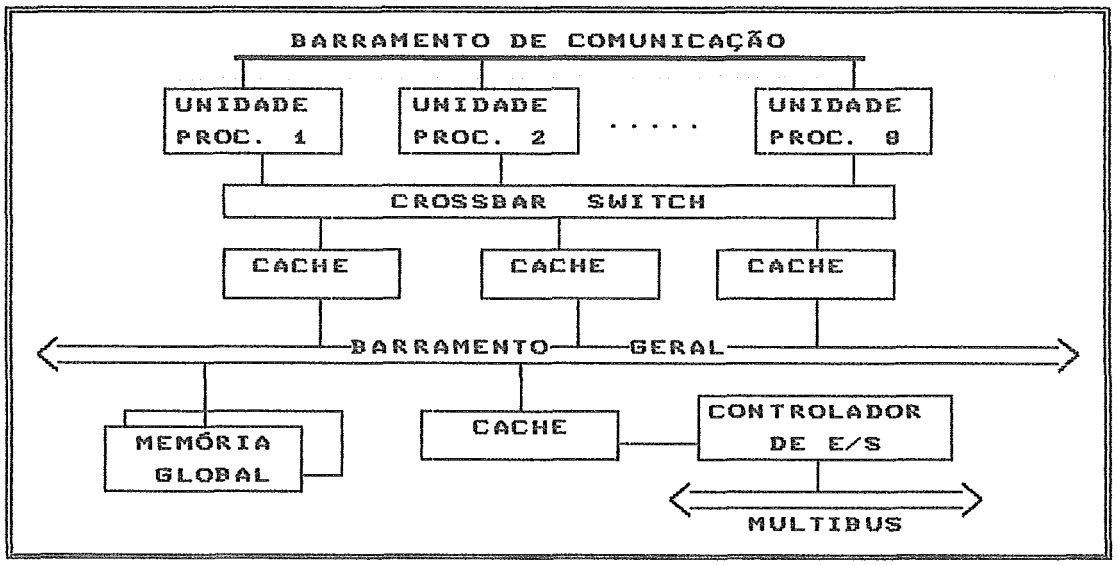


FIGURA VII.4 - ALLIANT - FX/8

O sistema é composto por até 8 unidades de processamento que compartilham um conjunto de memórias *cache* por meio de um sistema de controle de acesso empregando comutação do tipo CROSSBAR. O processador usado nas unidades de processamento é proprietário (construído em *gate-arrays*). Apresenta o mesmo modelo de programação e conjunto de instruções do microprocessador MC68020 da Motorola. Possui entretanto, características adicionais que o torna mais apropriado ao processamento paralelo. Suas características adicionais incluem capacidade para processamento vetorial e sincronização de processos através de instruções especiais que gastam apenas 3 ciclos de

máquina (BOND[25]).

A memória global é compartilhada por todas as unidades (inclusive controladores de E/S) através de um barramento proprietário. Um outro barramento é responsável pelo tráfego das insformações necessárias à sincronização e comunicação entre as unidades de processamento. O barramento MultiBus é usado para interligação de periféricos aos módulos de entrada e saída.

As características adicionais incorporadas às unidades de processamento permitem a execução de tarefas com paralelismo de média granularidade, conseguido com a utilização do compilador FORTRAN desenvolvido pelo fabricante. Neste caso, parte (ou todas) das unidades de processamento podem ser ser configuradas para executarem, simultaneamente, as partes de um mesmo processo. A performance de pico obtida quando todas as unidades de processamento executam um único processo pode chegar a 35.6 MIPS, em processamento escalar e cerca de 94 MFLOPS (HWANG[23]).

Os sistemas da série FX estão sendo utilizados como unidade básica de um *cluster* em desenvolvimento pela Universidade de Illinois. O *cluster* apresentará características e capacidade de processamento de um supercomputador.

VII.2.4. BBN Advanced Computers

Neste sistema, cada unidade de processamento é composta por um microprocessador MC68020 ligado a seu co-processador

aritmético (MC68881). Possui ainda um co-processador proprietário (PNC) chamado Processador Controlador de Nó, responsável pelo controle de todos os acessos à memória bem como pelo comando das unidades comutadoras. Cada unidade de processamento possui também um barramento próprio de E/S e até 4 *Mbytes* de memória local. Não existem módulos específicos de memória global. Todo o sub-sistema de memória é formado pelo conjunto de memórias locais. Quando um acesso é efetuado em um endereço acima da capacidade da memória local, o endereço é mapeado pela unidade de gerencia de memória (MC68851) e pelo PNC de forma a *cair* numa memória local de uma outra unidade de processamento.

A comunicação e sincronização de processos neste sistema são efetuadas através de regiões compartilhadas. A operação conjunta das instruções de *Test & Set* do microprocessador com o controle do PNC permitem a realização de acessos atômicos aos módulos de memória através das unidades comutadoras.

O sistema de interconexão permite que até 256 unidades de processamento possam ser interligadas. As transferências de informações entre unidades de processamento são realizadas por meio de *pacotes*. Quando uma informação requisitada pelo processador está na memória de um outro nó, o PNC local envia uma mensagem através de um *pacote* de dados ao PNC do nó que contém a informação. A informação é então enviada ao PNC local através de um outro *pacote*. A velocidade de transferência de informações em cada comutador é da ordem de 32 *Mbits/s*. Assim, um sistema com 64 unidades pode atingir uma taxa de transferência de

informações da ordem de 2048 *Mbits/s*. Em termos de desempenho de processamento, o sistema em sua configuração máxima (256 nós) pode atigir até 600 MIPS (GEHRINGER *et alli*[24]).

Apesar de apresentar um desempenho bastante superior às topologias baseadas em barramentos compartilhados, a interligação por meio de redes comutadas possui um grave problema. O número de fios necessários para interligar 100 unidades na arquitetura Butterfly por exemplo, é da ordem de 500 fios. Este elevado número de fios e consequentemente ligações, passa a ser um fator limitante para o crescimento do número de unidades de processamento.

VII.2.5. Intel Scientific Computers

Como exemplo de sistema comercial para processamento paralelo com topologia hipercúbica, escolhemos a máquina da Intel Scientific Computers (iPSC/2). Nestes sistemas podem ser interligados até 128 unidades de processamento. Cada unidade (nó) é composta por um microprocessador 80386 com seu co-processador aritimético (80387) além de 16 *Mbytes* de memória local. Cada unidade possui ainda 4 canais de DMA (*Direct Memory Access*) usados para a movimentação de grandes quantidades de dados. Dois destes canais são usados para comunicação com outros nós enquanto os outros dois são usados para comunicação com o Gerente do Sistema. O Gerente do Sistema é um nó especial responsável pela execução do sistema operacional e pelo tratamento das operações de entrada e saída.

O sistema possui dimensão máxima igual a 8. Assim, cada nó pode comunicar-se com outros 7 nós. A transferência de dados entre os nós vizinhos é efetuada pelos dois canais DMA através de um *hardware* especial responsável pelo roteamento dos dados para o nó desejado. A taxa de transferência de informações entre nós vizinhos é da ordem de 2.8 *Mbytes/s*. Toda a comunicação e sincronização de processos é realizada por meio de troca de mensagens.

O desempenho máximo alcançado pelo sistema atinge 512 MIPS e 2560 MFLOPS. Neste último caso são usadas unidades aceleradoras para processamento numérico acopladas aos nós.

VII.2.6. Sistemas Nacionais

Os esforços para o desenvolvimento de sistemas para processamento paralelo de que temos conhecimento no Brasil não têm ido além de algumas propostas, por parte das indústrias e alguns protótipos de laboratório, por parte de centros de pesquisa.

A nível comercial existe uma proposta da Cobra Computadores S.A. (RODRIGUES & SILVA[26]) para a construção de um sistema de múltiplos microprocessadores com topologia baseada em barramento compartilhado. As unidades de processamento são compostas por microprocessadore de 32 *bits* (atualmente Motorola) com co-processadores aritméticos (MC68881) e unidade de gerenciamento e proteção de memória (MC68851). Dispõe ainda de memória *cache* com mapeamento direto. A *cache* possui 16 *Kbytes* com 16 *bytes* por linha. Um sistema proprietário de monitoramento do barramento

assegura a consistência dos dados da *cache*.

A memória primária (global) possui bancos com organização entrelaçada e permite o mapeamento de até 4 *Gigabytes*. A unidade de gerenciamento utiliza o esquema paginado com páginas de 1 *Kbytes*. O barramento do sistema obedece ao padrão VME (assíncrono) e apresenta uma taxa máxima de transferência da ordem de 25 *Mbytes/s*. Não dispomos de informações quanto ao número máximo de unidades de processamento admitidas bem como o desempenho alcançado.

Outros sistemas de que temos conhecimento em nível de protótipo de laboratório ou em fase de desenvolvimento atualmente no Brasil são:

.Minissupercomputador - Sistema MS8701
Escola Politécnica da Universidade de São Paulo

.Processador Paralelo P3
CPqD - TELEBRÁS

.Máquina Paralela Híbrida
COPPE/Sistemas U.F.R.J.

.Máquina de Cálculo Coordenado
COPPE/Sistemas U.F.R.J.

CAPÍTULO VIII

CONSIDERAÇÕES FINAIS E CONCLUSÕES

A demanda crescente por máquinas com maior poder computacional tem obrigado aos projetistas de sistemas computacionais a procurar alternativas ao simples aumento da velocidade nominal dos processadores. Neste contexto, o processamento paralelo tem se destacado como uma das alternativas que vem apresentando os melhores resultados.

Os microprocessadores de 32 *bits* por outro lado, vêm encontrando um vasto campo de aplicações para sua utilização. Seu sucesso pode ser atribuído às seguintes características:

.Alta Velocidade de processamento

Obtida com os avanços da tecnologia VLSI e com a utilização paralelismo ao nível de unidades funcionais.

.Baixo Custo unitário

Conseguido através de técnicas de projeto estruturado e alta produção.

.Alta Capacidade de processamento

Resultante da utilização de barramentos de quatro *bytes* de largura e da incorporação de unidades funcionais

mais complexas (ponto-flutuante, *cache*, gerência de memória, etc.).

.Facilidade de Programação

Devido aos conjuntos de instruções que apresentam características importantes como ortogonalidade e simetria.

Deste modo, a utilização dos microprocessadores de 32 *bits* em sistemas para processamento paralelo tem demonstrado o alto potencial destes dispositivos através de máquinas comerciais com uma excelente relação entre o desempenho e o custo.

A arquitetura dos microprocessadores atuais de 32 *bits*, ainda bastante atrelada ao modelo tradicional Von Neumann, tem limitado sua utilização a sistemas com arquitetura de múltiplos processadores com topologia baseada principalmente em barramentos compartilhados. Podem ser encontrados também outros tipos arquiteturas e topologias porém, o grau de complexidade envolvido no *hardware* e *software* desses sistemas não tem encorajado a utilização dos microprocessadores. Além disso, o compromisso da compatibilidade de *software* com os sistemas monoprocessados tem obrigado, pelo menos no momento, a utilização das arquiteturas com paralelismo ao nível de múltiplos processadores operando como unidades autônomas.

VIII.1. Conclusões Finais

Como resultado deste trabalho destacamos duas importantes conclusões. Em primeiro lugar, os microprocessadores *CISC* de 32 *bits* da geração atual **não** estão perfeitamente adaptados à utilização em sistemas para processamento paralelo. Conseguem um sucesso relativo (significativo ante a demanda por poder computacional) numa faixa bastante restrita de sistemas paralelos. Normalmente têm sido empregados em arquiteturas de múltiplos processadores com topologia baseada em barramentos compartilhados. Estes sistemas apresentam graves limitações quanto a granularidade de paralelismo permitida e quanto ao número máximo de unidades computacionais operando em paralelo. Este número, como vimos, é determinado pela capacidade de transferência de informações apresentada pelos barramentos. Todos os sistemas comerciais para processamento paralelo que utilizam os microprocessadores de 32 *bits* apresentam algum tipo particular de adaptação para torná-los mais eficientes. As adaptações vão desde barramentos proprietários até controladores especiais (também proprietários) responsáveis pela implementação de esquemas mais eficientes para funcionamento de módulos como *cache*, gerência de memória, unidades comutadoras, etc.

Em segundo lugar, não nos parece haver no momento, uma preocupação perceptível por parte dos fabricantes de microprocessadores com relação ao suporte às arquiteturas e topologias para processamento paralelo. Os fabricantes têm concentrado maiores esforços no aumento da velocidade

nominal dos microprocessadores. É fato entretanto, que há um limite para a velocidade nominal de processamento, intrínseco a cada tecnologia de fabricação de componentes. Atingido este limite, uma opção bastante atraente será dotar os microprocessadores de maiores facilidades para o processamento paralelo. Os sistemas já contruídos e o *software* disponível deverão corroborar para a adoção desta opção pelos fabricantes de microprocessadores.

A análise das características dos quatro microprocessadores permitiu-nos concluir ainda que a arquitetura apresentada pelos microprocessadores da Motorola (MC68020 e MC68030) é a que oferece melhor suporte aos sistemas para processamento paralelo, principalmente no que diz respeito às facilidades oferecidas para adaptação de estruturas auxiliares através de sua interface para co-processadores. Os outros microprocessadores apresentam alguns pontos em que se destacam em relação às máquinas da Motorola. O sistema de tradução de endereços do 80386 da Intel por exemplo, nos parece melhor do que o apresentado pelo MC68030.

VIII.2. Direções Futuras e Sugestões para Outros Trabalhos

A utilização de unidades co-processadoras especiais nos parece a melhor alternativa para a adaptação dos atuais microprocessadores de 32 *bits* às diversas formas de paralelismo existentes. Neste caso, é imperativo que a interface para co-processadores apresentada pelo microprocessador ofereça uma forma de operação simples e

permita a ligação de um número variável de co-processadores de diferentes tipos. Deste modo, os microprocessadores poderão oferecer opções para processamento vetorial e matricial através de unidades co-processadoras. Poderão existir ainda unidades co-processadoras especializadas em comutação para a transferência de *pacotes* de informações (topologias de redes comutadas) ou especializadas em transferências de informações a altas velocidades com algoritmos de roteamento para um grande número de canais (topologias hipercúbicas). As interfaces para co-processadores apresentadas pelos microprocessadores da Motorola e National são as mais adequadas a adaptação de unidades especiais.

Para trabalhos futuros sugerimos uma análise semelhante a que desenvolvemos aqui porém, apresentando as características dos processadores com filosofia *RISC* relacionadas com o suporte ao processamento paralelo. Seria igualmente interessante a comparação entre suporte oferecido ao processamento paralelo pelos microprocessadores *CISC* e *RISC* de forma a identificar a filosofia que melhor se adapte (ou possa ser adaptada) ao processamento paralelo.

. REFERÊNCIAS BIBLIOGRÁFICAS:

[1] ENSLOW JR., P.H., "Multiprocessor Organization - A Survey", ACM Computing Surveys, vol. 9, nº 1, pp. 103-129, 1977.

[2] PAKER, Y., Multi-microprocessor Systems, Academic Press Inc., London, 1983.

[3] ENSLOW JR., P.H., "Multiprocessor and Parallel Processing", Proceedings of the 1976 International Conference on Parallel Processing, IEEE, 1976.

[4] HAYNES, L.S. *et alii*, "A Survey of Highly Parallel Computing", IEEE Computer, vol. 15, nº 1, pp. 9-24, 1982.

[5] THURBER, K.J. & WALD, L.D., "Associative and Parallel Processing", ACM Computing Surveys", vol. 7, nº 4, pp. 215-255, 1975.

[6] WATSON, I. & GURD, J., "A Practical Data Flow Computer", IEEE Computer, vol. 15, nº 2, pp. 51-57, 1982.

[7] GAJSKY, D.D., "A Second Opinion on Data Flow Machines and Languages", IEEE Computer, vol. 15, nº 2, pp. 58-69, 1982.

[8] WALLLICH, P. & ZORPETTE, G., "Minis and Mainframes", IEEE Spectrum, vol. 3, nº 1, pp. 36-39, 1986.

[9] KUNG, H.T., "Why Sistolic Architectures ?", IEEE Computer, vol. 15, nº 1, pp. 37-46, 1982.

[10] MOKHOFF, N., "Parallelism Breads a New Class od Supercomputers", Computer Design, vol. 26, nº 6, pp. 53-64, 1987.

[11] NG, K.W., "The High Level Language and Operating

System Support Features of Advanced Microprocessors - Part I", Microprocessing and Microprogramming, vol. 19, nº 3, pp. 203-218, 1987.

[12] BEIMS, B., "Multiprocessing Capabilities of the MC68020 32 Bit Microprocessor", AR220, Motorola Inc., Austin, Texas, 1984.

[13] SMITH, A.J., "Cache Memory Design: an Evolving Art", IEEE Spectrum, vol. 24, nº 12, pp. 40-44, 1987.

[14] SMITH, A.J., "Cache Memories", Computing Surveys, vol. 14, nº 3, pp. 473-530, 1982.

[15] GROCHOWSKI, E., "80386 Cache Memory Example", AP-404, Intel Corp., Santa Clara, California, 1987.

[16] PHILLIPS, D., "The Z80000 Microprocessor", IEEE Micro, vol. 5, nº 6, pp. 23-36, 1985.

[17] PATEL, A., "Z80000, 32 Bit Microprocessor", AFIPS Conf. Proc., vol. 54, pp. 225-231, 1985.

[18] FUHRT, B. & MILUTINOVIC, V., "A Survey of Microprocessor Architectures for Memory Menagement", IEEE Computer, vol. 20, nº 3, pp. 48-67, 1987.

[19] HENSLER, C. & SARNO, K., "Marrying UNIX and the 80386", Byte, vol. 13, nº 4, pp. 237-244, 1988.

[20] HWANG, K., "Advanced Parallel Processing with Super Computer Architectures", Proceedings of the IEEE, vol. 75, nº 10, pp. 1348-1379, 1987.

[21] STEINOVIC, J.A., "Software Communication Mechanisms Procedure Calls versus Messages", IEEE Computer, vol. 15, nº 4, pp. 19-25, 1982.

[22] MANUEL, T., "How Sequent's New Model Outruns most Mainframes", Electronics, vol. 60, nº 11, pp.

76-79, 1987.

[23] SEITZ, C.L., "The Cosmic Cube", Communications of the ACM, vol. 28, nº 1, pp. 22-33, 1985.

[24] GEHRINGER, E.F. *et alii*, "A Survey of Commercial Parallel Processors", ACM Computer Architecture News, vol. 16, nº 4, pp. 75-107, 1988.

[25] BOND, J., "Parallel Processing Concepts Finally Come Together in Real Sysytems", Computer Design, vol. 26, nº 11, pp. 51-74, 1987.

[26] RODRIGUES, R. & SILVA, G.P., "Um Supermini com Arquitetura Baseada em Múltiplos Microprocessadores de 32-Bits", Anais do XIX Congresso Nacional de Informática, pp. 131-135, Rio de Janeiro, 1986.

[27] GURD, J.R. *et alii*, "The Manchester Prototype Data Flow Computer", Communications of the ACM, vol. 28, nº 1, pp. 34-52, 1985.

[28] MENDELSON, B. & SIBERMAN, G.M., "Mapping Data Flow Programs on VLSI Array of Processors", ACM Computer Architecture News, vol. 15, nº 2, pp. 72-80, 1987.

[29] NG, K.W., "The High Level Language and Operating System Support Features of Advanced Microprocessors - Part II", Microprocessing and Microprogramming, vol. 19, nº 4, pp. 277-289, 1987.

[30] JACKSON, A.S., "A Basic Comparison of the MC68020, iAPX286/386, NS32032, DEC PDP11 & DEC VAX", BR330, Motorola Inc., 1985.

[31] HEERING, J., "The Intel 8086, the Zilog Z8000 and the Motorola MC68000 Microprocessors", Euromicro Journal, vol. 6, nº 6, pp. 135-143, 1980. 1980.

[32] WILSON JR., A.W., "Hierarquical Cache/Bus

Architecture for Shared Memory Multiprocessors", ACM Computer Architecture News, vol. 15, nº 2, pp. 244-252, 1987.

[33] YEN, W. *et alii.*, "Data Coherence Problem in a Multicache System", IEEE Transactions on Computers, vol. 34, nº 1, pp. 56-65, 1985.

[34] CHEUNG, K. *et alii*, "Organization and Analisis of a Gracefully-Degrading Interleaved Memory System", ACM Architecture News, vol. 15, nº 2, pp. 224-231, 1987.

[35] HYDE, R.L., "Overview of Memory Menagement", Byte, vol. 13, nº 4, pp. 219-225, 1988.

[36] WILKES, M.V., "Hardware Support for Memory Protection: Capacity Implementations", ACM SIGPLAN Notices, vol. 17, nº 4, pp. 107-116, 1982.

[37] INMOS, IMS T800 Transputer - Preliminary Data, Bristol, 1987.

[38] HAYES, J.P. *et alii*, "A Microprocessor-based Hypercube Supercomputer", IEEE Micro, vol. 6, nº 5, pp. 18-31, 1986.

[39] GAUDIOT, J.L. *et alii*, "The TX16: A Highly Programmable Multi-microprocessor Architecture", IEEE Micro, vol. 6, nº 5, pp.18-31, 1986.

[40] SEITZ, C.L., "Concurrent VLSI Architectures", IEEE Transactions on Computers, vol. 33, nº 12, pp. 1247-1265, 1984.

[41] JONES, A.K. & SCHWARZ, P., "Experience Using Multi-Microprocessor Systems - A Status Report", ACM Computing Surveys, vol. 12, nº 2, pp. 121-165, 1980.

[42] GHOSAL, D. & PATNAIK, L.M., "SHAMP: An Experimental Shared Memory Multimicroprocessor System for Performance Evaluation of Parallel Algorithms", Microprocessing and Microprogramming, vol. 19, nº 3, pp. 179-192, 1987.

[43] RAMACHANDRAN, U. *et alii*, "Hardware Support for Interprocess Communication", ACM Computer Architecture News, vol. 15, nº 2, pp. 178-188, 1987.

[44] TRIPATHI, S.K. *et alii*, "Report on the Workshop on Design & Performance Issues in Parallel Architectures", ACM Sigmetrics Performance Evaluation Review, vol. 14, nº 3, pp. 16-32, 1987.

[45] FIELLAND, G. & RODGERS, D., "32 Bit Computer System Shares Load Equally among up to 12 Processors", Electronic Design, vol. 32, nº 18, pp. 153-168, 1984.

[46] EL-AYAT, K. & AGARWAL, R.K., "The Intel 80386 - Architecture and Implementation", IEEE Micro, vol. 5, nº 6, pp. 4-22, 1985.

[47] SPERRY, T., "386 Vs 030: The Crowded Fast Lane", Dr. Dobb's Journal, vol. 13, nº 135, pp. 16-22, 1988.

[48] GROSSMAN, F., "Debbuging with the 80386", Dr. Dobb's Journal, vol. 13, nº 136, pp. 18-28 , 1988.

[49] CROSSWY, C.S. & PERES, M., "Upward to the 80386", PC Tech Journal, vol. 6, nº 2, pp. 51-66, 1987.

[50] GREHAN, R., "A Closer Look", Byte, vol. 12, nº 10, pp. 110-111, 1987.

[51] BYTE EDITORIAL STAFF, "High Tech Horse Power", Byte, vol. 12, nº 8, pp. 101-108, Extra Edition, 1987.

[52] BARNUM, J., "286/386 Protected Mode Programming", Byte, vol. 12, nº 12, pp. 125-129, 1987.

[53] MOTOROLA, Second Generation 32 Bit Enhanced Microprocessor - BR 508/D, Phoenix, Arizona, 1986.

[54] BEIMS, B., "The MC68020 32 Bit MPU: Openning New Applications Doors", AR232, Motorola Inc., Austin, Texas, 1985.

[55] COOPER, T.C. *et alii*, "A Benchmark Comparizon of the 32 Bit Microprocessors", IEEE Micro, vol. 6, nº 4, pp. 53-58, 1986.

[56] MOTOROLA, Motorola MC68020 Benchmark Report - BR322, Phoenix, Arizona, 1986.

[57] McGREGOR, D. *et alii*, "The Motorola MC68020", IEEE Micro, vol. 4, nº 4, pp. 101-118, 1984.

[58] KUBAN JR. & SALICK, J.E., "Testing Approaches in the MC68020", AR225, Motorola Inc., Phoenix, Arizona, 1985.

[59] INTEL, 80386 Hardware Reference Manual, Santa Clara, California, 1986.

[60] MORSE, S.P. & ALBERT, D.J., The 80286 Architecture, John Wiley & Sons Inc. - Wiley Press, New York, N.Y., 1986.

[61] NATIONAL, Series 32000 Data Book, Santa Clara, California, 1984.

[62] NATIONAL, The Specifics of 32-Bit Architecture and Implemantation - Series 32000, Santa Clara, California, 1984.

[63] Motorola MC68020 32 Bit Microprocessor User's Manual, Prentice-Hall Inc., Englewood Clifs, N.J., 1984.

[64] Enhanced 32-Bit Microprocessor User's Manual -

MC68030, Prentice-Hall Inc., Englewood Cliffs, N.J., 1988.

[65] SCANLON, L.J., The 68000: Principles and Programming, Howard W. Sams & Co. Inc., Indianápolis, Indiana, 1981.

[66] FREEMAN, M & KAPLINSKY, C., "Modeling the Resources of a System Demonstrates Memory Controller's Power", Electronic Design, vol. 32, nº 19, pp. 205-216, 1984.

[67] GARCIA MOLINA, *et alii*, "Performance Through Memory", Performance Evaluation Review, vol. 15, nº 1, pp. 122-131, 1987.

[68] RUDOLPH, L. & SEGALL, Z., "Dinamic Decentralized Cache Schemes for MIMD Parallel Processors", Canegie Mellon University, 1984.

[69] CHANG, J.H.; CHAO, H. & SO, K., "Cache Design of Sub-micron CMOS System/370", ACM Computer Architecture News, vol. 15, nº 2, pp. 208-213, 1987.

[70] COLE, B. C., "How a Cache Control Chip Supercharges 386 Processor", Electronics, vol. 60, nº12, pp. 74-79, 1987.

[71] COLE, B. C., "Advanced Cache Chips Make the 32-Bit Microprocessor Fly", Electronics, vol. 60, nº 12, pp. 78-79, 1987.

[72] SCHEURICH, C. & DUBOIS, M., "Correct Memory Operation of Cache Based Multiprocessors", ACM Computer Architecture News, vol. 15, nº 2, pp. 234-243, 1987.

[73] LEE, R.L.; YEW, P.C.; LAWRIE, D.H., "Multiprocessor

Cache Design Considerations", ACM Computer Architecture News, vol. 15, nº 2, pp. 253-262, 1987.

[74] DAREMA-ROGERS, F.; PFISTER, G.F. & SO, K., "Memory Access Patterns of Parallel Scientific Programs", ACM Performance Evaluation Review, vol. 15, nº 1, pp. 46-58, 1987.

[75] BRIL, R.J., "An Implementation Independent Approach of Cache Memories", ACM Computer Architecture News, vol. 15, nº 3, pp. 17-24, 1987.

[76] BRIL, R.J., "On Cacheability of Lock-Variables in Tightly Coupled Multiprocessor Systems", ACM Computer Achitecture News, vol. 15, nº 3, pp. 25-32, 1987.

[77] INTEL, 82385 High Performance 32-Bit Cache Controller - Architectural Overview - Santa Clara, California, 1987.

[78] STANSBERRY, M., "Cache Memory Design in 32-Bit Microprocessor Systems", VLSI Systems Design, vol. 9, nº 3, pp. 32-42, 1988.

[79] CRUESS, M.W., "Memory Menagement Chip for 68020 Tranlates Addresses in Less then a Clock Cicle", Electronic Design, vol. 34, nº 11, pp. 151-161, 1986.

[80] FREEMAN, M., "An Architectural Persperctive on A Memory Access Controller", ACM Computer Architecture News, vol. 15, nº 2, pp. 214-223, 1987.

[81] Design Entry, "Memory Controller Gives a Microprocessor a Big Mini's Throughput", Electronic

Design, vol. 32, nº 17, pp. 153-164, 1984.

[82] WULF, W.A., "Compilers and Computer Architecture", Computer Magazine, vol. 14, nº 7, pp. 41-47, 1981.

[83] BARISHANSKY, J., "UNIX Moves into New Hardware", UNIX/World, vol. 3, nº 6, pp. 26-38, 1986.

[84] RUSSEL, C. & WATERMAN, P.J., "Variations on UNIX for Parallel Processing Computers", Communications of the ACM, vol. 30, nº 12, pp. 1048-1055, 1987.

[85] CURRAN, L., "Here Comes High-Powered UNIX for Multiple CPU's", Electronics, vol. 60, nº 22, pp. 77-79, 1987.

[86] RITCHIE, D. & THOMPSON, K., "The UNIX Time Sharing System", Communications of the ACM, vol. 17, nº 7, pp. 365-375, 1974.

[87] BEIMS, B., "The MC68020 and System V/68", AR219, Motorola Inc., Austin, Texas, 1984.

[88] JACOBS, M. & TEST, J.A., "The UNIX System Adapts to a Parallel Processing Environment", UNIX/World, vol. 3, nº 6, pp. 48-63, 1986.

[89] DITZEL, D.R. & McLELLAN, H.R., "Register Allocation for Free: The C Machine Stack Cache", ACM SIGPLAN Notices, vol. 17, nº 4, pp. 48-56, 1982.

[90] DAHBURA, A.T. & SABNANI, K.K., "Performance Analisis of a Fault Detection Scheme in Multiprocessor Systems", Performance Evaluation Review, vol. 15, nº 1, pp. 143-154, 1987.

[91] BBN, Butterfly(TM) Parallel Processor - Overview, BBN Laboratories Inc., 1985.

[92] FALLER, N. *et alii*, "Técnicas de Projeto Utilizadas na Construção do Supermicro PEGASUS-32X e do

Sistema Operacional Plurix", Data News, nº 269, pp. 12-16, 1985.

[93] FALLER, N. & SALENBAUCH, P., "Plurix o Sistema Operacional Multiprocessador do NCE/UFRJ: Sincronização de Processos", Data News, nº 290, pp. 26-35, 1985.

[94] ZUFFO, J.A. *et alii*, "Projeto Minissupercomputador: Características Gerais do Sistema MS8701", II Simpósio Brasileiro de Arquitetura de Computadores - Processamento Paralelo, vol. 2, pp. 10.1.1-10.1.8, 1988.

[95] PESTANA, A. & CAVALLI, E., "Processador Paralelo P3", II Simpósio Brasileiro de Arquitetura de Computadores - Processamento Paralelo, vol. 2, pp. 10.2.1-10.2.8, 1988.

[96] FIGUEIRA, N.R., "Cache para Discos: Arquiteturas e Algoritmos", Tese Ms.C.-COPPE/UFRJ, Engenharia de Sistemas e COmputação, 1988.